

Neurális hálózatok

Altrichter, Márta

Horváth, Gábor

Pataki, Béla

Strausz, György

Takács, Gábor

Valyon, József

Neurális hálózatok

Altrichter, Márta

Horváth, Gábor

Pataki, Béla

Strausz, György

Takács, Gábor

Valyon, József

Ez a könyv az Oktatási Minisztérium támogatásával, a Felsőoktatási Tankönyv- és Szakkönyv-támogatási Pályázat keretében jelent meg.

Publication date 2006

Szerzői jog © 2006 Hungarian Edition Panem Könyvkiadó Kft., Budapest

A kiadásért felel a Panem Kft. ügyvezetője, Budapest, 2006

Minden jog fenntartva. Jelen könyvet, illetve annak részeit tilos reprodukálni, adatrögzítő rendszerben tárolni, bármilyen formában vagy eszközzel - elektronikus úton vagy más módon - közölni a kiadók engedélye nélkül

Tartalom

Előszó	xiv
Bevezetés	xvii
1. A neurális hálózatok felépítése, képességei	1
1. A neurális hálózat definíciója, működése	1
2. A neurális hálózat elemei, topológiája	2
2.1. A neuronok felépítése	2
2.2. A neurális hálózatban használt összeköttetések	7
2.3. A neurális hálózatok topológiája	7
3. A neurális hálózatok alapvető számítási képességei, felhasználási területei	11
3.1. A neurális háló, mint approximáló rendszer	11
3.2. A neurális hálózat, mint asszociatív memória	13
3.3. A neurális háló, mint optimalizáló rendszer	13
4. A neurális hálózatok approximációs képessége	14
4.1. Matematikai leképezések közelítése neurális hálóval, a probléma megfogalmazása	14
4.2. Két nemlineáris réteget használó approximációs hálózatok	14
4.3. Egy nemlineáris réteget használó approximációs hálózatok	16
2. Tanulás adatokból	19
1. Ellenőrzött tanulás (tanítóval történő tanítás)	21
2. Nemellenőrzött tanulás	29
3. A statisztikus tanuláselmélet alapjai	30
3.1. Az ERM konzisztenciája	30
3.2. Strukturális kockázat minimalizálás	37
4. Tanulás és statisztikai becslések	39
5. Determinisztikus és sztochasztikus szélsőérték-kereső eljárások	43
5.1. Gradiens alapú szélsőérték-kereső eljárások	44
5.1.1. Szélsőérték-keresés paramétereiben lineáris modellek esetén	45
5.1.2. A négyzetes kritériumfüggvény tulajdonságai	46
5.1.3. Szélsőérték-keresés Newton módszerrel	47
5.1.4. Szélsőérték-keresés a "legmeredekebb lejtő" (steepest descent) módszerével	48
5.1.5. A konjugált gradiens módszer	50
5.1.6. Az LMS algoritmus és néhány változata	53
5.1.7. Az LMS/Newton algoritmus	55
5.1.8. Transzformált tartománybeli LMS eljárás	56
5.1.9. Szélsőérték-keresés paramétereiben nemlineáris modellek esetén	59
5.1.10. További gradiens-alapú módszerek	60
5.1.11. Feltételes szélsőérték-kereső eljárások	61
5.2. Sztochasztikus szélsőérték-kereső eljárások	62
5.2.1. Véletlen keresés	63
5.2.2. Genetikus algoritmusok	64
5.2.3. A szkéma elmélet	68
3. Az elemi neuron	72
1. A Rosenblatt perceptron	72
1.1. A perceptron tanulása	73
1.2. A perceptron tanulás konvergenciája	75
1.3. A perceptron kapacitása	77
2. Az adaline	78
2.1. Az adaline tanítása	78
3. Egy processzáló elem szigmoid kimeneti nemlinearitással	80
4. További elemi neuronok	82
4. A többrétegű perceptron (MLP)	84
1. Az MLP felépítése	84
2. Az MLP tanítása, a hibavisszaterjesztéses algoritmus	85
3. Az MLP konstrukciójának általános kérdései	88
3.1. A hálózat méretének megválasztása	89
3.2. A tanulási tényező és a súlyok kezdeti értékeinek megválasztása	92
3.2.1. A tanulási tényező, μ megválasztása	92

3.2.2. A súlyok kezdeti értékeinek meghatározása	93
3.3. A tanító pontok felhasználása, a háló minősítése	94
3.3.1. A tanító pontok felhasználása	94
3.3.2. A háló minősítése	94
3.4. A hibavisszaterjesztési algoritmus változatai	99
3.4.1. Momentum módszer	99
5. Bázisfüggvényes hálózatok	102
1. Az RBF (Radiális Bázisfüggvényes) hálózat	103
1.1. Középpont kiválasztó eljárások	106
1.1.1. Az OLS eljárás	106
1.1.2. A K-közép eljárás	110
1.1.3. A középpontok ellenőrzött tanítása	111
1.2. A szélességparaméter meghatározása	111
1.3. RBF változatok	113
1.4. A kizáró VAGY (XOR) probléma megoldása RBF hálózattal	114
2. A CMAC hálózat	116
2.1. A háló komplexitása	120
2.2. A CMAC hálózat fontosabb tulajdonságai	123
2.2.1. A háló modellező képessége	123
2.3. A CMAC háló tanítása	131
2.4. CMAC változatok	133
2.4.1. Magasabbrendű CMAC hálózatok	135
2.4.2. Kredit hozzárendelés	137
2.4.3. Komplexitás csökkentés többdimenziós esetben	137
3. Az MLP és a bázisfüggvényes hálózatok összehasonlítása	139
6. Kernel módszerek	144
1. Egy egyszerű kernel gép	145
2. Kernel függvények	148
3. Szupport Vektor Gépek	149
3.1. Lineárisan szeparálható feladat lineáris megoldása	150
3.1.1. A maximális margójú lineáris szeparálás általánosítóképessége	153
3.2. A lineárisan nem szeparálható feladat lineáris megoldása	155
3.2.1. A szoft margójú lineáris szeparálás általánosító képessége	156
3.3. Nemlineáris szeparálás	157
3.4. Szupport vektor gépek regressziós feladatra	160
3.4.1. Az SVM működése	164
3.5. Az SVM neurális értelmezése	165
3.6. Az SVM hatékonyabb megvalósításai	166
3.6.1. A QP hatékonyabb megoldása	166
3.6.2. Az SVM módosítása	167
4. SVM változatok	168
4.1. Az LS-SVM	168
4.1.1. Az LS-SVM osztályozó	168
4.1.2. Az LS-SVM regresszió	170
4.1.3. Ritka LS-SVM	172
4.1.4. Súlyozott LS-SVM	173
4.2. Az LS-SVM hatékonyabb megoldása	175
4.2.1. A számítás gyorsítása	175
4.3. LS ² -SVM	175
4.3.1. Túlhatározott egyenletrendszer	175
4.3.2. A kiválasztási eljárás	178
4.3.3. Súlyozott LS ² -SVM	180
4.4. Ridge regresszió	180
4.4.1. Lineáris ridge regresszió	181
4.4.2. Nemlineáris kernel ridge regresszió	182
4.4.3. Csökkentett rangú kernel ridge regresszió	183
5. Kernel CMAC: egy LS-SVM gép véges tartójú kernel függvényekkel	185
5.1. Kernel CMAC súlykiegyenlítő regularizációval	187
6. A kernel gépek összefoglaló értékelése	188
7. Ellenőrzött tanítású statikus hálók alkalmazásai	192

1. Felismerési feladatok (képosztályozás)	193
1.1. Számjegyfelismerés speciális struktúrájú MLP-vel	193
1.2. Számjegyfelismerés egyrétegű hálózattal	194
1.3. Számjegyfelismerés MLP-vel, újabb eredmények	196
2. Szövegosztályozás	198
2.1. Spam szűrés SVM-mel	199
3. Ipari folyamatok modellezése	200
3.1. Gumigyártási folyamat modellezése RBF-fel	200
3.2. Acélgártási folyamat modellezése	202
3.3. Néhány egyéb alkalmazás	205
8. Időfüggő (szekvenciális) hálók	207
1. Regresszorválasztás, modellstruktúra választás	208
2. Dinamikus neurális modellek	209
3. Előrecsatolt időfüggő hálózatok	212
3.1. A FIR-MLP háló	212
3.1.1. A FIR-MLP háló tanítása	212
3.1.2. Temporális hibavisszaterjesztési eljárás	213
3.2. Egyéb NFIR hálóstruktúrák	217
4. Visszacsatolt (rekurzív) hálózatok	218
4.1. Rekurzív háló időbeli kiterítése	218
4.2. Rekurzív hálók tanítása pillanatnyi gradiens alapján	221
5. Dinamikus hálózatok kialakításának néhány további lehetősége	225
5.1. A parciális deriváltak számítása	225
5.2. Neurális hálózatot tartalmazó összetett struktúrák tanítása	227
6. Dinamikus hálók alkalmazása	230
6.1. Identifikáció, rendszermodellezés	230
6.1.1. Struktúraválasztás dinamikus neurális hálózatoknál	232
6.1.2. Kísérlettervezés	238
6.1.3. A modellezés további lépései	240
6.2. Szabályozás, modell alapú adaptív szabályozás	240
6.3. Néhány tipikus alkalmazási terület	241
6.3.1. Robotika	242
6.3.2. Előrejelzés	242
9. Moduláris hálók	248
1. Moduláris háló kialakítása feladat dekompozíció alapján	249
2. Szakértőegysétek	255
2.1. Hálók optimális lineáris kombinációja	255
2.1.1. Az egyedi hálók kiválasztása	257
2.1.2. Az integráló réteg súlyainak meghatározása	258
2.2. Pontos és különböző szakértők együttese	258
2.3. MOE (Mixture Of Experts) architektúra	260
2.3.1. A MOE tanítása	262
2.3.2. Tanítás valószínűségi értelmezés alapján	263
2.3.3. A MOE konstrukciója	265
3. Moduláris háló kialakítása a tanító mintakészlet módosításával	265
3.1. Boosting	266
3.1.1. Boosting szűréssel	266
3.1.2. AdaBoost	267
3.1.3. A boosting eljárások minősítése	269
4. A moduláris eljárások összefoglaló értékelése	270
10. Nemellenőrzött tanulású hálózatok	272
1. Hebb tanulás	273
2. Versengő tanulás	274
3. Kohonen háló, kompetitív hálózatok	275
3.1. A Kohonen háló tanítása	276
3.2. Felügyelt tanulás alkalmazása Kohonen hálózattal: tanuló vektorkvantálás (LVQ).	279
4. Adattömörítés Hebb tanulásra alapuló hálózatokkal, PCA, KLT	281
4.1. A KL transzformáció és optimalitása	282
4.2. Az Oja szabály, a legnagyobb sajátértéknek megfelelő sajátvektor meghatározása	284
4.2.1. Az Oja algoritmus, mint szélsőérték-kereső eljárás	286

4.3. Főkomponens- és altér hálózatok	287
4.3.1. Oja altér hálózat	287
4.3.2. Főkomponens hálók	288
4.3.3. Az APEX háló	289
4.4. Lineáris többrétegű perceptron, mint adattömörítő hálózat	292
5. Nemlineáris PCA és altér hálók	293
5.1. Kernel PCA	294
5.1.1. A nulla várhatóérték biztosítása a jellemzőtérben	296
5.1.2. Jelviszaállítás	297
5.2. Nemlineáris többrétegű perceptron, mint adattömörítő hálózat	300
6. Független komponens analízis	301
6.1. Független komponens analízis nemgauss-ság alapján	304
6.2. FastICA: egy kurtózis alapú fix-pont ICA algoritmus	305
6.3. Független komponens analízis nemlineáris PCA alkalmazásával	307
11. Analitikus tanítású hálózatok	312
1. A Hopfield hálózat	312
1.1. Minták tárolása	313
1.1.1. Egy minta tárolása	314
1.1.2. Több minta tárolása	315
1.2. Az energiafüggvény	317
1.3. Az energiafüggvény felhasználása	318
2. A Boltzmann gépek	319
2.1. Problémák a Hopfield hálózat alkalmazásakor	319
2.2. Szimulált lehűtés	320
3. Mean-field hálózatok	323
4. Hopfield típusú hálózatok alkalmazása optimalizációs problémákra	325
4.1. Az utazó ügynök probléma	327
4.2. Rádiófrekvenciák kiosztása neurális hálózattal	329
4.3. A/D konverter megvalósítása Hopfield hálózattal	329
12. Hibrid-neurális rendszerek	331
1. Az a priori tudás felhasználása virtuális minták generálására	332
2. Az a priori tudás beépítése a tanuló eljárásba	333
2.1. Monoton válaszü kernel regresszió	334
2.2. Tudásalapú szupport vektor gép	334
2.3. Súlyozott margójú szupport vektor gép	337
2.3.1. A priori tudás beépítése WMSVM-be	338
3. KBANN, a tudás alapú neurális hálózat	339
3.1. Apriori tudás használata a kezdeti hipotézis kialakításában: KBANN	341
3.2. Az elméleti tudás neurális hálózatba történő konvertálása	342
3.3. Szabály→neuron konverzió	345
3.4. A KBANN háló finomítása	346
3.5. Szabályok kinyerése – finomított domén elmélet	347
3.5.1. NofM módszer	348
3.6. Új csomópontok dinamikus hozzáadása a KBANN hálózathoz: TopGen algoritmus	350
3.7. A KBANN háló topológiájának megváltoztatása genetikus módszerrel: REGENT	352
13. Gyakorlati feladatmegoldás: adatelőkészítés, lényegkiemelés	355
1. Zajos adatok	355
1.1. Nem Gauss eloszlású zajok hatása	355
1.2. Zajos adatok felhasználása az adatok megfelelő súlyozásával (EIV módszer)	357
2. Az adatok előfeldolgozása	359
2.1. Adatok hihetőségvizsgálata	359
2.2. Az adatok normalizálása	360
2.3. Diszkrét értékekkel reprezentált szimbolikus adatok kezelése	361
2.4. Invariáns tulajdonságok elérését segítő előfeldolgozás	362
3. Kilógó adatok	363
3.1. Kilógó értékek modellezése klaszterezéssel, az EM algoritmus	364
3.1.1. Az EM-algoritmus	365
4. Hiányzó adatok	369
5. Lényegkiemelés	373
5.1. A dimenziócsökkentés (lényegkiemelés) célja	373

5.2. Problémafüggetlen, általános eljárások (PCA, ICA)	375
A. Összegzés, várható fejlődési irányok	380
B. Függelék	382
1. Mátrixok és vektorok [Gol96b], [Róz91]	382
2. Gauss elimináció [Gol96b]	386
3. A reduced row echelon alak	387
4. Feltételes szélsőérték-keresés, Lagrange multiplikátoros módszer [Fle86, Boy04]	388
5. Karush-Kuhn-Tucker feltételek [Cri00], [Boy04]	389
C. Jelölések	392
D. Irodalom	398

Az ábrák listája

1.1. A neuron általános, elvi felépítése	2
1.2. Egyenrangú bemenetekkel rendelkező memória nélküli neuron (perceptron) felépítése	3
1.3. Neuronokban használt tipikus nemlinearitások	3
1.4. Bemeneti összegzést nem használó neuron	5
1.5. Példa memóriával rendelkező neuronra	5
1.6. FIR/IIR neuron struktúra	6
1.7. Egy egy-(processzáló) rétegű hálózat	8
1.8. Teljesen összekötött, visszacsatolt (a), illetve rétegekbe szervezett, előrecsatolt (b) topológiájú hálózat	9
1.9. Lokális – elemi (a), laterális (b) és rétegek közötti (c) – visszacsatolás	9
1.10. Késleltető sorokkal dinamikussá tett neurális rendszer	12
1.11. Két nemlineáris aktivációs függvényű réteget alkalmazó háló	15
2.1. A tanítópontok alapján történő függvényapproximáció egy rosszul definiált feladat: a tanítópontok nem határozzák meg egyértelműen a bemenet és a kimenet közötti függvénykapcsolatot	22
2.2. Fontosabb veszteségfüggvények: (a) négyzetes függvény, (b) abszolútérték függvény, (c) ε érzéketlenségi sávval rendelkező igen-nem függvény	24
2.3. A szabad paraméterek számának hatása a túlilleszkedésre: szinusz függvény polinomiális approximációja. (a) 18-ad fokú polinommal, (b) 5-ödfokú polinommal	25
2.4. A torzítás-variancia dilemma.	26
2.5. Az ERM elv konzisztenciája	31
2.6. A növekedésfüggvény alakulása	34
2.7. A lineáris szeparálás kapacitását bemutató kétdimenziós példa. Két dimenzióban három pont mindig szeparálható lineárisan: az első 8 elrendezés ezt mutatja, de négy pontnál ez már nem minden esetben lehetséges: ld. utolsó eset.	35
2.8. Az eredő kockázat és a tapasztalati kockázat a VC dimenzió (h) függvényében	38
2.9. A tanulás, mint paraméterbecslési eljárás	40
2.10. A maximum likelihood becslés	42
2.11. Az a priori és az a posteriori sűrűségfüggvények alakulása	43
2.12. Az izokritérium görbék az eredeti és a sajátvektor irányú koordinátarendszerekkel	47
2.13. Példa konvergenciára "legmeredekebb lejtő" módszerrel a trajektória mentén (kis μ mellett) (a), "legmeredekebb lejtő" módszerrel nagyobb μ mellett (b); a konjugált gradiensek módszerével (c).	52
2.14. A hibaingadozás alakulása négyzetes paraméterfüggés esetén	54
2.15. A hibafelület (kétdimenziós) alakulása az eredeti (a), a transzformált (b) és a transzformált-normalizált (c) esetben	58
2.16. A genetikus algoritmus szélsőérték-kereső mechanizmusa	65
2.17. A keresztezés folyamata.	67
2.18. A szkémák megjelenése az egyes generációk populációiban	70
3.1. Az egyszerű perceptron felépítése a hibaképzéssel és a paramétermódosítással	72
3.2. Súlymódosítás a perceptron tanításánál	74
3.3. A perceptron kapacitásának alakulása	77
3.4. Az adaline felépítése a hibaképzéssel és a paramétermódosítással	79
3.5. Szigmoid kimeneti nemlinearitással felépülő elemi neuron a hibaképzéssel és a paramétermódosítással	80
4.1. A többrétegű perceptron felépítése a hibaszámítással	84
4.2. Az MLP tanítása back-propagation algoritmussal	87
4.3. A hálózat processzáló elemeinek kimenete a bemenet függvényében	89
4.4. A hálózat taníthatóságának alakulása változó hálózat-méret mellett	92
4.5. Tanulási görbe: a korai leállítási szerepe a túltanulás elkerülésében	95
4.6. A hálózat leképezésének túlzott illeszkedése a tanító pontokhoz	95
4.7. A szabad paraméterek számának hatása a túltanulás hajlamra MLP: (a), (c), (e) polinomiális approximáció (b), (d), (f)	96
5.1. A bázisfüggvényes hálózatok általános felépítése	102
5.2. Az (egykimenetű) RBF hálózat felépítése	103
5.3. A σ szélességparaméter hatása az RBF approximációs képességére.	112
5.4. A XOR problémát megoldó RBF háló	115

5.5. A mintapontok elhelyezkedése: (a) a bemeneti térben, (b) a transzformált térben (a kereszttel jelölt két mintapont a transzformált térben egybeesik)	115
5.6. A bemeneti tartomány lefedése bázisfüggvényekkel egydimenziós esetben $C=4$ mellett	117
5.7. A bemeneti tér lefedése bázisfüggvényekkel kétdimenziós CMAC-nál $C=4$ mellett.	117
5.8. Egy bemeneti mintapont tér lefedése bázisfüggvényekkel	118
5.9. A CMAC hálózat leképezései	119
5.10. Tömörítést is alkalmazó CMAC hálózat leképezései	121
5.11. A tömörítő leképezés ütközései	122
5.12. A CMAC válasza $C=8$ és (a) $t=8$, (b) $t=5$ mellett	124
5.13. Az egydimenziós CMAC relatív általánosítási hibája C/t függvényében	125
5.14. A címhozzárendelés elve kétdimenziós esetben	126
5.15. A szomszédos cellák lefedése kétdimenziós bináris CMAC-nál $C=4$ mellett	128
5.16. A szabadon felvehető (árnyékolt cellák) és a kiadódó (nem árnyékolt cellák) cellánkénti leképezések egy lehetséges elrendezése $C=4$ mellett	129
5.17. A 2D CMAC háló válasza $C/t=1$, $C=4$ mellett (a) és az eredeti függvény (b)	130
5.18. A CMAC háló válasza $C=4$ és $t=3$ mellett	130
5.19. A súlykiegyenlítő regularizáció hatása egydimenziós esetben: a háló válasza (a) regularizált esetben, (b) regularizáció nélkül	132
5.20. A súlykiegyenlítő regularizáció hatása kétdimenziós esetben	133
5.21. A CMAC átlagos négyzetes hibája C/t függvényében	134
5.22. Zajos tanítópontok mellett a CMAC válasza. (a) regularizációval, (b) regularizáció nélkül .	134
5.23. Egyváltozós k -ad rendű B-spline bázisfüggvények $k=1-4$ esetekre	135
5.24. Kétdimenziós elsőrendű bázisfüggvény $C=8$ és $t=(4,4)$ paraméterekkel, (a) folytonos változat, (b) kvantált változat	136
5.25. Kompakt tartójú Gauss bázisfüggvény CMAC háléhoz, $C=4$	136
5.26. A szorzatok összege (<i>sum-of-product</i> , SOP) struktúra	138
5.27. A HCMAC felépítése kétdimenziós Gauss CMAC hálóból	138
6.1. A nemlineáris leképezések az eredeti probléma tértől a kernel térig.	147
6.2. Egy lineárisan szeparálható feladat különböző megoldásai	150
6.3. A margó geometriai értelmezése.	151
6.4. A gyengítő változók használata: (a) a mintapont osztályozása helyes, de a mintapont a biztonsági sávba esik, (b) a mintapont osztályozása hibás.	155
6.5. A kettős spirál probléma megoldása SVM-el. Az SVM a világos és a négyzethálós tartományokat különböztette meg, mint a két osztályba tartozó pontokat.	158
6.6. Az ε -érzékenységi sávval rendelkező abszolútérték hibafüggvény és alkalmazása.	161
6.7. Az ε -érzékenységi sáv szerepe a függvényapproximációban (a) $\varepsilon=0,05$, (b) $\varepsilon=0,15$. A vastagabb folytonos vonal a háló választ, a szaggatott vonalak az ε szélességű sáv széleit mutatják. A vékonyabb folytonos vonal az approximálandó függvényt a rajta lévő tanítópontokkal (kisebb pontok) mutatja. A nagyobb fekete pontok a szupport vektorok	162
6.8. Az ε -érzékenységi sáv hatása a válasz simaságára függvényapproximációban (a) $\varepsilon=0,05$, (b) $\varepsilon=0,15$. A vastagabb folytonos vonal a háló választ, a szaggatott vonalak az ε szélességű sáv széleit mutatják. A vékonyabb folytonos vonal az approximálandó függvényt a rajta lévő tanítópontokkal mutatja. A nagyobb fekete pontok a szupport vektorok.	163
6.9. A $\text{sinc}(x)$ regressziója zajos adatok alapján. (Gauss kernel; $\sigma=\pi$; $C=10$, $\varepsilon=0,15$. A Gauss zaj szórása (a) $\sigma_n=0,01$, (b) $\sigma_n=0,04$	163
6.10. A kernel térbeli lineáris megoldás illusztrációja két szupport vektor és egydimenziós kimenet esetén. A fekete pontok a tanító mintákat illusztrálják, míg a fehér pont egy kiszámítandó kimenetet illusztrál.	164
6.11. A szupport vektor gépnek megfeleltethető neurális hálózat	165
6.12. A legelterjedtebben használt QP megoldók „szeletelési” (dekomponálási) stratégiái.	167
6.13. A kettős spirál probléma megoldása LS-SVM-el.	170
6.14. A zajos $\text{sinc}(x)$ LS-SVM modellje (Gauss kernel; $\sigma=\pi$; $C=10$; A Gauss zaj szórása (a) $\sigma_n=0,01$ (b) $\sigma_n=0,04$). Az összes tanítópont szupport vektor.	172
6.15. A metszési eljárás, azaz a tanító minták elhagyásának hatása a feladat méretére valamint az approximáció eredményére.	173
6.16. A metszési eljárás folyamatának négy állapota egy zajos $\text{sinc}(x)$ megoldása során (RBF kernel; $\sigma=\pi$; $C=10$; a zaj szórása: $\sigma_n=0,01$). A mintapontok közül végül (d) 22-t tartunk meg (nagy fekete pontok).	174

6.17. A zajos sinc(x) LS ² -SVM modellje (RBF kernel; $\sigma=\pi$; $C=100$; $\varepsilon'=0,01$; A zaj szórása: (a) $\sigma_n = 0,01$ (b) $\sigma_n = 0,04$). Az RREF módszer eredményeképp a megoldás 23 szupport vektoron alapul.	178
6.18. Egy kétdimenziós sinc függvény approximációja LS ² -SVM-el. A baloldali képen az összes (2500) tanítópont, a jobboldalin az LS ² -SVM eljárással kapott 63 szupport vektor (fekete pontok) és az approximáció eredménye látható. (Gauss kernel, $\sigma=\pi$, $C=1000$, $\varepsilon'=0,15$.)	179
6.19. A kettős spirál probléma megoldása LS ² -SVM alkalmazásával. A mintapontok száma 194, a szupport vektorok száma 119. (Gauss kernel, $\sigma=0,5$; $C=10$, $\varepsilon'=0,9$.)	179
6.20. A CMAC hálózat kernel függvényei (a) egydimenziós eset, (b) kétdimenziós C-szeres lefedésű, (c) kétdimenziós teljes lefedésű, (d) kétdimenziós teljes lefedésű kvantált.	186
6.21. Két lehetséges mintakészlet az osztályozós feladatokhoz.	190
7.1. Többretegű perceptron struktúra karakterfelismerési feladathoz	193
7.2. A kép előfeldolgozása	195
7.3. Egyretegű hálózat előfeldolgozott minták osztályozására	196
7.4. Számjegyfelismerés egy rejtett rétegű MLP-vel	196
7.5. Részlet a tesztelő készlet számjegyeiből	197
7.6. Az extruder működése, vázlatosan	201
7.7. Az előrejelzés, mint modellezési és inverz modellezési feladat.	203
8.1. Általános időfüggő neurális hálózat többdimenziós bemenet és egydimenziós kimenet mellett	209
8.2. Állapotváltozós modell neurális implementációja	210
8.3. Példa lokális visszacsatolásokra	211
8.4. A FIR-MLP háló felépítése	212
8.5. A FIR-CMAC háló	218
8.6. Egy két processzáló elemből álló teljesen visszacsatolt hálózat (a), és időben kiterített előreccsatolt megfelelője (BPTT) (b).	219
8.7. Az autótólatós feladat vázlata	220
8.8. A modell-kontroller együttes. Az ábrán Δ egy lépéses késleltetést jelöl.	221
8.9. A BPTT megközelítés alkalmazása az autótólatós feladatra	221
8.10. A két pótkocsi teherautó tolatási feladat egy tipikus eredménye	221
8.11. Rekurzív háló architektúra	222
8.12. A Narendra által javasolt struktúrák	226
8.13. A kimenet bemenet szerinti gradiensének számítása egybemenetű - egykimenetű hálózatonál	226
8.14. A gradiens származtatása az 8.12 (a) ábrán látható modellben	228
8.15. Négy dinamikus struktúra.	229
8.16. A rendszermodellezés folyamata	231
8.17. A Lipschitz index alakulása a foksám függvényében Mackey-Glass folyamat esetén zajmentes (a) és zajos (b-d) esetekben. A Gauss zaj szórása: (b) $\sigma=0,05$; (c) $\sigma=0,1$; (d) $\sigma=0,3$	236
8.18. A maradék hiba autókorrelációs függvénye	237
8.19. A maradék hiba és a bemenet keresztkorrelációs függvénye	238
8.20. Kisszintű jellel tanított háló válasza	239
8.21. Nagyobb szintű jellel tanított háló válasza	239
8.22. Az adaptív szabályozás	240
8.23. A modell alapú adaptív szabályozás (MRAC)	241
8.24. A közvetett szabályozás	241
8.25. A Mackey-Glass kaotikus idősor előrejelzése LS-SVM hálózattal. A felső ábrán a vékony vonal a folyamat időfüggvényét, a vastagabb vonal az előrejelzés eredményét mutatja. Az alsó ábrán az előrejelzési tartományban a hiba látható.	243
8.26. A Santa Fe lézer idősor tanítómintaként felhasználható első 1000 mintája	244
8.27. A Santa Fe lézer idősor előrejelzése FIR-MLP-vel. A folytonos vonal a kaotikus folyamat valódi alakulását, a szaggatott a neuronhálózattal történő előrejelzés eredményét adja meg.	246
9.1. Egy háromosztályos feladat felbontása 3 egyszerű kétsztályos feladatra	250
9.2. Az 9.1 ábrán látható háromosztályos feladat moduláris megoldása	252
9.3. Egy egyszerű, lineárisan nem szeparálható kétsztályos feladat (a zöld tanító mintapontok az egyik, a feketék a másik osztályba tartoznak)	253
9.4. A moduláris kialakítás lépései (az egyes részmegoldások a sötétrel jelzett területeken adnak közel 1 értéket, a világossal jelölt területeken közel 0 értéket)	254
9.5. A 9.3 ábrán látható feladat dekompozíciónak megfelelő moduláris háló felépítése	254
9.6. Hálók lineáris kombinációja több-bemenetű – egykimenetű esetben	256
9.7. A MOE (Mixture of Experts) architektúra	261
9.8. A szűréssel történő boosting eljárás teljesítőképesség-javító hatása	269
10.1. Egy lineáris neuron	273

10.2. Versengő tanulással tanított hálózat (Kohonen-háló)	274
10.3. Az oldalirányú kapcsolatokat meghatározó mexikói kalap függvény	276
10.4. A Kohonen térkép	278
10.5. A Karhunen-Loève transzformáció	281
10.6. A GHA működési elve	289
10.7. Az APEX háló	290
10.8. Lineáris többrétegű perceptron, mint adattömörítő autoasszociatív háló	292
10.9. Nemlineáris dimenzió redukció	294
10.10. Egy egyszerű példa a kernel PCA működésének illusztrálására [Sch96c]. (dr. Klaus-Robert Müller hozzájárulásával).	298
10.11. A kernel PCA zajtalanító hatása [Mik99]. (dr. Klaus-Robert Müller hozzájárulásával)	300
10.12. Nemlineáris adattömörítő MLP hálózat	301
10.13. Független komponens analízis hálózat	308
10.14. A független komponens analízis alkalmazása időfüggvényekre (beszédjelekre)	309
10.15. Független komponens analízis alkalmazása képekre	310
11.1. A Hopfield háló	312
11.2. A konfigurációs tér	313
11.3. A konfigurációs tér egy minta tárolása után	314
11.4. Véletlenszerűen választott minták stabilitása	316
11.5. Különböző T értékekhez tartozó szigmoid karakterisztikák	320
11.6. Tipikus lehűtési görbék Boltzmann és mean-field hálózatokkal a.) homogén energiafüggvényen b.) változatos, sok lokális minimumot tartalmazó energiafüggvényen	325
11.7. Egy jó és egy rossz megoldás az utazó ügynök problémára	326
11.8. Egy 4 várost tartalmazó utazó ügynök probléma.	328
11.9. 4 bites A/D konverter megvalósítása Hopfield hálózattal	330
12.1. Egy kétféle osztályozási feladat. (a) az eredeti SVM-mel történő megoldás; (b) az a priori tudás (árnyékolt konvex sokszögek, mint az adott osztályhoz tartozó területek) figyelembevételével származtatott tudás alapú SVM megoldása	335
12.2. Az WMSVM működésének a standard SVM-től való eltérésre. A háló négyzetekkel illetve körökkel jelölt mintapontok osztályozását végzi, ahol a objektum mérete a konfidenciát jellemzi. A vékony vonal az SVM válasza, a vastag fekete vonal a WMSVM válasza.	338
12.3. Az információ terjedése KBANN-ban	341
12.4. Mintapélda KBANN kialakítására	344
12.5. Szabálykonverzió. Konjunktív (a) illetve diszjunktív (b) szabály leképezése	345
12.6. Kimeneti neuron származtatott hibája keresztentrópia (folytonos) és négyzetes (szaggatott) hibafüggvény mellett	347
12.7. Példa az eliminációra.	348
12.8. Példa az NofM működésére egyszerűsítéssel	350
12.9. Példa a TopGen beszűrési algoritmusára	352
12.10. A REGENT eljárásnál alkalmazott keresztezés	353
13.1. Gauss zajjal terhelt szinuszos jel időfüggvényének egy szakasza és hisztogramja	355
13.2. Ugyanazon szinuszos jel impulzus zajjal terhelt időfüggvénye, illetve hisztogramja	356
13.3. Az impulzus zajjal torzított szinuszos jel egy részlete - 5 pont széles ablakú mozgó - átlagolással javítva (folytonos vonal: ideális jel, pontozott vonal: szűrővel kapott jel)	356
13.4. A bemeneten és kimeneten is megfigyelési zajjal terhelt rendszer és neurális modellje	358
13.5. A gyakorlati problémák tipikus feldolgozási folyamata	359
13.6. Egy [-1,1]-re normalizált szinuszjel, és hisztogramja	363
13.7. Egyetlen kilógó értékkel terhelt szinuszjel [-1,1]-re normalizálva, és a normált jel hisztogramja	363
13.8. A kilógó adatokat mindkét paraméterre egyszerre (a), illetve külön-külön ható (b) torzítás hozta létre (mindkét ábrán a vízszintes tengelyen az egyik, a függőleges tengelyen a másik paraméter értéke látható)	364
13.9. A korrelációs együttható mátrix elemeinek hisztogramja egy konkrét feladatban	371
13.10. Egymással szoros kapcsolatban álló, de különböző korrelációs koefficienset adó paraméterpárok (az (a) ábrán látható esetben közel 1, míg a (b) ábrán látható esetben közel 0)	371
13.11. Egymással szoros - de klaszterenként eltérő - kapcsolatban álló paraméterpár. Az ábrán a két paraméter különböző értékeinek előfordulási gyakorisága látható, szintvonalas ábrázolással. (Tehát 3 különböző csúcsa van a gyakoriságnak)	372
13.12. Egy konkrét - villamos energiafogyasztást jellemző - idősor, mérési problémák miatt hiányzó (0 értékkel feltüntetett) adatokkal	373
13.13. Dimenzióként 3-3 ponttal lefedett kétdimenziós térrész	374

13.14. Bemeneti - tipikusan PCA, illetve ICA - transzformációval történő lényegkiemelés	375
13.15. A mintahalmaz főkomponense mentén végzett osztályozása	376
13.16. A neuronháló tanítása közben iteratíván végzett dimenziómódosítás	377
F.3.1. Mintapélda a feltételes szélsőérték-keresésre. A vékonyabb vonallal rajzolt ellipszisek a minimalizálandó $f(\mathbf{x})$ függvény szintvonalai, a vastagabb vonallal rajzolt kör pontjai a mellékfeltételnek megfelelő pontok. A két fekete pont a feltételes minimumot biztosító pontok.	389

A táblázatok listája

2.1. A kromoszómák alakulása az egyes operációk után	68
6.1. A legelterjedtebben használt magfüggvények (kernel függvények).	148
7.1. A számjegyfelismerő pontossága különböző beállítások mellett	198
7.2. Példa dokumentum-term mátrix	198
7.3. Spam átengedés 5 %-os normál levél kiszűrés mellett	200
12.1. A különböző tudásreprezentációs tanuló rendszerek egymást kiegészítő erősségei (+) és gyengeségei (-)	340

Előszó

Mintegy 50 éve, jórészt biológiai kutatások eredményeképpen merült fel az a gondolat, hogy a természetes, "biológiai" neurális hálózatok mintájára is létrehozhatók számító rendszerek. Az alapgondolat térnyerése azonban csak az utóbbi, mintegy 20-25 évben következett be. Egy "új" számítási paradigma, a neurális számítástechnika (*neural computing*) jelent meg, amely a természetes (biológiai) neurális rendszerek felépítése és működési mechanizmusa mintájára hoz létre számító rendszereket. Olyan rendszereket, melyek a természetből ellesett módon mintákból, példákából nyert tapasztalatok felhasználásával, tanulás útján alakítják ki feladatmegoldó képességüket. E "mesterséges" neurális rendszerek felépítésükben is hasonlóságot mutatnak a biológiai neurális rendszerekkel; sok, egymással nagymértékben összekötött elemi műveletvégző egységből állnak, melyek párhuzamos működésük révén bonyolult feladatok igen gyors megoldására is képesek lehetnek.

A neurális számítástechnika mára önálló tudománnyá vált, amely szilárd elméleti alapokkal, egyre szélesebb alkalmazási körrel és egyre több alkalmazási tapasztalattal rendelkezik.

Ez a könyv azzal a céllal született, hogy áttekintést adjon a neurális hálózatokkal kapcsolatos legfontosabb elméleti és gyakorlati eredményekről. A könyv figyelembe veszi az elmúlt néhány év során megszületett új eredményeket is, amelyek jelentősen megváltoztatták, új megvilágításba helyezték az egész "neurális számítástechnikát" és általában az adatokból való tanulást. A könyv ugyanakkor nem törekedhet teljességre, sőt még csak nem is fedi le a neurális számítástechnika minden fontos területét. A neurális hálózatokkal foglalkozó irodalom és az eddigi eredmények olyan széleskörűek, hogy a témakör jelenlegi szintjének teljes áttekintése messze meghaladná egy korlátozott terjedelű könyv lehetőségeit. A könyv ezért nem „egyenszilárdságú”. Egyes témák talán a fontosságuknál is bővebben szerepelnek, míg más területek a szükségesnél is kisebb terjedelemben jelennek meg, sőt sok részterülettel egyáltalán nem foglalkozunk. Az ismertetett témakörök kiválasztásába jelentősen közrejátszott a szerzői kollektíva szemlélete is, tehát a témakörök kiválasztása bizonyos szubjektív elemeket is tartalmaz. Azon elméleti eredmények és hálózatok bemutatására törekszünk, amelyek a mérés-technikai, szabályozástechnikai és jelfeldolgozási feladatokhoz jobban illeszkednek, és amelyekhez kapcsolódóan a könyv szerzőinek önálló eredményei is vannak.

A könyv a neurális hálózatok biológiai vonatkozásait nem emeli ki. Ugyan megemlíti, hogy egyes hálózatok kialakulásában a biológiai, tehát a nem mesterséges, hanem a természetes neurális hálózatok jelentős szerepet játszottak, alapjában a hálózatok a könyvben, mint bizonyos operációk elvégzésére alkalmas rendszerek jelennek meg, amelyek eredetüktől függetlenül vizsgálhatók és alkalmazhatók.

A könyv alapját egy előadássorozat és egy csaknem egy évtizede készült tankönyv [Hor98] képezi. Ennek ellenére alapvetően új könyvről van szó. Az előadássorozat a BME Villamosmérnöki és Informatikai Kar különböző évfolyamos hallgatói részére az elmúlt években rendszeresen elhangzott. Az itt nyert tapasztalatok alapján az előző változattól mind szemléletében, mint felépítésében különböző, egyes fejezeteket tekintve jelentősen eltérő könyv megírását tűztük ki célul.

A nemzetközi irodalomban ma már található jónéhány olyan könyv, melyek jóval nagyobb terjedelemben foglalkoznak a neurális számítástechnikával. A teljes témakör bemutatására azonban ezek a könyvek sem vállalkoznak. Az utóbbi években szinte folyamatosan jelentek meg újabb és újabb könyvek, melyek megpróbálják követni a gyors fejlődést, a legújabb eredmények azonban csak folyóirat- és konferencia-publikációkban találhatók meg. A könyvek szemlélete, a témakörök feldolgozásának mélysége, vagy a feldolgozott témakörök szerteágazósága több-kevesebb mértékben különbözik jelen könyvétől. A neurális szakirodalom talán első és máig is egyik legjobb összefoglaló műve a John Hertz, Anders Krogh és Richard Palmer szerzői kollektíva könyve [Her91]. Az összefoglaló, viszonylag széles spektrumot átfogó könyvek között elsősorban Simon Haykin könyvét kell megemlíteni, melynek a második, jelentősen bővített és módosított kiadása 1999-ben jelent meg [Hay99]. E mellett fontos utalni a Mohamad Hassoun és a Christopher Bishop által jegyzett, 1995-ben megjelent könyvekre. Hassoun könyve szemléletében rokon Haykin könyvével, bár a hangsúlyok máshol vannak. A teljes spektrumból valamivel kisebb területet mutat be, viszont sok esetben ezt részletesebben teszi. Teljesen más szemléletű Bishop könyve [Bis95], mely talán jobban bemutatja, hogy a neurális számítástechnika milyen sok ponton kapcsolódik a mérnöki problémamegoldás és a matematika más témaköreihöz. A gyors fejlődést és az egyes szakterületek összekapcsolásának igényét jelzi a közelmúltban megjelent néhány újabb könyv is, melyek a neuronhálókat az intelligens információfeldolgozás egyéb eszközei között helyezik el, bemutatva a közöttük lévő kapcsolatokat is ld. pl. [Bish06], [Has09].

Számos, a neuronhálókhoz szorosabban vagy lazábban kapcsolódó, fontos témakör a felsorolt könyvekben nem vagy csupán érintőlegesen jelenik meg. Ezen témakörök között kell megemlíteni a szupport vektor gépeket és

általában a kernel gépeket, valamint az ezen hálózatok elvi megalapozásához is hozzájáruló statisztikus tanulásméleletet [Vap95], [Vap98]. E témakörökről az első áttekintő, mérnöki szemléletű összefoglaló művek angolul is csak 2000 után jelentek meg [Cri00], [Sch02], [Suy02], magyar nyelven pedig még semmilyen átfogó könyv nem született. A könyv ezért jelentős részben e témakörökkel is foglalkozik.

A könyv 13 fejezetből áll. A Bevezetés a mesterséges neurális hálók legfontosabb jellemzőit, a neurális számítástechnika kialakulásának, a neurális hálózatok fejlődésének néhány fontosabb állomását veszi sorra.

Az 1. fejezet a neurális hálózatokkal kapcsolatos alapfogalmakat, a hálózatok alapelemeinek, a processzáló elemeknek és az alapvető hálózatstruktúráknak a felépítését igyekszik bemutatni. Ugyancsak ez a fejezet ad áttekintést egyes, a későbbiek szempontjából fontos matematikai eredményekről: ez a fejezet foglalja össze a hálózatok approximációs képességeinek matematikai alapjait. Az elméleti eredményeket összefoglaló részek nem törekszenek matematikai precizításra; bár néhány tételt megfogalmazunk, ezek bizonyítását sehol sem közöljük. Ennek legfőbb oka, hogy a könyv a témakör mérnöki szemléletű bemutatását tűzi ki célul. Ugyancsak ezt a megközelítést támasztja alá, hogy a matematikai bizonyítások jelentős része nem konstruktív, tehát a probléma jobb megértését általában nem segítik elő. Ugyanakkor a tételek megléte fontos, hiszen ezek biztosítják a témakör szilárd, egzakt hátterét.

A 2. fejezet a gépi tanulással, mint a neuronhálók kialakításának egyik legfontosabb módjával foglalkozik. A fejezet bemutatja a gépi tanulás elvi alapjait, a legfontosabb elméleti eredményeket és azokat az általános tanulási eljárásokat, melyek a különböző neuronhálókánál alkalmazhatók. A fejezet a gépi tanuláshoz kapcsolódó általános eredményeket tekinti át, a konkrét tanulási eljárások bemutatására az egyes neuronhálók ismertetésénél kerül sor.

A 3. fejezet az elemi neuron különböző kialakításaival és azok tanításával foglalkozik. Bemutatja azt is, hogy az elemi neuronok milyen képességekkel és korlátokkal rendelkeznek.

A 4. fejezet az egyik leggyakrabban alkalmazott neuronháló, a többrétegű perceptron (MLP) felépítését, működését és tanítását tárgyalja. Bemutatja az MLP képességét is: ez teszi lehetővé, hogy az MLP a feladatok igen széles körében sikerrel alkalmazható.

Az 5. fejezet a bázisfüggvényes hálózatokkal, ezen belül is elsősorban a Radiális Bázisfüggvényes (RBF) hálózattal és a CMAC hálózattal foglalkozik. Részletesen elemzi e két hálótípus képességbeli korlátait, valamint azokat az előnyöket, amelyek a bázisfüggvényes felépítésből következnek.

A 6. fejezet az elmúlt mintegy 10 évben kifejlesztett, és mind elméleti, mind gyakorlati szempontból kiemelten fontos területtel, a szupport vektor gépekkel foglalkozik. A szupport vektor gépek – miközben számos ponton kapcsolódnak a "klasszikus" neuronhálókhoz – új szemléletet hoztak a tanuló gépek témakörbe. Ennek a megközelítésnek az alapján olyan kérdésekre is válasz adható, amelyekre a klasszikus neurális megközelítés mellett nem kaphatunk választ.

A 7. fejezet az előrecsatolt, statikus hálók néhány alkalmazási területét mutatja be, konkrét példákat is megadva.

A 8. fejezet a dinamikus hálókkal kapcsolatos eredményeket foglalja össze. A fejezetek végén néhány fontos vagy érdekes alkalmazási területet is megemlítünk. Egyes alkalmazási példák részletesebben is bemutatásra kerülnek.

A 9. fejezet a moduláris hálók témakörrel foglalkozik. A fejezet alapgondolata, hogy komplex feladatok megoldása általában csak összetett rendszerekkel lehetséges, ahol az egyes részrendszerek egymással vagy versengenek vagy együttműködnek. Egyes esetekben a kétfajta működés együttesen is megjelenhet. Bár a moduláris hálóknak nagyon sokféle változatát dolgozták ki, itt csak néhányat a bemutatására vállalkozunk. Ezek között is a legfontosabbnak az ún. szakértőegyüttes (Mixture of Experts, MOE) architektúrát tekintjük, mivel ez önmagában is érdekes elvi megállapításokon alapul. Hasonlóan fontos témakör a moduláris hálók olyan konstrukciója, ahol a tanítópontok egyféle válogatása is szerepet játszik a moduláris architektúra kialakításában. Az ún. boosting algoritmusok, valamint az erős és gyenge tanulással kapcsolatos eredmények a neuronhálók témakörén messze túlmutató jelentőségűek.

A 10. fejezet a nemellenőrzött tanítású hálókkal foglalkozik. Bemutatja a klasszikus eredményeket (pl. Kohonen hálók), de röviden tárgyalja a legújabb kernel PCA és ICA hálózatokat is, kiemelve ezen eljárások fontos gyakorlati alkalmazási lehetőségeit.

A 11. fejezet az ún. analitikus tanítású hálókkal foglalkozik. Ennek a résznek a súlya a könyvben minden bizonnyal kisebb a jelentőségénél. Indoklásként az hozható fel, hogy ezek a hálók kisebb szerepet játszanak a mérés-technikai, jelfeldolgozási alkalmazásoknál.

A neuronhálók meglehetősen univerzális feladatmegoldó eszközök. Összetett, bonyolult gyakorlati problémák megoldásában azonban legtöbbször önmagukban nem alkalmazhatók, inkább csak részfeladatok megoldásában jelentenek nagy segítséget. Általában elmondható, hogy egy komplex feladat megoldása az esetek döntő többségében különböző megközelítések együttes alkalmazásával lesz sikeres. Az utóbbi évtizedben ezért egyre inkább megfigyelhető az a tendencia, amely a különböző eszközök ötvözésére törekszik. A neurális számító eszközöket ezért előszeretettel alkalmazzák együtt más "szoft" számítási megközelítésekkel, a hagyományos mesterséges intelligencia eszközeivel, de a hagyományos algoritmikus megoldásokkal is. Az ilyen hibrid módszerek keresése és alkalmazása jelentheti a továbblépés egyik irányát. A 12. fejezet a hibrid-neurális rendszerekkel és ezek néhány konkrét megvalósítási lehetőségével foglalkozik röviden.

A 13. fejezet a neuronhálók néhány, a gyakorlati alkalmazások szempontjából fontos kérdését foglalja össze. Ezekkel a kérdésekkel szinte minden gyakorlati feladat megoldásánál találkozunk. Fontosságukat az indokolja, hogy e kérdések vizsgálata, az itt bemutatott problémákkal való foglalkozás nélkül a neuronhálók gyakorlati feladatokra való alkalmazása esetenként nagyon nehéz, sokszor sikertelen lesz.

A könyvet egy rövid kitekintés, egy Függelék, egy viszonylag sok tételes irodalomlista és egy rövid szótár zárja. A Függelék azokat a legfontosabb matematikai alapismereteket foglalja össze, melyek ismerete az egyes fejezetek megértéséhez feltétlenül szükséges. Az irodalomlista célja, hogy segítse mindazokat, akik a neurális hálózatokkal mélyebben szeretnének megismerkedni, a szótárban pedig a legfontosabb neurális hálózatokhoz kapcsolódó kifejezéseket (angolul és magyarul) gyűjtjük össze. A szótár kapcsán meg kell jegyeznünk, hogy a neurális számítástechnika számos szakkifejezésének máig nincs magyar megfelelője. A könyvben sem törekedtünk a mindenáron való „magyarításra”. Csak azoknál a kifejezéseknél, fogalmaknál használtunk magyar elnevezéseket, ahol ezeket a szakma többé-kevésbé már elfogadta, illetve ahol a magyar elnevezés nem tűnik nagyon erőltetettnek.

A könyv szándékaink ellenére nyilván számos hibát tartalmaz. Reméljük, hogy az olvasók ezen hibákat észrevéve hozzásegítenek a kijavításukhoz.

A könyv elkészültét több pályázat is segítette, melyek közül ki kell emelni a T 046771 sz. OTKA kutatási projektet, ugyanis a könyvben szereplő egyes részek kidolgozása az említett OTKA projekt keretében valósult meg.

A könyv elektronikus változata egy TAMOP projekt (TÁMOP-4.1.2-08/2/A/KMR-2009-0026) keretében született meg. Az elektronikus változat alapvetően a 2006-os nyomtatott kiadás megfelelője. Néhány különbség azonban található a kétféle verzió között. Egyrészt az elektronikus verzióban a nyomtatott verzió hibáit igyekeztünk kijavítani, másrészt egyes fejezeteknél az eredeti verzióhoz képest kisebb mértékben bővült a szöveg.

Budapest, 2011. január 15.

Horváth Gábor

Bevezetés

A neurális hálózatok olyan, számítási feladatok megoldására létrejött párhuzamos feldolgozást végző, adaptív eszközök, melyek eredete a biológiai rendszerektől származtatható. Az idegrendszer tanulmányozása, az idegsejt (neuron) felépítésének, illetve működésének valamilyen szinten való megismerése indította el azt a gondolkodást, hogy kíséreljünk meg az élő szervezetekben létező, bonyolult rendszerek mintájára létrehozni számító rendszereket. A biológiai, "természetes" neurális hálózatoknál nagyszámú, hasonló vagy azonos felépítésű, egymással összeköttetésben lévő építőelemekből, idegsejtekből felépülő hálózatok a legkülönbözőbb feladatok ellátására bizonyulnak alkalmasnak.

A természetes neurális hálózatok számos feladat megoldásánál nemcsak alkalmasnak, hanem alapvetően jobbnak is bizonyulnak, mint a hagyományos algoritmikus számítási rendszerek. Ilyen feladatok tipikusan a különféle felismerési problémák, kezdve a viszonylag egyszerű nyomtatott számok és karakterek felismerésétől a jóval bonyolultabb kézírás-, kép- és egyéb alakzatfelismerésekig. A felismerési feladatokat a "kellően összetett biológiai neurális hálózattal" rendelkező élőlények minták alapján tanulják meg. Egy arc felismerése az ember számára egy cseppet sem jelent megerőltető feladatot, annak ellenére, hogy nehéz megmondani milyen módon végezzük a felismerést, nehéz megnevezni azokat a jegyeket, melyek az adott arc egyértelmű vagy legalábbis nagy biztonságu felismerését lehetővé teszik, és ennek megfelelően nehéz olyan algoritmikus megoldást találni, amely hasonlóan sikeres e feladat megoldásában, mint az ember.

A biológiai neurális rendszerek nagymértékben párhuzamos felépítéssel és tanulási képességgel rendelkeznek; e két jellemzőjük következtében jelentősen különböznek a hagyományos számítástechnikai eszközöktől. Ráadásul bizonyos feladatokban – tipikusan azon feladatokban, melyekre hatékony algoritmikus megoldást eddig nem sikerült találni – jó eredmény elérését teszik lehetővé. Ezért vetődött fel olyan számítási rendszerek – "mesterséges" neurális hálózatok – létrehozásának lehetősége, amelyek a hagyományos algoritmikus eljárásokkal dolgozó eszközök helyett, a biológiai neurális hálózatokhoz hasonló felépítésűek és melyek tanulással nyerik el azt a képességüket, hogy bizonyos feladatokat meg tudnak oldani.

A felismerési feladatokon kívül sok egyéb olyan problémával találkozunk, ahol jelenleg nem ismert algoritmikus megoldás, vagy ha van is ilyen, az annyira bonyolult és/vagy olyan sok műveletet igényel, hogy valójában elfogadható idő alatt a mai legnagyobb teljesítményű számítógépekkel sem oldható meg.

E feladatok egy részénél – ilyenek pl. komplex ipari, gazdasági vagy pénzügyi folyamatok időbeli viselkedésének előrejelzése – a megoldás nehézsége általában abból ered, hogy nem rendelkezünk azzal a tudással, amely az algoritmikus megoldáshoz szükséges lenne. Alapvetően kétféle tudásra lenne szükségünk: egyrészt a folyamatok mögött meghúzódó fizikai, közgazdasági, stb. törvényszerűségeket kellene ismernünk, másrészt a törvényszerűségek ismeretében is csak akkor tudnánk a feladatot megoldani, ha a folyamatokat létrehozó rendszerek pillanatnyi állapotát és e rendszerekre ható összes vagy legalább minden lényeges hatást, a környezeti feltételeket is ismernénk. A feladatokról ugyanakkor más formában – adatokban megtestesülve – rendelkezésünkre áll tudás, amelyet ha fel tudunk használni, a feladat valamilyen megoldásához eljuthatunk. Egy adott problémáról mindig véges számú adatunk lehet, továbbá az adatok által hordozott tudás sohasem teljes. Mégis a természetes neurális hálók képesek az adatokból nyert ismeretek általánosítására: olyan szituációkban is jó választ adnak, melyek az adatok között nem találhatók meg. Egyes mesterséges neurális hálók is rendelkeznek ezzel a képességgel. Ez azt jelenti, hogy hiányos, esetleg pontatlan ismereteket hordozó, legtöbbször zajos adatokból is kinyerhető általános tudás. Ez a tudás azonban a feladat "tökéletes" megoldását rendszerint nem teszi lehetővé, viszont "jó" megoldás elérését biztosítja.

A feladatok másik csoportjánál a megfelelő tudás birtokában vagyunk, sőt létezik is algoritmikus megoldás. A megoldás bonyolultsága, a megoldás eléréséhez szükséges számítások mennyisége azonban a feladat méretével exponenciálisan nő. Az ilyen *NP*-teljes feladatok esetében is sokszor megelégszünk egy "jó", tehát nem a legjobb megoldással, ha a számítási komplexitást sikerül a polinomiális komplexitási tartományba leszorítani. E csoportba tartoznak pl. a kombinatorikus optimalizálási feladatok. A természetes neuronhálókhoz hasonlóan bizonyos mesterséges neuronhálók is alkalmasak optimalizálási feladatok megoldására, ahol a megoldás optimalitása általában nem garantálható, de az optimálishoz közeli megoldás elérésének a valószínűsége nagy.

A természetes neuronhálókban van még számos olyan tulajdonsága, melyek, ha megvalósíthatók a mesterséges neurális hálózatokban is, a gyakorlati alkalmazások körét jelentősen bővítik. Ezek közül az egyik legfontosabb a párhuzamos felépítés és működés. A neurális architektúrák nagymértékű párhuzamossága a nagysebességű működés egyik biztosítéka. A nagyfokú párhuzamosság – ami többnyire redundanciával is párosul – robusztus működést, egyfajta hibatűrő képességet is biztosít a neurális rendszereknek. Közismert, hogy ha valami, pl.

baleset vagy betegség folytán az emberi agy egyes területei megsérülnek, ez nem feltétlenül jelenti bizonyos képességek elvesztését. Sőt, az előbb említett sérülés olyan súlyos is lehet, hogy egyes agyterületek elvesztik működőképességüket, bizonyos funkciók nem működnek. Mégis idővel ezek a funkciók helyreállhatnak, más agyterületek átveszik a sérült részek feladatait. Ugyancsak fontos tulajdonság az adaptációs képesség, a változó környezethez való alkalmazkodás képessége.

A mesterséges neurális hálózatok párhuzamos hardver megvalósítása olyan működési sebesség elérését teheti lehetővé, hogy egyes alkalmazásokban a neurális megoldás már csak ezért is egyedülálló. Bizonyos feladatoknál a neurális megoldás nagymértékben párhuzamos működése miatt az algoritmikus megoldás neurális architektúrával történő implementációját érdemes alkalmazni. A hálók egy részére a redundáns struktúra jellemző, így a robusztus működés, a háló egyes részeinek hibája esetén a fokozatos és nem ugrásszerű működésbeli degradáció a mesterséges neuronhálónál is megtalálható. A folyamatos tanulás képessége pedig a környezet változásaihoz való alkalmazkodást is lehetővé teszi.

A neurális hálózatok tehát alkalmasak lehetnek olyan feladatok megoldására, amelyek a hagyományos algoritmikus megközelítéssel nem vagy nehezen oldhatók meg, de amelyeket még az ún. klasszikus mesterséges intelligencia módszereivel (pl. szabályalapú rendszerek) sem sikerül kielégítően megoldani.

Az alapgondolat, hogy a természetes neurális hálók mintájára hozzunk létre mesterséges számító rendszereket meglehetősen régi. A kezdeti lökést természetesen az idegsejttel kapcsolatos kutatási eredmények adták, elsősorban Warren McCulloch és Walter Pitts neuron modellje [McC43], majd Donald Hebb tanulással kapcsolatos eredményei [Heb49]. Jónéhány évnek el kellett azonban telnie addig, amíg az első mesterséges neuron és az ebből, mint alapegységből felépülő mesterséges neurális hálózat megjelent. Az egyszerű neuron modell valójában nem más, mint egy többdimenziós bemenet komponenseinek súlyozott összegét meghatározó, tehát lineáris kombinációt megvalósító hálózat, amelyet egy nemlineáris leképezés követ. A modell érdekessége elsősorban nem is a felépítésében, hanem abban az eljárásban keresendő, amely biztosítja, hogy egy neuron valamilyen kívánt működést mutasson. A kívánt működés – amely összerendelt be- és kimeneti értékekkel specifikálható – megfelelő konvergens iteratív eljárással, tanuló eljárással érhető el. A tanulás itt egyszerűen azt jelenti, hogy a kívánt működést reprezentáló, összetartozó be- és kimeneti értékek alapján a hálózat maga alakítja ki a megfelelő átvitelét, illetve a hálózat bemenetére kerülő adatokban önmaga próbál valamilyen hasonlóságot felfedni. Ezen egyszerű, tanulásra képes modell alapján hozta létre Frank Rosenblatt a *Perceptront* [Ros58], amely már egyszerű osztályozási feladatok megoldására megtanítható volt. Rosenblatt első eredményeit hamarosan követte a Bernard Widrow és munkatársai által definiált *Adaline* és a hozzá kapcsolódó Widrow-Hoff tanulási algoritmus [Wid60], ill. az egyedi "neuron", processzáló elemet hamarosan kiterjesztették neuronokból felépülő hálózatokká, lehetővé téve komplexebb feladatok megoldását.

A neurális hálózatok (neurális hálók, neuronhálók) kezdeti, igen ígéretesnek mutakozó fejlődése azonban hamarosan megtorpant, amikor Marvin Minsky és Seymour Papert kimutatták *Perceptrons* című könyvükben [Min69], hogy a perceptron valójában csak lineárisan szeparálható osztályozási feladatok megoldására képes, ami a megoldandó feladatoknak csak egy nagyon kis köre. Ez azt jelenti, hogy egy perceptron még egy olyan egyszerű feladatot sem képes megtanulni, mint két bináris változó kizáró VAGY (XOR) kapcsolata. Bár Minsky könyvében megmutatta, hogy ha az egyszerű processzáló elemeket rétegekbe rendezzük és többrétegű hálózatot hozzunk létre, akkor a hálózat nemlineáris szeparálásra is képes, azonban akkor nem találták meg azt az algoritmust, amely egy többrétegű hálózat be- és kimeneti mintapárok alapján történő tanítására alkalmas lett volna (sőt Minsky annak a sejtésének adott hangot, hogy ilyen algoritmust nem is lehet létrehozni). A hálózatnak e hiányossága (és a kritikus megfelelő tekintélye: Marvin Minsky az MIT professzora volt) azt eredményezte, hogy az e területre irányuló érdeklődés jelentősen csökkent, csaknem megszűnt; a terület kutatásával majdnem teljesen felhagytak.

Mintegy húsz évet kellett várni ahhoz, hogy ismét megélénküljön a neurális hálózatok iránti érdeklődés. Az élénkület az váltotta ki, hogy – az általános tendenciák ellenére – néhányan mégiscsak folytatták a hálókkal kapcsolatos kutatásaikat és olyan eredményeket értek el, amelyek ismét ráirányították a szakmai közvélemény figyelmét a neurális hálózatokra. Az érdeklődés felkeltésében a legfontosabb esemény talán az ún. Hopfield-háló publikálása volt [Hop82], amely bemutatta, hogy egy viszonylag egyszerű hálózat hogyan képes optimalizálási feladatok – köztük NP-teljes feladatok – megoldására. További jelentős eredmény volt az ún. hibavisszaterjesztéses (*back-propagation*) algoritmus "felfedezése", amely hatékonynak bizonyult a többrétegű hálózatok tanításánál, s amely cáfolatát adta Minsky sejtésének. Az algoritmust 1986-ban publikálta a David Rumelhart, Geoffrey Hinton, Ronald Williams szerzőhármas [Rum86] és egyben be is mutatták az algoritmus hatékonyságát. Később kiderült, hogy ugyanezt az algoritmust már többször is megkonstruálták, de különböző okok folytán akkor nem váltott ki komolyabb érdeklődést [Bry69], [Wer74].

A nyolcvanas évek második fele óta a neurális hálózatok kutatása reneszánszát éli. A megélénkült kutatások következtében egyre növekvő számban jelentek meg és jelennek meg a témakör különböző elméleti és gyakorlati kérdéseivel foglalkozó cikkek, könyvek; évente egyre több konferencia kerül megrendezésre a neurális hálózatok témakörben. A nagy érdeklődésnek káros hatásai is megmutatkoznak. A neurális hálózatokat időnként úgy állítják be, mint bármilyen feladatra alkalmazható univerzális eszközöket, és olyan esetekben is alkalmazzák, vagy javasolják alkalmazni, amikor más, hagyományos eljárások hasznosabbnak bizonyulnak. Az alkalmazási területek között vannak valóban olyanok, ahol a neurális hálózatoknak komoly előnyeik vannak. Ezek elsősorban olyan területek, ahol más, algoritmikus megoldás eddig nemigen vezetett eredményre vagy azért, mert eddig nem sikerült ilyen algoritmikus megoldást találni, vagy azért, mert ezen megoldások túl bonyolultaknak bizonyultak, s így mindenképpen indokolt más, egyszerűbb, gyorsabb, könnyebben megvalósítható megoldások keresése. Ilyen feladatoknak tekinthetők a már említett különböző felismerési problémák, vagy a különböző optimalizálási feladatok. Hasonlóan fontos alkalmazási terület a nemlineáris rendszerek vizsgálata, identifikációja, irányítása, ahol – eltérően a lineáris rendszerektől – nem alakult még ki olyan matematikailag megalapozott elmélet, amely biztosítaná az algoritmikus megoldások fölényét.

A neurális hálózatok tudománya egy fejlődésben lévő tudomány. Az újabban megélénkült kutatások eredményeképpen most már nemcsak azt sikerült bemutatni, hogy a neurális hálózatok komplex feladatok megoldásában igen hasznosnak bizonyulnak, hanem az ehhez szükséges elméleti alapokat – vagy legalábbis számos elméleti alaperedményt – is sikerült kidolgozni. Bizonyított tény pl., hogy megfelelő felépítésű neurális hálózatok képesek csaknem tetszőleges nemlineáris leképezések tetszőleges pontosságú approximációjára, vagyis, hogy a neurális hálózat univerzális approximátornak tekinthető. E képességük révén a hálózatok alkalmasak nemlineáris (statikus vagy dinamikus) rendszerek modellezésére. Ugyancsak bizonyítható, hogy megfelelő tanulási algoritmussal, adott felépítésű hálózat alkalmassá tehető adatokban meglévő hasonlóságok felismerésére, az adatokban rejtve meglévő ismeretek kinyerésére, tehát egyfajta adatbányászati eszköznek is tekinthető. Az approximációs képesség megfelelően interpretált feladat esetén osztályozási képességet is jelent, egyes hálók ezért univerzális osztályozók is. A neurális hálózatokhoz kapcsolódó kutatások számos, a tanulással kapcsolatos eredményre is vezettek, ezért nagyban hozzájárultak a gépi tanulás fejlődéséhez. Az elméleti eredmények elsősorban a hálózatok működésének megértésében segítenek, tisztázzák a hálózatok korlátait, lehetőségeit, új hálóarchitektúrák kialakítását teszik lehetővé, és nagymértékben segítik a gyakorlati felhasználást.

A neurális hálózatok kutatása számos ponton kapcsolódik az egyéb, lineáris és nemlineáris rendszerekkel foglalkozó kutatásokhoz. Így különösen sok kapcsolódási pont található a neurális hálózatok és az adaptív rendszerek között – lévén, hogy a neurális hálózatok is egyfajta adaptív működést valósítanak meg. Hasonlóan erős a kapcsolat a neurális információfeldolgozás és a parallel feldolgozás között, hiszen mint láttuk, a neurális hálók maguk is elosztott párhuzamos számító rendszerek. És az is természetes, hogy a neurális hálózatok kutatása kezdetektől fogva mindvégig szoros kapcsolatban áll bizonyos biológiai kutatásokkal, elsősorban az idegrendszeri- és agykutatásokkal, hiszen a mesterséges neurális hálók számos eredménye biológiai kutatásokon, megfigyeléseken alapul.

Az elmúlt években megszületett, matematikailag megalapozott eredmények ellenére számos megválaszolatlan kérdés maradt, ezért a hálózatokkal kapcsolatos feladatmegoldásokban heurisztikus elemekkel, tapasztalatokból levont következtetések alkalmazásával is találkozunk. Az intenzív kutatás következtében azonban várható, hogy egyre több kérdésre születik egzakt válasz, lehetővé téve így, hogy egyre pontosabban meghatározzuk azon feladatok körét, ahol lehet és érdemes is neurális hálózatokat alkalmazni.

1. fejezet - A neurális hálózatok felépítése, képességei

A fejezet célja a neurális hálózatok felépítésével kapcsolatos legfontosabb alapfogalmak, alapvető definíciók és háló-topológiák áttekintése, valamint annak bemutatása, hogy a legfontosabb neuronháló architektúrák milyen számítási képességekkel rendelkeznek. A fejezet röviden bemutatja azt is, hogy a számítási képességek alapján a neuronhálók milyen feladattípusok megoldásában alkalmazhatók sikerrel. A fogalmakat és a definíciókat tekintve elsősorban az igen gazdag és szerteágazó angol irodalomból indulunk ki. Bár neuronhálókkal kapcsolatos kutatások a 90-es évek kezdete óta már Magyarországon is folynak, sőt számos hazai neuronháló-alapú gyakorlati alkalmazás is született, a neuronhálók minden vonatkozására kiterjedő egységes magyar nyelvű terminológia a mai napig nem alakult ki. Ezért ennek a fejezetnek az is a célja, hogy megkísérelje a különböző angol nyelvű terminológiákat egy közös magyar elnevezésszerre leképezni. A fejezetben ezért igyekszünk megadni az általunk használt magyar elnevezések gyakrabban alkalmazott változatait és ezek angol megfelelőit is.

Az 1.1 alfejezet a neurális hálózatokhoz kapcsolódó legfontosabb definíciókat adja meg. Az 1.2 alfejezet a neurális hálózatok alapelemeit és a fontosabb neuronháló elrendezéseket mutatja be. Az 1.3 alfejezet a neuronhálók alapvető képességeit veszi sorra és ehhez kapcsolódóan néhány fontos matematikai eredményt ismertet.

1. A neurális hálózat definíciója, működése

Neurális hálózatnak nevezzük azt a hardver vagy szoftver megvalósítású párhuzamos, elosztott működésre képes információfeldolgozó eszközt, amely:

- azonos, vagy hasonló típusú – általában nagyszámú – lokális feldolgozást végző műveleti elem, neuron (*processing element, neuron*) többnyire rendezett topológiájú, nagymértékben összekapcsolt rendszeréből áll,
- rendelkezik tanulási algoritmussal (*learning algorithm*), mely általában minta alapján való tanulást jelent, és amely az információfeldolgozás módját határozza meg,
- rendelkezik a megtanult információ felhasználását lehetővé tevő információ előhívási, vagy röviden előhívási algoritmussal (*recall algorithm*).

A fenti definíció kulcsfogalmainak tisztázására a következőkben rendre megvizsgáljuk az alkalmazott műveleti elemek (neuronok) felépítését, az összeköttetéseket, illetve a tipikus topológiákat. A tanulási algoritmusok részletesebb áttekintésére a 2. fejezetben kerül sor, ebben a fejezetben csupán az alapfogalmakra utalunk.

A fentiek értelmében a neurális hálózatok működésénél tipikusan két fázist különböztethetünk meg. Az első fázis, melyet tanulási fázisnak nevezünk, a hálózat kialakítására szolgál, melynek során a hálózatba valamilyen módon beépítjük, eltároljuk a rendelkezésre álló mintákban rejtve meglévő információt. Eredményként egy információ-feldolgozó rendszert kapunk, melynek használatára általában a második fázisban, az előhívási fázisban kerül sor. A két fázis a legtöbb esetben időben szétválik. A tanulási fázis rendszerint lassú, hosszú iterációkat, tranzienseket, esetleg sikertelen tanulási szakaszokat is hordoz. Ezzel szemben az előhívási fázis tipikusan gyors feldolgozást jelent (bár – a visszacsatolt hálózatok esetén, ld. 1.2.3. alfejezet – ez is lehet tranziensekkel terhelt).

A két fázis mindamellett nem minden esetben válik szét, adaptív viselkedésű hálók az információ előhívási szakaszban is módosítják a képességeiket, tulajdonságaikat, tanulnak. Erre abban az esetben van szükség, ha a környezet vagy a minták által szolgáltatott információ időben változik, így az információ-feldolgozó eljárásnak is változni kell. Ezekre a hálózatokra is jellemző mindazonáltal, hogy egy hosszabb, tisztán tanulási szakasz előzi meg a párhuzamos tanulási-előhívási szakaszt.

Hangsúlyoznunk kell, hogy bár a fentiek értelmében más programozási eljárások is felfoghatók tanításnak, a neurális hálók esetén az eljárás döntően eltér a hagyományostól. Procedurális programozás esetén például valamilyen matematikai algoritmus kódolását, deklaratív programozás esetén pedig a probléma leírását, matematikai modelljének megalkotását végezzük el. Neurális hálók tanításánál viszont a bemeneti vagy a bemeneti és a kimeneti ismert minták alapján a neurális háló paramétereit módosítjuk a kívánt leképezés

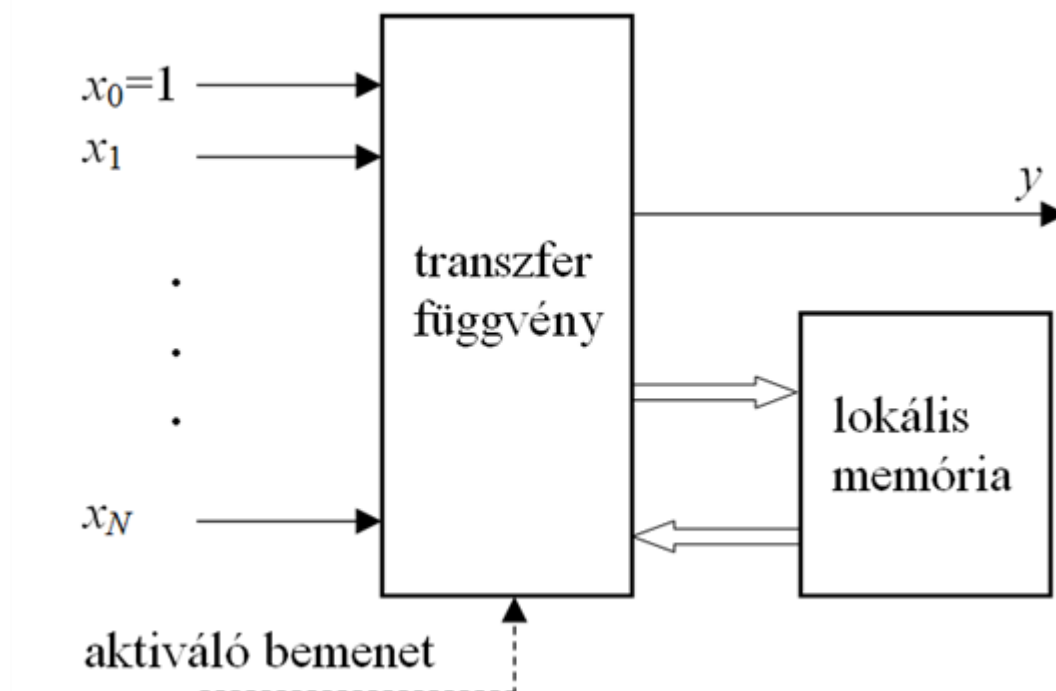
megvalósítása érdekében, anélkül, hogy előzetesen a problémához illeszkedő speciális modellt vagy algoritmust alkotnánk. A neurális háló felépítése, mint látni fogjuk, rendszerint általános, a megoldandó feladatok viszonylag széles körére alkalmas. Az egyes konkrét feladatok megoldása legtöbb esetben az általános struktúrájú eszköz paramétereinek a tanulás során való beállításával történik.

2. A neurális hálózat elemei, topológiája

2.1. A neuronok felépítése

Egy **neuron** vagy **műveleti elem**, **processzáló elem** (*processing element*) egy több-bemenetű, egy-kimenetű eszköz, amely a bemenetek és a kimenet között általában valamilyen nemlineáris leképezést valósít meg. Egy neuron rendelkezhet lokális memóriával is, amelyben akár bemeneti, akár kimeneti értékeket vagy a működés előéletére vonatkozó állapotinformációt tárolhat. A bemeneti- vagy a bemeneti- és a tárolt értékekből az aktuális kimeneti értéket egy tipikusan nemlineáris függvény alkalmazásával hozza létre, melyet **aktiváló** vagy **aktivációs függvénynek** nevezünk (*activation function*). (Használnak egyéb magyar és angol elnevezéseket is, mint pl. neuron esetén feldolgozó elem, vagy **csomópont** (*node*); aktivációs függvény esetén pl. **transzfer függvény** (*transfer function*).

1.1. ábra - A neuron általános, elvi felépítése

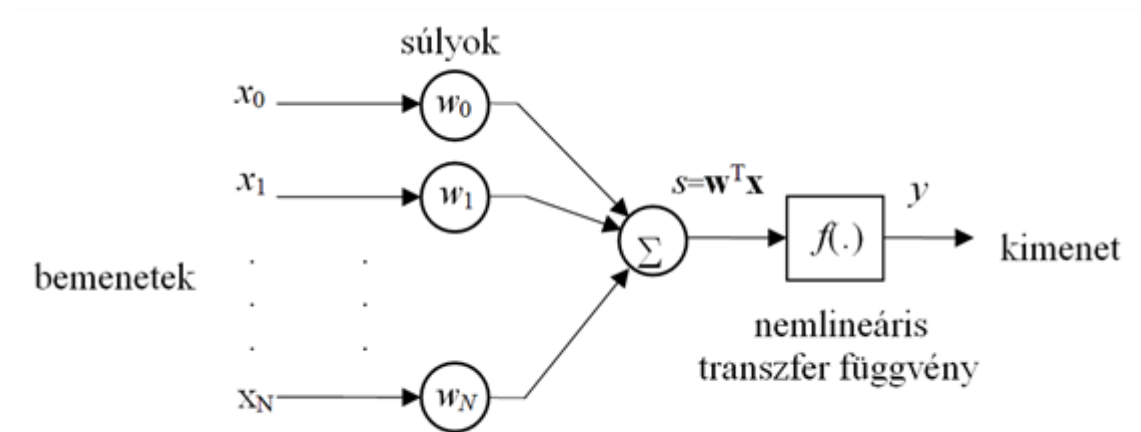


A neuronok rendelkeznek változó értékeket (a hálózat bemeneti jelei, más neuronok kimeneti jelei) hordozó bemenetekkel, illetve rendelkezhetnek állandó értéket hordozó bemenettel, pl. ilyen az 1.1 ábrán x_0 . Az ábrán kitüntetett szerepű aktiváló bemenet diszkrét idejű működés esetén a hálózat ütemezését vezérli, folytonos idejű működés esetén elmarad. A neuron működését az alábbi matematikai formában adhatjuk meg (diszkrét idejű hálózatra, ahol k az időlépés indexe):

$$y(k) = f(\mathbf{x}(k), \mathbf{x}(k-1), \dots, \mathbf{x}(k-M), y(k-1), \dots, y(k-L)) \quad (1.1)$$

ahol $\mathbf{x}(k) = [x_0(k), x_1(k), \dots, x_N(k)]^T$. Az áttekinthetőség kedvéért az N változó- és az egyetlen konstans bemenetet egy $N+1$ elemű $\mathbf{x}$ vektorba fogtuk össze. Ebben az esetben a neuron egy $f: \mathbb{R}^{(N+1) \times 1} \rightarrow \mathbb{R}$ leképezést valósít meg.

1.2. ábra - Egyenrangú bemenetekkel rendelkező memória nélküli neuron (perceptron) felépítése



A fenti általános felépítést az alábbiakban néhány fontos, elterjedt esetre konkretizáljuk. A műveleti elemek legegyszerűbb és egyben legelterjedtebb változata az egyenrangú bemenetekkel rendelkező **memória nélküli neuron**, melynek tipikus felépítése az 1.2 ábrán látható. A bemutatott neuron esetén az x_i skalár bemenetek w_i ($i=0,1,\dots,N$) súlyozással kerülnek összegzésre, majd a súlyozott összeg egy $f(\cdot)$ nemlineáris elemre kerül. Szokás az 1.2 ábrán látható neuronok esetén a bemeneti jelek súlyozott összegét (az összegző hálózat s kimeneti jelét, mely a nemlinearitás bemenete) ingernek (*excitation*), míg a kimeneti jelet válasznak (*activation*) nevezni. Ezek az elnevezések az egyes területeken ma is erősen hivatkozott biológiai analógiára utalnak.

Egyes neuronhálóknál a nemlineáris transzfer függvény elmarad, így lineáris neuronokról is beszélhetünk. Az ilyen lineáris súlyozott összeget megvalósító neuront önmagában viszonylag ritkán használják, legtöbbször egy nagyobb hálózat kimeneteinek előállítására szolgál. Vannak azonban olyan nemellenőrzött tanítású hálók, amelyek lineáris neuronokból épülnek fel. Ezeknél a hálókban a speciális tulajdonságot a súlyok kialakítása, a speciális tanulási eljárás biztosítja.

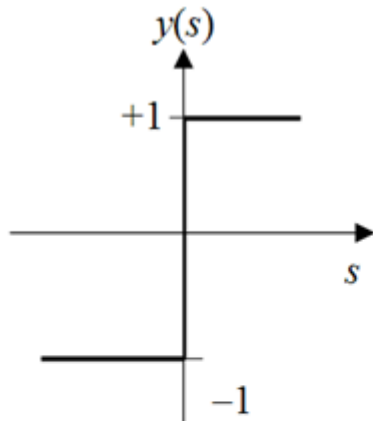
A hálózat összegző pontján a bemenetek lineáris kombinációját kapjuk

$$s = \sum_{i=0}^N w_i x_i = \mathbf{w}^T \mathbf{x} \quad (1.2)$$

A memória nélküli neuronok lineáris változata esetén az összegző kimenete közvetlenül a neuron kimenetét is jelenti:

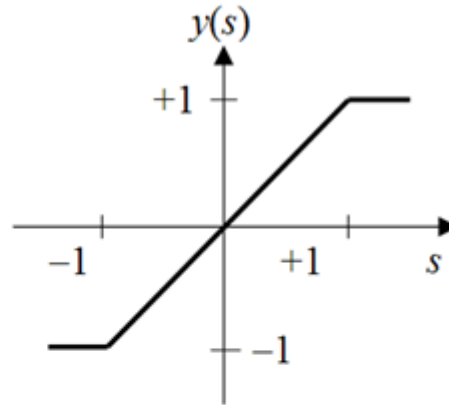
$$y = s = \mathbf{w}^T \mathbf{x} \quad (1.3)$$

1.3. ábra - Neuronokban használt tipikus nemlinearitások



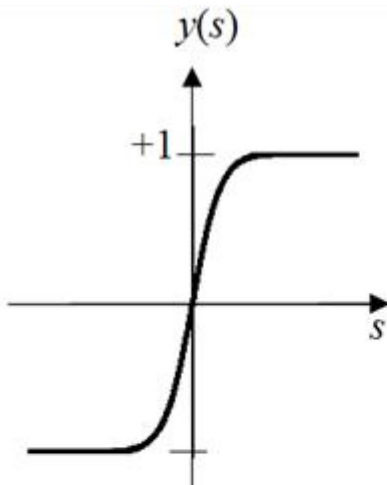
$$y = \begin{cases} +1 & s > 0 \\ -1 & s \leq 0 \end{cases}$$

(a)
lépcsősfüggvény



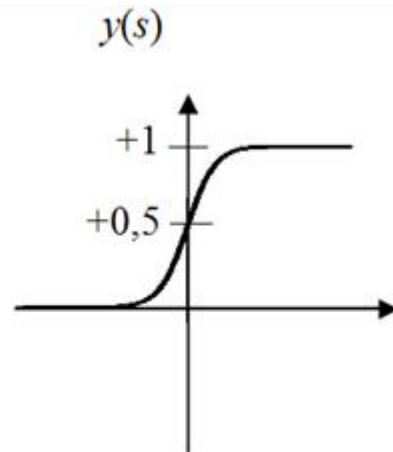
$$y = \begin{cases} +1 & s > 1 \\ s & -1 \leq s \leq 1 \\ -1 & s < -1 \end{cases}$$

(b)
telítéses lineáris függvény



$$y = \frac{1 - e^{-Ks}}{1 + e^{-Ks}}; \quad K > 0$$

(c)
tangens hiperbolikus függvény
($K=2$ -nél)



$$y = \frac{1}{1 + e^{-Ks}}; \quad K > 0$$

(d)
logisztikus függvény

Az 1.2 ábrán bemutatott, összegzővel sorbakapcsolt nemlinearitást használó egyszerű felépítésű neuront *perceptron* néven Frank Rosenblatt [Ros58], illetve *adaline* néven Bernard Widrow [Wid60] javasolta. A két neuron-modell felépítése megegyezik, eltérés csupán az alkalmazott tanítási eljárásokban van, melyeket a 3. fejezetben mutatunk be. Mind az adaline, mind a Rosenblatt-perceptron által megvalósított leképezést a (1.4) összefüggés adja meg:

$$y = f(s) = f(\mathbf{w}^T \mathbf{x}). \quad (1.4)$$

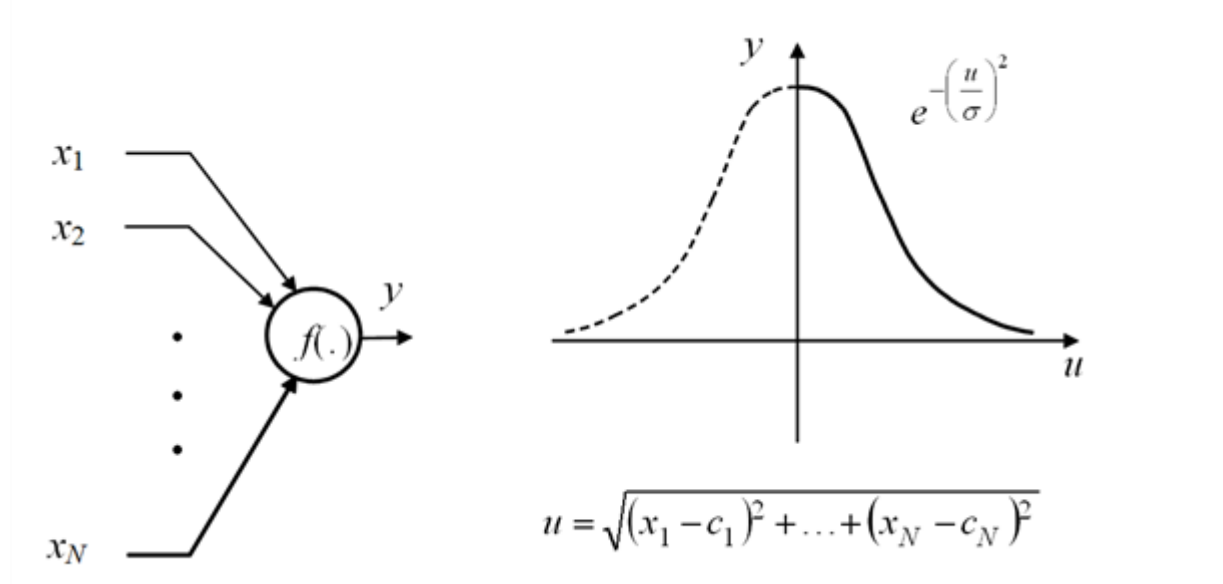
A nemlineáris aktivációs függvény tipikusan küszöbfüggvény jellegű leképezés, melynek értelmezési tartománya általában a valós számok halmaza, értékkészlete pedig a valós számok egy korlátos részhalmaza. Négy legelterjedtebb változata az 1.3 ábrán látható. Az első (a) az ugrásfüggvény, a második (b) a telítéses lineáris függvény, a harmadik (c) és negyedik (d) folytonos, monoton növekvő, telítődő jellegű függvények, melyeket a szigmoid függvényeknek nevezünk. A szigmoid függvényeknek kitüntetett szerepük van a neuronhálóknál. Két leggyakoribb megvalósítási formájuk a tangens hiperbolikus és a logisztikus függvény.

Egyes esetekben a neuron bemeneteit csoportokba osztjuk aszerint, hogy a bemeneti jel a neuron kimenetén a válaszjelet növeli, vagy csökkenti. Ezeket a bemeneti jelcsoportokat a biológiai analógia alapján gerjesztő (*excitatory inputs*), illetve gátló (*inhibitory inputs*) bemeneteknek nevezzük. A gerjesztő és gátló bemenetekkel is rendelkező memória nélküli neuronokra példa a Fukushima által képfeldolgozásra javasolt típus [Fuk88].

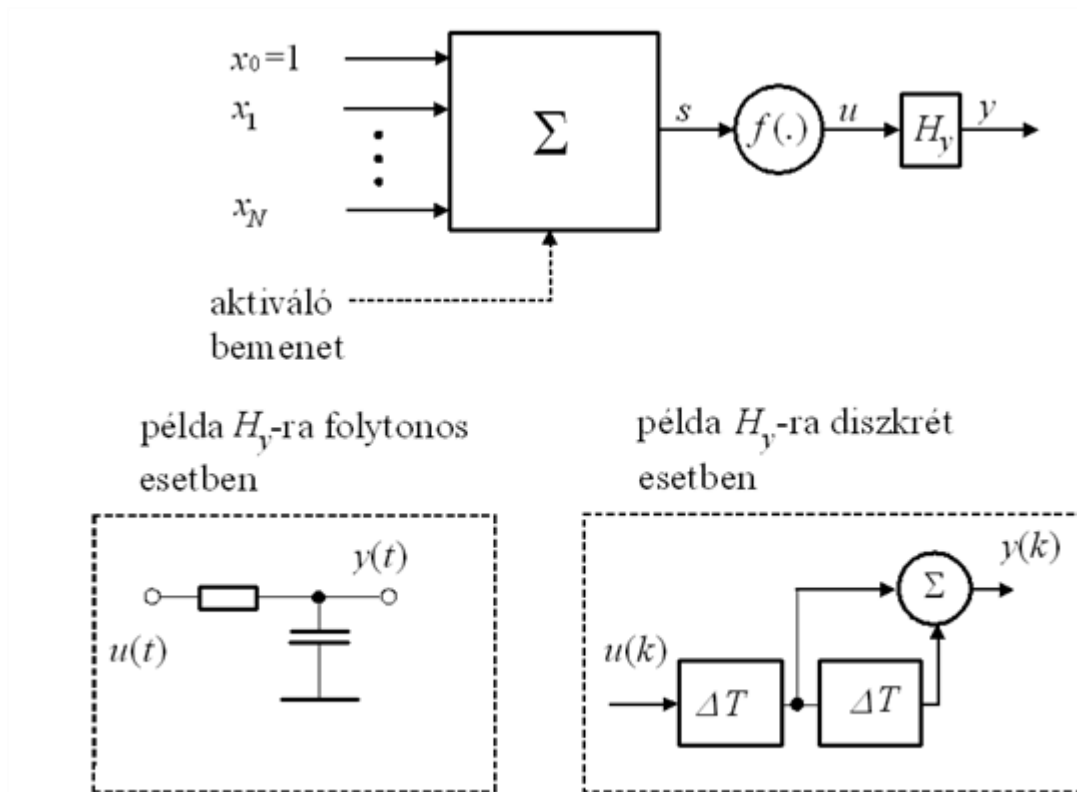
A memória nélküli neuronok másik típusát kapjuk, ha az 1.4 ábrán látható struktúra azon speciális változatát használjuk, mikor minden bemenet közvetlenül a nemlineáris elemre jut, azaz a nemlineáris aktivációs függvény előtt elmarad az összegzés, viszont nem egybemenetű, hanem N -bemenetű a nemlinearitás. Ennek elterjedt és napjainkban nagy jelentőségű típusa az RBF (*Radial Basis Function*) hálózatban használt neuron, de ebbe a csoportba sorolható az összes többi bázisfüggvényes hálózat is. A bázisfüggvényes hálózatokra jellemző még, hogy általában maga a bázisfüggvény is tartalmaz paramétereket, melyeket a háló konstrukciójakor valamilyen módon meg kell határozni.

Az 1.4 ábrán látható, hogy a kimenet tulajdonképpen a bemeneti vektor, $\mathbf{x}=[x_1, x_2, \dots, x_N]^T$ és egy, a neuronra jellemző $\mathbf{c}=[c_1, c_2, \dots, c_N]^T$ középpont-vektor távolságának nemlineáris függvénye. Az ábrán az euklideszi távolságdefiníciót használtuk, de nincs akadálya egyéb távolságfogalmak bevezetésének és használatának sem. A legelterjedtebb esetben Gauss függvényt használunk nemlinearitásként, melynek $\mathbf{c}$ középpont-vektora mellett paramétere még a függvény szélességét meghatározó σ is.

1.4. ábra - Bemeneti összegzést nem használó neuron



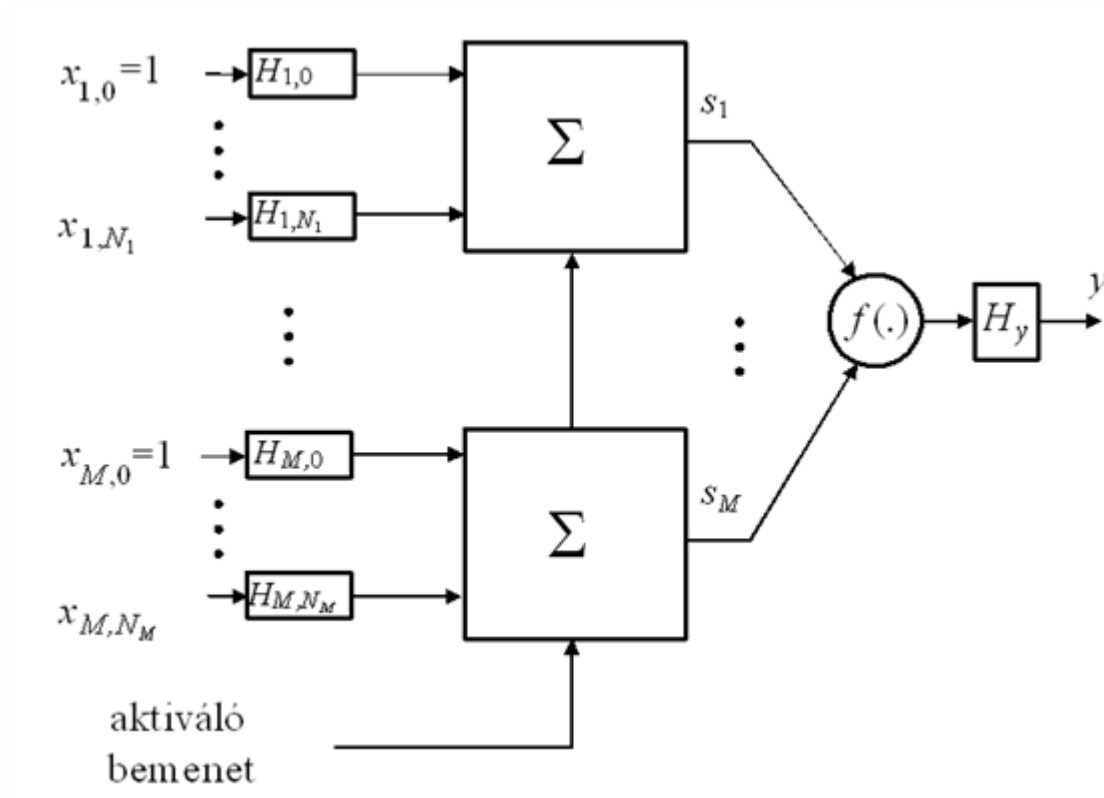
1.5. ábra - Példa memóriával rendelkező neuronra



Az 1.5 ábrán egy a kimenetén H_y lineáris dinamikus hálózattal (lineáris szűrővel) kiegészített (eredetileg) memória nélküli neuron látható. Az ábrán mind folytonos, mind diszkrét idejű esetre egy-egy konkrét, egyszerű dinamikus hálózatot is felrajzoltunk, csupán a szemléltetés kedvéért. (Ezek ebben a formában nem feltétlenül használhatók.)

Az ábrán látható eset kiegészíthető bemeneti lineáris szűrőkkel is, illetve kiterjeszthető több-bemenetű nemlinearitásra is, így egyre bonyolultabb modelleket nyerhetünk. (Természetesen míg a bonyolultabb modell komplexebb problémákra alkalmazható, addig megvalósítási, stabilitási, konvergencia stb. gondokat vethet fel.) Bonyolultabb esetre példa az 1.6 ábrán látható, memóriával és M -bemenetű aktivációs függvényrel rendelkező neuron, mely képes az 1.1 ábrán látható általános műveleti elemmel kialakítható legtöbb funkció megvalósítására. A neuron struktúrája azonban kötöttebb, mint az 1.1 ábrán látható esetben. A tárolási funkció itt is a neuron bemenetén és kimenetén felépített lineáris dinamikus rendszerekben valósul meg. (Implementációs előnyt jelenthet, ha az összegzők kimenetén – az $s_1, \dots, s_M$ jelek szűrésére – valósítjuk meg a lineáris szűrőket, ezek azonban az összegzők bemenetére transzformálhatók, így az elvi áttekintésre elég az 1.6 ábrán látható struktúra.) Diszkrét idejű hálózatokban attól függően, hogy a neuronban használt szűrőhálózatok véges impulzusválaszúak (*finite impulse response*, FIR szűrők), vagy végtelen impulzusválaszúak (*infinite impulse response*, IIR szűrők) **FIR neuronokról** vagy **IIR neuronokról** beszélünk [Wan90]; [Lei91]. Természetesen nincs akadálya egyéb kombinációk kialakításának, pl. a bemeneten FIR, a kimeneten IIR szűrőket használó változatnak sem.

1.6. ábra - FIR/IIR neuron struktúra



2.2. A neurális hálózatban használt összeköttetések

A neurális hálózatban használt összeköttetések egyirányú jelterjedést biztosítanak a hálózat bemenete vagy egy neuron kimenete felől egy neuron bemenete vagy a hálózat kimenete felé. Egy kimenet több (általában tetszőleges számú) bemenettel köthető össze, ekkor valamennyi bemenet ugyanazt a jelet kapja. Kimeneteket egymással nem köthetünk össze. Az összeköttetések késleltetés nélküliek (az esetleges késleltetéseket szűrővel modellezhetjük a neuron bemenetén).

Elsősorban az 1.2.1 fejezetben tárgyalt memória nélküli neuronoknál a bemeneti súlytényezőket gyakran az aktuális bemenetekhez vezető összeköttetésekhez rendelik, nem a neuronhoz. Ez semmilyen elvi különbséget sem jelent.

2.3. A neurális hálózatok topológiája

A hálózat topológiáján a neuronok összeköttetési rendszerét és a hálózat bemeneteinek és kimeneteinek helyét értjük. A hálózat topológiáját általában irányított gráffal reprezentáljuk. A gráf csomópontjai a neuronoknak felelnek meg, míg a kapcsolatokat a kimenetek és bemenetek között a gráf élei reprezentálják, a neuron-bemenettől a kimenet felé irányítva. (Az élekhez értelemszerűen hozzárendelhetjük a megfelelő w_{ij} bemeneti súlytényezőket vagy a $H_{ij}(z^{-1})$ szűrőket, ld. 1.2.1 alfejezet.)

A gráf egyes csomópontjai rendszerint nincsenek az összes többi csomóponttal kapcsolatban, csupán a csomópontok egy részhalmazával. Ez legtöbbször lehetőséget nyújt arra, hogy a gráf csomópontjainak – a neuronoknak – halmazát diszjunkt részhalmazokra bontsuk. Ennek megfelelően háromféle neuront különböztethetünk meg:

- **bemeneti neuronok**, melyek típusukban is különböznek a többi neurontól (egybemenetű, egykimenetű, buffer jellegű neuronok, melyeknek jelfeldolgozó, processzáló feladatuk nincs), bemenetük a hálózat bemenete, kimenetük más neuronok meghajtására szolgál,
- **kimeneti neuronok**, melyek kimenete a környezet felé továbbítja a kívánt információt, típusukra nézve nem feltétlenül különböznek a többi neurontól,

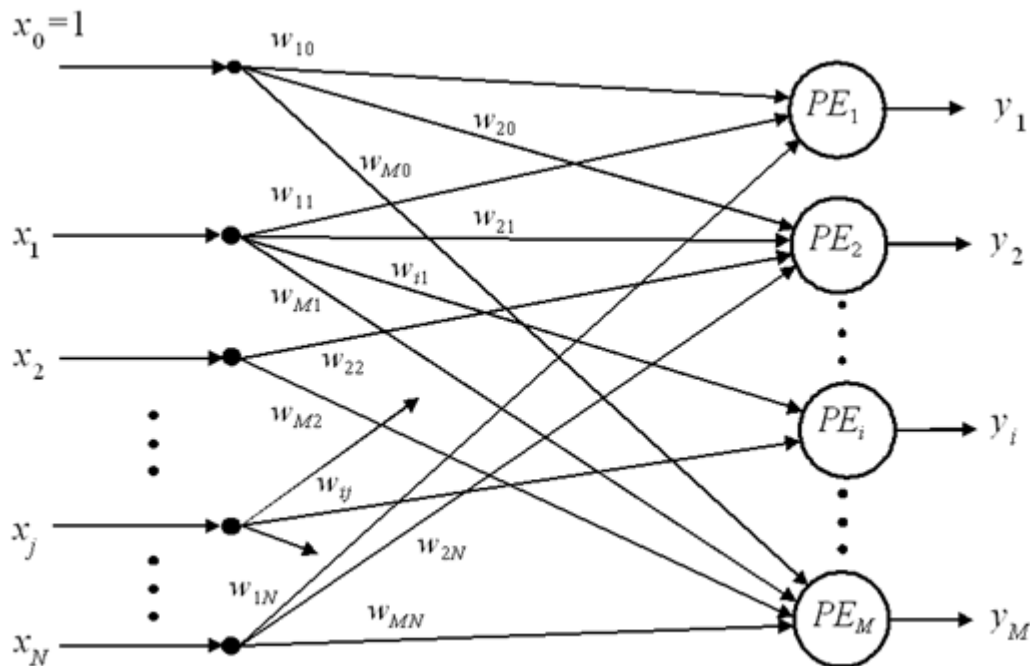
- **rejtett neuronok** (*hidden neurons*), melyek mind bemeneteikkel, mind kimenetükkel kizárólag más neuronokhoz kapcsolódnak.

A neuronokat sok esetben **rétegekbe** (*layers*) szervezzük, ahol egy rétegbe hasonló típusú neuronok tartoznak. Az egy rétegbe tartozó neuronokra még az is jellemző, hogy kapcsolataik is hasonlóak. Az azonos rétegbe tartozó neuronok mindegyikének bemenetei a teljes hálózat bemenetei, vagy egy másik réteg neuronjainak kimeneteihez kapcsolódnak, kimenetei pedig ugyancsak egy másik réteg neuronjainak bemeneteit képezik, vagy a teljes hálózat kimeneteit alkotják. Ennek megfelelően beszélhetünk **bemeneti rétegről** (*input layer*), **rejtett réteg(ek)ről** (*hidden layer(s)*) és **kimeneti rétegről** (*output layer*). A bemeneti réteg, amely buffer jellegű neuronokból épül fel, információfeldolgozást nem végez, feladata csupán a háló bemeneteinek a következő réteg bemeneteihez való eljuttatása. Ennek megfelelően egy rétegekbe szervezett háló legalább két réteggel, egy bemeneti és egy kimeneti réteggel kell rendelkezzen. E két réteg között elvben tetszőleges számú rejtett réteg helyezkedhet el.

A szakirodalomban a rétegszám definíciója sokszor nem egyértelmű. Egyes szerzők a rendszerint lineáris, bemeneti buffer réteget is beleszámítják a rétegszámba, mások csak a rejtett- és kimeneti rétegeket, tehát csak azokat a rétegeket, melyek információfeldolgozást is végeznek. Így az egyértelműség kedvéért az a szokás terjedt el, hogy az aktív, jelfeldolgozást is végző (processzáló) rétegek számát adják meg a hálózat jellemzőjeként.

A neuronhálókat az egyes neuronok közötti összeköttetési rendszer alapján két fő csoportba sorolhatjuk: beszélhetünk **előrecsatolt hálózatokról** (*feedforward networks*) és **viSSZacsatolt hálózatokról** (*recurrent networks*). ViSSZacsatoltnak nevezünk egy neurális hálót, ha a topológiáját reprezentáló irányított gráf tartalmaz hurkot, egyébként a neurális háló előrecsatolt. Előrecsatolt hálóra mutat példát az 1.7 és az 1.8 (b) ábra, míg egy viSSZacsatolt architektúra az 1.8 (a) ábrán látható.

1.7. ábra - Egy egy-(processzáló) rétegű hálózat

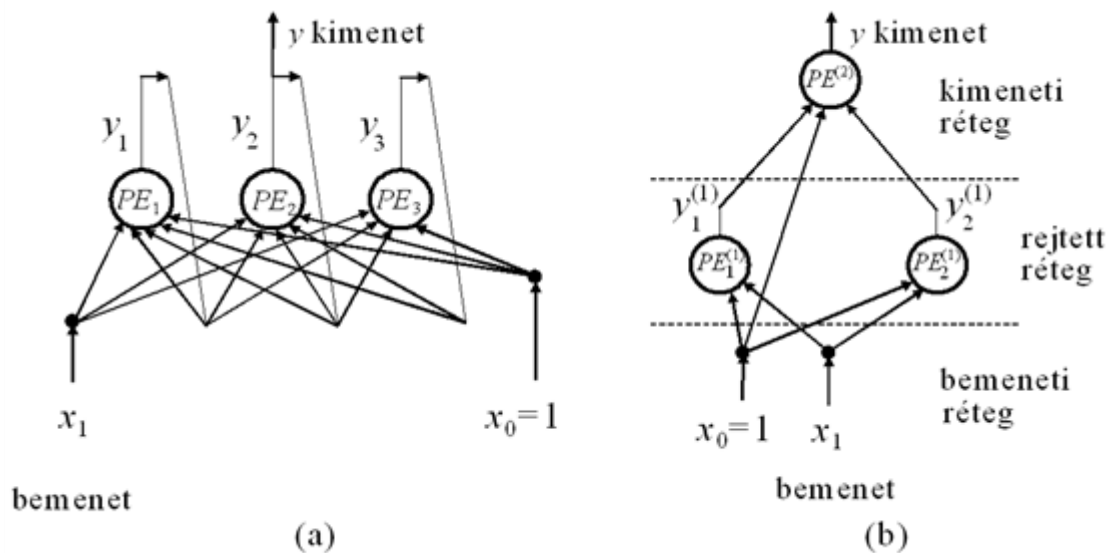


A viSSZacsatolt hálózatoknál beszélhetünk globális és lokális viSSZacsatolásról. Globális viSSZacsatolásnál a hálózat kimenetét csatoljuk vissza a bemenetére. A lokális viSSZacsatolásokat három csoportba sorolhatjuk (1.9 ábra):

- elemi viSSZacsatolásról (*recurrent connections*) beszélünk, ha egy réteg egy neuronjának kimenete közvetlenül egyik saját bemenetére van viSSZacsatolva,

- laterális (*lateral connections, intra-layer connections*) visszacsatolásoknál, valamely réteg(ek) neuronjainak kimenetei ugyanazon réteg neuronjainak bemeneteire kapcsolódnak, de nem értjük ide a neuron önmagára való visszacsatolását.
- a rétegek közötti visszacsatolások (*inter-layer connections*) több réteget tartalmazó hurkot hoznak létre a gráfon.

1.8. ábra - Teljesen összekötött, visszacsatolt (a), illetve rétegekbe szervezett, előrecsatolt (b) topológiájú hálózat



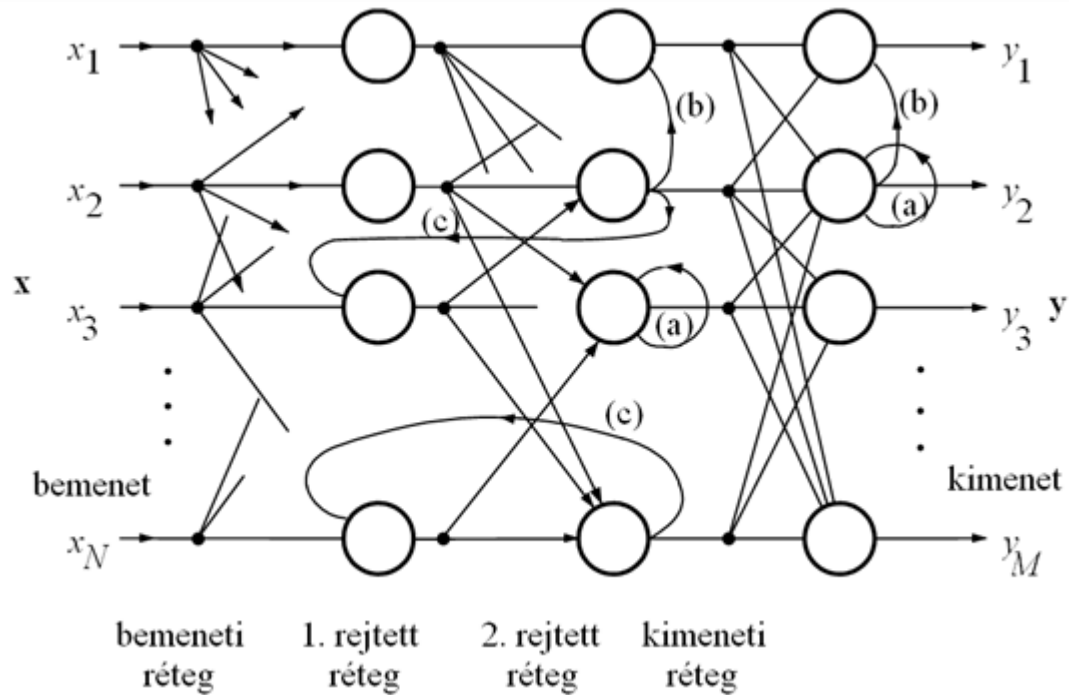
A jobb áttekinthetőség kedvéért az 1.9 ábrán a visszacsatolásokon kívül csak néhány további összeköttetést jelöltünk.

A legegyszerűbb, memória nélküli neuronokból (ld. 1.2 ábra) felépített egy processzáló rétegű előrecsatolt hálózatnak (1.7 ábra) egy bemeneti buffer rétege és egy feldolgozó rétege van, mely egyben a kimeneti réteg is. A hálózat N bemenettel és M kimenettel (M információfeldolgozást végző neuronnal, processzáló elemmel, PE) rendelkezik. Mind a bemeneteket, mind a neuronokat sorszámmal láttuk el. Az $1 \dots N$ sorszámú bemenetek a bemeneti réteget adják, a PE_i -vel jelölt neuronok a kimeneti réteget, ahol $i=1, \dots, M$. Ehhez kapcsolódik még a szokásos konstans értéket biztosító, az ábrán 0 sorszámmal jelölt bemenet. Természetesen itt az 1.2 ábrához képest – a több neuron miatt – a processzáló réteg bemeneti súlyait kettős indexeléssel látjuk el, az első index a neuronra, míg a második a neuronhoz kapcsolódó bemeneti komponensre utal.

Az 1.7 ábrán látható hálózat átvitele az (1.2) és az (1.3) összefüggések, (nemlineáris aktivációs függvény esetén az (1.4) összefüggés) alapján felírható. Értelemszerűen a súlyvektor helyét a súlyokból képzett $\mathbf{W}$ mátrix veszi át, a kimeneten pedig az $\mathbf{y}$ vektort kapjuk. Az alábbi összefüggés a nemlineáris aktivációs függvény esetét mutatja:

$$\mathbf{y} = \mathbf{f}(\mathbf{s}) = \mathbf{f}(\mathbf{W} \mathbf{x}) \quad (1.5)$$

1.9. ábra - Lokális – elemi (a), laterális (b) és rétegek közötti (c) – visszacsatolás



(Megjegyezzük, hogy az (1.5) összefüggés a szokásosnál általánosabb kapcsolatot is megenged $\mathbf{s}$ és $\mathbf{y}$ között. A legtöbb esetben ugyanis az $\mathbf{f}(\cdot)$ leképezés az $\mathbf{s}$ vektor komponenseire hat és így állítja elő $\mathbf{y}$ komponenseit.)

Az 1.8 ábrán egy teljesen összekötött topológiájú visszacsatolt hálózat és egy rétegekbe szervezett topológiájú egybemenetű-egykimenetű, előrecsatolt hálózat látható, mindkettő 3 - 3 neuronból áll. A második háló a fentiek értelmében egy kétrétegű (két processzáló rétegű) vagy másképpen egy rejtett rétegű hálózat. Az ábrán az információfeldolgozást végző neuronokat ismét $PE_i^{(l)}$ -vel jelöltük, ahol a felső index a rétegindex, az alsó pedig a rétegen belüli sorszám. Az egyes neuronok kimenetei – a többrétegű esetben – a neuronok jelöléséhez hasonlóan ugyancsak két indexet kaptak. A felső a rétegindex, az alsó a rétegen belül a sorszámot adja meg. (Hogy a későbbiek során a felső index ne legyen összetéveszthető a hatványkitevővel, a felső indexet minden esetben zárójelbe tesszük.) Az ábrán az áttekinthetőség érdekében a súlyokat nem jelöltük, de a bevezetett konvenciónak és annak megfelelően, hogy a súlyokat neuronokhoz rendeljük, általánosságban az l -edik réteg i -edik neuronjának a j -edik bemenetéhez kapcsolódó súlyát $w_{ij}^{(l)}$ fogja jelölni.

A neurális hálók – elsősorban a rétegekbe szervezett memória nélküli hálók – topológiáját nemcsak gráf reprezentációval adhatjuk meg, hanem az egyes rétegek közötti összeköttetést mátrixok segítségével is leírhatjuk.

Előrecsatolt hálónál (pl. 1.8 (b) ábra) ez értelemszerűen azt jelenti, hogy az 1.5 összefüggéshez hasonlóan az egyes rétegek neuronjaihoz tartozó súlyok egy mátrixba foghatók össze, ahol a mátrix egyes sorai az egyes neuronok súlyaiból képezett vektorok. $\mathbf{W}^{(l)}$ tehát az l -edik rétegben lévő neuronok súlyaiból, mint sorvektorokból képzett mátrix. Egy többrétegű, előrecsatolt háló egyes rétegei által megvalósított leképezés tehát, ha összeköttetések csak a szomszédos rétegekben lévő neuronok között vannak, a következő formában adható meg:

$$\mathbf{y}^{(l)} = \mathbf{f}(\mathbf{s}^{(l)}) = \mathbf{f}(\mathbf{W}^{(l)} \mathbf{y}^{(l-1)}) \quad (1.6)$$

Egy két rejtett réteggel rendelkező hálózat bemeneti-kimeneti leképezése ennek megfelelően:

$$\mathbf{y} = \mathbf{f}(\mathbf{W}^{(2)} \mathbf{f}(\mathbf{W}^{(1)} \mathbf{x})) \quad (1.7)$$

Itt $\mathbf{W}^{(1)}$ illetve $\mathbf{W}^{(2)}$ rendre az első és a második aktív (rejtett) réteg súlyait tartalmazó mátrix, az $\mathbf{f}(\cdot)$ függvényt komponensenként kell érteni. Az 1.8 (b) ábrán látható hálózathál $\mathbf{W}^{(1)}$ egy 2×2 -es mátrix, míg $\mathbf{W}^{(2)}$ egy 3 elemű vektor:

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} w_{10}^{(2)} & w_{11}^{(2)} & w_{12}^{(2)} \end{bmatrix} \quad (1.8)$$

A súlyok mátrixos formában történő felírása természetesen visszacsatolt hálónál is lehetséges, hiszen a súlyok itt is neuronok bemeneteihez kapcsolódnak. A konkrét összeköttetési rendszer szabja meg, hogy az egyes neuronok honnan kapnak bemenetet; visszacsatolt hálózatoknál általában mind közvetlen bemenetek, mind visszacsatolt kimenetek is szerepelnek bemenetként. Visszacsatolt hálónál természetesen figyelembe kell venni, hogy a visszacsatolt háló dinamikus működést valósítanak meg, és hogy egy neuronon a jeláthaladás késleltetéssel jár. Diszkrét idejű hálózatot feltételezve ez azt jelenti, hogy egy visszacsatolt háló kimenete a k -adik időpillanatban a bemeneteinek a $(k-1)$ -edik időpillanatbeli értékeitől függ. Például az 1.8 (a) ábrának megfelelő egy neuronrétegből álló visszacsatolt hálózat leképezésének általános formája

$$\mathbf{y}(k) = \mathbf{f}(\mathbf{W}\mathbf{u}(k-1)) \quad (1.9)$$

ahol $\mathbf{u}$ a neuronok összes bemenetét tartalmazó vektort jelöli. Az ábrának megfelelő konkrét hálózathoz a kimenet skalár, a bemeneti vektor pedig mind közvetlen bemeneti komponensekből, mind visszacsatolt kimenetekből áll:

$$\mathbf{y}(k) = y_2(k) \quad \text{és} \quad \mathbf{u}(k) = [x_0(k), x_1(k), y_1(k), y_2(k), y_3(k)]^T \quad (1.10)$$

3. A neurális hálózatok alapvető számítási képeségei, felhasználási területei

Az előbbieken láttuk, hogy a neuronháló a bemeneteik és a kimeneteik között általában valamilyen nemlineáris leképezést valósítanak meg. Egy statikus leképezésű elemi neuron ezért, mint elemi (nemlineáris) statikus rendszer kezelhető. Azt is láttuk, hogy megfelelő kiegészítésekkel, bővítésekkel az elemi neuron dinamikus működésűvé tehető. Egy neuronháló – amely tehát ilyen elemi neuronokból épül föl – úgy kezelhető, mint egy statikus vagy dinamikus (nemlineáris) rendszer, amelynek alkalmazhatósága attól függ, hogy milyen speciális képességekkel rendelkezik. Fontos kérdés tehát az egyes neuronháló architektúrák számítási képességeinek a meghatározása. Ebben a részben röviden ezt a kérdést vizsgáljuk, valamint kitérünk arra is, hogy ezeknek a képességeknek köszönhetően melyek a neuronháló legfontosabb alkalmazási területei.

A neurális hálózatok képeségeit és tulajdonságait a használt műveleti elemek tulajdonságai, a hálózat topológiája és a hálózat súlyainak értéke, – így közvetve a használt tanulási algoritmus – határozzák meg. A neuronháló legfontosabb képessége, hogy megfelelő kialakítás esetén, szinte tetszőleges be-kimeneti kapcsolatot, leképezést meg tudnak valósítani, vagyis a szokásos elnevezés szerint ezek a neuronháló ún. univerzális approximátorok. Az univerzális approximátor-képesség teszi lehetővé, hogy a neuronhálót mint univerzális osztályozó és modellező eszközöket tekintsük. Az osztályozó/modellező képesség nagyon szerteágazó alkalmazási lehetőségeket biztosít, kiemelt fontosságú, ezért e kérdéssel részletesebben is foglalkozunk. Ugyanakkor ezek a hálózatok más nézőpontból asszociatív memóriaként viselkednek, ezért röviden erre is ki fogunk térni. Végül megemlítjük, hogy egyes neuronháló hatékonyan alkalmazhatók bizonyos optimalizálási feladatok megoldásánál.

3.1. A neurális háló, mint approximáló rendszer

A megfelelő (statikus) bemenet-kimenet leképezés (függvénykapcsolat) előállításának központi feladata a mérnöki alkalmazások számos területén, pl. képfeldolgozásban, alakfelismerésben, osztályozási feladatokban, rendszer identifikációnál, szabályozástechnikában, jelfeldolgozásban stb. Ezért a neurális hálózatok igen fontos tulajdonsága, hogy a hálózatok számos típusa képes minták alapján a keresett leképezés közelítő, sok esetben tetszőleges pontossággal közelítő előállítására. Számos matematikai eredmény született a neurális háló függvényapproximációs (valamilyen előírt leképezést közelítő) képességeivel kapcsolatban. Ezek az eredmények a terület jelentősége miatt nagymértékben növelték a neurális háló iránti érdeklődést.

Kezdetben a neurális hálót szinte kizárólagosan olyan feladatok megoldására használták, ahol a pillanatnyi bemenetre adott válasz nem függött sem az időtől, sem az előző bementi és/vagy kimeneti értékektől, tehát a

megoldandó probléma statikus volt. A működés még akkor is statikus jellegű volt, mikor a kívánt választ csak valamilyen – időben lejátszódó – tranziens után érte el a hálózat, mivel nem a tranziens alakulása, hanem csak a végállapot jelenti a hálózat bemenetre adott választát.

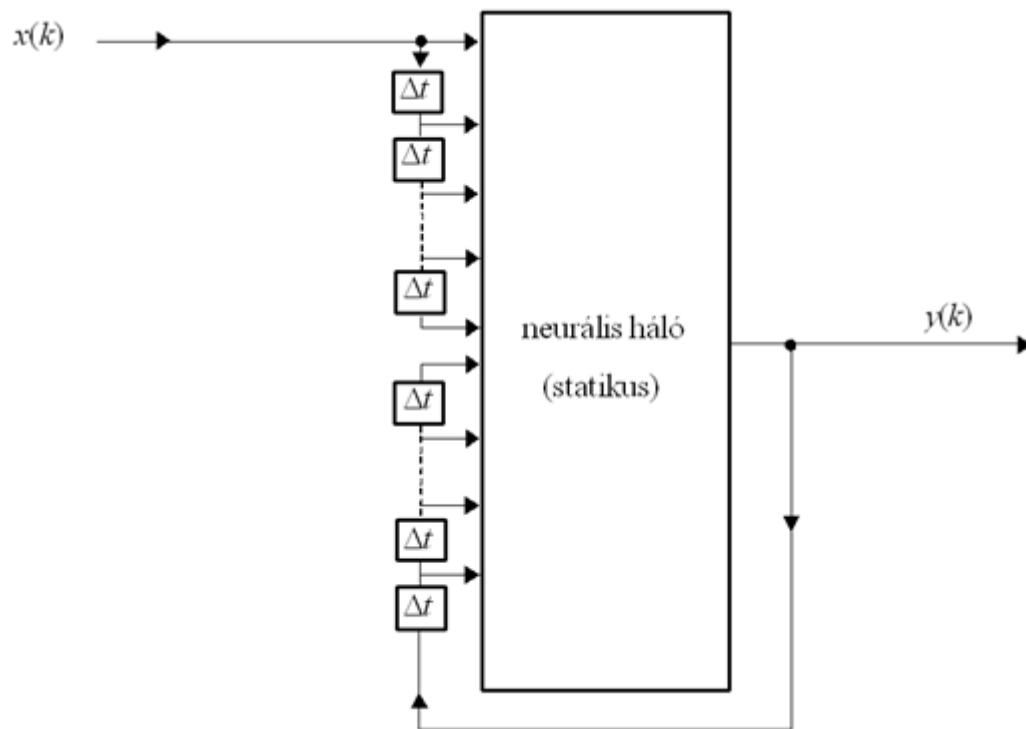
A feladatok jelentős része viszont valamilyen dinamikát mutat: vagy maga a vizsgált folyamat (melyet optimalizálunk, vagy melynek az eredményeit osztályozzuk, stb.), esetleg a környezet (zaj, stb.) változik az időben (idővariáns folyamatok), vagy a folyamat pillanatnyi értéke nem csupán az adott bemeneti jeltől, hanem az előzményektől is függ. Ilyen feladattal találkozunk akkor is, ha egy időben változó értéksorozat valahány elmúlt értékének (egy időben változó folyamat régebbi mintavételi értékeinek) ismeretében a következő értéket vagy értékeket kell előre jelezni, jósolni (a későbbi mintavételi értékekre becslést adni). Ezeket a feladatokat idősor előrejelzési feladatoknak (*time series prediction*) szokás nevezni, és a legkülönbözőbb alkalmazási területeken, pl. komplex ipari rendszerek működésének, gazdasági és pénzügyi folyamatok viselkedésének, természeti jelenségek lefolyásának, időbeli alakulásának, stb. előrejelzésénél jelennek meg.

Természetesen ilyen esetekben a feldolgozó algoritmusnak vagy modellező eljárásnak is a megfelelő időfüggést kell mutatnia. Ezt neurális hálók alkalmazása esetén legtöbbször úgy érjük el, hogy a statikus feladatok megoldására kialakított (rendszerint nemlineáris) neurális hálót kiegészítjük dinamikus (rendszerint lineáris) komponensekkel. Több szinten történhet a dinamikus elemek felhasználása, pl. maga a neuron is dinamikussá tehető, mint ahogy azt az 1.1 illetve 1.6 ábrán bemutattuk. Dinamikus feladatok megoldásánál ugyanakkor gyakrabban alkalmaznak olyan megoldást, amikor a tisztán statikus hálót kiegészítik dinamikus komponensekkel. Erre számos lehetőségünk van. Ilyen lehetőséget mutat az 1.10 ábra, de az 1.8 (a) ábrán bemutatott visszacsatolt háló is egy statikus hálóból megfelelő visszacsatolás alkalmazásával előállított dinamikus rendszer.

Az 1.10 ábrán látható megoldásban a (statikus) neurális hálót bemeneti és kimeneti megcsapolt késleltető sorokkal (*tapped delay line*) egészítettük ki, így az aktuális kimeneti érték az előző be- és kimeneti értékektől is függ. Természetesen használhatunk összetettebb kiegészítő lineáris komponenseket is, mind a visszacsatoló, mind az előreccsatoló ágban.

Akármilyen módon alakítunk is ki dinamikus neurális rendszereket, az alapvető problémát a hálózat tanítása jelenti. A tanítás rendszerint statikus esetben is lassú, nehézkes, dinamikus esetben pedig még ehhez képest is sokkal komplikáltabb, lassabb, sőt gyakran súlyos stabilitási problémákkal is meg kell küzdenünk.

1.10. ábra - Késleltető sorokkal dinamikussá tett neurális rendszer



Az approximációs képességhez számos matematikai eredmény kapcsolódik, melyeket fontosságuk miatt külön részben, az 1.4 alfejezetben foglalunk össze. A matematikai eredmények a statikus leképezésre vonatkoznak, azonban mint láttuk a dinamikus feladatok megoldására képes hálózatok is jórészt statikus neuronokon, statikus hálókra alapulnak, tehát a statikus leképezésre vonatkozó eredmények adják meg annak is a matematikai alapját, hogy megfelelő architektúrájú neuronhálókkal dinamikus feladatokat is hatékonyan meg tudunk oldani.

3.2. A neurális hálózat, mint asszociatív memória

A neurális hálók definíciójából is látható, hogy egy neuronháló egy speciális memóriaként is felfogható, melybe meghatározott módon (tanulás, *learning*) információt írunk be, s ezt később visszaolvassuk (előhívás, *recall*). Az általánosan használt írható/olvasható (RAM) memóriáknál az adat kiválasztott címre való beírását tekinthetjük tanulásnak, az előhívási algoritmus során pedig a címinformációt használjuk bemeneti jelként, melyre válaszként a tárolt adatot kapjuk meg. Ezzel szemben a neurális hálózatokat, mint adattal címezhető memóriát (*content addressable memory*), vagy mint asszociatív memóriát (*associative memory*) használjuk.

Adattal címezhető memória esetén a tanulás során a tanítási adathalmaz elemeit kapcsoljuk a hálózat bemenetére, s a hálózat autonóm vagy felügyelt módon azonos címhez (azaz a háló kimenetéhez) igyekszik hozzárendelni az összetartozó adatokat. Az előhívási fázisban is adatokat kapcsolunk a hálózat bemenetére, s a visszanyert információt az hordozza, hogy a megtanult címek közül melyiket adja vissza a hálózat. (Pl. ha osztályozási feladatot tanítunk, sikeres tanulás eredményeképp az azonos osztályba tartozó bemeneti adatok azonos, vagy egymáshoz közeli címeket adnak a kimeneten, míg a különböző osztályba tartozó bemeneti minták távoli címeket adnak.)

Asszociatív memóriaként alkalmazva a neurális hálót összetartozó adatkárokat használunk, s a hálózatot arra tanítjuk, hogy az adott bemeneti adathoz a megfelelő kimeneti adatot rendelje hozzá. Az előhívás során adatot használunk bemenetként, és válaszként is egy – a bemenethez kapcsolt, asszociált – adatot kapunk. (Pl. itt említhetjük a jelfeldolgozási feladatokat, mikor a bemeneti jelhez egy szűrt/módosított kimeneti jel tartozik.)

Az asszociáció két változatát, az autoasszociációt és a heteroasszociációt szokás megkülönböztetni. Autoasszociatív (*autoassociative*) hálóról akkor beszélünk, ha a tanítás során a neurális háló a bemenetére kapcsolt jelvektorokat, mintákat tárolja abból a célból, hogy a későbbiek során, ha a háló bemenetére a megtanított minták részleges vagy torzított (zajos) változatai kerülnek, a háló az eredeti, teljes, zajmentes mintát adja vissza válaszként a kimenetén. Ezek a hálók a bemeneti mintákat önmagukkal asszociálják. Autoasszociatív hálót használhatunk akkor is, ha adott jelek hatékony közelítő reprezentációját akarjuk előállítani. A háló ilyenkor olyan közbenső, közelítő reprezentációt állít elő, melyből maga a háló próbálja visszaállítani az eredeti jelet. A heteroasszociatív (*heteroassociative*) hálónál a bemeneti vektorokhoz, mintákhoz tetszőleges kimeneti vektorok kapcsolhatók. A háló tehát összetartozó párokat kell, hogy eltároljon, és tanítás után egy megadott bemenetre a hozzátartozó kimenetet kell válaszként előállítania, vagyis egy bemeneti mintát egy tetszőleges, de a bemeneti mintához párosított mintával asszociálja.

Általánosságban mind a bemeneti, mind a kimeneti minta lehet egy-egy mintaszekvencia is. Az ilyen összerendelésekre a dinamikus hálók képesek. A dinamikus hálók tehát mint bemeneti adatszekvenciákhoz kimeneti adatszekvenciákat asszociáló asszociatív memóriák is értelmezhetők.

3.3. A neurális háló, mint optimalizáló rendszer

Az optimalizálás a mérnöki tevékenység egyik alapvető lépése. Optimalizálás során általában valamilyen költséget kívánunk minimalizálni, a rendszer struktúrájának, illetve paramétereinek megfelelő megválasztásával. Nem feltétlenül klasszikus, pénzben kifejezhető költségre kell gondolnunk, költségként jelentkezhet a szükséges számítási kapacitás, a számítási idő, a memória igény, a modellezési hiba, stb. Általában egy költségfüggvénynek két fontos jellemzője van, egyrészt minden megoldáshoz egyértelműen hozzárendel egy skalár mennyiséget, másrészt ez a skalár mennyiség kifejezi a megoldás – valamilyen szempontból értelmezett – jóságát. Egy tipikus példa, ha egy gyártási folyamat során elvégzendő feladatokat akarjuk egy gépcsoporttal (melyben a gépek eltérő sebességgel azonos, illetve különböző részműveletek elvégzésére képesek) úgy megoldani, hogy a műveletek összes idejét, plusz a várakozási időket minimalizáljuk. Az optimalizálási feladat során azt keressük, hogy mely gépekhez mely részműveleteket milyen sorrendben kell hozzárendelnünk.

Neurális hálózatokat sikerrel alkalmaztak bizonyos optimalizálási feladatok megoldásában is. A megoldás menete általában az, hogy egy, a minimalizálni kívánt költséget kifejező skalár függvényt keresünk – a fenti példában az összidő lehet ilyen függvény –, ezután olyan hálózatot hozunk létre, melynek paramétereire

egyértelműen hozzárendelhető a költségfüggvény. (Pl. egy-egy neuron megfelel egy-egy részműveletnek, illetve gépnek, a részműveletet reprezentáló neuron kimenetét a használt géphez kapcsoló súly értéke pedig az adott műveleti idő és várakozási idő összege.) Az optimalizációs feladatokra alkalmas hálók többnyire visszacsatolt struktúrájúak, ami azt jelenti, hogy a háló valamely kiinduló állapotból egy tranzienst során jut el egy végállapotba. Amennyiben a hálózatot úgy tudtuk kialakítani, hogy tetszőleges állapotból indítva, a lejátszódó tranzienst során a költségfüggvény minden lépésben csökkenjen, akkor a végállapot valamilyen költségminimumot reprezentál.

A neuronhálókkal megoldott optimalizációs feladatokra általában az a jellemző, hogy a kapott megoldás nem feltétlenül biztosítja, hogy a globális költségminimumot, tehát az optimumot elérjük. Bonyolult problémák esetén azonban sokszor megelégszünk azzal is, ha az optimum helyett egy „jó” megoldást találunk. Különösen akkor igaz ez, ha az optimalizálási feladat komplexitása miatt a valódi optimum elérése nem reális, miközben egy neuronháló alapú „jó” megoldás kis számításgigánnyal is elérhető. Tipikusan ilyenek a kombinatorikus optimalizálási feladatok, melyek között számos NP-teljes problémát találunk.

4. A neurális hálózatok approximációs képessége

Ebben a részben az előrecsatolt, többrétegű hálózatok approximációs képességéhez kapcsolódó néhány fontosabb matematikai eredményt ismertetünk. Látni fogjuk, hogy ezek a matematikai tételek elsősorban elvi jelentőségűek: egyes előrecsatolt neurális architektúrákhoz kapcsolódóan általános approximációs eredményeket fogalmaznak meg, de ezt egzisztencia tételek formájában teszik. Azt mondják ki, hogy bizonyos képességek megvalósíthatók előrecsatolt többrétegű neuronhálókkal, de nem adnak támpontot ahhoz, hogy adott konkrét feladathoz hogyan konstruálhatunk megfelelő hálót. Fontosságuk a közvetlen gyakorlati felhasználhatóság hiánya ellenére vitathatatlan. A mérnöki feladatmegoldáshoz biztos támpontot adnak, hiszen abban biztosak lehetünk, hogy adott feladatosztályba tartozó problémákat megfelelő neuronhálóval meg tudunk oldani. A matematika ezt a biztos háttértudást adja, a mérnök feladata pedig a konkrét megoldás megtalálása, a megfelelő háló létrehozása.

Az approximációs képességekhez kapcsolódó eredmények bemutatásánál a lehetőségekhez képest az érthetőséget részesítjük előnyben a precíz matematikai megfogalmazással szemben. A tételeket és a feltétlenül szükséges definíciókat kimondjuk ugyan, de a bizonyításokat sehol sem közöljük, mivel ezek általában nem nyújtanak segítséget a mérnöki konstruktív eljárások kidolgozásához, vagy a probléma mélyebb megértéséhez.

A különböző eredmények egy része két nemlineáris réteget tartalmazó hálókra, míg más eredmények egy nemlineáris réteget tartalmazó hálókra vonatkoznak. Az alábbiakban e szerint csoportosítva tárgyaljuk az approximációs képességre vonatkozó tételeket.

4.1. Matematikai leképezések közelítése neurális hálóval, a probléma megfogalmazása

Az eddigiekben láttuk, hogy egy előrecsatolt, rétegekbe szervezett neuronokból felépülő neuronháló a bemenetei és a kimenetei között valamilyen (nemlineáris) leképezést valósít meg. E neuronháló-család alkalmazási körét az határozza meg, hogy a hálók által megvalósított leképezések milyen jellegű függvénykapcsolattal írhatók le. Precízebb megfogalmazásban arra keressük a választ, hogy egy előrecsatolt többrétegű hálózat által megvalósítható leképezés milyen függvényosztályba tartozó $y=f(x)$ függvényeket tud pontosan előállítani, vagy legalább valamilyen értelemben közelíteni. E hálók számítási képessége a függvényapproximáció témakörének speciális szelete.

4.2. Két nemlineáris réteget használó approximációs hálózatok

1900-ban a híres német matematikus, David Hilbert (1862-1943) 23 érdekes matematikai problémát fogalmazott meg, melyek megoldása véleménye szerint a XX. század matematikusainak fontos feladata lesz. A felsorolt problémák között a 13. volt az a sejtés, melynek eredeti alakja mérnöki szemmel nézve meglehetősen érdektelennek tűnik [Lor76]:

"Bizonyítsuk be, hogy az $x^7+ax^3+bx^2+cx+1=0$ hetedfokú egyenlet nem oldható meg pusztán kétváltozós függvények segítségével!"

Hilbert 13. problémájának más matematikusok által adott átfogalmazása már számunkra is több érdekességet mutat:

"Mutassuk meg, hogy van olyan háromváltozós folytonos függvény, mely nem írható fel véges számú kétváltozós folytonos függvény segítségével!"

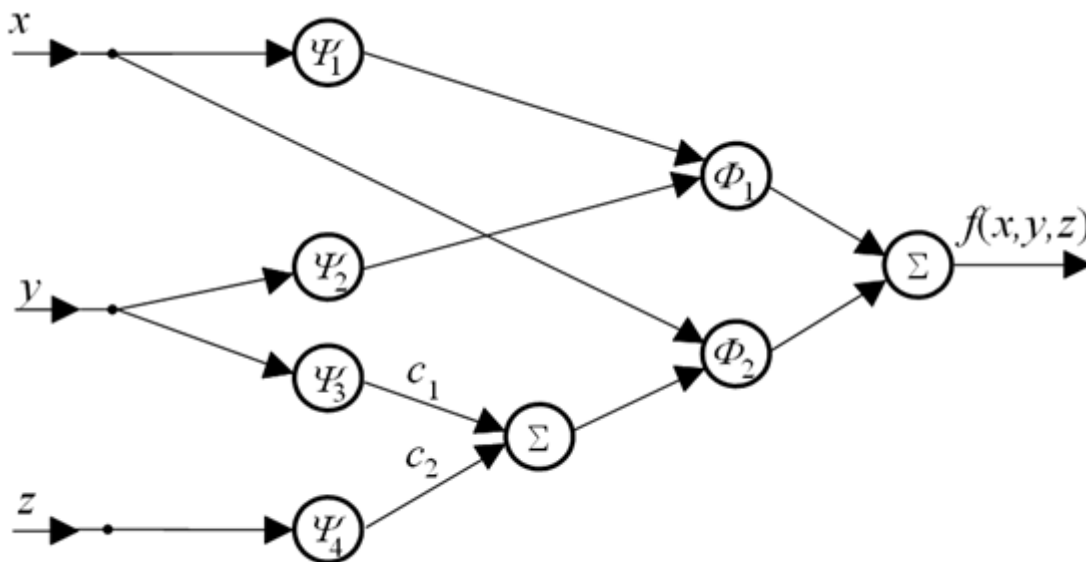
Ez a megfogalmazás ugyanis számunkra azt jelenti, hogy vannak olyan (nemlineáris) hárombemenetű rendszerek, melyek nem dekomponálhatók egy- vagy kétbemenetű rendszerek segítségével. Mivel a bonyolult problémák mérnöki megközelítése igen gyakran az egyszerűbb rendszerekre való dekompozíció, ha Hilbert sejtése igaznak bizonyul, akkor ezen rendszerek megvalósíthatósága várhatóan igen súlyos nehézségekbe ütközik.

Itt a kétváltozós függvényeket részint szuperpozícióban használhatjuk, illetve megengedett a függvény-függvénye struktúra használata is. Például a $\phi_1(\cdot, \cdot)$ és a $\phi_2(\cdot, \cdot)$ kétváltozós és a $\psi_1(\cdot), \psi_2(\cdot), \psi_3(\cdot), \psi_4(\cdot)$ egyváltozós függvények felhasználásával a háromváltozós f függvénynek ebben az értelemben legális felírása az alábbi:

$$f(x, y, z) = \phi_1(\psi_1(x), \psi_2(y)) + \phi_2(c_1\psi_3(y) + c_2\psi_4(z), x) \quad (1.11)$$

A fenti összefüggés tulajdonképpen egy neurális háló struktúrával megvalósítható leképezést takar (ld. 1.11 ábra), ahol a bemeneten használt lineáris buffer neuronok mellett összegző és kétbemenetű nemlineáris neuronokat használunk, a neuronok aktivációs függvényei pedig rendre az összefüggésben használt egyváltozós függvények.

1.11. ábra - Két nemlineáris aktivációs függvényű réteget alkalmazó háló



A helyzet a neurális hálókval való approximáció szempontjából akkor vált igazán érdekessé, mikor 57 évvel később Andrej Kolmogorov (1903-1987) megcáfolta a Hilbert sejtést, és bebizonyította, hogy nem csupán minden háromváltozós függvény, hanem tetszőleges N -változós folytonos függvény felírható csupán **egy**változós függvények és az összeadás segítségével. (Érdekességként megemlíjtük, hogy Kolmogorov és Vladimir Arnold versenyeztek egymással Hilbert 13. problémájának megoldásában, egyre újabb és újabb eredményeket értek el az 50-es évek során, míg 1957-ben Kolmogorov bebizonyította az eredeti Hilbert sejtés cáfolatán messze túlmutató reprezentációs tételét.)

1.1 tétel [Kol57]

Minden $N \geq 2$ egész esetén található $N(2N+1)$ olyan folytonos, monoton növekvő, egyváltozós, az $I=[0,1]$ intervallumon értelmezett $\psi_{\mathcal{M}}$ függvény, melyek segítségével tetszőleges N változós, valós, folytonos $f: I^N = [0,1]^N \rightarrow \mathbb{R}$ függvény az alábbi alakban előállítható

$$f(x_1, x_2, \dots, x_N) = \sum_{q=1}^{2N+1} \phi_q \left(\sum_{p=1}^N \psi_{pq}(x_p) \right) \quad (1.12)$$

Az összefüggésben felhasznált $\phi_q(\cdot)$ $q=1, \dots, 2N+1$ egyváltozós függvények különböző f -ek esetén különbözőek, de a $\psi_{pq}(\cdot)$ $p=1, \dots, N$; $q=1, \dots, 2N+1$ egyváltozós függvények csak N értékétől függenek (minden f -re közös). A fenti összefüggéssel megadott függvénykapcsolat egy olyan két rejtett rétegű neuronháló leképezésének is megfeleltethető, ahol a $\psi_{pq}(x_p)$ kapcsolatok adják meg az első rejtett rétegbeli neuronok leképezéseit, az

$$y_q = \phi_q \left(\sum_{p=1}^N \psi_{pq}(x_p) \right) \quad (1.13)$$

változók pedig a második rejtett réteg kimenetei. Ezekre a rejtett rétegbeli változókra hat a kimeneti réteg, mely az y_q függvényértékek összegét határozza meg. A Kolmogorov-féle reprezentációs összefüggés tehát egy speciális, két rejtett rétegű háló struktúrájának felel meg. Mindkét rejtett rétegbeli neuronok nemlineáris neuronok $\psi_{pq}(\cdot)$, illetve a $\phi_q(\cdot)$ transzfer függvényekkel, míg a kimeneti réteg egy egyszerű összegző, itt tehát nincs transzfer függvény.

Míg a Kolmogorov tétel rendkívül ígéretes a neurális háló típusú eszközökkel való approximáció szempontjából, a valós mérnöki feladatok megoldására nem használható közvetlenül. Ugyanis a tétel bizonyítása nem konstruktív, tehát adott N és f esetén sem a ψ_{pq} függvényrendszer, sem a ϕ_q függvényrendszer előállítására nem ad módszert.

A Kolmogorov tételt az elmúlt évtizedek során többen továbbfejlesztették. Ilyen pl. a David Sprecher által adott finomított tétel. A továbblépés abban áll, hogy a Kolmogorov tételben használt függvényrendszerek helyett csupán két nemlineáris függvény segítségével állítja elő az ismeretlen leképezést. A két függvényt többször is fel kell használniuk a pontos előállításhoz.

1.2 tétel [Spr65]

Létezik olyan monoton növekvő, valós $\psi(x): [0,1] \rightarrow [0,1]$ függvény, mely csak N -től ($N \geq 2$) függ, és található hozzá $\varepsilon > 0$ úgy, hogy minden folytonos, valós $f(x): [0,1]^N \rightarrow \mathbb{R}$ függvény felírható az alábbi formában:

$$f(x) = \sum_{q=1}^{2N+1} \phi_q \left(\sum_{p=1}^N \lambda_p \psi(x_p + q\varepsilon) \right), \quad (1.14)$$

ahol a ϕ_q függvények valós, folytonos függvények, míg a λ_p értékek alkalmas konstansok.

A Sprecher által kimondott tétel egy olyan – szintén két rejtett réteget tartalmazó – neurális hálózat irányába mutat, amelyben az első rejtett réteg neuronjainak aktivációs függvénye ψ (tehát ez már semmilyen formában nem függ a reprezentálandó függvénytől), míg a második réteg neuronjainál a ϕ_q aktivációs függvényeket használjuk. A fenti eredményekre alapozva ki is mondták a 3. fejezetben részletesen tárgyalásra kerülő többrétegű perceptron hálókra vonatkozó tételt.

1.3 tétel [Hec87]

Minden $f(x): [0,1]^N \rightarrow \mathbb{R}$ négyzetesen integrálható ($f \in L_2$) függvény átlagos négyzetes értelemben tetszőleges pontossággal közelíthető legfeljebb két feldolgozó réteget, és a neuronokban szigmoid jellegű nemlinearitást alkalmazó neurális hálózat segítségével.

4.3. Egy nemlineáris réteget használó approximációs hálózatok

A fenti tétel, melyet Robert Hecht-Nielsen fogalmazott és bizonyított be, két feldolgozó réteget alkalmazó hálóra vonatkozik. A következő tétel azonban ezen gyengíteni tudott, hiszen azt állítja, hogy egy aktív réteg is elegendő ahhoz, hogy egy folytonos függvényt tetszőleges pontossággal közelíteni tudjunk.

1.4 tétel [Fun89]

Legyen $g(x)$ nemkonstans, korlátos, monoton növekvő folytonos függvény. Legyen továbbá $\mathbf{A}$ egy kompakt (korlátos és zárt) részhalmaza $\mathbb{R}^n$ -nek, és legyen $f(x_1, \dots, x_M)$ egy valós értékű folytonos függvény $\mathbf{A}$ -n. Ekkor tetszőleges $\varepsilon > 0$ esetére létezik egy olyan M egész szám, és léteznek c_i $i=1, \dots, M$, valamint w_{ij} , $i=1, \dots, M$, $j=1, \dots, N$ konstansok, hogy:

$$\hat{f}(x_1, \dots, x_M) = \sum_{i=1}^M c_i g\left(\sum_{j=1}^N w_{ij} x_j\right); \quad x_0 = 1 \quad (1.15)$$

melyre:

$$\max_{\mathbf{x} \in \mathbf{A}} |f(x_1, \dots, x_M) - \hat{f}(x_1, \dots, x_M)| < \varepsilon. \quad (1.16)$$

Vegyük észre, hogy a Tatsushi Funahashi által megfogalmazott 1.4 tétel nemcsak azt állítja, hogy az (1.16) összefüggés értelmében vett approximációs képességhez egy rejtett réteg elegendő, hanem azt is, hogy ezt a képességet lineáris kimeneti réteggel biztosíthatjuk.

Különösen fontosnak tűnik az utóbbi időben az egy nemlineáris réteggel közelítő hálózatok kutatásának egy másik iránya. A nemlinearitás ezekben a közelítésekben nem az eddigiekben tárgyalt folytonos, monoton, telítődő jellegű függvények közül kerül ki, hanem tipikusan nem monoton, esetleg nem is folytonos függvényeket alkalmazunk. Érdekes, hogy szükséges és elégséges feltételt tudtak kimondani az alkalmazott nemlinearitásra, az alábbi tétel szerint.

1.5 tétel [Les93]

Tetszőleges folytonos $f: \mathbb{R} \rightarrow \mathbb{R}$ függvényt *akkor és csak akkor* tudunk tetszőleges pontossággal közelíteni lokálisan korlátos aktivációs függvényt alkalmazó neuronokból felépített, egy nemlineáris réteget tartalmazó neurális hálóval, ha az alkalmazott aktivációs függvény **nem** polinom.

Az 1.5 tétel az aktivációs függvény folytonosságát sem tételezi fel, csupán annyi megkötést tesz, hogy a szakadási helyek nullmértékű halmazzal kell alkossanak.

A Funahashi által a perceptron típusú hálózatokra megfogalmazott tétel igen fontos. Hasonló eredményt mások is megfogalmaztak (többek között George Cybenko, [Cyb89], Sean Carroll és Bradley Dickinson [Car89], Takahashi [Tak93]). E tételek részben abban térnek el egymástól, hogy az approximációt milyen értelemben fogalmazzák meg, részben a bizonyítás matematikai részletei mások.

Az eddigiekben olyan approximációs tételeket ismertettünk, melyek valós számok közötti leképezést megvalósító hálókra vonatkoztak, tehát ahol a háló $f: \mathbb{R} \rightarrow \mathbb{R}$ vagy $f: \mathbb{R}^n \rightarrow \mathbb{R}$ típusú leképezést valósít meg. A feladatok egy részében (ezek az osztályozási feladatok) azonban a kimenet véges számú lehetséges diszkrét érték közül vehet csak fel értéket.

Cybenko eredménye [Cyb89] kiterjeszthető osztályozós típusú leképezésekre is. Ekkor a háló által megvalósított leképezés: $f: \mathbf{A} \subset \mathbb{R}^n \rightarrow \{1, 2, \dots, k\}$, és

$$f(\mathbf{x}) = j \text{ akkor és csak akkor, ha } \mathbf{x} \in \mathbf{A}_j, \quad (1.17)$$

ahol $\mathbf{A}_j$, $j=1, 2, \dots, k$ az $\mathbf{A}$ tartomány diszjunkt részhalmazait jelöli, vagyis $\mathbf{A} = \bigcup_{j=1}^k \mathbf{A}_j$ és $\mathbf{A}_i \cap \mathbf{A}_j = \emptyset$,

ha $i \neq j$. Ez az eredmény azt mondja ki, hogy egy egy rejtett réteggel és szigmoid típusú nemlinearitással dolgozó háló nemcsak univerzális approximátor, hanem univerzális osztályozó is. Ezt az eredményt számos kétosztályos osztályozási feladat megoldásával kísérletileg igazolta William Huang és Richard Lippman [Hua88].

Végezetül a neurális hálózatok egyik, egyre jobban terjedő változatára mutatunk approximációs tételt. A 3. fejezetben részletesen tárgyaljuk az RBF (*radial basis function*) hálózatokat. Ezek tipikusan egy nemlineáris rejtett réteget tartalmaznak, a neuronok aktivációs függvényének jellegzetessége pedig az N -elemű bemeneti

vektorok terében mutatott körszimmetria, melyre neve is utal. (*radial* = sugaras, ui. a neuronra jellemző középpont paramétértől azonos távolságra – egy r sugarú hipergömbön – elhelyezkedő bemeneti vektorok között nem tesz különbséget a neuron.)

1.6 tétel [Par91]

Legyen a $\psi: \mathbb{R}^n \rightarrow \mathbb{R}$ aktivációs függvény a $(-\infty, \infty)$ -ben integrálható, folytonos és nem azonosan nulla függvény, mely körszimmetriát mutat az N -elemű bemeneti vektorok halmazán, az euklideszi távolságra nézve. Ekkor bármely $f \in C^n$ függvény tetszőleges pontossággal közelíthető

$$f(x) = \sum_{i=1}^M w_i \psi\left(\frac{\|x - c_i\|}{\sigma_i}\right) \quad (1.18)$$

alakban, ahol $c_i \in \mathbb{R}^n$, σ_i és w_i $i=1, 2, \dots, M$ a hálózatra jellemző valós paraméterek.

A fentiekben tárgyalt, alapvetően approximációelméleti eredmények olyan modellek irányába mutatnak, melyek struktúrája az 1.2.3 alfejezetben felvázolt előrecsatolt neurális háló. Az eredmények egyik része az 1.3 ábrán látható rendkívül elterjedten használt telítődő jellegű aktivációs függvények használatát sugallja, míg másik része az 1.4 ábrán látható bázisfüggvények használatának irányába mutat.

Az utóbbi években jelentős matematikai kutatások kezdődtek, melyek a fenti eredmények gyakorlati alkalmazhatóságára irányulnak. A kutatások egyrészt az adott pontosságú közelítéshez szükséges neuronszám meghatározását, vagy legalább egy felső korlát felállítását célozzák. Ezek az eredmények azonban már általában nemcsak egy általános neuronháló architektúrát tételeznek fel, hanem kapcsolatot keresnek a háló approximációs hibája, a háló mérete, a tanítópontok száma és a megoldandó probléma bonyolultsága között is; tehát már azzal a kérdéssel is foglalkoznak, hogy adott képességű háló hogyan konstruálható, és azzal is, hogy mennyiben hat a háló képességeire az, hogy véges számú tanítópont felhasználásával konstruáljuk a hálókat. Ezek a kérdések az eddig tárgyaltakhoz hasonló fontosságúak. A legfontosabb eredményeket a következő, a tanúlással foglalkozó fejezetben foglaljuk össze, illetve néhány eredményt az egyes háló architektúrákat tárgyaló 3. fejezetben mutatunk be.

Feladatok

1.1 Rajzoljon fel egy FIR/IIR neuront (ld. 1.6 ábra) részletesen felrajzolva a dinamikus hálózatokat! A neuron nemlineáris függvénye legyen egyváltozós ($M=1$), a bemenetek száma legyen $N=2$. A neuron mindegyik bemenetén egy-egy két tároló elemet tartalmazó FIR szűrővel, kimenetén pedig egy mind előrevezető, mind visszacsatoló ágában három-három tároló elemet tartalmazó általános IIR szűrővel rendelkezzen! Hány időegységgel késleltetve jelennek meg a bemeneti változások a kimeneten?

1.2 Rajzoljon fel egy FIR/IIR neuront (ld. 1.6 ábra), részletesen felrajzolva a dinamikus hálózatokat. A neuron nemlineáris függvénye legyen egyváltozós ($M=1$), a bemenetek száma legyen $N=2$. Mutassa meg, hogy a bemeneteken megvalósított FIR szűrő hogyan transzformálható át az összegző kimenetére!

1.3 Az alábbi aktivációs függvényeket használó neuronok közül, mely(ek) segítségével lehet univerzális approximációs hálózatot építeni (ld. 1.4 – 1.6 tételek)?

$$g_1(x) = \frac{1}{1+x^2}; \quad g_2(x) = x^2; \quad g_3(x) = \sin(x), \quad (-\pi/2 \leq x \leq \pi/2); \quad g_4(x) = \frac{1}{|x|}; \quad g_5(x) = x^T x; \quad g_6(x) = \exp\left(-\frac{\|x - c_i\|^2}{\sigma_i}\right).$$

2. fejezet - Tanulás adatokból

A neurális hálózatok egyik legfőbb jellemzője az adaptációs, tanulási képesség. Az adaptáció és a tanulás olyan fogalmak, amelyek alapján az élő szervezetek képességeit jellemzik; azt a képességüket, hogy viselkedésüket a környezetből nyert ismeretek, tapasztalatok felhasználásával módosítani, javítani tudják.

A tanulás egzakt definiálása meglehetősen nehéz, hiszen tanulásnak nevezhető az ismeretek, tapasztalatok egyszerű memorizálása, de tanulásnak tekinthető egy, a tapasztalatok alapján kialakított teljes tudományos elmélet létrehozása is.

Az élő szervezetek tanulásán túl – annak analógiájára – beszélhetünk gépi tanulásról is, illetve olyan gépekről is, melyek tanulásra képesek. A gépi tanulás során egy gép – a tanuló rendszer – a környezetéből nyert ismeretek alapján javítja a teljesítőképességét. A tanuló gépek konstrukciója valójában olyan algoritmusok kifejtését jelenti, melyek segítségével ez a teljesítőképesség-javítás automatikusan történik meg [Mit97], [Rus05].

Az élő szervezetek tanulásához hasonlóan a gépi tanulás is nagyon sokféle lehet. Neurális hálónál a minták alapján történő tanulás a jellemző, amikor az általában nagymennyiségű adatból akarunk megfelelő ismereteket kinyerni, és ezzel egy rendszer viselkedését módosítani.

A viselkedés módosítása általában arra irányul, hogy a rendszer adott bemenetekre előzetesen megadott ún. kívánt válaszokat adjon, de irányulhat arra is, hogy csupán a bemenetek ismeretében – tehát a kívánt válaszok ismerete nélkül – képes legyen a rendszer a bemenetekben valamilyen szabályosságot, hasonlóságot vagy különbözőséget feltárni. Van olyan eset is, amikor a kívánt választ pontosan ugyan nem tudjuk, de arról kapunk információt, hogy a rendszer jó vagy rossz választ adott-e.

A tanulási képesség lehetővé teheti azt is, hogy egy eddig megfelelő viselkedésű rendszer a változó körülményekhez való alkalmazkodás céljából módosítsa a viselkedését, adaptálódjon a változó körülményekhez. Az adaptív, tanuló rendszerek alapvető jellemzője tehát, hogy nem rögzített képességekkel rendelkeznek, amelyek egy adott feladat ellátására teszik őket alkalmassá, hanem képességeiket fejleszteni tudják, továbbá alkalmazkodni tudnak a változó körülményekhez, környezethez.

Neurális hálózatokban döntően a tanulás alábbi főbb formáival találkozhatunk:

- tanulás tanítóval, ellenőrzött, vagy felügyelt tanulás,
- tanulás tanító nélkül, nemellenőrzött vagy felügyelet nélküli tanulás,
- analitikus tanulás.

Meg kell említenünk, hogy az elmúlt néhány évben intenzív kutatás tárgya lett az ún. félig ellenőrzött tanulás (*semi-supervised learning*) [Cha06]. Ennek sajátossága, hogy a felhasználható adatainknak csak egy része biztosítja az ellenőrzött tanulás lehetőségét, míg további adatoknál az ellenőrzött tanításhoz szükséges minden ismeret nem áll rendelkezésre. A félig ellenőrzött tanulás lehetővé teszi, hogy ez utóbbi adatok se vesszenek kárba, a tanuló rendszer konstrukciójához valamilyen módon az általuk szolgáltatott ismeretek is felhasználásra kerüljenek.

Ellenőrzött tanulásnál (*supervised learning*) a hálózat összetartozó be- és kimeneti értékei, tanító mintapont párok állnak rendelkezésre. A tanítás ezeken az ismert összerendelt mintapárokon alapul. A háló feladata, hogy megtanulja a mintapont párok által reprezentált bemenet-kimenet leképezést. Mivel ismertek a kívánt válaszok (a bemenetekhez tartozó kimenetek), a hálózat tényleges válasza minden esetben közvetlenül összehasonlítható a kívánt válasszal. Az összehasonlítás eredménye – a tényleges és a kívánt válasz különbsége – felhasználható a hálózat olyan módosítására, hogy a tényleges válaszok a kívánt válaszokkal minél inkább megegyezzenek, és a hálózat tényleges viselkedése és a kívánt viselkedés közötti eltérés csökkenjen. Az ellenőrzött tanulás általában iteratív eljárással, a környezetből származó ismeret fokozatos felhasználásával, a tanító mintapontok egyenkénti, esetenként többszöri, ismételt felhasználásával történik. Az ellenőrzött tanulásnak ezt a változatát tanítóval történő tanulásnak (*learning with a teacher*) is szokás nevezni.

A tanítóval történő tanításnál feltételezzük, hogy adott bemenetekhez tartozó kívánt válaszok korrekt értékei rendelkezésünkre állnak. Számos esetben nem rendelkezünk ezen információval, mindössze annyit tudunk, hogy

adott bemenetekre a hálózat tényleges válasza helyes vagy hibás; tehát a környezetből csak egy bitnyi visszacsatolt információ áll rendelkezésünkre. Ezen információ a tanuló eljárás során csak arra elegendő, hogy eldöntsük: szükség van-e a hálózat módosítására vagy sem, de a módosítás mértékének meghatározására már nem elegendő. Az ellenőrzött tanuláshoz ezt a fajtáját megerősítéses tanuláshoz (*reinforcement learning*) nevezzük. A megerősítéses tanuláshoz a visszajelzés tehát csak, mint egy általános kritika jelentkezik, ezért az ilyen tanuló eljárásokat szokás kritikussal történő tanításhoz is nevezni (*learning with a critic*). A megerősítéses tanulás jól alkalmazható azon esetekben, ha a hálózat egy adott bemenetre nem csupán egy jó választ adhat [Her91].

Előfordulhat olyan eset is, amikor a környezet sztochasztikus jellege miatt adott bemenetre kapott válasznál csak a megerősítés valószínűségét tudjuk megadni. Ez a valószínűség azonban minden be-kimeneti párra rögzített. Ilyen jellegű feladattal találkozunk többek között gazdasági rendszerek modellezésénél és egyszerű játékoknál. A legáltalánosabb esetben mind a környezet – tehát a tanuló rendszer választ értékelő visszajelzés –, mind a rendszer bemenetei függenek a rendszer korábbi bemeneteitől.

Nemellenőrzött tanuláshoz (*unsupervised learning*) nem állnak rendelkezésünkre adott bemenetekhez tartozó kívánt válaszok. A hálózatnak a bemenetek és a kimenetek alapján kell valamilyen viselkedést kialakítania; a környezetből azonban nincs semmiféle visszajelzés, ami a hálózat viselkedésének helyességére utalna. A nemellenőrzött tanuló hálózatoknak azt kell felderíteniük, hogy van-e a hálózat bemenetére kerülő adatokban, jelekben valami hasonlóság, van-e az adatok között korreláció, kialakíthatók-e a bemeneti adatok között kategóriák, csoportok. Itt tehát a háló által megvalósítandó leképezés pontosan nem definiálható. Ennek ellenére a hálózat képes önmaga módosítására, hogy a kategorizálás egyre sikeresebb legyen. E képességük miatt ezen hálózatokat szokás önszervező hálózatoknak (*selforganizing networks*) is nevezni.

Általában a nemellenőrzött tanulásra is az iteratív jelleg a jellemző.

Analitikus tanuláshoz a megfelelő viselkedést biztosító hálózat kialakítása szintén a mintapontok alapján, de nem az azokban megtestesülő ismeretek fokozatos felhasználásával, hanem közvetlen számítások útján, matematikai összefüggések kiértékelésével, analitikus módszerekkel történik. Analitikus tanuláshoz az ismétlés, mint a tanuló eljárások egyik fontos eleme hiányzik. Valójában ebben az esetben olyan értelemben nincs szó tanulásról, hogy kis lépésekben fokozatosan növeljük egy rendszer "tudását". Abban az értelemben azonban most is adatokból történő tanulásról beszélhetünk, hogy a környezetből származó információ általában itt is tanító minták formájában áll rendelkezésünkre.

Amikor analitikus tanulásról beszélünk, akkor nem az információ jellege, hanem sokkal inkább az információ felhasználási módja alapján kategorizáljuk a tanuló eljárást. Így analitikus tanulás lehetséges mind ellenőrzött, mind nemellenőrzött tanuláshoz, de vannak olyan analitikus tanuló hálózatok, melyek sem az ellenőrzött, sem a nemellenőrzött tanuló hálók közé nem sorolhatók be.

Ha a hálóarchitektúrák oldaláról vizsgáljuk a kérdést, akkor is azt látjuk, hogy egyes neuronháló típusoknál a hálók konstrukciója alapvetően analitikus úton történik, más típusoknál mind analitikus, mind iteratív tanulás lehetséges, és vannak olyan hálóarchitektúrák, melyek kialakítása analitikus módszerekkel nem lehetséges. A súlymeghatározás analitikus módszereit a konkrét hálózatoknál fogjuk bemutatni.

A fejezet célja, hogy összefoglalja a mintákból történő tanúlással kapcsolatos legfontosabb alapfogalmakat, a neurális hálózatoknál alkalmazott főbb tanuló eljárásoknál felmerülő fontosabb elvi kérdéseket és a tanuló eljárások általános jellemzőit. Nem foglalkozik azonban az egyes konkrét hálózatok tanításával, ezeket – alkalmazva a tanulásra vonatkozó általános ismereteket – az egyes hálózatokat bemutató későbbi fejezetek tárgyalják.

A 2.1 alfejezet az ellenőrzött tanítási eljárásokkal foglalkozik, ezen belül is a tanítóval történő tanúlással – mint a neuronhálónál alkalmazott egyik legfontosabb tanuló eljárással. A megerősítéses tanulás – bár számos alkalmazásban fontos szerepet tölt be – neuronhálónál közvetlenül ritkán kerül alkalmazásra, ezért a megerősítéses tanúlással részletesen nem foglalkozunk, csak a megfelelő irodalomra utalunk pl. [Nar89b], [Sut98].

A 2.2 alfejezet a nemellenőrzött tanulás néhány fontos kérdéséről szól röviden. A 2.3 alfejezet a véges számú tanítómintából történő tanulás általános kérdéseivel foglalkozik a statisztikus tanuláselmélet alapjainak bemutatásán keresztül. A 2.4 alfejezet azt mutatja be, hogy az adatokból való tanulás és a statisztikai becslések (paraméterbecslés) rokon területek. Az egyes tanuló eljárások statisztikai becslési eljárásaként is értelmezhetők. A

két terület közötti kapcsolat bemutatása lehetővé teszi, hogy a statisztikai becslések terén megszületett egyes általános eredmények a tanuló eljárások elemzésénél is felhasználhatók legyenek.

A fejezet utolsó részében a neuronhálók tanításánál fontos szerepet betöltő szélsőérték-kereső eljárásokkal foglalkozunk részletesebben. A felügyelt tanítás mindig visszavezethető szélsőérték-keresési feladatra, de megmutatható – és erre a megfelelő későbbi fejezetek ki is térnek –, hogy a nemellenőrzött tanítási eljárások is megfogalmazhatók szélsőérték-kereső feladatként.

1. Ellenőrzött tanulás (tanítóval történő tanítás)

A tanítóval történő tanításnál a tanítandó rendszer változtatása összetartozó be- és kimeneti tanító mintapárok felhasználásával történik. Ilyenkor a tanuló eljárás célja az, hogy a tanítandó rendszer felépítését (struktúráját és/vagy paramétereit) olyan módon válasszuk meg vagy alakítsuk ki, hogy a megtanított rendszer – amely valamely vizsgált rendszer modelljeként jelenik meg – működése minél inkább feleljen meg a vizsgált rendszer működésének. Úgy is fogalmazhatunk, hogy modell-illesztési feladatot kell megoldanunk.

A modell-illesztés feladata minden olyan esetben megjelenik, amikor "identifikálni", egyszerűbben kifejezve "meghatározni", "azonosítani", "mérni" szeretnénk valamit, vagy amikor "követni" szeretnénk folyamatokat. De lényegében ugyanerről van szó akkor is, amikor bonyolult, nehezen kezelhető jelenségekhez egyszerűbb – könnyebben kezelhető –, ugyanakkor főbb vonásaiban mégis hasonlóan viselkedő "modelleket" hozunk létre, illetőleg amikor a klasszikus regresszió-számítás módszerét alkalmazzuk.

A modell-illesztési feladat általánosan a következő formában fogalmazható meg: Adott egy rendszer, mely a bemenetei és a kimenetei között valamilyen $g(\cdot)$ leképezést valósít meg. A leképezés nem ismert, viszont rendelkezésünkre áll egy $\{\mathbf{x}_i\}_{i=1}^I = \{\mathbf{x}_i, d_i\}_{i=1}^I$ mintapont-készlet, melynél az összetartozó bemeneti és kimeneti értékek közötti kapcsolat az ismeretlen leképezés adja meg. A mintakészlet elemei, a mintapontok általában többdimenziós vektorok, melyek a mintatérben helyezkednek el.

A tanulási feladat az, hogy egy $f(\mathbf{x}, \mathbf{w})$ paraméteres függvényhalmazból, – ahol $\mathbf{w} \in W$ a függvényhalmaz paramétere és W a paramétertér – válasszunk egy függvényt olyan módon, hogy adott $\mathbf{x}_i$ bemenetre a kiválasztott függvény $y_i = f(\mathbf{x}_i, \mathbf{w})$ válasza valamilyen értelemben a lehető legjobban közelítse az ismeretlen leképezés azonos bemenetre adott d_i válaszát. A d_i válaszokat kívánt válaszoknak (*desired response*) nevezzük. A függvényhalmaz szabad paramétereit tartalmazó $\mathbf{w}$ vektort a W paramétertér megfelelő pontjaként a rendelkezésre álló mintapontok alapján kell megválasztani vagy meghatározni.

Az $\{\mathbf{x}_i, d_i\}_{i=1}^I$ mintapont-készletet **tanító készletnek** vagy **tanító mintakészletnek** (*training set*) nevezzük, azt az eljárást pedig, melynek során a tanító mintakészlet felhasználásával az $f(\mathbf{x}, \mathbf{w})$ leképezést kialakítjuk, **tanulásnak** (*learning*). A tanuló eljárás célja tehát egy valamilyen értelemben optimális $\mathbf{w}^*$ paraméter meghatározása a tanító mintakészlet alapján.

Amennyiben az $f(\mathbf{x}, \mathbf{w})$ leképezést egy neuronháló valósítja meg, a függvénykészlet megválasztása a neuronháló típusának és architektúrájának megválasztásával történik, míg a tanítás valójában a $\mathbf{w}$ paramétervektor meghatározását, becslését szolgálja. Ebben az értelemben a tanulás egy paraméterbecslési eljárás. Nyilvánvaló tehát, hogy a tanuló eljárások és a statisztikai becslések között szoros kapcsolat található. Erről a kapcsolatról részletesebben a 2.4 alfejezetben szólunk.

Osztályozás és regresszió

Az általános modell-illesztési feladat osztályon belül több feladatcsoportot különböztethetünk meg. Ezek közül a neuronhálókhoz kapcsolódóan az osztályozási és a regressziós feladatcsoportokkal foglalkozunk.

Az osztályozási feladatoknál a rendszer által képviselt leképezést az

$$(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \dots, (\mathbf{x}_I, d_I) \in \mathbb{R}^N \times \{\pm 1\} \quad (2.1)$$

tanító mintapontok alapján kell közelítenünk, míg a regressziós feladatnál a rendelkezésre álló összerendelt tanító mintapárok:

$$(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \dots, (\mathbf{x}_I, d_I) \in \mathbb{R}^N \times \mathbb{R} \quad (2.2)$$

egy, általában zajos leképezés $d_i = g(\mathbf{x}_i) + \varepsilon_i$ eredményeképpen nyerhetők. Itt $g(\cdot)$ egy determinisztikus függvény, a regressziós függvény, ε_i pedig a kimeneti megfigyelési zaj. A (2.1) összefüggés kétosztályos osztályozási feladatot tételez fel, de nyilvánvalóan tetszőleges (véges) számú osztályra is kiterjeszthető. A különbség mindössze annyi, hogy a válasz nemcsak két érték közül kerülhet ki, hanem a véges számú lehetséges értékek közül bármelyik lehet. A (2.2) által definiált mintapontokról feltételeztük, hogy egy több-bemenetű – egykimenetű leképezésből származnak. Az egydimenziós kimenet feltételezése nem korlátozza a feladat általános megfogalmazását. A többdimenziós kimenetre való kiterjesztés természetes módon lehetséges.

A megfigyelési zajról általában feltételezzük, hogy várható értéke bármely $\mathbf{x}$ mellett nulla, továbbá, hogy a megfigyelési zaj és a regressziós függvény korrelálatlanok egymással, vagyis

$$E_{\varepsilon|\mathbf{x}}\{\varepsilon|\mathbf{x}\} = 0, \quad (2.3)$$

illetve

$$E\{\varepsilon g(\mathbf{x})\} = 0. \quad (2.4)$$

Nulla várható értékű, korrelálatlan zaj mellett a regressziós feladat valójában egy feltételes várhatóérték-képzési feladat, vagyis $g(\mathbf{x})$ a megfigyelés feltételes várhatóértéke

$$g(\mathbf{x}) = E_{d|\mathbf{x}}\{d|\mathbf{x}\}. \quad (2.5)$$

Minősítés, hibamértékek

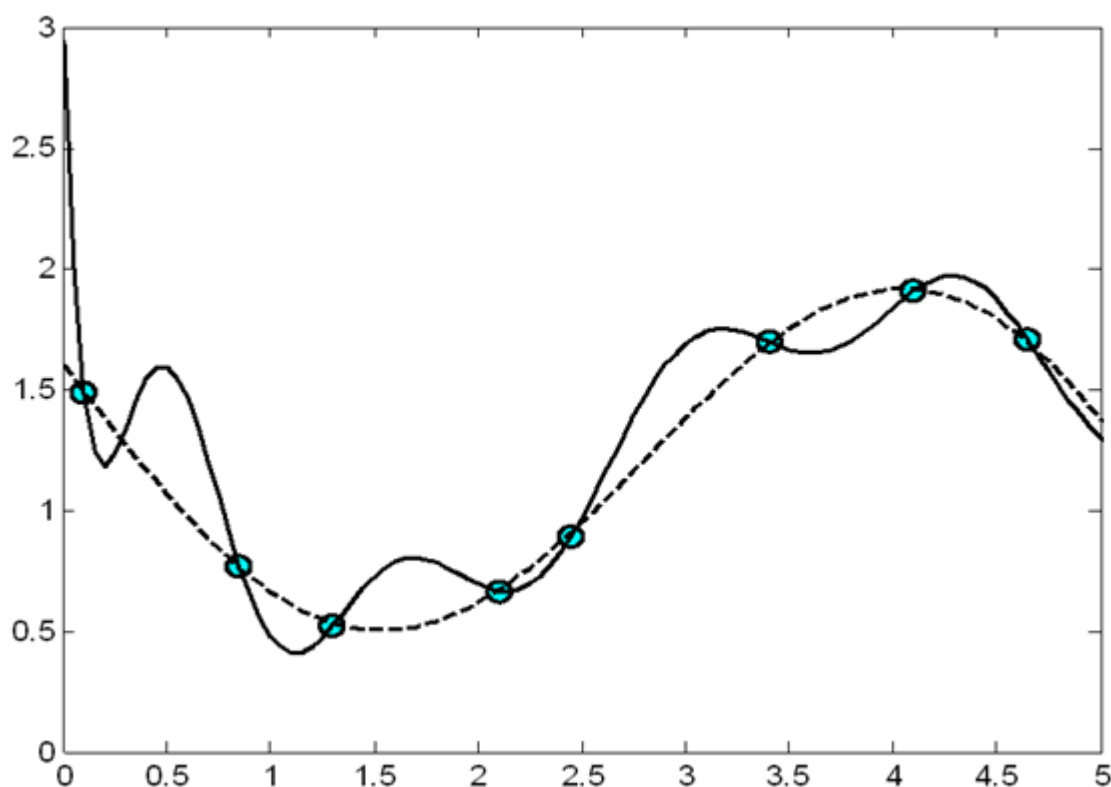
A legjobb approximációt biztosító függvény meghatározásához az approximáció minőségét jellemző mértékre van szükség. Ez a mérték egy **hibafüggvény** (*error function*) felhasználásával határozható meg, amit **költségfüggvénynek** (*cost function*) vagy **vesztésgfüggvénynek** (*loss function*) is szokás nevezni. A vesztésgfüggvény – amit a továbbiakban $L(d, f(\mathbf{x}, \mathbf{w}))$ -vel fogunk jelölni – mind a háló válaszának, $f(\mathbf{x}, \mathbf{w})$ -nek, mind a rendelkezésre álló $d(\mathbf{x})$ kívánt válasznak a függvénye: leggyakrabban az approximáció $\varepsilon(\mathbf{x}, \mathbf{w})$ hibájának a függvénye, ahol a hibát a kívánt válasz és a tényleges válasz különbségeként definiálhatjuk:

$$\varepsilon(\mathbf{x}, \mathbf{w}) = d(\mathbf{x}) - f(\mathbf{x}, \mathbf{w}). \quad (2.6)$$

A tanuló rendszer konstrukciója – akár osztályozási, akár regressziós probléma megoldására irányul – véges számú tanító mintapont alapján történik. A tanuló eljárás célja azonban általában nem az, hogy a véges számú tanítópontra kapjunk megfelelő válaszokat, hanem egy olyan leképezés megtanulása, amely a tanítópontra által reprezentált bemenet-kimenet kapcsolatot is megadja. A tanuló rendszer tehát **általánosítóképességgel** (*generalization capability*) kell rendelkezzen: olyan esetekben is „jó” választ kell adjon, mely esetek nem szerepelnek a tanítópontra között. A tanulás nehézségét épp ez adja: véges számú tanítópontra alapján kell egy problémáról „általános tudást” szerezni.

Az adatokból történő tanulás egy rosszul definiált feladat. Példaképp nézzünk egy függvényapproximációs problémát. Mint a 2.1. ábrán látható, a pontokkal jelölt tanítópontokra több – akár végtelen sok – különböző függvény illeszthető, melyek mindegyikére igaz lehet, hogy átmegy a megadott pontokon, vagy azokat tetszőleges pontossággal közelíti, miközben a tanítópontoktól eltérő pontokban az egyes függvények viselkedése nagymértékben különbözhet.

2.1. ábra - A tanítópontra alapján történő függvényapproximáció egy rosszul definiált feladat: a tanítópontra nem határozzák meg egyértelműen a bemenet és a kimenet közötti függvénykapcsolatot



A tanuló rendszer minősítésére ezért olyan mértékre van szükségünk, amely nem csupán azt mondja meg, hogy a tanuló rendszer a tanító mintapontokat milyen pontosan tanulta meg, hanem a rendszer általánosítóképességéről is mond valamit.

A veszteségfüggvény adott bemenet mellett minősíti a tanuló rendszert. A minősítésre azonban alkalmasabb mérték, ha a veszteség összes pontra vonatkozó átlagos értékét, a veszteség várható értékét, az ún. kockázatot (*risk*) határozzuk meg, ahol a kockázat a veszteségfüggvényhez rendelt számérték:

$$R(\mathbf{w}) = E_{\mathbf{x}, d} \{ L(d, f(\mathbf{x}, \mathbf{w})) \}. \quad (2.7)$$

A kockázat meghatározásánál a várható értéket a minták $p(\mathbf{x}, d)$ együttes sűrűségfüggvénye szerint kell vennünk. Ugyanakkor – ahogy ezt az $R(\mathbf{w})$ jelölés mutatja is – a kockázat a paramétervektor függvénye, amit szokás kritériumfüggvénynek (*criterion function*) is nevezni és $C(\mathbf{w})$ -vel jelölni. A továbbiakban a két elnevezést és a kétfajta jelölést egymás alternatívájaként kezeljük: adott szövegkörnyezetben azt a jelölést/elnevezést használjuk, melyet az aktuális témakörben általánosabban használnak.

Osztályozási feladatoknál veszteségfüggvényként általában az osztályozási hibát használhatjuk, vagyis

$$L(d, f(\mathbf{x}, \mathbf{w})) = \begin{cases} 0 & \text{ha } d = f(\mathbf{x}, \mathbf{w}) \\ 1 & \text{ha } d \neq f(\mathbf{x}, \mathbf{w}) \end{cases} \quad (2.8)$$

Ekkor a (2.7) szerinti kockázat a téves osztályozás valószínűségét adja meg. A tanulás tehát olyan $f(\mathbf{x}, \mathbf{w}^*)$ osztályozó konstrukcióját célozza, amely a hibás osztályozás valószínűségét minimalizálja. Itt $\mathbf{w}^*$ az adott függvényosztályt és a rendelkezésre álló tanító mintapont-készletet tekintve az optimális paramétervektor.

A regressziós feladatoknál az egyik leggyakoribb veszteségfüggvény a hiba négyzetes függvénye:

$$L(d, f(\mathbf{x}, \mathbf{w})) = (d(\mathbf{x}) - f(\mathbf{x}, \mathbf{w}))^2, \quad (2.9)$$

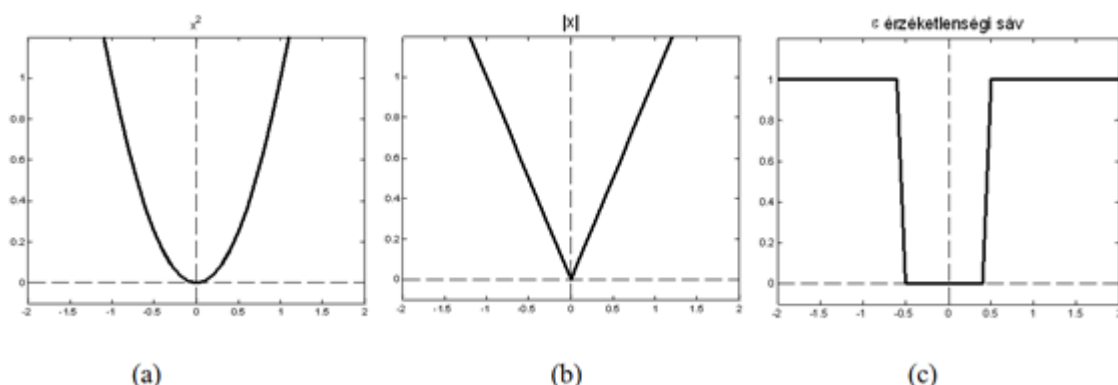
bár más veszteségfüggvények alkalmazása is szokásos. Ilyen függvény lehet pl. az abszolútérték függvény vagy ennek kissé módosított változata az ún. ε érzéketlenségi sávval rendelkező abszolútérték függvény. Egyes speciális alkalmazásokban szerepet kaphatnak a különböző abszolútérték-hatvány függvények, és ezek lineáris kombinációi, az "igen-nem" függvény, amely az optimumnál, ill. annak egy rögzített, általában szűk környezetében nulla, egyébként egységnyi (vagy más pozitív) értéket vesz fel. Speciális alkalmazásokban használnak logaritmikus, trigonometrikus, hiperbolikus függvényekből alkalmasan előállított hibafüggvényeket is. Egyes osztályozási feladatoknál előnyösen alkalmazható az

$$L(d, f(\mathbf{x}, \mathbf{w})) = - (d \ln f(\mathbf{x}, \mathbf{w}) + (1-d) \ln(1 - f(\mathbf{x}, \mathbf{w}))) \quad (2.10)$$

keresztentrópia függvény is. (A téma iránt mélyebben érdeklődők az irodalomban pl. [Cic93] találhatnak ilyeneket, pl. a logisztikus, a Huber, a Talvar és a Hampel függvényt.)

A hibafüggvény konkrét alakja – bizonyos közös lényeges tulajdonságokon (pl. az optimumhoz tartozó szélsőérték) túlmenően – a feladat jellegétől, az a priori ismeretektől (pl. különböző zaj eloszlások esetén más-más függvény biztosítja a "jobb" viselkedést), és bizonyos mértékig szubjektív tényezőktől is függ. A hibafüggvény megválasztása befolyásolja a tanuló eljárás jellemzőit: az eljárás konvergenciáját, illetve a konvergencia sebességét, sőt magát a megoldást is. Néhány fontosabb hibafüggvényt a 2.2 ábrán mutatunk be.

2.2. ábra - Fontosabb veszteségfüggvények: (a) négyzetes függvény, (b) abszolútérték függvény, (c) ε érzéketlenségi sávval rendelkező igen-nem függvény



A kockázat négyzetes veszteségfüggvény mellett a tanuló rendszer átlagos négyzetes hibája. Az átlagos négyzetes hiba vizsgálata több hasznos következtetésre ad lehetőséget. Ennek érdekében bontsuk fel a kockázat kifejezését több tagra:

$$\begin{aligned} R(\mathbf{w}) &= E\left\{\left(d - f(\mathbf{x}, \mathbf{w})\right)^2\right\} = E\left\{\left(d - g(\mathbf{x}) + g(\mathbf{x}) - f(\mathbf{x}, \mathbf{w})\right)^2\right\} \\ &= E\left\{\left(d - g(\mathbf{x})\right)^2\right\} + E\left\{\left(f(\mathbf{x}, \mathbf{w}) - g(\mathbf{x})\right)^2\right\} + 2E\left\{\left(d - g(\mathbf{x})\right)\left(g(\mathbf{x}) - f(\mathbf{x}, \mathbf{w})\right)\right\} \end{aligned} \quad (2.11)$$

Feltételezve, hogy – amint azt az előbbiekben kikötöttük – a megfigyelési zaj nulla várható értékű, továbbá, hogy a zaj és a $g(\mathbf{x})$ leképezés korrelálatlan, a (2.11) összefüggés utolsó tagja zérus, hiszen $d - g(\mathbf{x}) = \eta$. Így a kockázat az alábbi két tagra bontható:

$$R(\mathbf{w}) = E\left\{\left(d - g(\mathbf{x})\right)^2\right\} + E\left\{\left(f(\mathbf{x}, \mathbf{w}) - g(\mathbf{x})\right)^2\right\} = E\{\eta^2\} + E\left\{\left(f(\mathbf{x}, \mathbf{w}) - g(\mathbf{x})\right)^2\right\} \quad (2.12)$$

Az első tag nem függ a tanuló rendszer leképezésétől, csupán a megfigyelt kívánt válasz és a $g(\mathbf{x})$ ideális leképezés eltérésétől, ami épp a megfigyelési zaj. Az első tag tehát a megfigyelési zaj variáciájával egyenlő. Ez a hibakomponens nyilvánvalóan független a tanuló eljárástól, és nem is csökkenthető $\mathbf{w}$ megfelelő megválasztásával: az optimális $\mathbf{w}$ meghatározásánál tehát figyelmen kívül hagyható. Ebből következik, hogy a kockázat minimumát biztosító $\mathbf{w}^*$ paramétervektor egyben a tanuló rendszer és a regressziós függvény válaszainak átlagos négyzetes eltérését is minimalizálja.

A feltételezett zajmodell mellett tehát a zajos megfigyelések ellenére a kockázatminimalizálás a regressziós függvény átlagos négyzetes ért. $f(\mathbf{x}, \mathbf{w})$ vett legjobb közelítését eredményezi. Másfelől, ha a tanulás célját úgy d galmazzuk meg, hogy azt az $\mathbf{g}(\mathbf{x})$ -t keressük, amely adott $\mathbf{x}$ mellett a lehető legjobban közelíti a megfelelő kívánt választ, akkor a megoldást az

$$MSE(\mathbf{w}) = E\left\{\left(f(\mathbf{x}, \mathbf{w}) - \mathbf{g}(\mathbf{x})\right)^2\right\} \quad (2.13)$$

minimalizálása biztosítja. A válasz helyességének természetes mértéke tehát a tanuló rendszer leképezése és a regressziós függvény kimenete közötti eltérés átlagos négyzetes értéke (MSE).

Az approximációs hiba (2.13) kifejezése alkalmas a megoldás minősítésének további elemzésére is. Az átlagos négyzetes eltérés ugyanis tovább dekomponálható, és ennek a dekompozíciónak az eredményeképpen juthatunk a torzítás-variancia dilemmához.

Torzítás-variancia dilemma

Bontsuk fel a (2.13) kifejezést a következő módon:

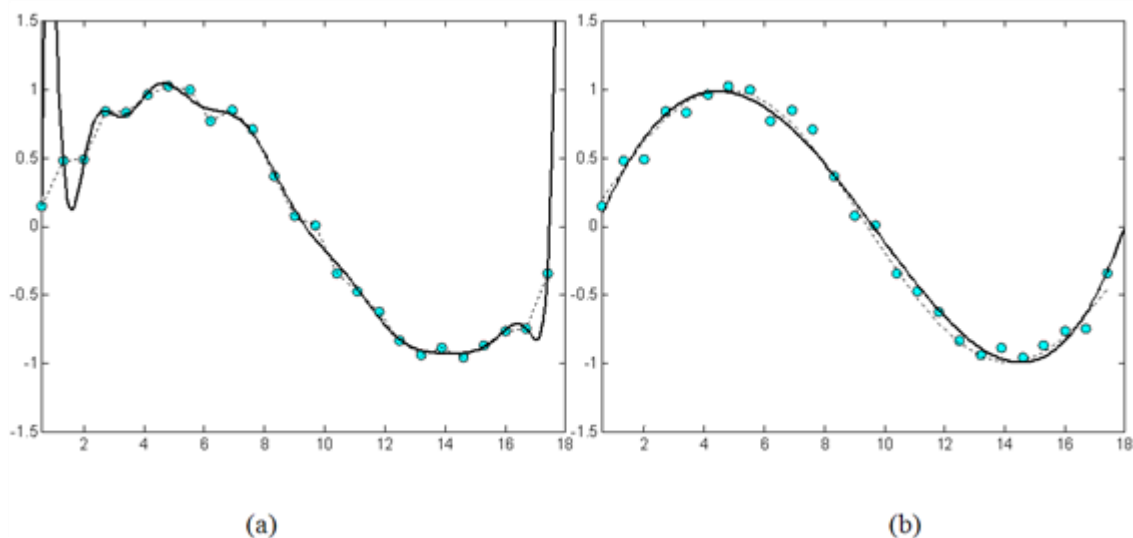
$$\begin{aligned} MSE(\mathbf{w}) &= E\left\{\left(f(\mathbf{x}, \mathbf{w}) - \mathbf{g}(\mathbf{x})\right)^2\right\} \\ &= E\left\{\left(\mathbf{g}(\mathbf{x}) - E\{f(\mathbf{x}, \mathbf{w})\} + E\{f(\mathbf{x}, \mathbf{w})\} - f(\mathbf{x}, \mathbf{w})\right)^2\right\} \\ &= \left(\mathbf{g}(\mathbf{x}) - E\{f(\mathbf{x}, \mathbf{w})\}\right)^2 + E\left\{\left(f(\mathbf{x}, \mathbf{w}) - E\{f(\mathbf{x}, \mathbf{w})\}\right)^2\right\} \\ &= B^2(\mathbf{w}) + V(\mathbf{w}) \end{aligned} \quad (2.14)$$

A felbontás szerint az átlagos négyzetes eltérés két tagjából az első, $B^2(\mathbf{w})$ a torzítás négyzete, a második, $V(\mathbf{w})$ pedig a megoldás varianciája. A torzítás (*bias*) azt mutatja meg, hogy a tanuló rendszer leképezésének várható értéke mennyiben tér el a regressziós függvénytől, míg a variancia a leképezésnek a saját várható értékétől való átlagos négyzetes eltérését adja meg. A torzítás arra ad választ, hogy az adott függvényosztály – amit a háló típusa és architektúrája rögzít – mennyire alkalmas a kérdéses approximációs feladat megoldására, a variancia pedig a véges számú tanítópont következménye: azt adja meg, hogy a véges számú tanítópont felhasználásával konstruált háló válasza, $f(\mathbf{x}, \mathbf{w})$ mennyire érzékeny arra, hogy egy adott problémából származó aktuális tanítóhalmaz épp milyen mintapontokból áll.

Azonos négyzetes hiba előállhat nagyobb torzítás és kisebb variancia vagy kisebb torzítás és nagyobb variancia mellett; a két eset eltérő jellegű approximációt jelent. A szabad paraméterek számának növelése és így a háló által megvalósított függvényosztály komplexitásának növelése a torzítás csökkentését eredményezi. Minél több a szabad paraméter, annál komplexebb leképezés megvalósítására lehet alkalmas a neuronháló. Elegendően nagyszámú szabad paraméter mellett a háló válasza a tanítópontokban akár tetszőlegesen kis hibájú is lehet, miközben a tanítópontok között a válasz a valódi leképezéstől jelentős mértékben eltérhet. Ezt az esetet, amikor a torzítás kicsi, de a variancia nagy **túlilleszkedésnek** (*overfitting*) nevezzük. A variancia csökkentése elsősorban a felhasznált tanítópontok számának a növelésével, pontosabban a szabad paraméterek számának és a tanítópontok számának megfelelő összehangolásával lehetséges.

A túlilleszkedésre példa egy olyan polinomiális regresszió, amikor kevés mintapontra akarunk nagyfokszámú polinomot illeszteni. A 2.3 ábra egy szinuszos függvény polinomiális approximációját mutatja kissé zajos mintavételi pontok alapján. Az (a) ábrán egy túl nagy fokszámú (18-ad fokú) polinommal történő közelítés, a (b) ábrán egy 5-ödfokú polinom illesztésének az eredménye látható. Látható, hogy a fokszám csökkentésével – esetleg a mintapontokbeli tökéletes illeszkedés rovására – a pontsorozat általános trendjének követése inkább lehetséges.

2.3. ábra - A szabad paraméterek számának hatása a túlilleszkedésre: szinusz függvény polinomiális approximációja. (a) 18-ad fokú polinommal, (b) 5-ödfokú polinommal



A torzítás-variancia dilemma tehát azt a kérdést veti fel, hogy hogyan viszonyuljon egymáshoz a kétféle hibakomponens. A kérdés szorosan összefügg a háló típusának és méretének (komplexitásának) a megválasztásával, de kapcsolatban van a tanítópontok számával és magával az alkalmazott tanuló eljárással is. Az átlagos négyzetes hiba ilyen felbontása tehát azáltal, hogy segít a hiba értelmezésében, fontos szerepet tölt be a hálók megfelelő konstrukciójában.

A torzítás-variancia dilemmát szemlélteti a 2.4. ábra. Látható, hogy az átlagos négyzetes értelemben legjobb megoldást nem a nagykomplexitású háló eredményezi, hanem az, ha a komplexitás összhangban van a tanítópontok számával és a feladat bonyolultságával.

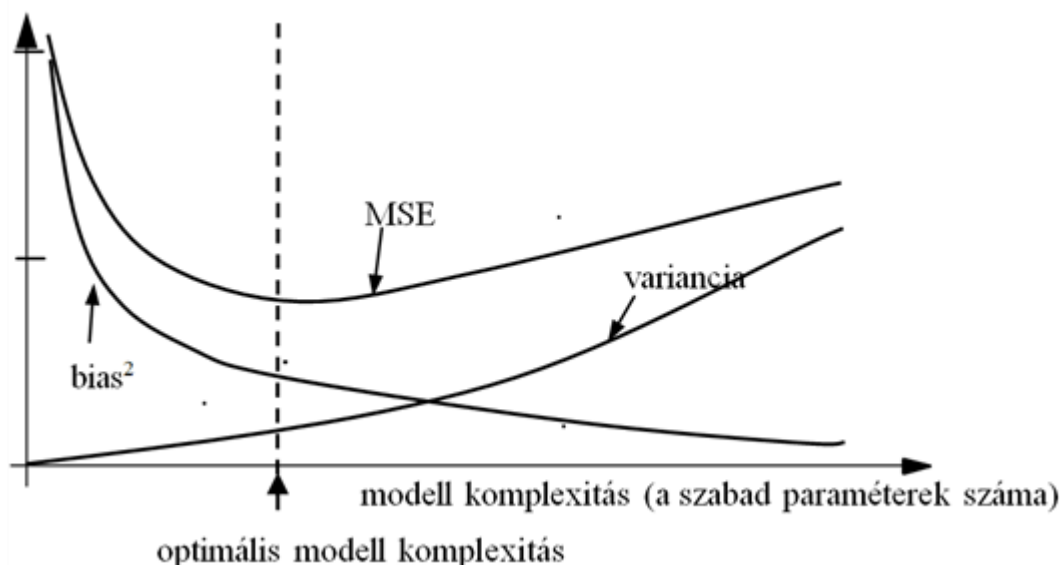
Regularizáció

Mint láttuk, a véges számú mintapont alapján történő tanulási feladat rosszul definiált probléma. Egy feladatot rosszul definiálnak (*ill-posed*) [Had02] nevezünk, ha

- nem létezik megoldása, vagy
- a megoldás nem egyértelmű, vagy
- a megoldás a kiinduló adatoknak nem folytonos függvénye.

Mint ahogy a 2.1. ábra is illusztrálta, a véges számú pontból egy bemenet-kimenet leképezés megtanulása rosszul definiált probléma, mert a második feltételt nem teljesíti.

2.4. ábra - A torzítás-variancia dilemma.



Az adatokból történő tanulás ugyanakkor ún. inverz probléma is. Tekintsünk egy $y = f(\mathbf{x})$ függvénykapcsolattal definiált leképezést. Ha a függvény ismert, adott $\mathbf{x}_i, i=1, 2, \dots, l$ független változó értékekhez a megfelelő y_i függvényértékek egyértelműen meghatározhatók. Ezzel szemben, ha az összetartozó $\{\mathbf{x}_i, d_i\}_{i=1}^l$ mintapontokat ismerjük, és ebből akarjuk visszaállítani az $y = f(\mathbf{x})$ függvényt, akkor egy inverz problémával állunk szemben. Az inverz problémák között számos rosszul definiált problémát találunk.

A neuronhálók tanításánál a rosszul definiáltság valójában azt jelenti, hogy a rendelkezésre álló információ – a tanítópontok együttese – az egyértelmű megoldáshoz nem elegendő. Egyértelmű megoldást csak további információ birtokában kaphatunk. Ilyen további információ lehet, hogy a megtanulandó leképezés – azon túl, hogy illeszkednie kell a tanítópontokra – legyen minél simább. Hasonló járulékos információként szolgálhat, ha egy kétféle osztályozási feladatnál arra törekszünk, hogy a két osztályt elválasztó felület ne csak elválassza a tanítópontokat, hanem az elválasztás minél nagyobb biztonsággal, tartalékkal valósuljon meg, az elválasztó felület az egyes osztályokba tartozó tanítópontoktól minél távolabb legyen.

A járulékos információkat be kell építenünk a probléma megfogalmazásába: a kockázat funkcionált járulékos taggal (tagokkal) kell kiegészítenünk. A járulékos tag mellékfeltételként vagy büntető tagként igyekszik a tanuló eljárást rákényszeríteni, hogy ne csak a kockázatot minimalizáló, hanem a mellékfeltételnek is eleget tevő megoldást kapjunk. Ha a kritériumfüggvényünket úgy módosítjuk, hogy a mintapontokra vonatkozó kockázatot kiegészítjük megfelelő mellékfeltétel(ek)et megfogalmazó taggal (tagokkal), regularizációról beszélünk [Tik77]. Regularizált esetben a kockázat helyett egy

$$R_{\lambda}(\mathbf{w}) = R(\mathbf{w}) + \lambda \Omega(\mathbf{w}) \quad (2.15)$$

kifejezés minimumát (szélső értékét) biztosító paramétervektort keresünk, ahol $R(\mathbf{w})$ az eredeti kockázat, $\Omega(\mathbf{w})$ a regularizációs feltételt megfogalmazó járulékos komponens és λ az ún. regularizációs együttható. A járulékos tag valamilyen plusz feltétel, kényszer figyelembevételét szolgálja, amely tükrözhet meglévő a priori információt, megfogalmazhat valamely, a realizációval kapcsolatos kényszert, vagy csak egyszerűen segíthet a szélsőérték probléma megoldásában. A regularizációs együttható szerepe, hogy az eredő kockázatban biztosítsa az eredeti kockázat és a regularizációs feltétel megfelelő arányát. A regularizáció, melynek elméleti megalapozása Andrej Tyihonovtól származik [Tik77], általában a rosszul definiált feladatok megoldásában nyújt segítséget. A regularizációs technika alkalmazására példákat a további fejezetekben, konkrét hálók kapcsán mutatunk.

Minősítő és teszt minták alkalmazása, kereszt kiértékelés

A kockázat alkalmazása a tanítás alatt álló vagy a megtanított háló minősítésére csak elvi lehetőség, hiszen a várható érték számításához vagy a $p(\mathbf{x}, d)$ együttes sűrűségfüggvény vagy a $P(\mathbf{x}, d)$ együttes eloszlásfüggvény

ismeretére lenne szükség. A mintákból történő tanulásnál ezeket azonban nem ismerjük, így a tanuló rendszer minősítésére a kockázat közvetlenül nem használható.

A kockázat közvetett meghatározásával lehetőségünk nyílna a tanuló rendszer minősítésére. Ekkor a tanítópontok alapján előbb az ismeretlen sűrűségfüggvényt becsüljük, majd annak ismeretében már tudunk becslést adni a kockázatra. A közvetett megoldás nehézsége, hogy a sűrűségfüggvény megfelelő pontosságú becsléséhez nagyszámú mintára van szükségünk. Hasznosabbnak bizonyul, ha nem ezt az utat választjuk, hanem a rendelkezésre álló mintapontokat mind a háló tanítására, mind annak minősítésére is közvetlenül felhasználjuk. Ez a megközelítés abból az általános elvből indul ki, hogy ne kíséreljünk meg egy speciális problémát egy nála nehezebb általános probléma megoldásának felhasználásával, indirekt úton megoldani [Vap95].

Amennyiben a sűrűségfüggvényt nem ismerjük, a kockázat alapján történő minősítés helyett a minősítésre is csak a rendelkezésünkre álló mintapontokat használhatjuk. Mást nem tehetünk, minthogy az ismert válaszú pontok esetében meghatározzuk a tanuló rendszer válaszában hibáját. Ezen pontok hibájának átlagát **empirikus** vagy **tapasztalati kockázatnak** (*empirical risk*) nevezzük. Az l ismert tanító pontpár esetében a tapasztalati kockázat:

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{l} \sum_{i=1}^l L(d_i, f(\mathbf{x}_i \mathbf{w})), \quad (2.16)$$

vagyis a tanítópontokban meghatározott veszteség átlaga.

Amennyiben a tapasztalati kockázatot a tanulásnál felhasznált mintapontokra számítjuk, a háló általánosító-képességéről általában semmit sem mondhatunk. A már említett túlilleszkedés miatt ugyanis előfordulhat, hogy a háló a tanítópontokat nagy pontossággal megtanulta, miközben a tanítópontoktól eltérő pontoknál a hiba nagyon nagy is lehet. A háló általánosítóképességének jellemzésére olyan mintapontokra kell meghatároznunk a tapasztalati hibát, melyeket a tanításnál nem használtunk fel.

Ennek érdekében egy külön kiértékelő (*validation*) mintakészletet kell használnunk, melynek mintái ugyancsak a megoldandó feladatból származnak, de amelyek függetlenek a tanító készlettől. A kiértékelő mintakészletet sokszor a tanítás folyamata alatt, a tanulás előrehaladtánál értékelésére használjuk. Így, annak ellenére, hogy ezen minták nem tartoznak a tanításnál közvetlenül felhasznált mintapontok közé, közvetve részei a tanuló eljárásnak. A megtanított háló minősítésére ekkor egy ettől is független mintakészlet, a tesztkészlet szolgálhat. A tesztkészlet szintén az adott problémából származó összetartozó be-kimeneti mintapárok halmaza. A tesztkészlet mintapontjait azonban a tanításnál sem közvetlenül, sem közvetve nem használjuk fel. A tesztkészlet a megtanított háló végső értékelésére, a háló általánosítóképességének a becslésére használható.

A rendelkezésre álló mintapontokat egy problémánál ezért három részre, tanító-, értékelő- és tesztelő készletre kell bontanunk. Kellően nagyszámú adat mellett ez nem okoz nehézséget. Ha azonban kevés adat áll rendelkezésünkre (ez számos gyakorlati feladatnál előfordulhat vagy az adatok beszerzésének nagy költsége, vagy az adatoknak a probléma természetéből adódó kis száma miatt) a három diszjunkt készletre bontás azt eredményezheti, hogy az egyes részfeladatok elvégzéséhez túl kevés mintapontunk marad. Így, ha viszonylag sok pontot használunk fel a tanításra és kevés marad a megtanított hálózat minősítésére, nem lehetünk biztosak a megtanított hálózat képességeit illetően. Ezzel szemben, ha a minősítésre hagyunk több mintapontot és ezért kevés tanító pontunk lesz, csak azt konstatálhatjuk – bár ezt kellő biztonsággal tehetjük –, hogy a hálónk meglehetősen gyengén tanulta meg a feladatot. Ekkor, mint ahogy a (2.14) összefüggésből is látható az eredő hiba variancia komponense lehet nagy. Az egyes részhalmazok közötti megfelelő arányok meghatározása meglehetősen nehéz feladat. Erre vonatkozó néhány eredményt, melyet Shun-ichi Amari és munkatársai dolgoztak ki [Ama97], a 4. fejezetben fogunk röviden bemutatni.

Az ellentmondás egy lehetséges feloldására ún. **kereszt kiértékelési** (*cross validation*) eljárás alkalmazható (pl. [Sto78]). Ennek során a rendelkezésre álló mintapontokat véletlenszerűen felosztjuk k diszjunkt részhalmazra, majd $k-1$ részhalmaz pontjait tanításra, a maradék részhalmaz pontjait pedig kiértékelésre használjuk. Az eljárást megismételjük az összes részhalmazra úgy, hogy a tanítópontok közül mindig más-más részhalmazt hagyunk ki és használunk kiértékelésre. A hálózat eredő minősítését az egyes kiértékelések átlagaként nyerjük. Az eljárás előnye, hogy a rendelkezésre álló pontok jelentős részét felhasználjuk a hálózat tanítására, miközben az eredő értékelést az összes pont alapján végezzük, de úgy, hogy egy adott tanítást követően a háló értékelése mindig csak olyan pontokkal történik, melyeket az adott tanításnál nem használtunk fel. Hátrányként a hosszabb számítási időt említhetjük, hiszen a feladatot most k -szor kell megtanítanunk. Nagyon kevés rendelkezésre álló pont esetén a részhalmazok száma megegyezhet a pontok számával ($k=l$), ekkor minden tanítás $l-1$ ponttal

történik, a minősítésre pedig a kihagyott pontot használjuk. Az eljárást itt is *l*-szer végezzük el, tehát végül itt is az összes pont alapján történik a megtanított háló minősítése. A kereszt kiértékelést ebben az esetben egy-kihagyásos (*leave-one-out*, *loo*) eljárásnak hívjuk.

Az itt említett eljárások a tapasztalati hibát alkalmazzák a megtanított háló minősítésére. Láttuk azonban, hogy ha a tapasztalati hibát a tanításnál ténylegesen felhasznált pontokra számítjuk, az általánosítóképességről nem kapunk megbízható információt. Amennyiben viszont független tesztkészletet használunk, a rendelkezésre álló ismert válaszü pontok korlátozott száma okozhat nehézségeket. További gondot jelent, hogy, még ha a tanító készlettől független tesztkészletet használunk is a minősítésre, a tapasztalati hibát mindenképpen véges számú pont alapján számítjuk. A tapasztalati kockázat tehát csupán becslése a valódi kockázatnak, és – különösen kevés teszt adat mellett – a becslés nagyon pontatlan is lehet. Alapvetően fontos ezért annak a kérdésnek a megválaszolása, hogy a tapasztalati hiba alapján állíthatunk-e valamit a (2.7) összefüggéssel definiált valódi kockázatról, és ha igen, akkor ez milyen feltételek mellett tehető meg. Erre a kérdésre az elsősorban Vladimir Vapnik nevéhez fűződő statisztikus tanuláselmélet (*statistical learning theory*) ad választ, melynek alapjait a 2.3. alfejezetben foglaljuk össze.

2. Nemellenőrzött tanulás

Ellenőrzött tanulásnál a tanuló eljárás céljának megfogalmazása természetes módon adódott: a tanulással olyan rendszert kívántunk kialakítani, amely valamilyen előre meghatározott viselkedést mutat, vagy egy ilyen viselkedést legalább közelít. Mindezt felcímkézett tanítópontok alapján tettük, ahol a címke a bemenethez tartozó kívánt válasz volt.

Nemellenőrzött tanításnál címkézetlen tanítópontjaink vannak, így egy előírt viselkedés megtanulása fel sem merül. A hálózathoz a bemenetei és a kimenetei alapján, de kívánt válaszok ismerete nélkül kell valamilyen viselkedést kialakítani; a környezetből azonban nincs közvetlen visszajelzés, ami a hálózat viselkedésének a helyességére utalna. Azt a kérdést kell tehát feltennünk, hogy ilyen esetben lehet-e egyáltalán értelmes célt megfogalmazni, amit tanulással, a mintákban meglévő ismeret kinyerésével érhetünk el. Mivel csak a címkézetlen minták állnak rendelkezésünkre, mást nem tehetünk, minthogy a mintákat elemezzük. Azt vizsgáljuk, hogy a mintáink mennyiben hasonlítanak egymáshoz, vagy milyen mértékben különböznek egymástól; a mintapontok terében vannak-e olyan tartományok, ahol a mintáink sűrűsödnek, lehet-e a mintatérben csoportokat, klasztereket találni, stb.

A nemellenőrzött tanulás a mintákból való tanulásnak talán még az ellenőrzött tanulásnál is elemibb formája; olyan adatelemzés, amikor azt próbáljuk felderíteni, hogy az adataink egyáltalán hordoznak-e valamilyen információt. Ha igen, akkor azt valamilyen módon meg akarjuk „ragadni”, az adatokból ki akarjuk nyerni. Azt is mondhatjuk, hogy a nemellenőrzött tanulás egyfajta adatbányászat.

Tipikus nemellenőrzött tanulási feladat a sűrűségfüggvénybecslés vagy a klaszterezés. Nemellenőrzött tanulási feladatként megfogalmazható az is, hogy sokdimenziós adattérben vannak-e kitüntetett „irányok”, melyek az adatok reprezentálásánál elsődlegesen fontosak, míg más irányok szerepe kisebb, esetleg elhanyagolható. Az ilyen típusú feladatok az adatok hatékony közelítő reprezentációjára is irányulhatnak.

A nemellenőrzött tanulási eljárásokat sokszor adatelőkészítésre, az adatok előfeldolgozására használjuk, mielőtt egy ellenőrzött tanulással megoldható feladatot oldanánk meg. Ilyenkor a nemellenőrzött tanulás bizonyos, a későbbi feldolgozást elősegítő lényeges információ kiemelésére, lényegkiemelésre, jellemzőkinyerésre (*feature extraction*) szolgál.

Számos esetben a nemellenőrzött tanulási eljárásoknál is megfogalmazhatók kritériumfüggvények, így a nemellenőrzött tanulás is interpretálható olyan feladatként – pl. szélsőérték-keresési feladatként –, ahol valamilyen kritérium kielégítése a cél. Nemellenőrzött tanulásnál az alkalmazott kritériumfüggvények leggyakrabban az adatok eloszlásával, az adatok egymáshoz viszonyított elhelyezkedésével, az adatkomponensek közötti korrelációval kapcsolatosak, de itt is megfogalmazhatók négyzetes kritériumfüggvények. Ilyen eset például, ha az adatok hatékony, átlagos négyzetes értelemben vett közelítő reprezentációja a cél.

A nemellenőrzött tanulásnál a kritériumfüggvények sokszor információelméleti fogalmakhoz kapcsolódnak. Így kritériumként szerepelhet az entrópia (olyan adatrepresentációt keresünk, ahol az entrópia valamilyen extrémális értéket vesz fel), a kölcsönös információ (az adatok statisztikai függetlenségét szeretnénk biztosítani) vagy a Kullback-Leibler divergencia (olyan tanuló eljárásnál, ahol két sűrűségfüggvény hasonlósága alapján történik a tanulás).

Számos, a nemellenőrzött tanulású hálónál alkalmazott kritériumfüggvényt az egyes hálók tárgyalásánál (10. fejezet) fogunk bemutatni.

3. A statisztikus tanuláselmélet alapjai

Az előzőekben láttuk, hogy a valódi kockázat meghatározása a megfelelő sűrűség- vagy eloszlásfüggvény ismeretének hiányában nem lehetséges, így akár a tanulás eredményének folyamatos követése, akár a megtanított háló minősítése csak a tapasztalati kockázat alapján lehetséges. A tanítópontok átlagos hibájának, a tapasztalati kockázatnak a minimalizálása útján nyert megoldás kisszámú tanítópont esetén automatikusan nem biztosítja, hogy ez a megoldás a valódi kockázat minimumát is biztosítani fogja. A tapasztalati kockázat minimalizálásának elve (*empirical risk minimization – ERM – principle*) ennek ellenére azt mondja ki, hogy eredménynek a tapasztalati hiba minimumát biztosító megoldást fogadjuk el.

A véges számú mintából való tanulás és statisztikai becslés elméleti megalapozása Vladimir Vapniktól és Alekszej Cservonyenkisztől származik [Vap95], [Vap98]. Ez az elmélet, melyet statisztikus tanuláselméletnek (*statistical learning theory*) vagy Vapnik-Cservonyenkisz (VC) elméletnek is neveznek, megadja a tapasztalati kockázat alkalmazhatóságának egzakt matematikai feltételét. Megmutatja, hogy bizonyos feltételek teljesülése esetén az R_{emp} tapasztalati kockázat, a mintapontok számának növelésével tart a valódi kockázathoz, R -hez, sőt a konvergencia sebességéről (vagyis arról, hogy a mintapontok számának függvényében a tapasztalati kockázat milyen gyorsan közelíti a valódi kockázatot) is tud bizonyított állítást megfogalmazni.

A statisztikus tanuláselmélet fontos eredménye az is, hogy az általánosítási hibára, vagyis a valódi kockázatra a tapasztalati hibától függő felső korlátot tud adni, valamint, hogy általános eljárást ad olyan tanuló algoritmus konstrukciójára, mely mellett kontrolálható az általánosítási hiba.

A statisztikus tanuláselmélet az alábbi négy fő részből áll. Megadja, hogy

- mik a feltételei az ERM elv konzisztenciájának,
- milyen gyors a tanulási folyamat konvergenciája,
- milyen felső korlátok határozhatók meg az általánosítási hibára, és
- milyen konstruktív eljárást lehet követni tanuló eljárások létrehozásánál, hogy a fenti eredmények érvényesek legyenek.

A statisztikus tanuláselmélet elsősorban osztályozási feladatokra fogalmaz meg állításokat, de az elméletet kiterjesztették más tanulási problémákra, így regressziós feladatokra is.

A statisztikus tanuláselmélet teljes mélységű tárgyalását [Vap98]-ban találjuk. Az elmélet mögötti alapvető gondolatok, az eredmények matematikai megfogalmazásával, de bizonyítások nélkül [Vap95]-ben szerepelnek, míg egy rövid áttekintést [Vap99] ad. Jelen könyvben – jórészt [Vap95] és [Vap99] alapján – csak a legfontosabb eredmények rövid áttekintése szerepel a matematikai részletek mindennemű bemutatása nélkül. A téma iránt érdeklődő olvasó elsősorban az említett három irodalomban talál bővebb és mélyebb ismereteket.

3.1. Az ERM konzisztenciája

Az ERM elv szerint a kockázat minimalizálása elérhető a tapasztalati kockázat minimalizálása alapján. Az ERM konzisztenciája azt jelenti, hogy a tapasztalati kockázat tart a valódi kockázathoz, ha a mintapontok számát minden határon túl növeljük, tehát az ERM konzisztenciájára vonatkozó elméleti eredmények a tapasztalati hiba aszimptotikus alakulására vonatkoznak és megadják a tapasztalati hiba konvergenciájának szükséges és elégséges feltételét.

Felmerül a kérdés, hogy mi szükség van egy aszimptotikus elméletre, amikor véges számú minta alapján dolgozó tanuló algoritmusokat vizsgálunk. A konzisztencia elmélet szükségességét az támasztja alá, hogy ez az elmélet a tapasztalati hiba minimalizálás konzisztenciájának nemcsak szükséges, de elégséges feltételeit is megadja. Ezért az ERM elv konzisztenciájára vonatkozó bármely elméletnek ki kell elégítenie ezeket a feltételeket.

A probléma megfogalmazásához induljunk ki az eddigi jelölésekből. Jelölje $\mathbf{z} = (\mathbf{x}, d)$ egy tanító be-
kimeneti mintapárt. A tanítási feladat során rendelkezésünkre áll egy $\mathcal{W}$ tanítókészlet, melynek
elemeiről feltételezzük, hogy egymástól független, azonos – de ismeretlen – $L(\mathbf{z}, \mathbf{w})$ $\mathbf{w} \in \mathcal{W}$ tapontok. A $p(\mathbf{z})$
sűrűségfüggvény $R(\mathbf{w})$ hát nem ismerjük. Adott továbbá $\mathbf{w}_0$ gy veszteségfüggvény, $R(\mathbf{w}_0)$, és a (2.7)-
ben definiált $R(\mathbf{w})$ kockázat $R(\mathbf{w}_0)$ inimumát biztosító paramétervektor meghatározása. Mivel a mintapon $\mathbf{w}_0$
eloszlását nem ismerjük, $R(\mathbf{w}_0)$ -t sem tudjuk meghatározni, így a valódi kockázat minimumát biztosító $R_{\text{emp}}(\mathbf{w})$
paramétervektort sem. Az ERM elv szerint a valódi (ismeretlen) $R(\mathbf{w}^*)$ ázat helyett a (meghatározható)
tapasztalati kockázattal dolgozunk és az ennek minimumát biztosító $R_{\text{emp}}(\mathbf{w}^*)$ paramétervektort keressük.

A tanuló eljárások többségénél ezt az eljárást követjük anélkül, hogy az eljárás elméleti megalapozottsága
bizonyított lenne. Ez ugyan természetesen adódó megoldás, hiszen a mintapontok eloszlásának ismerete nélkül
csak a tapasztalati hiba határozható meg, azonban az eljárás alkalmazhatóságához a matematikai igazolás is
szükséges lenne. Az igazolás és a megfelelő feltételek meghatározása ezért mind elméleti, mind gyakorlati
szempontból kiemelten fontos.

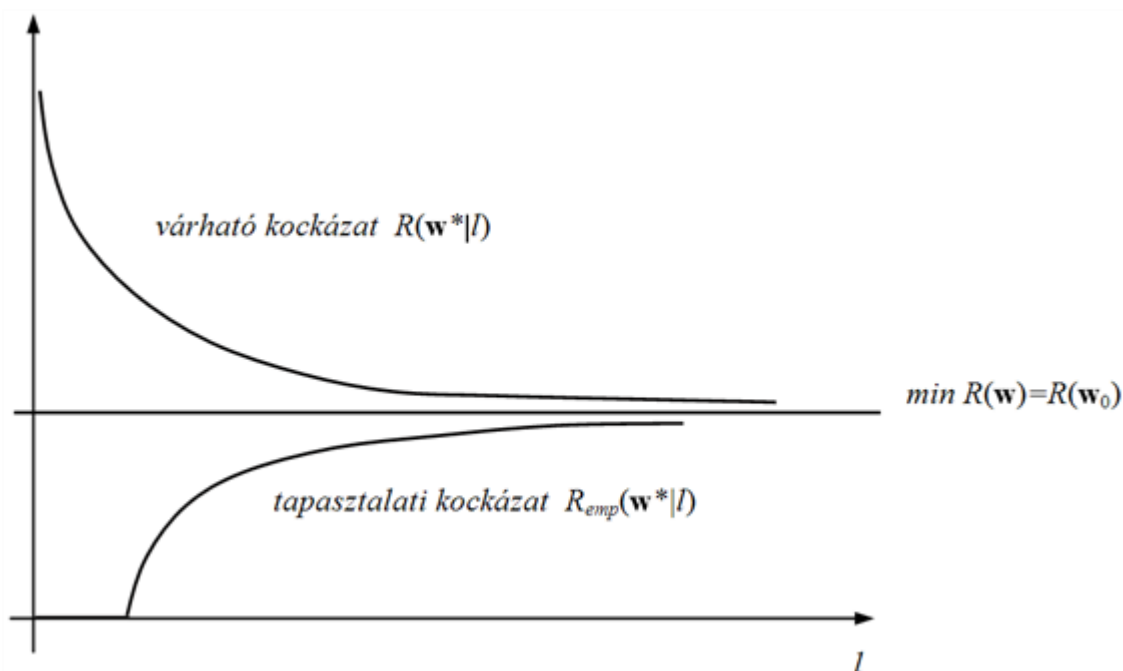
Az ERM elv alkalmazhatóságához azt kell belátni, hogy az ERM elv alapján kapott becslés, $\mathbf{w}^*$ a valódi
paramétervektorhoz konvergál, ha a tanítópontok számát növeljük. A statisztikus tanuláselmélet fő eredménye
éppen az, hogy megadja az ERM konzisztenciájának feltételét.

Először nézzük, hogy mit is jelent az ERM elv konzisztenciája. Jelölje $R_{\text{emp}}(\mathbf{w}^*|l)$ az l mintapont alapján
meghatározott tapasztalati hiba optimumát, és $R(\mathbf{w}^*|l)$ azt az ismeretlen valódi kockázatot, melyet ugyancsak az
 l tanítópont alapján számított tapasztalati hiba minimumát biztosító $\mathbf{w}^*$ mellett kapnánk. Az ERM elv akkor
konzisztens, vagyis az ERM elv alapján számított kockázat becslés akkor konzisztens, ha mind a tapasztalati
kockázat, $R_{\text{emp}}(\mathbf{w}^*|l)$, mind a valódi kockázat, $R(\mathbf{w}^*|l)$ ugyanahhoz az $R(\mathbf{w}_0) = \min_{\mathbf{w}} R(\mathbf{w})$ határértékhez tart,
ha $l \rightarrow \infty$. Vagyis

$$\begin{aligned} R(\mathbf{w}^*|l) &\rightarrow R(\mathbf{w}_0), & \text{ha } l \rightarrow \infty \\ R_{\text{emp}}(\mathbf{w}^*|l) &\rightarrow R(\mathbf{w}_0), & \text{ha } l \rightarrow \infty \end{aligned} \quad (2.17)$$

A konzisztenciát szemléletesen a 2.5 ábra illusztrálja. Amint az ábra mutatja, feltételezhető, hogy
 $R_{\text{emp}}(\mathbf{w}^*|l) < R(\mathbf{w}^*|l)$, hiszen a tanuló eljárás a tapasztalati hiba és nem a valódi hiba minimalizálása útján
határozza meg $\mathbf{w}^*$ -ot. Intuitív módon azt várjuk, hogy $l \rightarrow \infty$ esetén a tapasztalati kockázat bármely $\mathbf{w}$ mellett
konvergál az ugyanahhoz a $\mathbf{w}$ paramétervektorhoz tartozó valódi kockázathoz. Ebből azonban még nem
következik, hogy a tapasztalati hibát minimalizáló paramétervektor a valódi hibát is minimalizálni fogja, ha $l \rightarrow$
 ∞ . Hiszen a tapasztalati hiba mindig egy adott l elemű mintapontkészlet alapján határozható meg, így
óhatatlanul az épp rendelkezésre álló mintakészlethez igazodó megoldást biztosít. A valódi kockázat viszont
nem függ az aktuális mintakészlettől.

2.5. ábra - Az ERM elv konzisztenciája



A statisztikus tanuláselmélet fő tételét Vladimir Vapnik és Alekszej Cservonyenkisz fogalmazták meg.

2.1 tétel [Vap91]

Adott korlátos veszteségfüggvény mellett az ERM elv akkor és csak akkor konzisztens, ha a tapasztalati kockázat egyenletesen konvergál a valódi kockázathoz a következő értelemben:

$$\lim_{l \rightarrow \infty} P\left(\sup_{\mathbf{w} \in \mathbb{W}} |R(\mathbf{w}) - R_{\text{emp}}(\mathbf{w})| > \varepsilon\right) = 0 \quad \forall \varepsilon > 0 \quad (2.18)$$

Itt P a valószínűséget, $R_{\text{emp}}(\mathbf{w})$ az l mintapont alapján számított tapasztalati kockázatot, $R(\mathbf{w})$ pedig a valódi kockázatot jelöli azonos $\mathbf{w}$ mellett.

Látni kell, hogy a konzisztenciatétel legrosszabb esetre vonatkozó, ún. *worst case* tétel, vagyis ha a konzisztencia biztosított, a fenti összefüggésnek fenn kell állnia bármely, a tanuló rendszer által megvalósított függvény (bármely $\mathbf{w} \in \mathbb{W}$) mellett még akkor is, ha az adott függvény mellett a tapasztalati és a valódi kockázat eltérése a legnagyobb.

A fenti konzisztenciatétel ugyanakkor gyakorlati felhasználásra nem alkalmas. Egyrészt nem konstruktív, vagyis ennek alapján nem tudunk a gyakorlatban tanuló rendszert konstruálni, másrészt a konvergencia ténye önmagában nem elegendő, hiszen a tapasztalati kockázat tetszőlegesen lassan is konvergálhat a valódi kockázathoz. A konvergencia-sebességről is kellene valamit állítani; meg kellene fogalmazni azon feltételeket, melyek mellett a konvergencia gyors.

Az aszimptotikus konvergenciát gyorsnak nevezzük, ha tetszőleges $l > l_0$ esetére teljesül a következő egyenlőtlenség:

$$P\left\{|R(\mathbf{w}^*|l) - R(\mathbf{w}_0)| > \varepsilon\right\} < e^{-cl\varepsilon^2}, \quad (2.19)$$

ahol $c > 0$ egy konstans.

A statisztikus tanuláselmélet megadja mind a konzisztencia szükséges és elégséges feltételét, mind a gyors konvergencia elégséges feltételét. A feltétel megfogalmazásához néhány további fogalom bevezetésére van szükség. Az elmélet bevezet egy új entrópia fogalmat: egy függvényhalmaz entrópiájának a fogalmát és egy ún. növekedés függvényt (*growth function*), végül ehhez kapcsolódóan az ún. VC-dimenzió (*VC-dimension*) fogalmát. A függvényhalmaz entrópiájának bevezetése viszonylag természetes módon osztályozós esetre

lehetséges, bár e fogalmat kiterjesztették valós értékű függvényekre is. Osztályozós esetben a veszteségfüggvény csak két értéket, nullát vagy egyet vehet fel. Az ilyen függvényeket indikátorfüggvényeknek nevezzük. A valós értékű függvényekre való kiterjesztés alapján az eredmények általánosíthatók regressziós feladatokra is; a kiterjesztéssel azonban itt nem foglalkozunk.

Tekintsük az $L(\mathbf{z}, \mathbf{w})$, $\mathbf{w} \in \mathbb{W}$ indikátorfüggvények egy osztályát és legyen adott egy $\mathbf{Z}_l = \{\mathbf{z}_i, i = 1, \dots, l\}$ mintakészlet. Minden egyes indikátorfüggvény két részhalmazra particionálja a mintakészletet. A particionálás kétosztályos osztályozást jelent, vagyis azt is mondhatjuk, hogy a particionálásnál a mintakészlet pontjait két eltérő színnel kiszínezzük. A függvényhalmaznak az adott mintakészletre vonatkozó sokféleségét (*diversity*) jelöljük $N(\mathbf{Z}_l)$ -vel. $N(\mathbf{Z}_l)$ egy olyan szám, ami azt adja meg, hogy az adott függvényosztály elemeivel az adott mintakészletet hányféle eltérő módon tudjuk kiszínezni. A függvényosztály véletlen entrópiájának (*random entropy*) e szám logaritmusát nevezzük:

$$H(\mathbf{Z}_l) = \ln N(\mathbf{Z}_l). \quad (2.20)$$

A kapott mennyiség egy valószínűségi változó, hiszen függ a $\mathbf{Z}_l$ mintakészlettől. A véletlen entrópiának a minták együttes eloszlása szerinti várható értéke az indikátorfüggvény entrópiája vagy VC-entrópia.

$$H(l) = E\{\ln N(\mathbf{Z}_l)\}. \quad (2.21)$$

A VC-entrópia mind az indikátorfüggvény-halmaztól, mind a mintakészlet eloszlását jellemző együttes eloszlás-vagy sűrűségfüggvénytől függ. A VC-entrópia az ERM elv konzisztenciájában kap szerepet, ugyanis az ERM elv konzisztenciájának szükséges és elégséges feltétele, hogy teljesüljön a

$$\lim_{l \rightarrow \infty} \frac{H(l)}{l} = 0 \quad (2.22)$$

feltétel.

Mivel ez az összefüggés a mintapontok eloszlásától függő VC-entrópiára vonatkozik, a közvetlen gyakorlati felhasználásra nem alkalmas. További „hiányossága”, hogy a feltétel csak magának a konvergenciának a tényére vonatkozik, a konvergencia-sebességről nem állít semmit. A statisztikus tanuláselmélet ugyanakkor a konvergencia eloszlásfüggetlen feltételét is megadja, amely egyben biztosítja a (2.19) összefüggés által megfogalmazott módon a gyors konvergenciát is. Ehhez a következőnek kell teljesülnie:

$$\lim_{l \rightarrow \infty} \frac{G(l)}{l} = 0 \quad (2.23)$$

A statisztikus tanuláselmélet a $G(l)$ függvényt nevezi növekedésfüggvénynek (*growth function*), amelyet a véletlen entrópia alapján a következőképpen definiál:

$$G(l) = \ln \max_{\mathbf{z}_l} N(\mathbf{Z}_l). \quad (2.24)$$

$G(l)$ értékei eloszlásfüggetlen mennyiségek, melyek meghatározásánál az összes lehetséges l elemű mintakészletet tekintve kell a maximumot keresni. A növekedésfüggvény a mintapontok számának függvényében azon lehetséges particionálások (kétosztályos szeparálások) maximális számát (pontosabban annak logaritmusát) adja meg, melyeket az $L(\mathbf{z}, \mathbf{w})$ függvényosztályba tartozó indikátorfüggvényekkel lehet megvalósítani. $G(l)$ definíciója szerint elegendő, ha egyetlen olyan l elemű mintakészletet találunk, amely mellett a maximális számú szeparálás lehetséges. Az összes l elemű mintakészletnél nem feltétlenül kapunk azonos számban kétosztályos szeparálásokat. A növekedésfüggvény, ami az alkalmazott függvényosztálytól függ, az eloszlásfüggetlen entrópia felső korlátja. Ugyanakkor, mivel egy l elemből álló mintakészletet maximálisan 2^l -féleképpen lehet particionálni, nyilvánvaló, hogy

$$G(l) \leq l \ln 2, \quad (2.25)$$

vagyis a bevezetett mennyiségek között az alábbi egyenlőtlenségek állnak fenn:

$$H(l) \leq G(l) \leq l \ln 2. \quad (2.26)$$

Ennek igazolásához a VC-elmélet a Jensen egyenlőtlenség alapján felhasználja, hogy

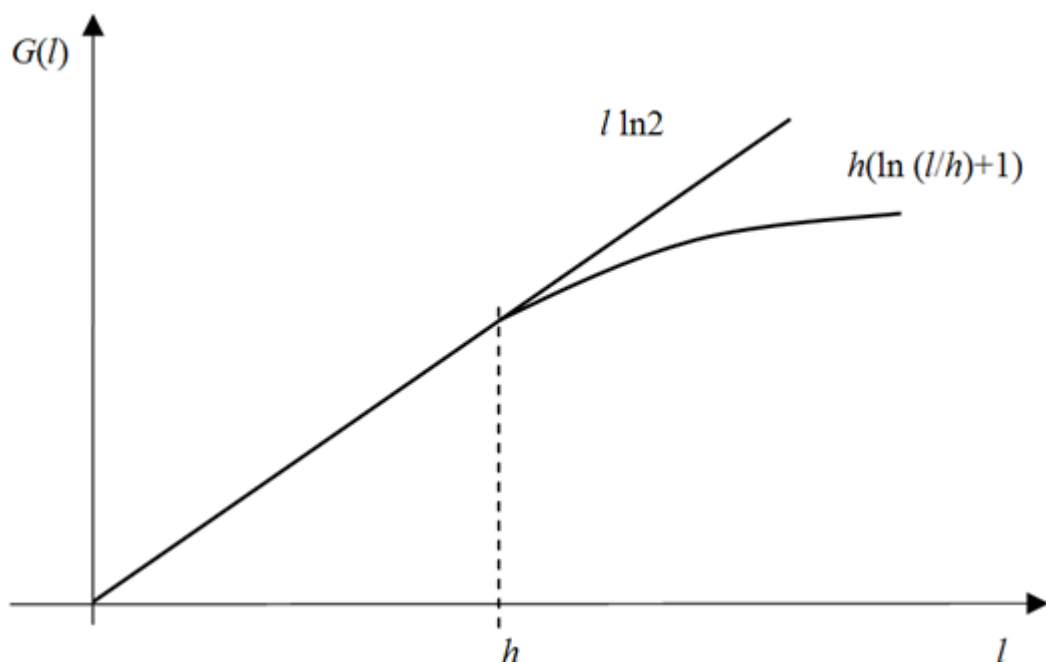
$$H(l) = E\{\ln N(\mathbf{Z}_l)\} \leq \ln E\{N(\mathbf{Z}_l)\}, \quad (2.27)$$

továbbá, hogy

$$\ln E\{N(\mathbf{Z}_l)\} \leq \ln \max_{\mathbf{z}_l} N(\mathbf{Z}_l) = G(l) \quad (2.28)$$

Az általánosítóképességre vonatkozó eloszlásfüggetlen eredmény a növekedésfüggvény vizsgálata alapján nyerhető. Vapnik és Cservonyenkis bebizonyították, hogy a növekedésfüggvény vagy lineárisan növekszik l -lel, vagy l valamely logaritmikus függvényével felülről korlátozható, ahogy ezt a 2.6 ábra mutatja.

2.6. ábra - A növekedésfüggvény alakulása



Az előbbi esetben

$$G(l) = l \ln 2, \quad (2.29)$$

az utóbbiban pedig

$$G(l) \leq h \left(1 + \ln \frac{l}{h} \right). \quad (2.30)$$

A mintapontok számának az az l értéke, ahol $G(l)$ növekedése a lineáris növekedéshez képest lelassul nevezetes érték. Ezt az értéket nevezzük VC-dimenzióknak, és a továbbiakban h -val fogjuk jelölni. Ha h véges érték, akkor nagy mintaszámokra a növekedésfüggvény a lineárisnál lassabban nő.

Figyelembe véve a gyors konvergencia (2.23)-ban megfogalmazott feltételét, az ERM elv konzisztenciájának és a gyors konvergenciának eloszlásfüggetlen szükséges és elégséges feltétele, hogy a VC-dimenzió, h véges legyen. Ugyanakkor, ha a növekedésfüggvény bármekkora l mellett l -lel lineárisan nő, az adott indikátorfüggvény-készlet VC-dimenziója definíciószerűen végtelen. Ebben az esetben az adott függvényosztály elemeivel bármely l elemű mintakészletet minden lehetséges módon, tehát mind a 2^l -féleképpen lehet particionálni. Ez pedig azt jelenti, hogy az a tanuló rendszer, amelyik ilyen függvényosztályba tartozó függvények megvalósítását végzi, általánosításra nem lesz képes. Vagyis akármilyen tanítókészletünk is lesz, a tanítópontokhoz bármilyen módon is rendeljük hozzá a kívánt válaszokat – a címkéket – a tanuló rendszer a

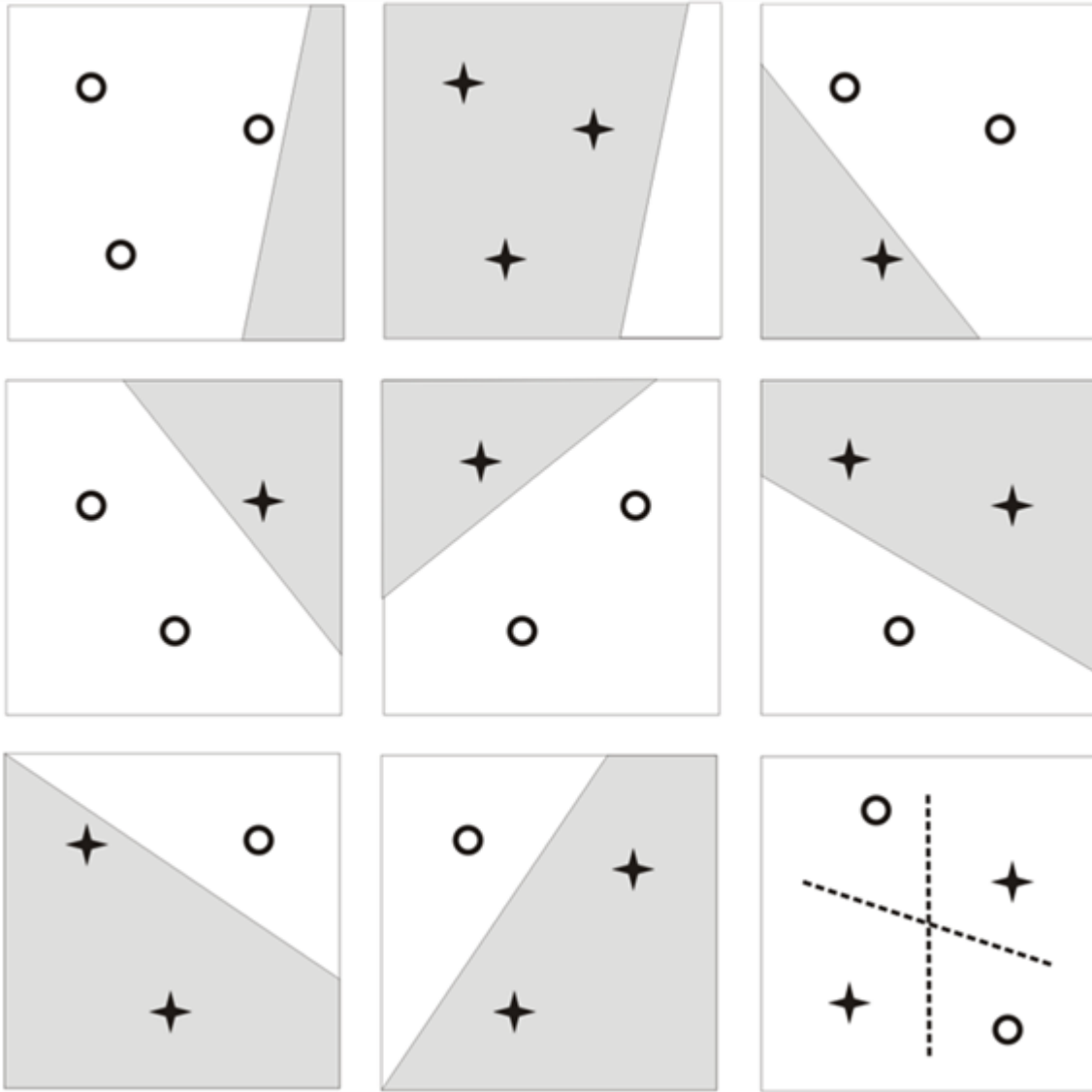
tanítópontokat mindig hibátlanul meg tudja tanulni; nem az osztályozási feladatot, hanem az adott mintapontokat tanulja meg, és így a tanítópontokra kapott válaszok alapján az általánosítóképességéről semmit sem tudunk állítani.

Fentiek alapján a VC-dimenziónak egy alternatív definíciója is megadható: egy indikátorfüggvény-készletnek h a VC-dimenziója, ha létezik h olyan mintapont, melyeket a függvénykészlet elemeivel minden lehetséges módon be tudunk sorolni két osztály valamelyikébe (a h mintapontot minden lehetséges módon két színnel ki lehet színezni), de $h+1$ ilyen mintapont már nem létezik. A VC-dimenzió tehát az a maximális mintaszám, amit adott függvényosztály elemeivel hibátlanul particionálhatunk minden lehetséges módon. A VC-dimenzió egy függvényosztály komplexitásának egyfajta mértékeként is tekinthető.

A VC-dimenzió fogalmát illusztrálja a 2.7 ábra. Tekintsük a lineáris függvények osztályát, vagyis a mintapontok particionálására a lineáris függvények osztályából válasszunk függvényeket. Ha $l=3$, tehát három mintapontunk van, a pontthalmaz bármely particionálása (bármilyen címkézése) lehetséges megfelelően megválasztott egyenessel. Négy pont esetén ez már nem mindig oldható meg: az összes (2^4) lehetséges szeparálás mindegyike lineáris szeparáló függvénnyel már nem lehetséges. Az alábbi ábra a három ponthoz tartozó nyolc lehetséges kombinációt mutatja be. Jól látható, hogy a pontok bármely címkézése mellett szeparáló egyenes elhelyezhető. A négypontos esetek közül csak egy lineárisan nem megoldhatót mutatunk. Az ábra alapján meg is állapítható, hogy a lineáris szeparáló függvények osztályának VC-dimenziója kétdimenziós mintapontoknál 3.

A statisztikus tanuláselmélet fontos eredménye, hogy annak megállapításán túl, hogy véges VC-dimenzióra van szükség, felső korlátokat is meg tud fogalmazni a tanuló rendszer valódi hibájára (a valódi kockázatra). Ezek a felső korlátok összekapcsolják a tapasztalati- és a valódi kockázatot. Arra a kérdésre adnak egyfajta választ, hogy milyen közel van egymáshoz az $R(\mathbf{w}^*|l)$ valódi kockázat és az $R_{\text{emp}}(\mathbf{w}^*|l)$ tapasztalati kockázat. A kapcsolatban h értékének is fontos szerep jut.

2.7. ábra - A lineáris szeparálás kapacitását bemutató kétdimenziós példa. Két dimenzióban három pont mindig szeparálható lineárisan: az első 8 elrendezés ezt mutatja, de négy pontnál ez már nem minden esetben lehetséges: ld. utolsó eset.



Az elmélet keretében számos ilyen korlát született. Itt példaként csak két eloszlásfüggetlen korlátot említünk meg, amelyek a bináris osztályozós (kétosztályos), illetve a regressziós problémákra vonatkoznak.

Egy osztályozó valószínűségi hibájára vonatkozó következő egyenlőtlenség legalább $1-\eta$ valószínűséggel érvényes az összes $L(\mathbf{z}, \mathbf{w})$ függvényre, köztük arra az $L(\mathbf{z}, \mathbf{w}^* | l)$ -re is, amely az l mintapont alapján számított tapasztalati kockázatot minimalizálja:

$$R(\mathbf{w}) \leq R_{\text{emp}}(\mathbf{w}) + \frac{\varepsilon(h)}{2} \sqrt{1 + \frac{4R_{\text{emp}}(\mathbf{w})}{\varepsilon(h)}}, \quad (2.31)$$

ahol

$$\varepsilon(h) = 4 \frac{h(\ln(2l/h) + 1) - \ln(\eta/4)}{l}. \quad (2.32)$$

Itt h a modell VC-dimenziója, és l a tanítóminták száma.

A (2.31) összefüggés alapján látható, hogy a valódi kockázat felső korlátja két tagból áll: az első a mintapontok alapján számítható tapasztalati kockázat, míg a második a tanuló rendszer komplexitására jellemző VC-dimenzió értékétől (pontosabban a h/l aránytól) is függ. Jó általánosító-képesség eléréséhez ezért mind a két tag minimalizálása szükséges: önmagában a kis tapasztalati hiba nem elegendő, hiszen ez csak azt biztosítja, hogy a

mintapontokat a tanuló rendszer jól megtanulta. Ha emellett a második tag értéke nagy, akkor az általánosítóképesség gyenge lesz. A már említett túlilleszkedés ilyen esetnek felel meg. A második tag, amit konfidencia tagnak is szokás nevezni, akkor vesz fel nagy értéket, ha h értéke nagy, pontosabban, ha a mintapontok számához viszonyítva nagy a VC-dimenzió értéke. Mint említettük a VC-dimenzió tekinthető úgy, mint az osztályozási feladatot megoldó rendszer által megvalósítható függvényosztály komplexitásának valamilyen mértéke. Minél nagyobb komplexitású rendszerrel valósítjuk meg a feladatot, annál nagyobb az esély, hogy a tanítópontokban a hibát tetszőlegesen kicsivé tudjuk tenni, vagyis a (2.31) összefüggésben a tapasztalati hibakomponens tetszőlegesen kicsi lehet. Ugyanakkor, ha ezzel egyidőben a második tag nő, az általánosítóképesség gyenge lesz, a valódi kockázat értéke nagy lesz.

A tanuló rendszer komplexitása függ a háló szabad paramétereinek számától. A statisztikus tanulásmélet eredményei tehát összhangban vannak azzal a megállapítással, hogy a túl sok szabad paraméter a túlilleszkedés lehetősége miatt gyengébb általánosítóképességet eredményezhet, mint ha a szabad paraméterek számát korlátozzuk. Meg kell jegyezni azonban, hogy a statisztikus tanulásmélet a komplexitás mérésére nem a szabad paraméterek számát, hanem a VC-dimenziót használja, ami azt sejteti, hogy a megvalósítható függvényosztály komplexitása nem csak a szabad paraméterek számával befolyásolható. Erről részletesebben a kernel gépekkel – köztük a szupport vektor gépekkel is – foglalkozó 6. fejezetben lesz szó.

A (2.31) egyenlőtlenség $1-\eta$ valószínűséggel érvényes, ahol az $(1-\eta)$ -t konfidenciaszintnek is szokás nevezni. Minél magasabb a konfidenciaszint (minél nagyobb valószínűséggel igaz a felső korlátra vonatkozó egyenlőtlenség), annál nagyobbá válik a konfidencia tag. Kis η mellett a felső korlát túl nagy értéket vesz fel, ami a felső korlát gyakorlati használhatóságát már megkérdőjelezi. Ez különösen kis mintapontszámnál igaz. Ha a mintapontok száma egyre nő, miközben a többi érték változatlan marad, a konfidencia tag értéke csökken, és így a tapasztalati hiba egyre jobb becslése lesz a valódi kockázatnak. Ez szintén összhangban van azzal az intuitív érzésünkkel, hogy minél nagyobb számú mintaponttal tanítunk egy hálót, annál megbízhatóbb az eredmény, és ekkor a mintapontokra vett átlagos hiba – a tapasztalati kockázat – önmagában is egyre inkább alkalmas lehet a háló minősítésére.

Az VC-dimenzióra és így az általánosító-képességre vonatkozó eredmények kiterjeszthetők regressziós feladatokra is. Regressziós esetben is megfogalmazható a valódi kockázat felső korlátja. Egy regressziós feladatnál a valódi kockázatra vonatkozó következő egyenlőtlenség legalább $1-\eta$ valószínűséggel érvényes az összes $L(\mathbf{z}, \mathbf{w})$ függvényre, köztük az empirikus kockázatot minimalizáló $L(\mathbf{z}, \mathbf{w}^* | \mathcal{D})$ -re is:

$$R(\mathbf{w}) \leq \frac{R_{\text{emp}}(\mathbf{w})}{\left(1 - c\sqrt{\varepsilon(h)}\right)_+} \quad (2.33)$$

ahol $\varepsilon(h)$ -t most is (2.32) adja meg és $(a)_+ = \max(a, 0)$. A legtöbb gyakorlati problémánál a (2.33) összefüggésben a $c=1$ választás ajánlott.

3.2. Strukturális kockázat minimalizálás

Az előbbieket alapján nyilvánvaló, hogy a tapasztalati hiba kis értéke nem jelent egyben jó általánosítóképességet is. Az ERM elv konzisztenciájára és a gyors konvergenciára vonatkozó eredmények azt mutatták, hogy ha a mintapontszám nagy, pontosabban ha az $1/h$ arány nagy, akkor (2.31) jobb oldalának második tagja elhanyagolható: a tapasztalati kockázat közel azonos a valódi kockázat értékével. Ha azonban $1/h$ értéke kicsi, akkor nyilvánvaló, hogy a kis empirikus hibából nem következik, hogy a valódi kockázat értéke is kicsi lesz. Megfelelően kis általánosítási hibához ezért a (2.31) egyenlőtlenség jobb oldalának mindkét tagját (illetve regressziós esetben a (2.31) egyenlőtlenség jobb oldalát) minimalizálni kell. Láttuk, hogy a két hibakomponens egymás rovására csökkenthető, hiszen a komplexitás növelésével a tapasztalati hiba csökken, de a második tag nő, míg a komplexitás csökkentése ezzel épp ellentétesen hat: a második tagot tudjuk csökkenteni, de a tapasztalati hibát a kisebb komplexitású tanuló rendszer csak kevésbé tudja minimalizálni.

A komplexitás és a hiba együttes minimalizálásának formális kereteit a strukturális kockázatminimalizálás (*structural risk minimization, SRM*) megközelítés adja meg. Az SRM eljárás pont azt mondja ki, hogy véges (kis) mintaszámnál mindkét hibakomponenst együttesen kell minimalizálni. A módszert eredetileg Vapnik és Cservonyenkisz osztályozási feladatokra javasolták, de bármilyen tanulási feladatnál alkalmazható, amikor a kockázat értékének a minimalizálása a cél [Vap79]. Az SRM eljárás szerint a tanuló rendszer konstrukciójánál

nemcsak a tapasztalati kockázatot, hanem a VC-dimenzió keresztül a modell komplexitását is kézben kell tartani.

Ennek érdekében az SRM egy egymásba ágyazottan növekvő komplexitású modellkészletet használ, azaz egyre komplexebb függvényosztályok elemeivel próbálja a feladatot megoldani. A megoldásnál használt S függvényosztály egymásba ágyazott részosztályokat

$$S_1 \subset S_2 \subset \dots \subset S_k \subset \dots \quad (2.34)$$

tartalmaz. Minden egyes S_k részosztály véges h_k VC-dimenzióval jellemezhető és az egyes részosztályok növekvő VC-dimenzió (növekvő komplexitás) szerint sorbarendezettek:

$$h_1 \leq h_2 \leq \dots \leq h_k \leq \dots \quad (2.35)$$

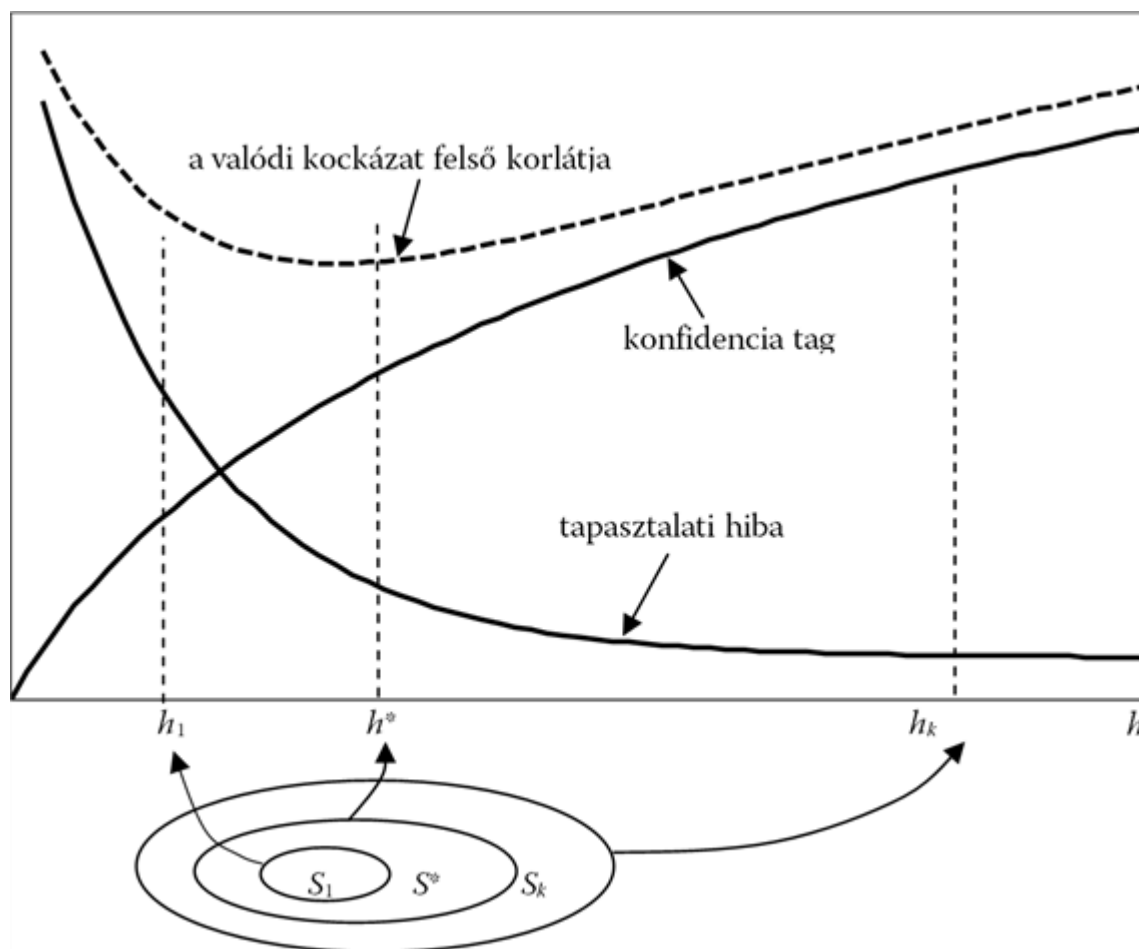
Az egyre növekvő komplexitás hatását a 2.8 ábra mutatja. Az ábrán a kockázat felső korlátja (a garantált kockázat) látható, és az, hogy ez két komponensből tevődik össze: a tapasztalati kockázatból és az ún. konfidencia-tagból. Minél komplexebb rendszert használunk, vagyis minél nagyobb a VC-dimenzió, annál kisebb lesz a tapasztalati kockázat, de ezzel együtt annál nagyobb a konfidencia-tag. Az optimumot h^* mellett érjük el.

Adott mintakészlet mellett az optimális modell konstrukciója ezért az alábbi két lépésből áll: előbb a megfelelő komplexitású modellstruktúrát – a megfelelő S_k részosztályt – választjuk meg, majd ezen belül végezzük el a modell konstrukciót a tanító mintapontok alapján.

A statisztikus tanuláselmélet általánosítási képességeire vonatkozó eredményeinek a jelentősége az, hogy – még ha csak felső korlát formájában is, de – formális kapcsolatot teremt a tanítópontokban számított átlagos kockázat, a tapasztalati kockázat és a tényleges általánosítási hiba, a valódi kockázat között. Az elmélet gyakorlati alkalmazása azonban nehézségekbe ütközik. Egyrészt egy függvényosztály VC-dimenziójának a meghatározása általában igen nehéz. A VC-dimenzió csak néhány egyszerűbb esetben kapható meg könnyen, gyakran pedig csak meglehetősen pontatlan felső becslés adható rá. Másrészt a megfogalmazott felső korlátok általában nem éles korlátok; a gyakorlati feladatoknál az általánosítási hiba jóval kisebb, mint amit a felső korlát megad. Ezen még gyengít, ha a VC-dimenziót is csak becsüljük felülről.

A kockázatra vonatkozó felső korlátok véges VC-dimenzió mellett értékelhetők. Ebből azonban nem következik, hogy nem létezhetnek jó általánosítóképességgel rendelkező olyan tanuló rendszerek, ahol a VC-dimenzió végtelen. Egyes neuronháló típusoknál – ilyen például a leggyakrabban alkalmazott többrétegű perceptron, az MLP is – a VC-dimenzió lehet végtelen, miközben e hálótípus a gyakorlatban igen sikeresen alkalmazható. A végtelen VC-dimenzió csak azt jelenti, hogy a statisztikus tanuláselmélet alapján az általánosítóképességről itt semmit sem tudunk mondani.

2.8. ábra - Az eredő kockázat és a tapasztalati kockázat a VC dimenzió (h) függvényében



Fentiek miatt a korlátokra vonatkozó eredmények neuronhálóknál általánosan nem alkalmazhatók. Hasonló mondható el az SRM megközelítésről is. Ennek ellenére az elmélet elvi jelentőségén túl fontos gyakorlati következményekkel is jár: a tanuló rendszereken belül meghatározó szerepet játszik a döntően a kilencvenes évek derekán kidolgozott szupport vektor gépek elméleti megalapozásában. A szupport vektor gépekről részletesen a 6. fejezetben szólnunk.

4. Tanulás és statisztikai becslések

Amikor neuronhálók tanulásáról beszélünk a tanulás egyik legalapvetőbb, elemi formájáról van szó. Mérési adatokból, megfigyelésekből kell egy megfigyelt rendszerről, jelenségről, folyamatról általános ismereteket nyerni. Láttuk, hogy a tanuló eljárás értelmezhető úgy is, mint egy, a mintapontokkal jellemzett rendszer egyfajta modelljének a létrehozása. A rendszerről egy olyan modellt szeretnénk megalkotni, melynek bemenet-kimenet kapcsolata minél inkább megegyezik a rendszer bemenetei és kimenetei közötti kapcsolattal. Ha a modellezési feladat valójában csak arra irányul, hogy a rendszer által megvalósított leképezést minél pontosabban adjuk meg, akkor ún. fekete doboz modellezési feladatról beszélünk. A fekete doboz modellezésnél nem törekszünk arra, hogy a modell felépítése kövesse a rendszer felépítését, csupán azt célozzuk, hogy kívülről nézve, tehát adott bemenetekre kapott válaszokat tekintve a modell minél inkább úgy viselkedjen, mint a modellezendő rendszer.

A fekete doboz modellezésnél ennek megfelelően nem használunk a rendszer belső felépítését tükröző ismereteket, kizárólag összetartozó bemeneti és kimeneti adatokból, mintapontokból történik a modell létrehozása.

Egy modell konstrukciójánál először meg kell határozni a modell felépítését, struktúráját, majd meg kell adni a modellben megjelenő szabad paraméterek értékeit. A struktúra rögzítése egy modellosztály rögzítését jelenti, mely modellosztályba tartozó konkrét modellek a szabad paraméterek meghatározása útján nyerhetők.

Neuronhálónál a modell struktúráját a háló típusa és mérete határozza meg. Ezek rögzítése után a háló tanítása a szabad paraméterek meghatározását jelenti.

Mint azt az 1. fejezetben láttuk, egyes neuronháló architektúrák univerzális approximátorok, vagyis alkalmasak meglehetősen általános bemenet-kimenet leképezést megadó függvények tetszőleges pontosságú közelítésére. A megfelelő pontosságú approximáció a háló méretének a megválasztásával és a szabad paramétereknek a meghatározásával biztosítható. A háló struktúrájának – típusának és méretének – a meghatározása általában nem része a tanulási folyamatnak, a tanulás a szabad paraméterek meghatározására szolgál. Valójában tehát a tanulás során egy paraméterbecslési feladattal állunk szemben (2.9 ábra).

A paraméterbecslés során mindig valamilyen cél elérése vagy kritérium teljesítése érdekében kívánjuk az „optimális” paraméterértékeket meghatározni. A tanulási eljárashoz ezért elsődlegesen egy célfüggvényt vagy kritériumfüggvényt kell megfogalmaznunk. A kritériumfüggvény a modell minőségének a mérésére szolgál, tehát az előzőekben megfogalmazott kockázat épp ilyen kritériumfüggvény szerepet tölthet be, és a kockázat minimalizálása lehet az eljárás célja. A tanulás tehát egy paraméterbecslési eljárásnak is felfogható, amikor adott modell struktúra mellett az ismeretlen paramétereket a mintapontok alapján egy kritériumfüggvény szélsőértékének (általában minimumának) elérése érdekében határozzuk meg.

A kritériumfüggvény, továbbá a rendelkezésünkre álló egyéb információ alapján különböző paraméterbecslési eljárásokról beszélhetünk. Amennyiben a kívánt és a tényleges válaszok közötti eltérés négyzetét tekintjük hibafüggvénynek (veszteségfüggvénynek) és az ebből származtatott tapasztalati kockázat minimumát biztosító paramétereket szeretnénk megkapni, legkisebb átlagos négyzetes hibájú (LS) becslésről beszélünk.

LS becslésnél a megoldás a

$$C(\mathbf{w}) = \frac{1}{l} \sum_{i=1}^l (d_i - f(\mathbf{w}, \mathbf{x}_i))^2 \quad (2.36)$$

kritériumfüggvény minimumát biztosító paramétervektor (a keresett paramétervektor becslése: $\hat{\mathbf{w}}$). A megoldás tehát

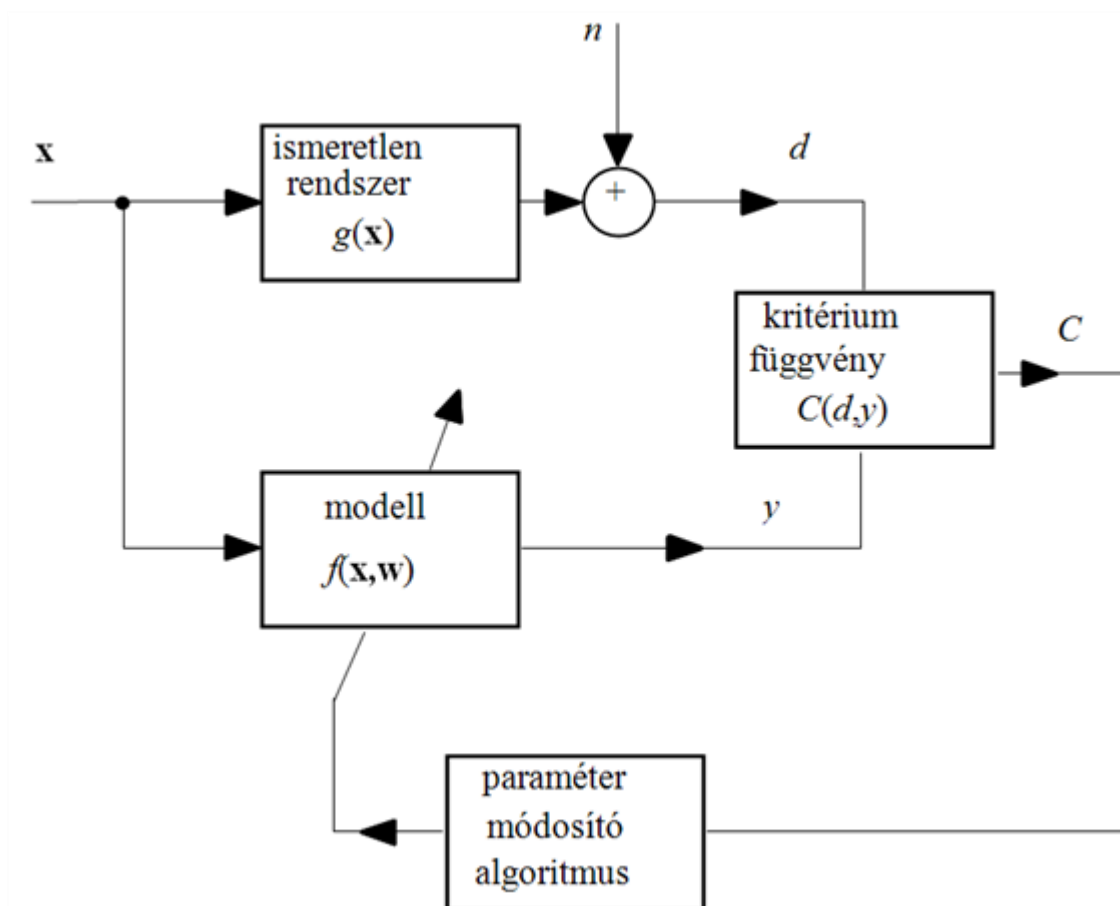
$$\hat{\mathbf{w}}_{LS} = \arg \min_{\mathbf{w}} C(\mathbf{w}) \quad (2.37)$$

Látható, hogy $\hat{\mathbf{w}}_{LS}$ a (2.16) összefüggéssel definiált tapasztalati kockázat minimumát biztosító paramétervektor, ha négyzetes veszteségfüggvényt alkalmazunk. Az LS becslő a megfigyeléseken kívül semmilyen további információt nem használ fel a becslés meghatározásához.

Az adataink, a megfigyelések azonban általában zajosak. Zajos kimenet mellett a bemenet-kimenet leképezést egy $d_i = g(\mathbf{x}_i) + \varepsilon_i$ kapcsolat írja le (ld. 2.9 ábra). Ha az ε megfigyelési zaj statisztikai jellemzése is ismert, a paraméterbecslésnél már valószínűségi megközelítés is alkalmazható. A zaj hatását a $p(d|\mathbf{x})$ feltételes sűrűségfüggvénnyel írhatjuk le. Az adatok alapján a $g(\mathbf{x})$ leképezést kívánjuk becsülni, ahol a becslést a modell $y_i = f(\mathbf{x}_i|\mathbf{w})$ leképezése jelenti. Adott $\mathbf{x}$ mellett a modell y válasza tehát az f függvénytől (és annak $\mathbf{w}$ paramétervektorától) függ. A becslés jóságának mérésére ezért felhasználható a

$$\begin{aligned} & p(\{\mathbf{x}_1, \dots, \mathbf{x}_l\}, \{d_1, \dots, d_l\} | f(\mathbf{x}_i|\mathbf{w})) \\ &= \prod_{i=1}^l p(\mathbf{x}_i, d_i | f(\mathbf{x}_i|\mathbf{w})) \\ &= \prod_{i=1}^l p(d_i | \mathbf{x}_i, f(\mathbf{x}_i|\mathbf{w})) p(\mathbf{x}_i) \end{aligned} \quad (2.38)$$

2.9. ábra - A tanulás, mint paraméterbecslési eljárás



feltételes sűrűségfüggvény, amely megadja, hogy az $(\mathbf{x}_1, d_1), (\mathbf{x}_2, d_2), \dots, (\mathbf{x}_i, d_i)$ tanító mintapontok milyen valószínűséggel lennének kaphatók, feltéve, hogy a közöttük lévő kapcsolatot egy adott $f(\mathbf{x}|\mathbf{w})$ függvény írja le. Mivel a megfelelő f függvény (illetve a $\mathbf{w}$ paramétervektor) meghatározása a cél, és $p(\mathbf{x}_i)$ nem függ f -től, továbbá, ha a bemeneteket egyenletes eloszlással generáljuk, $p(\mathbf{x}_i)$ a (2.38) kifejezés jobb oldalából el is hagyható. Gyakorlati szempontok miatt a szorzat helyett annak negatív logaritmusával érdemes dolgozni. Az így kapott

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^I \ln p(d_i | \mathbf{x}_i, f(\mathbf{x}_i | \mathbf{w})) \quad (2.39)$$

log-likelihood függvény képezi a maximum likelihood (ML) becslés kritériumfüggvényét [Lju99].

A maximum likelihood (ML) becslés olyan paraméterértékeket keres, melyek mellett a rendelkezésre álló megfigyeléseink a legnagyobb valószínűségűek.

$$\hat{\mathbf{w}}_{\text{ML}} = \arg \max_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \quad (2.40)$$

A valószínűség mértékét a paramétervektor függvényében a (log-)likelihood függvény adja meg. A likelihood függvény tehát a megfigyeléseink hihetőségének a mértékeként is értelmezhető. Az ML becslés alap gondolatát illusztrálja a 2.10 ábra.

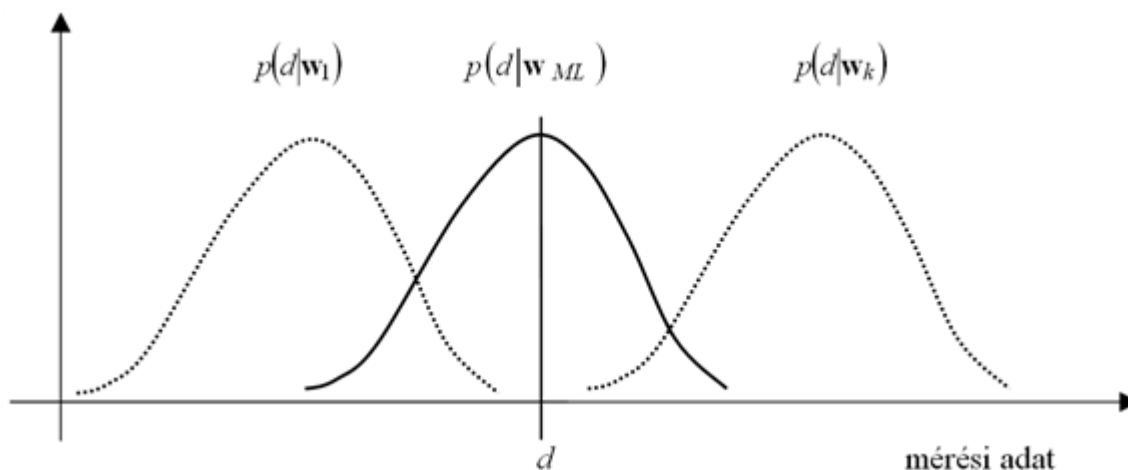
Az ábra azt mutatja, hogy hogyan befolyásolja a paramétervektor megválasztása a megfigyelések eloszlását. Azt a paramétervektort fogadjuk el ML becslésnek, mely mellett az aktuális megfigyelésünk (megfigyeléseink), (az ábrán d) a legnagyobb valószínűségű(ek).

A paramétereink tekinthetők valószínűségi változóknak is. Amennyiben ezen valószínűségi változók eloszlása (sűrűségfüggvénye) ismert, származtatható a Bayes becslés, amely abból indul ki, hogy az ismeretlen

paraméterről van a priori ismeretünk, adott a paraméter ún. a priori eloszlása. Az a priori sűrűségfüggvény azt adja meg, hogy a keresett paraméter a megfigyelésekből származó ismeretek hiányában a paramétertérben milyen értékeket milyen valószínűséggel vehet fel.

A becslési eljárás célja, hogy a paraméterről az ismereteinket pontosítsuk a megfigyelések felhasználásával. Minthogy valószínűségi változóról van szó, a

2.10. ábra - A maximum likelihood becslés



pontosítás a paraméter eloszlásának pontosítását jelenti. A pontosított eloszlás a megfigyelések felhasználása után nyert eloszlás, amit a posteriori eloszlásnak hívnak. Az a priori és az a posteriori eloszlásokat a Bayes szabály kapcsolja össze:

$$p(\mathbf{w} | \text{megfigyelések}) = \frac{p(\text{megfigyelések} | \mathbf{w}) p(\mathbf{w})}{p(\text{megfigyelések})} \quad (2.41)$$

ahol

$p(\mathbf{w})$ a paraméter a priori (a megfigyelések előtti) sűrűségfüggvénye,

$p(\text{megfigyelések})$ a kapott megfigyelések (tanító adatok) sűrűségfüggvénye,

$p(\mathbf{w} | \text{megfigyelések})$ az a posteriori (a megfigyelések által szolgáltatott ismereteket is figyelembe véve) sűrűségfüggvénye, és

$p(\text{megfigyelések} | \mathbf{w})$ egy olyan feltételes sűrűségfüggvény, amely azt jellemzi, hogy az adott megfigyelések milyen eloszlásúak, feltéve, hogy azt a $\mathbf{w}$ paraméterű modell generálta.

A Bayes becslés az a posteriori sűrűségfüggvény meghatározására vezet. Amennyiben az ismeretlen paraméterről hordoznak információt a megfigyelések (a tanító mintapontok), akkor az a posteriori sűrűségfüggvény a konkrét paraméterérték szűkebb környezetére terjed ki (2.11 ábra).

Az a posteriori sűrűségfüggvény felhasználásával felírható a Bayes kockázat:

$$R(\mathbf{w}) = \int_{\mathbf{z}, \mathbf{w}} L(\mathbf{z}, \mathbf{w}) p(\mathbf{w}, \mathbf{z}) d\mathbf{z} d\mathbf{w} \quad (2.42)$$

ahol $p(\mathbf{w}, \mathbf{z})$ a paramétervektor és a megfigyelések együttes sűrűségfüggvénye. A keresett paramétervektor Bayes becslése a Bayes kockázat minimalizálása útján határozható meg:

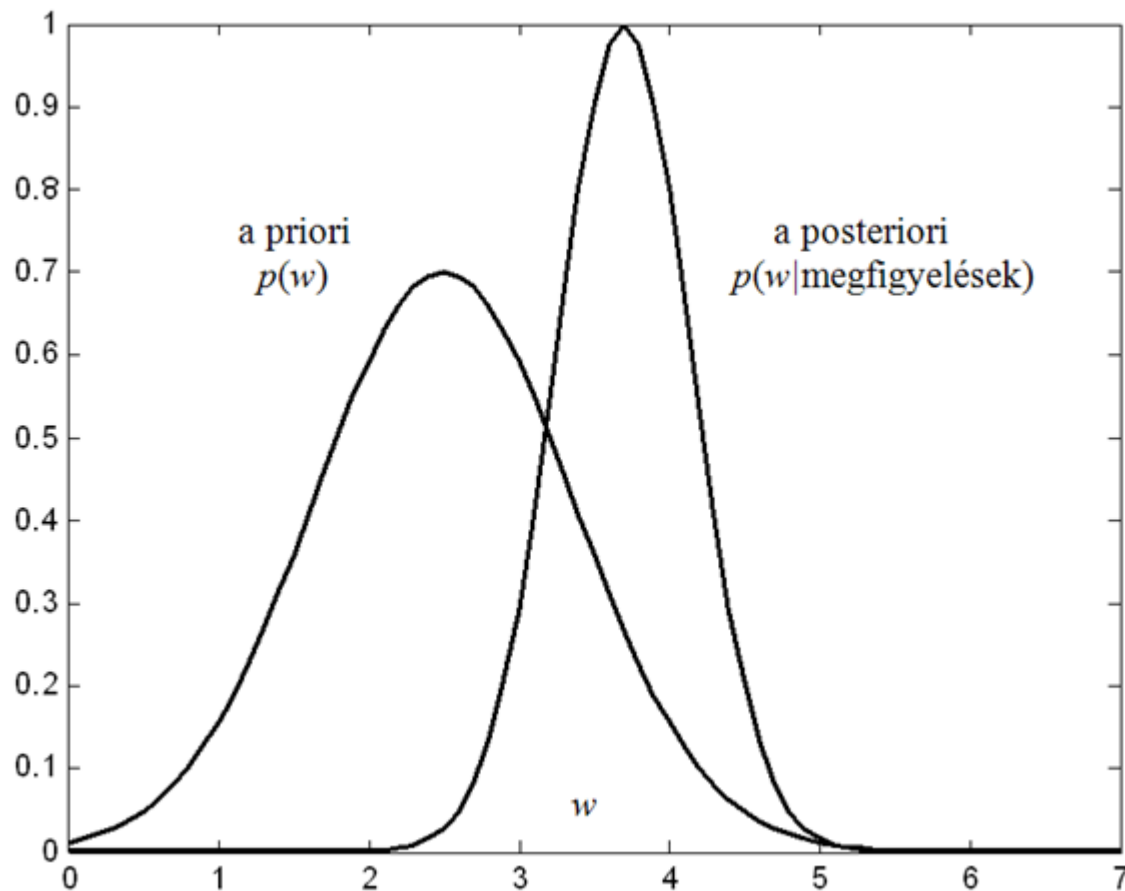
$$\hat{\mathbf{w}}_{\text{BAYES}} = \arg \min_{\mathbf{w}} R(\mathbf{w}) \quad (2.43)$$

Az a posteriori sűrűségfüggvény a megfigyelések figyelembevételével a keresett paramétervektor teljes statisztikai leírását adja. A teljes statisztikai ismeretre azonban nincs mindig szükség (ráadásul az a posteriori sűrűségfüggvény meghatározása általában meglehetősen nehéz is), ezért célszerű, ha a lehetséges értékek közül egyet kiválasztunk, és azt tekintjük a Bayes becslésnek. Leggyakrabban ez az a posteriori sűrűségfüggvény maximumához tartozó paraméterérték, amit ezért maximum a posteriori vagy MAP becslésnek is szokás nevezni. A MAP becslés tehát szintén megfogalmazható szélsőérték-keresési problémaként:

$$\hat{\mathbf{w}}_{\text{MAP}} = \arg \max_{\mathbf{w}} p(\mathbf{w} | \text{megfigyelések}) \quad (2.44)$$

A Bayes becslés az előző becslési eljárásoknál több információt használ fel. A paraméter sűrűségfüggvényét a legtöbb esetben nem ismerjük, így ilyenkor vagy feltételezéssel élünk (pl. Gauss sűrűségfüggvényt tételezünk fel) vagy a Bayes becslést nem alkalmazhatjuk.

2.11. ábra - Az a priori és az a posteriori sűrűségfüggvények alakulása



A neuronhálók nagy többségénél a tanulás LS becslést jelent, hiszen egy négyzetes hibafüggvény minimumát biztosító paraméterértékek meghatározása a cél. Amennyiben a hálózhoz, illetve az általa megvalósított leképezéshez valószínűségi modell is rendelhető, maximum likelihood vagy Bayes becslésként értelmezhető a tanulási folyamat. A valószínűségi megközelítések azzal az előnnyel járnak, hogy az eredmény optimalitásáról határozottabb állítások fogalmazhatók meg, illetve eredményként nem csupán a paraméterek értékét kapjuk meg, hanem ezen értékekhez egy konfidenciaintervallum is rendelhető, így valójában az eredményeknek valamilyen minősítése is megtörténik.

5. Determinisztikus és sztochasztikus szélsőérték-kereső eljárások

Az eddigiekben láttuk, hogy a tanulás során egy kritériumfüggvény vagy a kockázat funkcionál szélsőértékét keressük, vagyis azt a $\mathbf{w}$ értéket, ahol a kritériumfüggvény extrémális értéket vesz fel. Szemléletesen a

kritériumfüggvény a paraméterter felett egy felülettel jellemezhető – ezt szokás kritériumfelületnek vagy hibafelületnek is nevezni –, és ennek a felületnek a minimumához vagy maximumához tartozó paraméterérték meghatározása a cél. A tanuló eljárások tehát valójában szélsőérték-kereső eljárások.

A szélsőérték-kereső eljárásoknak egész tárháza áll rendelkezésre, és ezen eljárások szinte mindegyikét alkalmazzák is a neuronhálónál. Adott neuronháló tanításánál az eljárások széles választékából a megfelelő eljárás kiválasztása számos körülménytől, bizonyos feltételek teljesülésétől vagy hiányától függ. Az egyes eljárások alkalmazhatóságát döntően befolyásolja a kritériumfüggvény jellege. A kritériumfüggvény jellegét egyrészt a veszteségfüggvény, másrészt a tanuló rendszer leképezésének a keresett paraméterektől való függése határozza meg. Kiemelten fontos, ha ez utóbbi kapcsolat lineáris. Ebben az esetben a leggyakrabban alkalmazott négyzetes veszteségfüggvény mellett a kritériumfüggvény a keresett paraméter kvadratikusan függvénye, ami – megfelelő feltételek fennállta esetén – biztosítja az egyértelmű szélsőérték létezését.

A továbbiakban az egyszerűbb tárgyalás lehetőségének kedvéért általában minimumkeresésről fogunk beszélni, de az eredmények értelemszerűen érvényesíthetők maximumkeresés estére is.

Ha a kritériumfüggvény a keresett paraméter folytonos és differenciálható függvénye, a minimumkeresés során azt a $\mathbf{w}^*$ értéket keressük, ahol a kritériumfüggvény $\mathbf{w}$ szerinti gradiense zérus. A gradiens alapú szélsőérték-kereső eljárásoknak kiemelten fontos szerepük van a neuronhálók tanításánál. Ha gradiens módszerek nem alkalmazhatók, vagy a gradiens analitikus vagy numerikus meghatározása túl bonyolult, kereső eljárásokkal lehet a minimumhelyet meghatározni. A kereső eljárások általában nem tételeznek fel semmit a kritériumfüggvényről. A keresés során valamilyen véletlen vagy determinisztikus algoritmus szerint feltérképezik a $\mathbf{W}$ paraméterteret, miközben kiértékelik a hibafelületet. A megoldást a megfelelő szélsőértékhez tartozó $\mathbf{w}^*$ jelenti.

5.1. Gradiens alapú szélsőérték-kereső eljárások

Gradiens alapú eljárásoknál a

$$\nabla [C(\mathbf{w})] = \frac{\partial C(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{0} \quad (2.45)$$

feltételt biztosító paraméter meghatározása a cél. A $C(\mathbf{w})$ kritériumfüggvény ismeretében a $\nabla [C(\mathbf{w})] = \mathbf{0}$ megoldása a keresendő paraméterre nézve explicit kifejezést eredményezhet. Ekkor analitikus eredményt kapunk.

Az analitikus kifejezés meghatározása azonban bonyolultabb esetekben, különösen, ha a kritériumfüggvény a keresett paraméter nemlineáris függvénye, komoly matematikai nehézségeket okozhat. Sok esetben a kritériumfüggvényre analitikus kifejezés nem is adható. Ezekben az esetekben iteratív eljárások alkalmazhatók, ahol a tanuló rendszer paramétereit valamilyen algoritmus szerint addig változtatjuk, amíg a kritériumfüggvény minimumát el nem érjük vagy kellő mértékben meg nem közelítjük; vagyis itt a $\nabla [C(\mathbf{w})] \rightarrow \mathbf{0}$ összefüggésnek megfelelően jutunk a megoldáshoz.

A tanuló eljárások konstrukciója döntően ilyen algoritmusok kidolgozását jelenti, ahol az iteratív eljárás során a mintapontok nyújtotta információt fokozatosan, sok esetben a mintapontok többszöri, ismételt alkalmazása útján használjuk fel. Az iteratív szélsőérték-kereső eljárásoknak is igen széles skáláját dolgozták ki, igazodva a különféle feltételekhez és az eltérő kritériumfüggvényekhez. Ezek nagy részét neurális hálózatoknál is alkalmazni szokták. Leggyakrabban azonban azokat az eljárásokat alkalmazzák, ahol az egy iterációhoz szükséges műveletek száma viszonylag kicsi, és ahol a műveletek egyszerű számításokat igényelnek. A számítások egyszerűségén kívül további követelményeket is ki kell elégíteniük az eljárásoknak.

A fontosabb követelmények:

- legyen az eljárás konvergens,
- a konvergencia sebessége legyen minél nagyobb,
- az eljárás a kritériumfüggvények által meghatározott hibafelületek széles körére, vagyis a feladatok széles osztályára legyen alkalmazható.

A követelmények kielégítése neurális hálózatoknál különösen nehéz, hiszen a szélsőérték-kereső eljárások többségénél a konvergencia csak kvadratikusan vagy konvex kritérium-felület mellett bizonyítható és neurális hálózatoknál ezt a feltételt a kritériumfelületek csak kivételes esetben teljesítik.

A neuronhálók döntő többsége nemlineáris bemenet-kimenet leképezést valósít meg. A nemlineáris leképezés a háló paramétereinek szerepét tekintve többféleképpen megvalósítható: a leképezés lehet paramétereiben lineáris vagy nemlineáris. Az előbbi esetben a kapcsolatot

$$y = \mathbf{w}^T \Phi(\mathbf{x}), \quad (2.46)$$

alakban, az utóbbiban

$$y = \psi(\mathbf{w}, \mathbf{x}) \quad (2.47)$$

formában írhatjuk le, ahol mind $\Phi(\cdot)$, mind $\psi(\cdot)$ valamilyen nemlineáris függvényt jelöl.

5.1.1. Szélsőérték-keresés paramétereiben lineáris modellek esetén

A paramétereiben lineáris modellek a neuronhálók témakörén belül is kitüntetett fontosságúak. Ekkor, továbbá, ha négyzetes hibafüggvényt alkalmazunk, kvadratikusan hibafelületet kapunk, amelynek a minimumpontjához tartozó paramétervektor meghatározása a cél.

A négyzetes hibafelületre vonatkozó eljárásokat és eredményeket az

$$y = \mathbf{w}^T \mathbf{x} \quad (2.48)$$

lineáris kapcsolat feltételezésével fogjuk megvizsgálni, de az eredmények értelemszerűen alkalmazhatók a (2.46)-ben megfogalmazott általánosabb kapcsolat esetére is.

Ha kritériumként az átlagos négyzetes hibát tekintjük, a minimalizálandó összefüggés:

$$\begin{aligned} C(\mathbf{w}) &= E \left\{ (d - \mathbf{w}^T \mathbf{x})^2 \right\} \\ &= E \{ d^2 \} - 2E \{ d \mathbf{x}^T \} \mathbf{w} + \mathbf{w}^T E \{ \mathbf{x} \mathbf{x}^T \} \mathbf{w} \\ &= E \{ d^2 \} - 2\mathbf{p}^T \mathbf{w} + \mathbf{w}^T \mathbf{R} \mathbf{w}, \end{aligned} \quad (2.49)$$

ahol $\mathbf{R} = E \{ \mathbf{x} \mathbf{x}^T \}$ a bemenet autokorrelációs mátrixa, $\mathbf{p} = E \{ d \mathbf{x} \}$ pedig egy olyan oszlopvektort jelöl, amely a kívánt kimenet és a bemenet közötti keresztkorrelációt mutatja.

A szélsőérték a $C(\mathbf{w})$ négyzetes hiba $\mathbf{w}$ szerinti

$$\nabla C(\mathbf{w}) = \frac{\partial C(\mathbf{w})}{\partial \mathbf{w}} = 2(\mathbf{R} \mathbf{w} - \mathbf{p}) \quad (2.50)$$

parciális deriváltjának nulla értékéhez tartozik, amelyből – amennyiben $\mathbf{R}^{-1}$ létezik – az optimális paraméterkészlet a Wiener-Hopf egyenlet szerint határozható meg:

$$\mathbf{w}^* = \mathbf{R}^{-1} \mathbf{p}. \quad (2.51)$$

A (2.51) összefüggés szerint az átlagos négyzetes hiba minimumát biztosító megoldásvektor analitikusan is meghatározható, ha a d_i kívánt válaszok és az $\mathbf{x}_i$ bemenetek statisztikai jellemzői, $\mathbf{R}$ és $\mathbf{p}$ ismertek, ami a gyakorlatban a legritkább esetben teljesül. Ha csak a mintapontok állnak rendelkezésünkre, az analitikus összefüggés alkalmazásához előbb $\mathbf{R}$ -et és $\mathbf{p}$ -t a véges számú mintapont alapján becsülnünk kell. Ennek, továbbá a (2.51) összefüggés szerinti mátrix invertálásnak az elkerülése érdekében a továbbiakban olyan módszereket fogunk vizsgálni, amelyek vagy eleve kevesebb ismeret meglétét igénylik, vagy amelyek a tanítópontokból iteratív úton igyekeznek minél több ismeretet összegyűjteni (pontosítani).

Mielőtt azonban ezekre rátérnénk, a [Wid85] irodalmat követve bemutatjuk a fentiekben vázolt, ideálisnak tekinthető megoldás néhány lényeges tulajdonságát.

Először a (2.51) összefüggést a (2.49) hibakifejezésbe helyettesítve megkapjuk az elérhető négyzetes hiba minimumát:

$$C_{\min} = E\{d^2\} - \mathbf{p}^T \mathbf{R}^{-1} \mathbf{p} = E\{d^2\} - \mathbf{p}^T \mathbf{w}^* \quad (2.52)$$

Sokszor előnyös, ha egy paraméter tényleges értéke helyett az ideális paraméter-értéktől való eltéréssel, azaz a paraméter-hibával dolgozunk. Ezért alakítsuk át ennek megfelelően a (2.49) összefüggést (2.52) felhasználásával. A hiba

$$C(\mathbf{w}) = C_{\min} + (\mathbf{w} - \mathbf{w}^*)^T \mathbf{R} (\mathbf{w} - \mathbf{w}^*) = C_{\min} + \mathbf{v}^T \mathbf{R} \mathbf{v} \quad (2.53)$$

alakba írható, ahol $\mathbf{v} = \mathbf{w} - \mathbf{w}^*$. A (2.53) összefüggés jól mutatja, hogy a kritériumfüggvény a $\mathbf{W}$ paraméterter felett egy kvadratikus hibafelülettel jellemezhető, amelynek – ha $\mathbf{R}$ pozitív definit – a $\mathbf{w} = \mathbf{w}^*$ mellett minimumpontja van.

A (2.49) és a (2.53) összefüggésekből kiindulva ebben a formába is felírható a gradiens vektor:

$$\nabla C(\mathbf{v}) = \frac{\partial C(\mathbf{v})}{\partial \mathbf{v}} = 2\mathbf{R}\mathbf{v} = 2\mathbf{R}(\mathbf{w} - \mathbf{w}^*) = 2(\mathbf{R}\mathbf{w} - \mathbf{p}) \quad (2.54)$$

5.1.2. A négyzetes kritériumfüggvény tulajdonságai

Az átlagos négyzetes hiba (2.53) szerinti összefüggése tömör formában jelzi, hogy a kritériumfüggvény paraboloid hibafelületet ír le, melynek jellemzőit az $\mathbf{R}$ mátrix határozza meg. Ebből egyértelműen arra a következtetésre juthatunk, hogy a (2.49) összefüggéssel definiált hiba a négyzetes jelleg megtartása mellett csak a bemeneti jeltől, pontosabban annak korrelációs mátrixától függ. A későbbiekben látni fogjuk, hogy a szélsőérték-kereső eljárások hatékonyságát befolyásolják a hibafelület tulajdonságai – a hibafelület alakja –, ezért ezzel a kérdéssel most kiemelten foglalkozunk.

Elsőként – kétdimenziós paraméterter feltételezése mellett – gondolatban rajzoljuk fel a paraméterek síkján a (2.53) összefüggés szerinti hibafelület "szintvonalait". A szintvonalak – az ún. izokritérium görbék – egy görbesereget alkotnak. A görbesereg egy eleme – egy izokritérium görbe – azon pontok mértani helyét adja meg, amelyekhez (egy adott) állandó kritériumfüggvény-érték tartozik. Minden ilyen görbe egy-egy ellipszist jelöl ki, amely a síkban általános helyzetben (de egymáshoz képest kötötten), helyezkedik el a sík koordináta tengelyeihez képest. Egy ilyen ellipszis előállítható úgy is, hogy a paraboloid hibafelületet egy, a paraméterek síkjával párhuzamos, ettől a síktól a rögzített négyzetes hiba értéknek megfelelő távolságra lévő síkkal el metszünk. Különböző hibaértékekhez különböző, de koncentrikus, azaz olyan ellipsziseket kapunk, melyek főtengely irányai egybeesnek (ld. 2.12 ábra). Ez a geometriai kép tetszőleges dimenzióra általánosítható. Könnyen belátható, hogy a metszék-ellipszisek (ill. ellipszoidok) főtengelyei kitüntetett irányok. A hibafelület ezen irányokban található bármely pontjánál ugyanis a gradiens irányok egybeesnek a főtengely irányokkal, azaz mindig a középpont felé mutatnak. Így ezen pontokból kiindulva a gradiensek mentén közvetlenül a középpont felé mozoghatunk, illetve ennek megfelelően a hibafelületen a minimumhely felé "ereszkedhetünk". A mértani helyet tovább vizsgálva arra az érdekes következtetésre juthatunk, hogy a főtengely irányokat az $\mathbf{R}$ mátrix sajátvektorai jelölik ki, tehát a későbbiekben jól felhasználható eredményre vezet az $\mathbf{R}$ mátrix sajátérték/sajátvektor rendszere. Ha feltesszük, hogy a sajátértékek multiplicitása egy és az $\mathbf{R}$ mátrix egy $N \times N$ -es mátrix (N a bemeneti vektorok dimenziója), akkor $\mathbf{R}$ felírható az alábbi formában

$$\mathbf{R} = \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^{-1}, \quad (2.55)$$

ahol $\mathbf{\Lambda} = \text{diag}\{\lambda_0, \lambda_1, \dots, \lambda_{N-1}\}$ a sajátértékekből képezett diagonál mátrix, $\mathbf{Q}$ pedig a sajátvektorok mátrixa. Ha a sajátvektorokat egységnyi hosszúságúra normáljuk, és felhasználjuk azt a tulajdonságot, hogy a szimmetrikus négyzetes mátrixok sajátvektorai ortogonálisak (vagyis $\mathbf{Q}$ ortonormált), akkor:

$$\mathbf{Q}^{-1} = \mathbf{Q}^T, \quad \text{azaz} \quad \mathbf{R} = \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^T. \quad (2.56)$$

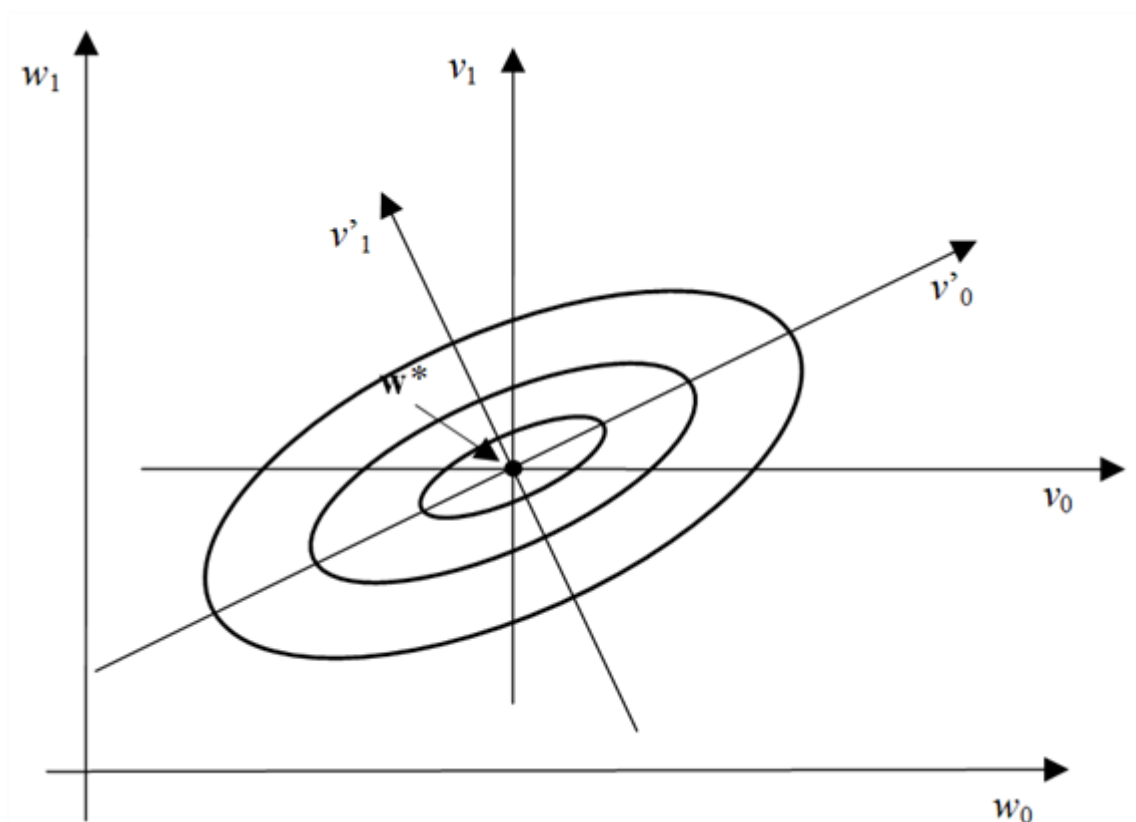
Behelyettesítve a (2.56) kifejezést a (2.53) összefüggésbe, a hibára

$$\begin{aligned} C(\mathbf{v}) &= C_{\min} + \mathbf{v}^T \mathbf{R} \mathbf{v} = C_{\min} \mathbf{v}^T (\mathbf{Q} \Lambda \mathbf{Q}^T) \mathbf{v} = C_{\min} + (\mathbf{Q}^T \mathbf{v})^T \Lambda (\mathbf{Q}^T \mathbf{v}) = \\ &= C_{\min} + \mathbf{v}'^T \Lambda \mathbf{v}' \end{aligned} \quad (2.57)$$

adódik. Az új kifejezésben $\mathbf{v}'$ -t egy forgatási transzformációval kaphatjuk meg $\mathbf{v}$ -ből, amit értelmezhetünk úgy, hogy a hibafelület "alatt" elforgattuk a paraméterek koordináta-rendszerét olyan mértékben, hogy a koordinátatengelyek éppen főtengely irányúak legyenek. Ebben a koordináta-rendszerben a (2.54) összefüggésnek megfelelő gradiens vektor a következőre adódik:

$$\nabla C(\mathbf{v}') = \frac{\partial C(\mathbf{v}')}{\partial \mathbf{v}'} = 2\Lambda \mathbf{v}' = 2 \begin{bmatrix} \lambda_0 v'_0 & \lambda_1 v'_1 & \dots & \lambda_{N-1} v'_{N-1} \end{bmatrix} \quad (2.58)$$

2.12. ábra - Az izokritérium görbék az eredeti és a sajátvektor irányú koordináta-rendszerekkel



A gradiens komponensei tehát a megfelelő sajátértékek és a paraméterhiba-komponensek szorzataként számíthatók. A sajátértékekhez szemléletes jelentés is rendelhető, ugyanis a főtengely-irányokban számított második derivált komponensek az $\mathbf{R}$ mátrix sajátértékeinek kétszeresei lesznek:

$$\frac{\partial^2 C(\mathbf{v}')}{\partial v_i'^2} = 2\lambda_i \quad (2.59)$$

Mindezekből képet alkothatunk a paraboloid hibafelület geometriai tulajdonságairól. Ha a bemeneti jel autokorrelációs mátrixának sajátértékei jelentősen eltérnek egymástól, akkor a paraboloid hibafelület a különböző főtengely irányokban is jelentősen eltérő meredekségű lesz. Ez a tulajdonság – mint látni fogjuk – általában kedvezőtlen az ún. gradiens-alapú szélsőérték-kereső eljárások szempontjából. A 2.12 ábra nemcsak a hibafelületet jellemző izokritérium görbéket, hanem a $\mathbf{w}$, $\mathbf{v}$ és $\mathbf{v}'$ koordináta-rendszereket is mutatja.

5.1.3. Szélsőérték-keresés Newton módszerrel.

Ha a gradiens

$$\nabla C(\mathbf{w}) = 2\mathbf{R}(\mathbf{w} - \mathbf{w}^*) \quad (2.60)$$

szerinti felírását balról megszorozzuk $\frac{1}{2}\mathbf{R}^{-1}$ -gyel, akkor átrendezés után a

$$\mathbf{w}^* = \mathbf{w} - \frac{1}{2}\mathbf{R}^{-1}\nabla C(\mathbf{w}) \quad (2.61)$$

alakot kapjuk, amely négyzetes hibafelület esetén tetszőleges kiinduló $\mathbf{w}$ mellett közvetlenül megadja az optimális megoldást.

Ha a hibafelület nem négyzetes, a (2.61) összefüggés akkor is alkalmas kiindulás egy iteratív eljárásra. (Nem kvadrátikus függvényeknél tekinthetjük úgy, mintha ezek másodfokú Taylor-sorával dolgoznánk, annál is inkább, mivel a megoldás – a minimumhely – közelében a nem másodfokú függvények jó közelítéssel úgy viselkednek, mint a négyzetes függvények.) Az iteratív eljárásoknál lépésenként módosítjuk a megoldást: az épp érvényes megoldásból kiindulva, annak korrekciójával kapjuk a következő lépés érvényes megoldását.

A Newton eljárás iteratív összefüggése, ha az egyes iterációs lépések indexét k -val jelöljük

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu \mathbf{R}^{-1} \nabla C(k) \quad (2.62)$$

ahol $\nabla C(k) = \nabla C(\mathbf{w}(k))$ a gradiensvektort jelöli a k -adik lépésben érvényes paramétervektor mellett, míg $0 < \mu < 1$ az ún. **tanulási tényező** (*learning rate*), amelyet négyzetes esetben az egy lépéses konvergenciához $1/2$ -re kell választani. Megjegyezzük, hogy a μ tanulási tényezőt bátorsági tényezőnek is szokás nevezni, mivel értéke befolyásolja, hogy adott gradiens irányában mekkora lépésekben haladunk. Ha az utóbbi összefüggésben visszaírjuk a derivált (2.60) szerinti alakját, és mindkét oldalból levonjuk $\mathbf{w}^*$ -ot, akkor a

$$\mathbf{w}(k+1) - \mathbf{w}^* = (1 - 2\mu)(\mathbf{w}(k) - \mathbf{w}^*) = (1 - 2\mu)^{k+1}(\mathbf{w}(0) - \mathbf{w}^*), \quad (2.63)$$

összefüggést, illetve az ennek megfelelő

$$\mathbf{v}(k+1) = (1 - 2\mu)^{k+1} \mathbf{v}(0) \quad (2.64)$$

iterációs kapcsolatot kapjuk, amely konvergens, ha $0 < \mu < 1$, és négyzetes hibafelületnél egy lépésben optimumot ad, ha $\mu = 1/2$.

5.1.4. Szélsőérték-keresés a "legmeredekebb lejtő" (steepest descent) módszerével

Ez a módszer a negatív gradiens irányában "ereszkedik" a hibafelületen. Iteratív összefüggése:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu (-\nabla C(k)) \quad (2.65)$$

A (2.65) összefüggésben a fentiekhez hasonlóan visszaírva a gradiens (2.60) szerinti alakját, és ismét mindkét oldalból levonva $\mathbf{w}^*$ -ot, a

$$\mathbf{w}(k+1) - \mathbf{w}^* = (\mathbf{I} - 2\mu\mathbf{R})(\mathbf{w}(k) - \mathbf{w}^*) = (\mathbf{I} - 2\mu\mathbf{R})^{k+1}(\mathbf{w}(0) - \mathbf{w}^*), \quad (2.66)$$

illetve az ennek megfelelő

$$\mathbf{v}(k+1) = (\mathbf{I} - 2\mu\mathbf{R})^{k+1} \mathbf{v}(0) \quad (2.67)$$

összefüggést kapjuk. Ez utóbbi kifejezésből látható, hogy ennél az eljárásnál a μ tanulási tényező megválasztása függ $\mathbf{R}$ -től. Ha áttérünk a főtengety-irányú koordinátákra, mint ahogy azt korábban a (2.57) összefüggésben tettük, akkor (2.67) helyére a

$$\mathbf{v}'(k+1) = (\mathbf{I} - 2\mu\mathbf{\Lambda})^{k+1}\mathbf{v}'(0) \quad (2.68)$$

összefüggés írható, amely lehetővé teszi, hogy a paraméter-hibákat komponensenként vizsgáljuk. A (2.68) összefüggésből jól látható, hogy a konvergencia feltétele ekkor:

$$0 < \mu < \frac{1}{\lambda_{\max}}, \quad (2.69)$$

ahol $\lambda_{\max}$ $\mathbf{R}$ legnagyobb sajátértéke.

A (2.62), és (2.65) összefüggések összevetéséből látszik, hogy a Newton módszer szerinti iterációban elkerülhetetlen $\mathbf{R}$ ismerete, míg a gradiens menti iteráció $\mathbf{R}$ ismeretét közvetlenül nem igényli. A μ tanulási tényező helyes megválasztásához azonban mégis kell valamennyi ismeret a bemeneti jelről. Sokszor megfelelő a

$$\lambda_{\max} \leq \sum_{n=1}^N \lambda_n = \text{tr}(\mathbf{\Lambda}) = \text{tr}(\mathbf{R}) \quad (2.70)$$

összefüggés használata, amellyel a

$$0 < \mu < \frac{1}{\text{tr}(\mathbf{R})} \quad (2.71)$$

előírás adódik, ahol $\text{tr}(\mathbf{R})$, az autokorrelációs mátrix nyoma, amelyet a mintapontokból az alábbi módon becsülhetünk:

$$\text{tr}(\mathbf{R}) \approx \sum_{j=1}^N \frac{1}{l} \sum_{i=1}^l x_{ji}^2. \quad (2.72)$$

Itt x_{ji} az i -edik bemeneti vektor j -edik komponense ($i=1, 2, \dots, l$; $j=1, 2, \dots, N$). A tanulási tényező megválasztása a szélsőérték-keresés igen kritikus lépése, mert közvetlenül befolyásolja az eljárás stabilitását, ill. a konvergencia sebességét.

A (2.69) összefüggésből látható, hogy a konvergencia sebessége a bemeneti jel korrelációs mátrixától, pontosabban annak sajátértékeitől függ. Az is látható, hogy az eljárás konvergenciájának biztosításához a tanulási tényezőre felső korlát adható. A felső korlát azonban valóban csak a konvergencia tényét garantálja, miközben az eljárás gyakorlati alkalmazása szempontjából a konvergencia sebessége is fontos.

Az előzőekben láttuk, hogy a λ_i sajátértékekhez szemléletes jelentés is rendelhető. A paraboloid hibafelület főtengety irányait $\mathbf{R}$ sajátvektorai adják meg. Az i -edik sajátvektor, $\mathbf{q}_i$ irányában a hibafelület második deriváltja viszont arányos a sajátvektorhoz tartozó λ_i sajátértékkel. Minél nagyobb egy sajátérték, annál nagyobb a hibafelület adott irány menti meredekség-változása, azaz a görbülete. A konvergenciára vonatkozó (2.69) összefüggés a konvergencia tényét még a hibafelület legmeredekebb iránya mentén is garantálja, viszont más irányok mentén még nagyobb μ mellett is garantált lenne a konvergencia. A főtengety irányok mentén az adott sajátvektorhoz tartozó sajátérték reciproka jelenti a konvergenciát még biztosító tanulási tényező felső korlátját.

A (2.69) által megfogalmazott korlát betartása azért szükséges, mert általában nem ismerjük, hogy a hibafelület mely pontjából indulunk ki, így a konvergencia csak akkor garantálható, ha az a kiindulási ponttól függetlenül minden körülmény mellett fennáll. A hibafelület kevésbé meredek részein a túl kis tanulási tényező ugyanakkor a lehetségesnél kisebb konvergencia-sebesség elérését eredményezi. A konvergencia-sebesség szempontjából tehát az a kedvező, ha a sajátértékek egyformák vagy közel azonosak, vagyis ha $\mathbf{\Lambda} \approx \mathbf{I}$. Különösen kedvezőtlen esettel állunk szemben, ha a legkisebb és a legnagyobb sajátértékek aránya, $\lambda_{\max} / \lambda_{\min} \gg 1$. Ez egy keskeny hibafelületet jelent. Keskeny hibafelületnél kedvezőtlen kiinduló pont mellett a konvergencia nagyon lassú is lehet.

Felmerül a kérdés, hogy ilyen esetekben lehet-e valami olyan módosítást tenni, ami a konvergencia-sebesség növelését eredményezi. A válasz pozitív. Ha a bemeneteket olyan módon transzformáljuk, hogy a transzformált bemenetek autokorrelációs mátrixa közel egységmátrix, a sajátértékek közel egyformák lesznek, így kiindulóponttól függetlenül a legkedvezőbb tanulási tényező választható. Ennél a megoldásnál a legmeredekebb lejtő gradiens eljárást a „transzformált” térben végezzük. A módszer részletesebb bemutatását a későbbiekben a transzformált tartománybeli LMS eljárásnál adjuk meg.

5.1.5. A konjugált gradiensek módszere

A (2.51) összefüggés a $\mathbf{w}$ paraméterekre nézve egy lineáris egyenletrendszer megoldása, amelyhez az $\mathbf{R}$ mátrix inverze szükséges. Az

$$\mathbf{R}\mathbf{w}^* = \mathbf{p} \quad (2.73)$$

egyenlet megoldása egyszerűsíthető az ún. konjugált irányok ismeretében. A konjugált irányokat azok a $\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_{N-1}$ vektorok jelölik ki, amelyekre fennáll, hogy

$$\mathbf{q}_i^T \mathbf{R} \mathbf{q}_j = 0, \quad i \neq j. \quad (2.74)$$

Könnyen belátható, hogy a $\{\mathbf{q}_i\}$ vektorok lineárisan függetlenek és egy olyan bázist alkotnak, amely kifeszíti a $\{\mathbf{w}(k)\}$ paraméterek terét, így segítségükkel előállítható a paraméterek terének az a vektora is, amely a $\mathbf{w}_0$ vektorból elvezet a $\mathbf{w}^*$ vektorba:

$$\mathbf{w}^* - \mathbf{w}_0 = \sum_{j=0}^{N-1} \alpha_j \mathbf{q}_j. \quad (2.75)$$

Itt $\{\alpha_j\}$ a lineáris kombináció súlytényező készletét jelöli. Ha a (2.75) összefüggés mindkét oldalát megszorozzuk balról $\mathbf{q}_j^T \mathbf{R}$ -rel, és $\mathbf{R}\mathbf{w}^*$ helyére behelyettesítjük $\mathbf{p}$ -t ((2.73) alapján), akkor a konjugált irányok (2.74) feltétele miatt az α_j -re nézve N -ismeretlenes lineáris (2.75) egyenletrendszer "szétesik", és α_j -k közvetlenül megadhatók lesznek (és egyben nyilvánvalóvá válik a konjugált $\mathbf{q}$ vektorok használatának az előnye is):

$$\alpha_j = \frac{\mathbf{q}_j^T (\mathbf{p} - \mathbf{R}\mathbf{w}(0))}{\mathbf{q}_j^T \mathbf{R} \mathbf{q}_j}. \quad (2.76)$$

Ezek után definiálhatunk egy iteratív eljárást $\mathbf{w}(k+1)$ számítására:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha_k \mathbf{q}_k, \quad (2.77)$$

ahol

$$\mathbf{w}(k) = \mathbf{w}(0) + \sum_{j=0}^{k-1} \alpha_j \mathbf{q}_j, \quad \text{és} \quad \mathbf{w}(N) = \mathbf{w}^*, \quad (2.78)$$

tehát N lépésben elérjük az optimumot.

A (2.77) összefüggésben szereplő α_k tényező egy alternatív kifejezése nyerhető, ha felhasználjuk a (2.74) és (2.78) összefüggéseket annak igazolására, hogy

$$\mathbf{q}_k^T \mathbf{R} \mathbf{w}(k) = \mathbf{q}_k^T \mathbf{R} \mathbf{w}(0). \quad (2.79)$$

Ekkor ugyanis a (2.76) összefüggésben $\mathbf{w}(0)$ $\mathbf{w}(j)$ -re cserélhető, és így (2.60) felhasználásával

$$\alpha_j = -\frac{1}{2} \frac{\mathbf{q}_j^T \nabla C(j)}{\mathbf{q}_j^T \mathbf{R} \mathbf{q}_j} \quad (2.80)$$

adódik.

A $\{\mathbf{q}_i\}$ konjugált gradiens irányok meghatározása iteratíván történik úgy, hogy a kezdőérték $\mathbf{q}_0 = -\nabla C(0)$, azaz a kezdőpontban érvényes negatív gradiens adja a legelső irányt, majd a következő irányok rendre az aktuális gradiens és a megelőző irány lineáris kombinációjaként kerülnek kiszámításra:

$$\mathbf{q}_{k+1} = -\nabla C(k+1) + \beta_k \mathbf{q}_k, \quad (2.81)$$

Ha ezt a kifejezést jobbról megszorozzuk $\mathbf{R} \mathbf{q}_k$ -val, megkapjuk β_k ahhoz szükséges új értékét, hogy az új irány konjugált irány legyen:

$$\beta_k = \frac{\nabla C^T(k+1) \mathbf{R} \mathbf{q}_k}{\mathbf{q}_k^T \mathbf{R} \mathbf{q}_k}. \quad (2.82)$$

Mivel (2.60) és (2.77) felhasználásával

$$\nabla C(k+1) - \nabla C(k) = \alpha_k \mathbf{R} \mathbf{q}_k, \quad (2.83)$$

a (2.82) összefüggésből $\mathbf{R} \mathbf{q}_k$ kiejthető és így β_k -ra a következő kapható

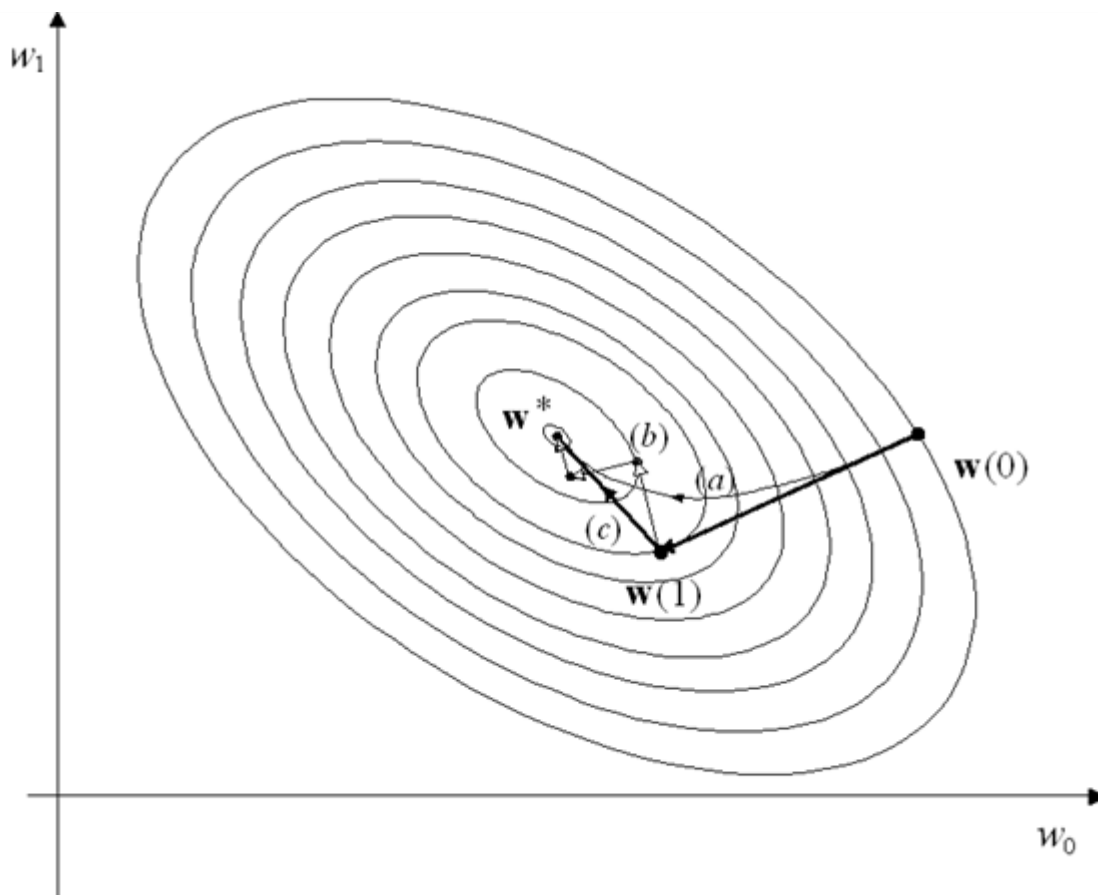
$$\beta_k = \frac{\nabla C^T(k+1) (\nabla C(k+1) - \nabla C(k))}{\mathbf{q}_k^T (\nabla C(k+1) - \nabla C(k))}. \quad (2.84)$$

Az algoritmus tehát a kezdőpontbeli negatív gradiens, mint kezdeti irány választása mellett számolja az új paramétert a (2.77) egyenlet szerint úgy, hogy olyan α_k értéket vesz, melynél a hibakritérium értéke minimális (az α_k szerinti derivált nulla; ún. vonalmenti keresés, *line search* módszer). Erre azért van szükség, mert az esetek többségében $\mathbf{R}$ ismeretének hiányában α_k (2.80) alapján nem számolható közvetlenül. Ezután az új paraméterhez tartozó gradienst számítjuk, amelynek birtokában (2.82) alapján új β_k -t határozzunk meg. Ezzel az új irány (2.81)-ből számítható.

Meg kell jegyezni, hogy a β_k együtthatók meghatározására léteznek egyéb módszerek is, melyek más kifejezésre vezetnek. Ennek jelentősége abban rejlik, hogy azon esetben, ha az egymás után következő gradiensok különbsége nullává válik (vagyis az (2.84) szerinti összefüggés használhatatlan lesz a nevező nullává válása miatt), a konjugált gradiens módszer akkor is használható a β_k -kra vonatkozó más kifejezés alkalmazásával. Vizsgálatokat végeztek annak meghatározására is, hogy különböző feladattípusoknál a β_k -k meghatározásának melyik módszere a legjobb.

A módszer fontos jellemzője, hogy kvadratikus, N -dimenziós kritériumfelület esetén a (2.78) összefüggés értelmében a konjugált gradiens módszer N lépésben konvergál. A gyorsabb konvergencia annak tudható be, hogy minden egyes lépésnél a (2.77), (2.78) és (2.80) egyenletek által meghatározott új $\mathbf{w}(k)$ a (2.49) egyenletben definiált négyzetes hibafüggvényt minimalizálja $\mathbf{w}$ szerint az addigi irányok ($\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_k$) által kifeszített altér felett. (Bizonyítható, hogy az algoritmus szerint számított gradiens irányok mindig merőlegesek a korábbi lépésekben meghatározott $\mathbf{q}$ irányokra. Ebből az ortogonalitásból pedig következik az előbbi kijelentés [Joh92]. Ezzel szemben a többi gradiens alapú eljárásnál, mint pl. a legmeredekebb lejtő módszernél, az új irány menti minimalizálás "leronthatja" a korábbiak hatását. Ezt illusztrálja a 2.13 ábra, ahol a konjugált gradiens módszeren (c) kívül a legmeredekebb lejtő módszer esetére is láthatunk két példát. Amennyiben a lépésmagyság az egyes iterációknál kicsi (kis μ tanulási tényezőt használunk), sok lépésben, de a trajektória (az ideális útvonal) mentén közelítjük meg a minimumhelyet (a), míg nagyobb lépésmagyság mellett láthatóan felesleges paraméterváltoztatásokra is sor kerül (b).

2.13. ábra - Példa konvergenciára "legmeredekebb lejtő" módszerrel a trajektória mentén (kis μ mellett) (a), "legmeredekebb lejtő" módszerrel nagyobb μ mellett (b); a konjugált gradiensek módszerével (c).



Összefoglalva az algoritmus a következő iterációs lépésekből áll:

1. Válasszunk egy kezdeti pontot: $k=0$, $\mathbf{w}(0)$, és határozzuk meg ebben a pontban a kritériumfelület gradiensét ($\nabla C(0)$).
2. Ha $\nabla C(0)=\mathbf{0}$, szélsőértékre jutottunk, állítsuk le az eljárást, egyébként legyen $\mathbf{q}_0 = -\nabla C(0)$.
3. Határozzuk meg a megfelelő lépésmagyságot (α_k), és számítsuk ki az új súlyértéket (2.80)-nek, ill. (2.77)-nek megfelelően. Amennyiben $\mathbf{R}$ nem áll rendelkezésünkre, α_k (2.80) alapján nem határozható meg. Ilyenkor α_k meghatározására azt a tényt használjuk ki, hogy megfelelő lépésmagyság választás esetén $\mathbf{q}_k$ irányban elmozdulva a kritériumfelületen feltételes szélsőértéket érünk el, vagyis:

$$\alpha_k = \arg \min_{\alpha} \{C(\mathbf{w}(k) + \alpha \mathbf{q}(k))\}$$
4. Végezzük el a súlymódosítást, és az új pontban határozzuk meg az új gradiens, $\nabla C(k+1)$ értékét. Ha $\nabla C(k+1)=\mathbf{0}$, állítsuk le, egyébként folytassuk az eljárást.
5. A (2.81) és (2.84) összefüggések felhasználásával határozzuk meg az új konjugált irány, $\mathbf{q}_{k+1}$ értékét.
6. Ha $k=N$, az eljárásnak vége, egyébként növeljük meg k értékét: $k=k+1$, és folytassuk az eljárást a 3. lépéssel.

A konjugált gradiens eljárást, vagy az eljárás valamilyen változatát kedvező konvergencia tulajdonságai miatt gyakran alkalmazzák neuronhálók tanításánál. Egyes neuronhálókban kedvelt az ún. momentum módszer, amely a konjugált gradiens eljáráshoz hasonló heurisztikus gradiens alapú eljárás. A hasonlóság abban áll, hogy a paramétermódosítás adott iterációbeli iránya itt is a gradiens és az előző iterációban alkalmazott irány súlyozott összegeként határozható meg. A módosítás irányának meghatározása tehát itt is a (2.81) összefüggéshez formailag hasonló módon történik, azzal a különbséggel, hogy a súlyozó együtthatók meghatározása nem (2.84) alapján, hanem tapasztalati úton történik. Ennek megfelelően a momentum módszer egyes lépéseiben használt irányok nem is lesznek konjugált irányok. A momentum módszerrel részletesebben a 4. fejezetben foglalkozunk.

5.1.6. Az LMS algoritmus és néhány változata

Az eddigi eljárások mind a (2.49) összefüggésben definiált hibakritériumból, vagyis a négyzetes hiba várható értékéből indultak ki. A várhatóérték-képzés következtében a kapott eljárások közvetlen vagy közvetett formában igényelték az $\mathbf{R}$ autokorrelációs mátrix ismeretét. Neuronhálókban általában a tanító mintapont-készleten kívül más ismeretünk nincs, így az autokorrelációs mátrix is csak a mintapontokból becsülhető. A becslés viszonylag nagy számításigénye, továbbá az $\mathbf{R}$ mátrixszal történő további műveletek számításigénye miatt hatékonyabb lenne egy olyan eljárás, ami $\mathbf{R}$ ismerete nélkül is alkalmazható. A Bernard Widrow által javasolt LMS (*Least Mean Squares*) algoritmus épp ezt teszi. Az LMS algoritmus és az eddigi gradiens alapú eljárások közötti alapvető különbség az, hogy az LMS eljárás a (2.49)-ben definiált átlagos négyzetes hiba, mint kritériumfüggvény helyett a pillanatnyi hiba alapján dolgozik.

Az LMS eljárásnál a hibakritériumot, mint a pillanatnyi hiba négyzetét definiáljuk:

$$C(\mathbf{w}(k)) = \varepsilon^2(k) = (d(k) - \mathbf{x}^T(k)\mathbf{w}(k))^2. \quad (2.85)$$

A pillanatnyi hiba alapján – gradiens módszert alkalmazva – a paramétermódosítás a kritériumfüggvény „pillanatnyi” gradiense alapján történhet.

$$\hat{\nabla} C(k) = \frac{\partial \varepsilon^2(k)}{\partial \mathbf{w}(k)} = -2\varepsilon(k)\mathbf{x}(k). \quad (2.86)$$

Az LMS eljárás alap gondolata tehát az, hogy az átlagos négyzetes hiba helyett a pillanatnyi négyzetes hibából indulunk ki. Természetesen meg kell vizsgálnunk, hogy a várhatóérték-képzés elhagyásának milyen következményei lesznek.

Könnyen belátható, hogy (2.86) várható értéke megegyezik a gradiens vektor (2.54) összefüggésével [Wid85]. A pillanatnyi gradiens vektorral a "legmeredekebb lejtő" módszert alkalmazva a paraméter változtatás összefüggésére a következőt kapjuk:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu \hat{\nabla} C(k) = \mathbf{w}(k) + 2\mu \varepsilon(k)\mathbf{x}(k). \quad (2.87)$$

Az is megmutatható, hogy a paraméterváltoztatás eredményeképpen kapott megoldás várható értéke

$$\begin{aligned} E\{\mathbf{w}(k+1)\} &= E\{\mathbf{w}(k)\} + 2\mu \left(\mathbf{p} - \mathbf{R} E\{\mathbf{w}(k)\} \right) \\ &= (\mathbf{I} - 2\mu \mathbf{R}) E\{\mathbf{w}(k)\} + 2\mu \mathbf{R} \mathbf{w}^* \end{aligned} \quad (2.88)$$

amelyből, ha áttérünk az $\mathbf{R}$ sajátvektorai által meghatározott koordináta-rendszerre, adódik, hogy

$$E\{\mathbf{v}^i(k)\} = (\mathbf{I} - 2\mu \Lambda)^k \mathbf{v}^i(0). \quad (2.89)$$

Fentiekből látszik, hogy ha k minden határon túl nő, az LMS megoldás várható értéke megegyezik az optimális megoldással (ami a zérus vektor a sajátvektor koordináta-rendszerben, vagyis az eredeti koordináta-rendszerben épp a $\mathbf{w}^*$), ha (2.89) jobb oldala nullához konvergál. E feltétel pedig azonos a legmeredekebb lejtő módszer konvergenciájának (2.69)-ben megfogalmazott feltételével, vagyis a tanulási tényezőre vonatkozó konvergencia-feltétel az LMS algoritmusnál is

$$0 < \mu < \frac{1}{\lambda_{\max}}. \quad (2.90)$$

Az LMS eljárás legnagyobb előnye az eddig látott eljárásokkal szemben, hogy alkalmazásához a mintapontokon kívül másra nincs szükség, továbbá, hogy nagyon egyszerűen megvalósítható. Az eljárás egyszerűsége mellett azonban azt is látni kell, hogy a konvergencia feltételének megfelelő tanulási tényező megválasztásához itt is ismerni kell $\lambda_{\max}$ -ot (vagy legalább annak (2.70) alapján nyerhető felső becslését), sőt a megfelelően gyors konvergenciához most is szükség van arra, hogy a $\lambda_{\max}/\lambda_{\min} \cong 1$ arány teljesüljön. Meg kell jegyezni, hogy a sajátértékek becslésének viszonylag nagy számításgénye miatt a gyakorlati alkalmazások többségében a tanulási tényezőt tapasztalati úton határozzák meg.

Rendkívüli egyszerűsége következtében az LMS eljárást igen elterjedten használják. Az a tény azonban, hogy minden lépésben a pillanatnyi gradiens irányában korrigálunk, az optimum közelében problémákat okozhat. E problémák jellegét érzékelhetjük, ha az ideális gradienstől való eltérés statisztikai jellemzőit vizsgáljuk. Az eltérést modellezzük egy $\{\mathbf{n}(k)\}$ zaj folyamattal, amivel

$$\hat{\nabla}C(k) = \nabla C(k) + \mathbf{n}(k). \quad (2.91)$$

Ha elegendően kicsi μ tanulási tényezővel dolgozunk, akkor elég jól megközelíthetjük az optimális $\mathbf{w}^*$ értéket, ahol már $\nabla C(k) \cong \mathbf{0}$, így

$$\mathbf{n}(k) \cong \hat{\nabla}C(k) = -2\varepsilon(k) \mathbf{x}(k). \quad (2.92)$$

Az $\{\mathbf{n}(k)\}$ zaj kovarianciája

$$E\{\mathbf{n}(k)\mathbf{n}^T(k)\} = 4E\{\varepsilon^2(k) \mathbf{x}(k)\mathbf{x}^T(k)\}. \quad (2.93)$$

A (2.51) összefüggés segítségével és felhasználva, hogy

$$\varepsilon(k) = (d(k) - \mathbf{x}^T(k)\mathbf{w}(k)) \quad (2.94)$$

belátható, hogy az optimum közelében $E\{\varepsilon(k)\mathbf{x}(k)\} \cong \mathbf{0}$, ezért (2.93) átírható a következő alakba:

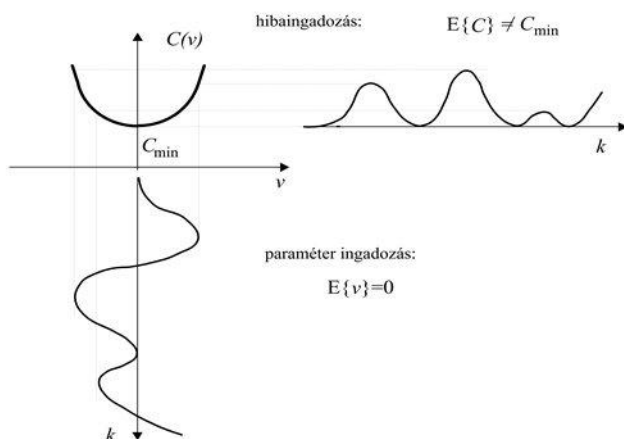
$$E\{\mathbf{n}(k)\mathbf{n}^T(k)\} = 4E\{\varepsilon^2(k)\} E\{\mathbf{x}(k)\mathbf{x}^T(k)\} \cong 4C_{\min} \mathbf{R}. \quad (2.95)$$

Ha ennek a varianciának a hatását nyomon követjük az iterációk során, akkor a paraméter hiba-vektor ($\mathbf{v}(k)$) kovarianciáját is becsülhetjük. A levezetés részletezése nélkül

$$E\{\mathbf{v}(k)\mathbf{v}^T(k)\} \cong \mu C_{\min} \mathbf{I}, \quad (2.96)$$

azaz a paraméter hiba kovarianciavektora egyenesen arányos a tanulási tényezővel, ami azt jelenti, hogy a varianciát úgy tudjuk alacsony szinten tartani, hogy az optimum közelében kis μ értékkel dolgozunk.

2.14. ábra - A hibaingadozás alakulása négyzetes paraméterfüggés esetén



A paraméter hiba kovarianciája a 2.14 ábrán látható módon lehetetlenné teszi a minimum elérését. A paraboloid hibafelület tulajdonságai következtében ugyanis a bekövetkező hibának nem $C_{\min}$ a várható értéke. A (2.57) összefüggésre alapozva és felhasználva a $C_{\min}$ (2.52) összefüggését a $C_{\min}$ -t meghaladó komponensre – az ún. többlet átlagos négyzetes hibára (*excess mean-square error, EMSE*) – a következő adódik:

$$EMSE = E\{\mathbf{v}^T(k) \mathbf{A} \mathbf{v}(k)\} \approx \mu C_{\min} \text{tr}(\mathbf{R}), \quad (2.97)$$

amelynek felhasználásával a hiba elérhető minimuma

$$C'_{\min} = C_{\min} (1 + \mu \text{tr}(\mathbf{R})) \quad (2.98)$$

lesz. A pillanatnyi gradiens alkalmazásának következménye önmagában is kifejezhető:

$$M = \frac{EMSE}{C_{\min}} = \mu \text{tr}(\mathbf{R}). \quad (2.99)$$

M értékét, amelyet az irodalomban „beállítatlanságnak” (*misadjustment*) neveznek, tehát a többlet hibának és a Wiener-Hopf megoldáshoz tartozó hibának (a minimális átlagos négyzetes hibának) az arányaként definiáljuk.

A továbbiakban az LMS eljárás néhány további változatával foglalkozunk: előbb az LMS és a Newton algoritmus ötvözeteként származtatható algoritmust mutatjuk be röviden, majd egy olyan változat főbb jellemzőit foglaljuk össze, amelynek során egy kedvezőtlen konvergencia tulajdonságú eset a bemeneti mintatér transzformálásával kedvezőbb esetté alakítható.

5.1.7. Az LMS/Newton algoritmus

Ez az algoritmus a Newton és az LMS algoritmus egyes elemeit együttesen magába építi. A Newton algoritmus paramétermódosító algoritmusát a (2.62) összefüggés adta meg és láttuk, hogy amennyiben mind a gradiens, mind $\mathbf{R}^{-1}$ ismert, akkor egy lépésben is elérhető a $\mathbf{w}^*$ megoldás. Ha $\mathbf{R}^{-1}$ -et nem ismerjük, és az iterációnkénti pontos gradiens helyett csak a pillanatnyi gradiens áll rendelkezésünkre, akkor az LMS eljárás alkalmazható. Az LMS/Newton algoritmusnál feltételezzük, hogy $\mathbf{R}^{-1}$ ismert, de a pillanatnyi gradiens alapján számolunk. Így az iterációs összefüggés a (2.62) és a (2.86) összefüggések kombinálásából származtatható, vagyis a Newton algoritmusban a gradiens értékét a pillanatnyi gradienssel helyettesítjük:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu \mathbf{R}^{-1} \varepsilon(k) \mathbf{x}(k). \quad (2.100)$$

A kapott összefüggés csak annyiban tér el az LMS algoritmustól, hogy a súlymódosító részben $\mathbf{R}^{-1}$ is szerepel. Ha $\mathbf{R}$ azonos $\lambda_{\mathbf{w}}$ sajátértékekkel rendelkező diagonálmátrix lenne, akkor $\lambda_{\mathbf{w}} \mathbf{R}^{-1} = \mathbf{I}$ -re adódna, vagyis, ha μ helyett $\lambda_{\mathbf{w}} \mu$ -t használnánk, lényegében visszakapnánk az LMS algoritmust, ahol μ -t a sajátérték reciprokánál kisebbre kell választanunk. Alkalmazzuk ezt a választást akkor is, ha $\mathbf{R}^{-1}$ nem diagonálmátrix, sőt a sajátértékek

sem egyformák. Ekkor legyen λ_{av} az $\mathbf{R}$ mátrix sajátértékeinek az átlaga. Az így kapott algoritmus az LMS/Newton algoritmus:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu \lambda_{av} \mathbf{R}^{-1}(k) \mathbf{x}(k) \varepsilon(k). \quad (2.101)$$

Mindamellett, hogy az LMS/Newton algoritmust kedvező tulajdonságai miatt szokták "ideális adaptív algoritmusnak" is nevezni, a gyakorlati esetek többségében nem alkalmazható, mert $\mathbf{R}^{-1}$ pontos ismeretét igényli. Létezik azonban olyan, a gyakorlatban is használható módosítása, amelyben rekurzívan becsüljük az $\mathbf{R}$, ill. $\mathbf{R}^{-1}$ mátrixot. A (2.100) összefüggés módosításával

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu \mathbf{R}^{-1}(k+1) \mathbf{x}(k) \varepsilon(k). \quad (2.102)$$

ahol ($0 < \eta \leq 1$, $0 < \nu \leq 1$ feltételezésével)

$$\mathbf{R}(k+1) = \eta \mathbf{R}(k) + \nu \mathbf{x}(k) \mathbf{x}^T(k), \quad (2.103)$$

a korrelációs mátrix becsült értéke a $(k+1)$ -edik időpillanatban, $\mathbf{R}^{-1}(k+1)$ pedig az ún. mátrix inverziós lemma (ld. Függelék) felhasználásával az alábbi formában adható meg:

$$\mathbf{R}^{-1}(k+1) = \frac{1}{\eta} \left(\mathbf{R}^{-1}(k) - \frac{\mathbf{R}^{-1}(k) \mathbf{x}(k) \mathbf{x}^T(k) \mathbf{R}^{-1}(k)}{\frac{\eta}{\nu} + \mathbf{x}^T(k) \mathbf{R}^{-1}(k) \mathbf{x}(k)} \right). \quad (2.104)$$

A (2.103) összefüggésben szereplő konstansok kapcsolata gyakran $\eta = 1 - \nu$. (Továbbá gyakori ajánlás η -ra a $[0.9 \dots 0.99]$ közötti érték). A (2.102)-(2.104) összefüggésekkel jellemzett algoritmusnak léteznek további változatai, amelyek között érdemi különbség csak az $\mathbf{R}$ mátrix rekurzív becslésében van. Formai szempontból a (2.102)-(2.104) összefüggések az iteratív, gradiens alapú szélsőérték-kereső eljárások széles családját lefedik.

5.1.8. Transzformált tartománybeli LMS eljárás

A legmeredekebb lejtő módszer tárgyalásánál láttuk, hogy a konvergencia-sebességet az $\mathbf{R}$ mátrix sajátértékeinek alakulása nagymértékben befolyásolja. A konvergencia-sebesség szempontjából a legkedvezőbb eset, ha az összes sajátérték azonos, és különösen kedvezőtlen, ha a legkisebb és a legnagyobb sajátérték nagymértékben különböző, vagyis ha $\lambda_{\max} / \lambda_{\min} \gg 1$. Láttuk azt is, hogy a konvergencia-sebesség szempontjából azért kedvezőtlen a sajátértékek nagyfokú különbözősége, mert ilyenkor a hibafelület különböző részein a felület meredeksége jelentősen eltér, így a $\lambda_{\max}$ alapján választott tanulási tényező – bár a konvergenciát biztosítja – az esetek jelentős részében a szükségesnél óvatosabb lépések megtételét eredményezi. Minél meredekebb szakaszán mozgunk a hibafelületnek, annál kisebb a megengedhető μ . Ha nem tudjuk, hogy a hibafelület mely szakaszán haladunk, akkor a μ megválasztása szempontjából a legrosszabb esetet kell feltételeznünk, vagyis azt, mintha a felület legmeredekebb részén mozognánk. Ezt fejezi ki a (2.69) összefüggéssel megadott felső korlát, amely szerint μ -t a legnagyobb sajátérték reciprokánál kisebbre kell választani.

A hibafelület alakját az $\mathbf{R}$ mátrix jellemzi, ami a bemeneti mintákból számítható. A konvergencia-sebesség szempontjából kedvezőbb helyzet tehát úgy teremthető, ha a bemeneteket módosítjuk. A módosításnak arra kell irányulnia, hogy a módosított bemenetek autokorrelációs mátrixa minél inkább az egységmátrix legyen. A bemenetek módosítása valamilyen alkalmas transzformációval lehetséges. A transzformáció az $\{\mathbf{x}(k)\}$ bemeneti mintapontokból $\{\mathbf{X}(k)\}$ transzformált mintapontokat állít elő. A transzformáció következtében természetesen az eredeti probléma megoldásában szerepet játszó $\mathbf{w}$ paramétervektor helyett az ugyancsak megfelelő módon transzformált $\mathbf{w}^*$ paramétervektor fog szerepelni [Nar83].

A szélsőérték-keresési probléma a transzformált bemenetekre is megfogalmazható, így az eddig bemutatott eljárások mindegyike alkalmazható transzformált esetben is.

Megfelelően választott transzformációt és azt követő normalizálást alkalmazva biztosítható a $\lambda_{\max} / \lambda_{\min} = 1$ arány. Jelölje $\mathbf{T}$ az $\mathbf{x}(k)$ és $\mathbf{X}(k)$ közötti transzformációt leíró mátrixot:

$$\mathbf{X}(k) = \mathbf{T}\mathbf{x}(k). \quad (2.105)$$

Ha a kiinduló bemenetek autokorrelációs mátrixa $\mathbf{R}_{\mathbf{x}} = E\{\mathbf{x}\mathbf{x}^T\}$, a transzformált bemenetek autokorrelációs mátrixát

$$\mathbf{R}_{\mathbf{xx}} = E\{\mathbf{X}\mathbf{X}^T\} = E\{\mathbf{T}\mathbf{x}\mathbf{x}^T\mathbf{T}^T\} = \mathbf{T}\mathbf{R}_{\mathbf{x}}\mathbf{T}^T \quad (2.106)$$

formában kapjuk (a kétféle autokorrelációs mátrix megkülönböztetése miatt alkalmazzuk a bemenet autokorrelációs mátrixára az $\mathbf{R}_{\mathbf{x}}$ jelölést az eddigi $\mathbf{R}$ helyett és az $\mathbf{R}_{\mathbf{xx}}$ jelölést a transzformált bemenetek autokorrelációs mátrixára). Legyen $\mathbf{T}$ egy ortonormált transzformáció, vagyis teljesüljön a $\mathbf{T}\mathbf{T}^T = \mathbf{I}$ összefüggés.

Ha még azt is előírjuk, hogy $\mathbf{T} = \mathbf{Q}^T$, ahol $\mathbf{Q}$ az $\mathbf{R}_{\mathbf{x}}$ mátrix normált sajátvektoraiból képezett mátrix, akkor (2.55)-et figyelembevéve

$$\mathbf{R}_{\mathbf{xx}} = \mathbf{Q}^T \mathbf{R}_{\mathbf{x}} \mathbf{Q} = \mathbf{\Lambda}, \quad (2.107)$$

vagyis a transzformált bemenetek autokorrelációs mátrixa az $\mathbf{R}_{\mathbf{x}}$ sajátértékeiből képezett diagonálmátrix. A diagonálmátrixból egységmátrix megfelelő normalizálással nyerhető. Ezért, ha a transzformáció mátrixát $\mathbf{T} = \mathbf{\Lambda}^{-1/2} \mathbf{Q}^T$ -ra választjuk, a transzformált bemenet autokorrelációs mátrixára

$$\mathbf{R}_{\mathbf{xx}} = E\{\mathbf{X}\mathbf{X}^T\} = E\{\mathbf{T}\mathbf{x}\mathbf{x}^T\mathbf{T}^T\} = \mathbf{T}\mathbf{R}_{\mathbf{x}}\mathbf{T}^T = \mathbf{\Lambda}^{-1/2} \mathbf{Q}^T \mathbf{R}_{\mathbf{x}} \mathbf{Q} \mathbf{\Lambda}^{-1/2} = \mathbf{I} \quad (2.108)$$

adódik, vagyis minden sajátérték azonos és egységnyi lesz.

Az autokorrelációs mátrix sajátvektoraiból képezett $\mathbf{Q}$ mátrixszal elvégzett ún. főtengety transzformációt Karhunen-Loève transzformációnak (KL transzformáció, KLT) is nevezik. A kitűzött feladat szempontjából ez az optimális transzformáció, hiszen ha a főtengety transzformáció után a sajátértékek reciprokéval (2.108) szerint még normalizálunk is, a kitűzött célt el is érjük, a sajátvektorok azonosak lesznek.

A konvergencia-sebesség növelésére a KL transzformációt, annak viszonylag jelentős számításigénye miatt, ritkán használják. Helyette könnyebben számítható ortogonális transzformációkat, például a diszkrét Fourier transzformációt vagy a diszkrét koszinusz transzformációt szokás alkalmazni. Ha a KL transzformációtól eltérő ortogonális transzformációt alkalmazzuk, meg kell vizsgálnunk, hogy ennek milyen hatása lesz a $\lambda_{\max} / \lambda_{\min}$ arány alakulására.

Írjuk fel ehhez komponensenként a transzformált tartománybeli LMS eljárás súlymódosítását:

$$\mathbf{w}_i'(k+1) = \mathbf{w}_i'(k) + 2\mu_i \varepsilon(k) X_i(k), \quad (2.109)$$

ahol a $\mathbf{w}_i'$ értékek a transzformált tartománybeli súlyokat jelölik. A súlymódosításnál most minden komponens esetén más-más tanulási tényezőt (μ_i) alkalmazhatunk. Ha a

$$\mu_i = \frac{\mu}{E\{X_i^2(k)\}}, \quad i = 0, 1, \dots, N-1 \quad (2.110)$$

választással élünk, akkor ez a transzformált tartományban normálást is jelent.

A súlymódosítást vektorosan is felírhatjuk, ha bevezetjük a $\hat{\mathbf{\Lambda}}$ $N \times N$ -es diagonálmátrixot, melynek az (i,i) -dik eleme $E\{X_i^2(k)\}$.

$$\mathbf{w}_i'(k+1) = \mathbf{w}_i'(k) + 2\mu \hat{\mathbf{\Lambda}}^{-1} \varepsilon(k) \mathbf{X}(k). \quad (2.111)$$

A transzformált tartományban a konvergencia sebessége $\hat{\mathbf{A}}^{-1}\mathbf{R}_{\mathbf{xx}}$ sajátértékeinek szóródásától vagy a legnagyobb és legkisebb sajátértékének arányától függ.

Ennek az aránynak a becsléséhez felhasználjuk, hogy egy $\mathbf{A}$ mátrix $\lambda_{\max} / \lambda_{\min}$ aránya felülről becsülhető:

$$\frac{\lambda_{\max}}{\lambda_{\min}} \leq \frac{\text{Tr}(\mathbf{A})}{\text{Det}(\mathbf{A})}, \quad (2.112)$$

mivel az $\mathbf{A}$ mátrix maximális sajátértéke a mátrix nyomával majorálható, minimális sajátértéke pedig a mátrix determinánsával minorálható:

$$\lambda_{\max} \leq \text{Tr}(\mathbf{A}), \quad (2.113)$$

illetve, ha $N > 2$

$$\lambda_{\min} \geq \text{Det}(\mathbf{A}). \quad (2.114)$$

Ha feltételezzük, hogy $E(x^2(k)) = 1$, akkor

$$\text{Tr}(\hat{\mathbf{A}}^{-1}\mathbf{R}_{\mathbf{xx}}) = \text{Tr}(\mathbf{R}_{\mathbf{xx}}) = N, \quad (2.115)$$

továbbá

$$\text{Det}(\hat{\mathbf{A}}^{-1}\mathbf{R}_{\mathbf{xx}}) = \text{Det}(\hat{\mathbf{A}}^{-1})\text{Det}(\mathbf{R}_{\mathbf{xx}}) = \text{Det}(\hat{\mathbf{A}})^{-1}\text{Det}(\mathbf{R}_{\mathbf{xx}}). \quad (2.116)$$

és

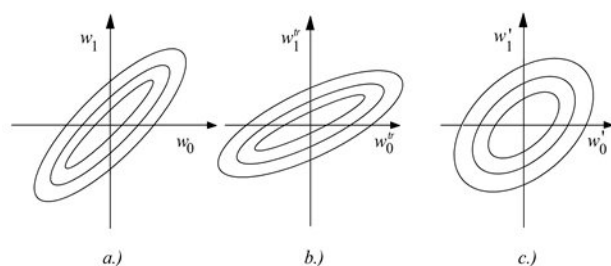
$$\frac{\text{Tr}(\hat{\mathbf{A}}^{-1}\mathbf{R}_{\mathbf{xx}})}{\text{Det}(\hat{\mathbf{A}}^{-1}\mathbf{R}_{\mathbf{xx}})} = \text{Det}(\hat{\mathbf{A}}) \frac{\text{Tr}(\mathbf{R}_{\mathbf{xx}})}{\text{Det}(\mathbf{R}_{\mathbf{xx}})}. \quad (2.117)$$

Mivel $\text{Tr}(\hat{\mathbf{A}}) = N$, ezért feltételezhető, hogy $\text{Det}(\hat{\mathbf{A}})$ legfeljebb 1 lehet, ezért

$$\frac{\text{Tr}(\hat{\mathbf{A}}^{-1}\mathbf{R}_{\mathbf{xx}})}{\text{Det}(\hat{\mathbf{A}}^{-1}\mathbf{R}_{\mathbf{xx}})} \leq \frac{\text{Tr}(\mathbf{R}_{\mathbf{xx}})}{\text{Det}(\mathbf{R}_{\mathbf{xx}})}, \quad (2.118)$$

tehát megfelelően megválasztott transzformációnál – bár (2.118) erre feltétlen garancát nem ad – esély van arra, hogy a sajátérték-arány csökkenjen és így kedvezőbb feltételeket biztosítson a gyorsabb konvergenciához. A transzformált tartományra való áttérés hatását illusztrálja a 2.15 ábra.

2.15. ábra - A hibafelület (kétdimenziós) alakulása az eredeti (a), a transzformált (b) és a transzformált-normalizált (c) esetben



Az ábra a hibafelület izokritérium görbéit mutatja a paramétertérben az eredeti bemenőjelre (a), a transzformált tartományban, normalizáció nélkül (b) és a transzformált tartományban normalizációval (c). Megjegyezzük, hogy az optimális transzformációt jelentő KL transzformáció esetén a kvadratikus hibafelület izokritérium görbéi koncentrikus körök lesznek.

A fentiek után még azt kell megnéznünk, hogy eredményként mit kapunk, ha a transzformált bemeneti térben alkalmazzuk a legmeredekebb lejtő módszert vagy az LMS algoritmust. Láttuk, hogy az eredeti bemeneti tartományban végzett LMS eljárás a (2.51) Wiener-Hopf egyenlet által megadott súlyvektorhoz tart

$$\mathbf{w}^* = \mathbf{R}_{xx}^{-1} \mathbf{p}_{xd} \quad (2.119)$$

ahol $\mathbf{p}_{xd}$ a bemenet és a kívánt kimenet keresztkorrelációs vektora.

Könnyen belátható, hogy

$$\mathbf{w}'^* = \mathbf{R}_{xx}^{-1} \mathbf{p}_{xd} = \mathbf{T} \mathbf{R}_{xx}^{-1} \mathbf{p}_{xd} = \mathbf{T} \mathbf{w}^* \quad (2.120)$$

ahol $\mathbf{R}_{xx}$, mint (2.107)-ben megadtuk a transzformált bemenet autokorrelációs mátrixa, $\mathbf{p}_{xd}$ pedig a transzformált bemenet és a kívánt válasz keresztkorrelációs vektora. Tehát a transzformált bemenetekre alkalmazott LMS eljárás eredményeképpen kapott súlyvektor az eredeti súlyvektor transzformáltja lesz.

5.1.9. Szélsőérték-keresés paramétereiben nemlineáris modellek esetén

Az eddigiekben feltételeztük, hogy kvadratikus kritériumfelülettel van dolgunk, amit a legegyszerűbben az $y = \mathbf{w}^T \mathbf{x}$ lineáris kapcsolat és a négyzetes hibafüggvény biztosíthat. A súlytér felett azonban akkor is kvadratikus kritériumfelületet kapunk, ha a vizsgált leképezést $y = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x})$ alakban adhatjuk meg. Ekkor valójában először a bemeneti vektorok $\boldsymbol{\varphi}(\mathbf{x})$ általi (nemlineáris) transzformációja történik, a transzformált térről a kimeneti térre pedig már lineáris leképezést valósítunk meg. Az optimális paramétervektor meghatározása szempontjából azonban ez az eset nem különbözik az eddig tárgyalttól, hiszen a modellünk most is a paramétervektor lineáris függvénye. Egyes neuronhálók (pl. a radiális bázisfüggvényes – RBF – hálózatok) leképezése ez utóbbi általános formulával írható le, míg más hálónál – pl. az MLP-knél – a nemlineáris leképezés már a paraméterekben is nemlineáris kapcsolattal történik. Ez utóbbi esetben – még ha négyzetes hibafüggvényt alkalmazunk is – a kritériumfüggvény nem lesz a paraméterek kvadratikus függvénye, sőt még az sem garantálható, hogy a kritériumfüggvény konvex függvény legyen. Ilyen esetekre az előzőekben bevezetett gradiens alapú eljárásokat és az eljárások eredményét külön kell értékelnünk.

A paramétereiben nemlineáris modellek egyik legfőbb jellemzője, hogy a hibafelület nem unimodális: a hibafelületen nem csak egy – globális – szélsőérték (maximum vagy minimum) található, hanem számos – lokális – szélsőérték is lehet. Vagyis a

$$\nabla C(\mathbf{w}) = \frac{\partial C(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{0} \quad (2.121)$$

feltételt nemcsak a globális szélsőértékhez tartozó $\mathbf{w}^*$ paramétervektor, hanem egyéb paramétervektorok is biztosíthatják. Sőt, általános esetben az is előfordulhat, hogy egy egész paramétertartományban a felület gradiense zérus, ami azt jelenti, hogy a hibafelületen valamekkora kiterjedésű „lapos” terület is található.

A gradiens alapú eljárások ilyenkor is alkalmazhatók, az eredményeket azonban külön értékelni kell. A gradiens eljárásokkal kapott eredmény nem feltétlenül jelenti a legkisebb kritériumértékhez tartozó megoldást, így az eredmény minősége – a kritériumérték – alapján mérlegelni kell, hogy a kapott megoldásvektor a hibafelület globális minimumpontját adja meg, vagy csupán egy lokális minimumpontot. A megoldást még ezen utóbbi esetben is elfogadhatjuk, ha a kapott kritériumérték az adott alkalmazást tekintve elegendően kicsi. Ha nem, akkor újabb (lokális) minimumpontot kell keresni, ami szerencsés esetben egyben a globális minimumpont is lehet.

A nemkvadratikus hibafelületek további következménye, hogy olyan eljárásoknál, ahol a konvergenciához szükséges lépések száma kvadratikus hibafelület mellett meghatározható, most ez sem lesz érvényes. A konjugált gradiens eljárásnál például nem kvadratikus hibafelület mellett N iteráció után az algoritmus nem

állítható le. Ilyenkor az az általánosan javasolt eljárás, hogy N iterációnként indítsuk újra az egész algoritmust egy új kezdeti gradiens meghatározásával. Meg kell azonban jegyeznünk, hogy valóságos esetekben az N -lépéses konvergencia még kvadratis hibafelület mellett sem biztosított, hiszen numerikus problémák miatt a gradiens értéke sohasem fogunk nullát kapni. Ilyenkor az algoritmus leállítása sem történhet meg a gradiens értéke alapján. Helyette vagy egy adott iterációs szám, vagy ha az egymást követő iterációkhoz tartozó kritériumértékek különbsége megfelelően kicsi jelentheti a leállás feltételét. A tapasztalat szerint azonban a konjugált gradiens módszer még nem ideális esetben (numerikus problémák, vagy nem "teljesen" négyzetes hibafelület) és az optimumtól távol is kedvezőbb konvergenciát mutat, mint a direkt gradiens módszerek.

5.1.10. További gradiens-alapú módszerek

Az előzőekben láttuk, hogy egyes eljárások, mint a Newton, a Newton/LMS, de a konjugált gradiens módszer is a hibafelületről az $\mathbf{R}$ mátrix figyelembevétele által a gradiens túl további információt is felhasznál. Kvadratis hibafelület esetén $\mathbf{R}$ a hibafelület második parciális deriváltjaiból képezett $\mathbf{H}$ ún. Hesse mátrixszal egyezik meg, tehát $\mathbf{R}$ valójában a hibafelület görbületéről szolgál információval. Kvadratis hibafelületet – mint láttuk – paramétereiben lineáris modell és négyzetes hibafüggvény mellett kapunk. A hibafelületre vonatkozó járulékos információ a konvergenciasebességet jelentősen megnöveli a csupán a gradiens felhasználásával dolgozó LMS eljáráshoz képest. A Newton módszerrel pl. négyzetes hibafelület mellett akár egy lépésben is eljuthatunk a hibafelület minimumpontjába.

A Newton illetve a Newton/LMS eljárások előnyös konvergencia tulajdonságai ellenére számos nehézséget is rejtenek. Egyrészt alkalmazásukhoz kell, hogy $\mathbf{R}$ pozitív definit legyen, másrészt $\mathbf{R}^{-1}$ meghatározása számításigényes. További gondot jelenthet, ha a hibafelület nem négyzetes. Az alábbiakban bemutatott Levenberg-Marquardt eljárás ezekre a nehézségekre igyekszik megoldást találni.

A nemkvadratis kritériumfelület kezelhető olyan módon hogy a $C(\mathbf{w})$ kritériumfüggvényt Taylor-sorának első néhány tagjával közelítjük. Ha az első és másodfokú tagokat vesszük figyelembe, akkor egy nemkvadratis kritériumfelület kvadratis közelítésével dolgozunk.

$$C(\mathbf{w}) \cong C(\mathbf{w}(k)) + \nabla C(\mathbf{w}(k))^T (\mathbf{w} - \mathbf{w}(k)) + \frac{1}{2} (\mathbf{w} - \mathbf{w}(k))^T \mathbf{H}(\mathbf{w}(k)) (\mathbf{w} - \mathbf{w}(k)) \quad (2.122)$$

ahol

$$\mathbf{H} = \frac{\partial \nabla C(\mathbf{w}(k))}{\partial \mathbf{w}} \quad (2.123)$$

a kritériumfelület második parciális deriváltjaiból felépített Hesse-féle mátrix. A (2.122) összefüggésnek a (2.53) összefüggéssel való összevetéséből látszik, hogy a $\mathbf{H}$ mátrix az $\mathbf{R}$ mátrixnak feleltethető meg.

Ha a (2.122) összefüggés minimumát $\mathbf{w}$ szerinti deriválással keressük, akkor a közelítő négyzetes hibafelület minimumpontját egy lépésben is elérhetjük. Ehhez a

$$\nabla C(\mathbf{w}(k+1)) = \mathbf{0} = \nabla C(\mathbf{w}(k)) + \mathbf{H}(\mathbf{w}(k)) (\mathbf{w}(k+1) - \mathbf{w}(k)) \quad (2.124)$$

feltételt kell kielégítenünk, amelyből a Newton módszert (lásd (2.61)) kapjuk:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mathbf{H}(\mathbf{w}(k))^{-1} \nabla C(\mathbf{w}(k)). \quad (2.125)$$

Az 1/2-es szorzó a korrekciós tagból most hiányzik, mert a (2.122) összefüggés jobboldalának harmadik tagjában a (2.61) összefüggéssel ellentétben $\mathbf{H}^{-1}$ szerepel. Az így kapott eljárás nemkvadratis hibafelület esetén nem feltétlenül jobb, mint a gradiens alapú eljárás. Ezért hatékonyabb lehet, ha a Newton eljárás és az egyszerű gradiens alapú eljárás „keverékét” alkalmazzuk. Ezt javasolja a Newton módszer Levenberg-Marquardt féle módosítása:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - (\mathbf{H}(\mathbf{w}(k)) + \lambda(k) \mathbf{I})^{-1} \nabla C(\mathbf{w}(k)). \quad (2.126)$$

Látható, hogy a (2.126) összefüggés valójában a legmeredekebb lejtő módszer és a Newton eljárás kombinációja, ahol a kétféle eljárás arányát a $\lambda(k)$ lépésfüggő együttható szabja meg. Ha $\lambda(k)$ nagy, a (2.126)

alapján történő súlymódosítás a legmeredekebb lejtő módszert, míg ha $\lambda(k)$ kicsi, a Newton eljárást közelíti. A $\lambda(k)\mathbf{I}$ beépítése az eljárásba regularizációként is értelmezhető, ami a mátrix invertálhatóságát akkor is biztosítja, ha $\mathbf{H}$ közel szinguláris.

A neurális hálózatok tanításánál hatékonynak bizonyult a $\mathbf{H}$ mátrix egy közelítése, amely a következő gondolatmeneten alapszik. Tételezzük fel, hogy a kritériumfüggvény önmagában négyzetes, de a paraméterfüggés nemlineáris, tehát

$$C(\mathbf{w}) = \frac{1}{2} E \left\{ (d - y(\mathbf{w}))^2 \right\} = \frac{1}{2} E \left\{ \varepsilon^2 \right\} \quad (2.127)$$

Ekkor a gradiens vektor

$$\nabla C(\mathbf{w}) = -E \left\{ \varepsilon \nabla y(\mathbf{w}) \right\} \quad (2.128)$$

alakban írható, illetve ennek minden sorát deriválva minden paraméter szerint megkapjuk a $\mathbf{H}$ mátrixot:

$$\mathbf{H} = E \left\{ \nabla y(\mathbf{w}) \nabla y(\mathbf{w})^T - \varepsilon \left[\frac{\partial^2 y(\mathbf{w})}{\partial w_i \partial w_j} \right] \right\}, \quad (2.129)$$

ahol a szögletes zárójelben szereplő kifejezés $y(\mathbf{w})$ másodrendű parciális deriváltjainak mátrixát szimbolizálja. Mivel a (2.129) kifejezés második tagja ε -nal szorozódik, a $\mathbf{H}$ mátrix az optimumhoz közeledve egyre inkább az első tagtól függ, hiszen akkor általában mind ε , mind $y(\mathbf{w})$ görbülete egyre kisebb, ezért felvethető a

$$\mathbf{H} \cong E \left\{ \nabla y(\mathbf{w}) \nabla y(\mathbf{w})^T \right\} \quad (2.130)$$

közelítés. Ha $\mathbf{H}$ (2.130) szerinti közelítését (2.126)-ban alkalmazzuk, akkor számítási hatékonyság szempontjából kedvezőbb megoldást nyerünk, hiszen a Hesse mátrix (2.130) szerinti becslése jóval kisebb számításigényű, mintha a valódi második deriváltak alapján számítanánk. Meg kell jegyezni, hogy sok sikeres alkalmazás fűződik egy olyan megoldáshoz, ahol az $\mathbf{I}$ egységmátrix helyett egy olyan $\mathbf{\Omega}$ diagonálmátrixot alkalmazunk, melynek elemei megegyeznek $\mathbf{H}$ (2.130) szerinti becslésének diagonális elemeivel. Az így kapott eljárást az irodalomban Levenberg-Marquardt módszernek nevezik [Mor77].

5.1.11. Feltételes szélsőérték-kereső eljárások

Neuronhálók, illetve egyes tanuló rendszerek konstrukciójánál az optimális paraméterkészlet meghatározása sokszor feltételes szélsőérték-keresési eljárásra vezet. Ilyen esetekkel találkozunk például, ha a leképezés hibájára vonatkozó kritérium vagy kockázat minimalizálását bizonyos mellékfeltételek teljesülése mellett kell elvégeznünk. Mint az egyes hálóarchitektúráknál látni fogjuk, ilyen mellékfeltétel lehet a háló leképezésére megfogalmazott valamilyen simasági feltétel, vagy a paraméterekre vonatkozó megkötés: a paraméterek értékei a lehető legkisebbek legyenek, vagy a paramétervektor komponenseinek összege legyen egy adott érték, tipikusan 1. A mellékfeltételek megfogalmazásának egy lehetséges módja a 2.1 részben bemutatott regularizáció is.

A feltételes szélsőérték problémák gyakran a következő formában fogalmazhatók meg: keressük egy skalár értékű konvex $C(\mathbf{w})$ kritériumfüggvény extrémumához tartozó $\mathbf{w}^*$ -ot azzal a feltétellel, hogy egyidejűleg a $g(\mathbf{w})=0$ formában adott mellékfeltétel is teljesüljön.

Az ilyen típusú feltételes szélsőérték-keresési problémák hatékony megoldására szolgál a Lagrange multiplikátoros módszer. A Lagrange multiplikátoros eljárásnál meg kell fogalmazni egy módosított kritériumfüggvényt, amely mind az eredeti kritériumfüggvényt, mind a mellékfeltételt magában foglalja:

$$L(\mathbf{w}, \lambda) = C(\mathbf{w}) + \lambda g(\mathbf{w}) \quad (2.131)$$

A feltételes szélsőérték a $\nabla L(\mathbf{w}, \lambda) = 0$ megoldásából adódik.

Amennyiben nem egyenlőség, hanem egyenlőtlenség formájában vannak mellékfeltételeink, vagy az egyenlőség típusú feltételek mellett egyenlőtlenség típusú mellékfeltételeink is vannak, a Karush-Kuhn-Tucker (KKT) elmélet alkalmazásával kaphatunk megoldást, amely elmélet a Lagrange módszer kiterjesztésének tekinthető.

Mind a Lagrange, mind a KKT elmélet rövid összefoglalását a Függelékben adjuk meg.

5.2. Sztochasztikus szélsőérték-kereső eljárások

A gradiens módszerek hatékony működéséhez a kritériumfelületről bizonyos tulajdonságokat fel kell tételezni. A kedvező konvergencia-sebességet általában csak kvadratikusan hibafelület mellett biztosítják, továbbá a minimumhely elérése csak akkor biztos, ha a felületen nincsenek lokális minimumok (unimodális felület).

A neurális hálózatokra tipikusan a nem kvadratikusan hibafelületek, a lokális minimumok fennálltának lehetősége és általában a konvergencia szempontjából kedvezőtlen tulajdonságok a jellemzők. A lokális minimumhelyekben való bennragadás elkerülésére sztochasztikus gradiens eljárásokat dolgoztak ki, amelyek fő jellemzője, hogy a kritériumfelületen valamilyen – általában kis és az algoritmus során változó – valószínűséggel a felfelé mozgást is megengedik, lehetővé téve a lokális minimumból való kiszabadulást. A sztochasztikus gradiens eljárások célja, hogy a tanulási folyamat során véletlen zaj felhasználásával lehetővé tegyék, hogy a hálózat a lokális minimumból kikerüljön. Ennek egy egyszerű, heurisztikus módja, ha olyan esetekben, amikor a lokális minimumba kerülés gyanúja fennáll, vagyis a tanulási folyamat több lépése sem eredményez hibacsökkenést, a hálózat aktuális súlyaihoz zajt adunk. Additív zajként általában egyenletes vagy Gauss eloszlású valószínűségi változó aktuális értékeit tekintjük. A súlyoknak az ilyen véletlenszerű "megrázása" kiugrathatja a hálózatot a lokális minimumból.

A zaj felhasználás egy további lehetséges módja, ha az eddigi kritériumfüggvényt az alábbiak szerint módosítjuk:

$$\tilde{C}(\mathbf{w}) = C(\mathbf{w}) + c(k) \sum_{i=1}^N w_i n_i(k), \quad (2.132)$$

ahol $\tilde{C}(\mathbf{w})$ a módosított kritériumfüggvény, az $n_i(k)$ értékek az $\mathbf{n}(k) = [n_1(k), n_2(k), \dots, n_N(k)]^T$ független fehérzaj komponensei, a w_i -k pedig a $\mathbf{w}$ súlyvektor komponensei. A $c(k)$ lépésfüggő együttható az additív zaj amplitúdóját határozza meg; ezt olyan módon kell megválasztani, hogy a k diszkrét időindex előrehaladtával tartson nullához. Az így definiált kritériumfüggvény esetén a súlymódosítás az i -edik súlykomponens esetében:

$$w_i(k+1) = w_i(k) - \mu \left(\frac{\partial C(\mathbf{w})}{\partial w_i} + c(k) n_i(k) \right). \quad (2.133)$$

Az eljárás egyik kulcskérdése a $c(k)$ paraméter megválasztása. Legtöbbször $c(k)$ k monoton csökkenő függvénye, azonban néhány esetben bonyolultabb módon változtatjuk $c(k)$ -t. Pl. $c(k)$ értékét átmenetileg növelhetjük, ha észrevesszük, hogy lokális minimumba kerültünk. A legegyszerűbb választás az alábbi lehet:

$$c(k) = \beta e^{-\alpha k}, \quad (2.134)$$

ahol $\beta > 0$ és $\alpha > 0$. Ekkor β a kezdeti ($k=0$) zajamplitúdót határozza meg, míg α az additív zaj csillapodásának mértéke.

A sztochasztikus szélsőérték-kereső algoritmusok között kell megemlíteni a szimulált lehűtést (*simulated annealing*, SA) [Aar89] is, amely szintén megengedi, hogy a hibafelületen valamilyen valószínűséggel felfelé, a nagyobb hiba irányában is elmozduljunk. A felfelé mozgás valószínűsége a minimalizáló eljárás előrehaladtával fokozatosan csökken, mígnem a minimum közelében 0-hoz tart. A szimulált lehűtés előnye, hogy a globális minimumhoz konvergál 1 valószínűséggel, hátránya az eljárás nagyfokú lassúsága. A szimulált lehűtessel részletesebben a 11. fejezetben foglalkozunk.

Tulajdonképpen az LMS eljárások is felfoghatók sztochasztikus gradiens módszerekként, hiszen a pillanatnyi gradienszen alapulnak, ami a valódi gradiens zajos értékének feleltethető meg. Láttuk is az LMS eljárás tárgyalásánál, hogy konstans μ mellett – még ha a μ a konvergenciát biztosító tartományon belül is van – a $\mathbf{w}^*$ megoldásvektort nem fogjuk elérni, az eljárás nem áll le, hanem a súlyvektor $\mathbf{w}^*$ valamekkora környezetében fog "vándorolni". Az eljárás konvergenciája csak akkor biztosítható, ha konstans μ helyett lépésenként csökkenő μ -t alkalmazunk. A lépésfüggő μ megválasztására a sztochasztikus approximáció ad támpontot. A sztochasztikus approximáció ún. *Robbins-Monro*-algoritmus egy sztochasztikus zérus-kereső módszer, amely zajos megfigyelések ellenére is biztosítja a konvergenciát (ld. még [Bat92]). A konvergencia akkor biztosítható, ha a lépésfüggő μ -re betartjuk az alábbi feltételeket:

$$\sum_{k=1}^{\infty} \mu(k) = \infty, \text{ és } \sum_{k=1}^{\infty} \mu^2(k) < \infty. \quad (2.135)$$

Megjegyezzük, hogy ezek a feltételek kielégíthetők pl. $\mu(k) = \mu_0/k$ választással (ahol μ_0 alkalmasan megválasztott konstans).

Bár a felügyelt tanítású hálózatoknál többségében determinisztikus vagy sztochasztikus gradiens eljárásokat alkalmaznak, számos esetben egyéb, nem a hibafület gradiensén alapuló eljárást használnak. A gradiens eljárások hátránya ugyanis – akár determinisztikus, akár sztochasztikus eljárásról van szó –, hogy a kritériumfelületről valamilyen a priori ismerettel kell rendelkezünk. Az általános, nem gradiensén alapuló kereső eljárások viszont nem tételeznek fel semmit a kritériumfelületről. A keresés során valamilyen véletlen (egyes esetekben determinisztikus) algoritmus szerint feltérképezik a felületet. A legkézenfekvőbb megoldás a kritériumfelület teljes feltérképezése (a megoldási tér teljes végigtesztelése), azaz minden lehetséges paraméterérték (súlyvektor) mellett a kritériumfüggvény meghatározása. Mivel csak véges számú pontban határozható meg a kritériumfüggvény értéke, a paraméterek változtatása diszkrét lépésekben történhet. Ez azt jelenti, hogy a súlytérre egy megfelelő finomságú (nem feltétlenül egyenletes felbontású) rácsot helyezünk és a kritériumértékeket a rácpontokban határozzuk meg. Ez az ún. "kimerítő (teljes) keresés" (*exhaustive search*) megfelelően finom felbontás mellett biztosan megtalálja a globális minimumot.

A dimenziószám növelésével viszont a rácpontok száma hatványozottan növekszik, ezért a legegyszerűbb esetektől eltekintve – tehát szinte minden valós problémánál – az eljárás elfogadhatatlan mértékű számítás igényel. A nagy számításigény mérséklésére a determinisztikus teljes keresés helyett egyéb kereső eljárások alakultak ki, melyek a kritériumfelület feltérképezésének eltérő módját alkalmazzák. Ezen eljárások közös jellemzője, hogy a kritériumfelület feltérképezését nem előre meghatározott módon, hanem lépésről lépésre, a pillanatnyi állapotától függően végzik. A következőkben a (nem gradiens alapú) kereső eljárások két, jelentősen eltérő családjával foglalkozunk.

5.2.1. Véletlen keresés

Véletlen kereső módszereknél a paraméterértékek véletlenszerű megváltoztatásával próbálkozunk. Ha egy próbálkozás eredményeképpen kapott paraméterhez tartozó kritériumérték az előző pontbelinél kisebb, az új pontot tekintjük a következő lépés kiinduló pontjának, ellenkező esetben a régi paraméterértéket tartjuk meg és a keresést tovább folytatjuk. A véletlen keresési eljárásoknak – a hatékonyság növelése érdekében – különböző változatait dolgozták ki. Az alábbiakban a véletlen keresés alapeljárásának és egy módosított változatának fő lépéseit foglaljuk össze.

Mátyás véletlen optimalizálási módszere [Mát65] 1 valószínűséggel biztosítja a globális minimumhoz való konvergenciát, abban az esetben, ha a keresési tér kompakt (korlátos és zárt). Jelölje W a kompakt keresési teret, $\mathbf{w}(k)$ pedig a keresési tér egy pontját.

A módszer a következő lépésekből áll:

1.

válasszunk egy kezdeti pontot: $k=0$, $\mathbf{w}(k) \in W$,

2.

generáljunk egy nulla várható értékű, Gauss véletlen vektort: $\xi(k)$, módosítsuk a véletlen vektorral a pillanatnyi pont értékét: $\mathbf{w}(k) + \xi(k)$ és ha $\mathbf{w}(k) + \xi(k) \in W$, ugorjunk a 3. lépésre, egyébként ugorjunk a 4. lépésre,

3.

ha $C[\mathbf{w}(k) + \xi(k)] < C[\mathbf{w}(k)]$, akkor legyen $\mathbf{w}(k+1) = \mathbf{w}(k) + \xi(k)$, vagyis a módosított pontot (paraméterértéket) tekintjük a megoldás következő közelítésének, ha $C[\mathbf{w}(k) + \xi(k)] \geq C[\mathbf{w}(k)]$, akkor a paraméterértéket ne változtassuk meg, tehát $\mathbf{w}(k+1) = \mathbf{w}(k)$ maradjon,

4.

növeljük meg k értékét: $k=k+1$ és folytassuk az eljárást a 2. lépéssel.

A konvergencia gyorsítható, ha a 3. lépést az alábbival helyettesítjük:

3.

ha $C[\mathbf{w}(k) + \xi(k)] < C[\mathbf{w}(k)]$, akkor $\mathbf{w}(k+1) = \mathbf{w}(k) + \xi(k)$ és

$\mathbf{m}(k+1) = 0,2\mathbf{m}(k) + 0,4\xi(k)$ legyen,

ha $C[\mathbf{w}(k) + \xi(k)] \geq C[\mathbf{w}(k)]$ és $C[\mathbf{w}(k) - \xi(k)] < C[\mathbf{w}(k)]$, akkor:

$\mathbf{w}(k+1) = \mathbf{w}(k) - \xi(k)$ és $\mathbf{m}(k+1) = \mathbf{m}(k) - 0,4\xi(k)$ legyen,

egyébként pedig $\mathbf{w}(k+1) = \mathbf{w}(k)$ és $\mathbf{m}(k+1) = 0,5\mathbf{m}(k)$,

ahol $\mathbf{m}(k)$ a Gauss valószínűségi vektorváltozó várható értéke a k -adik lépésben.

A módosított eljárás tehát változó várható értékű Gauss véletlen vektort használ. Kezdetben a várható érték itt is nulla: $\mathbf{m}(0)=\mathbf{0}$. Bizonyítható [Sol81], hogy a módosított eljárás is 1 valószínűséggel konvergens és a globális minimumhoz konvergál, ha a keresési tér kompakt.

Neurális hálózatokra való alkalmazásnál általában nem biztosítható, hogy a keresési tér kompakt legyen. A súlyértékek nem szükségszerűen korlátos tartományban vehetnek fel csak értékeket. Ez azt jelenti, hogy a véletlen keresési módszerek nem feltétlenül konvergálnak a globális minimumhoz. Abban az esetben azonban, ha a keresési teret korlátozzuk, akkor e tartományon belül a globális minimumhoz való 1 valószínűségű konvergencia igaz. E nehézség ellenére, alkalmazástól függően a véletlen keresési módszerekkel más eljárásoknál gyorsabb konvergencia is elérhető.

A véletlen keresési módszerek sok esetben sikerrel alkalmazhatók egyéb szélsőérték-kereső eljárásokkal, pl. gradiens alapú módszerekkel kombinálva. Ilyenkor a gradiens módszert alkalmazzuk mindaddig, amíg a kritériumfüggvény értékének csökkenése egy adott korlátnál kisebb nem lesz, majd áttérünk a véletlen keresésre. A kritériumcsökkenés lelassulása vagy megállása annak jele, hogy a hibafelület meredeksége nagyon lecsökkent, esetleg lokális minimumhelyet értünk el. A minimumhelyben való bennragadás elkerülésére alkalmazzuk most a véletlen keresést, mindaddig, amíg a hibafelületen meredekebb részre nem érünk ismét, ahol célszerűen visszatérünk a gradiens alapú módszerre.

5.2.2. Genetikus algoritmusok

A genetikus algoritmusok [Hol75], [Gol89a] az evolúció számítógépes modelljei, a természetes szelekciót utánozzák. Az evolúció a természetes kiválasztódással központi szerepet tölt be a biológiában. A mesterséges intelligencia témakörében, a mesterséges intelligens rendszerek létrehozásánál az utóbbi időben egyre gyakrabban alkalmazzák a természetes kiválasztódás modelljét.

A genetikus algoritmusok sztochasztikus keresési algoritmusoknak tekinthetők, ahol a paramétertér feltérképezése és az iterációnként egyre jobb megoldások előállítása az eddigi eljárásoktól gyökeresen eltérő módon történik. Az eddigi szélsőérték-keresési eljárások – akár a gradiens módszerek, akár a véletlen keresés – a paramétertérben egy pontot mozgattak és a pont minden egyes állapotában – ami egy lehetséges megoldásnak felelt meg – kiértékeltek a kritériumfüggvényt, majd ennek eredményétől függően folytatták az eljárást. Az alapvető eltérés e két módszer között csak az egymást követő pontok meghatározásában volt.

A genetikus algoritmusok ezzel szemben nem különálló pontokban vizsgálják a kritériumfelületet, hanem egyszerre több ponton értékelik ki, tehát egy lépésben a megoldások egész halmazával dolgoznak. Genetikus algoritmusoknál a megoldások egy adott lépésben érvényes halmazát *populációnak* nevezik. A populáció elemei különböző sikerrel oldják meg a feladatot, ami szélsőérték-keresésnél azt jelenti, hogy a populáció elemei a kritériumfüggvény különböző pontjait határozzák meg. E pontok között lesznek olyanok, amelyek a globális minimumhoz közelebb esnek – ahol a kritériumfüggvény értéke kisebb –, és lesznek a megoldástól távolabb eső pontok is. A populáció elemei tehát a feladat szempontjából eltérő tulajdonságúak, képességűek. A genetikus algoritmusoktól azt várjuk, hogy a természetes szelekció mintájára egy látszólag spontán fejlődés során egy

adott populációból olyan újabb populációt, ill. olyan újabb és újabb populációkat hozzanak létre, amelyekben a jó megoldáshoz közeli megoldások egyre nagyobb számban lesznek megtalálhatók, és amelyekben a gyenge eredményt szolgáltató elemek egyre ritkulnak. Ez azt jelenti, hogy valójában a genetikus algoritmusok is egy hálót helyeznek a kritériumfelületre, azonban e háló nem rögzített felbontású, hanem lépésenként változik. Ezen túl a kritériumfüggvény kiértékelése a háló rácspontjaiban nem egyenként történik, hanem minden egyes lépésben a lépéshez tartozó háló összes rácspontjában. A kiértékelés eredményeképpen egy új háló jön létre, amelynek lehetnek közös rácspontjai az előző hálóval, de alapvető jellemzője, hogy miközben egyes régi rácspontok eltűnnek, új rácspontokkal is fog rendelkezni, és ezen rácspontok a kritériumfüggvény kedvezőbb tartományain sűrűsödnek: a rácspontok sűrűsége arányos lesz az adott tartomány jóságával. Bizonyos szempontból tehát a genetikus algoritmus a véletlen kereséssel rokonítható, hiszen itt is a teljes keresési tér valamilyen feltérképezéséről van szó. A genetikus algoritmus azonban – szemben a véletlen kereséssel – nem vakon keres. A keresés az egymást követő lépésekben egyre inkább a kritériumfüggvény minimumhelyei környezetére koncentrálódik.

Az egymást követő populációkat – biológiai mintára – generációknak nevezzük, tehát az algoritmus az egymást követő generációk során egyre jobb megoldás-halmazokat állít elő. E folyamat követhető a 2.16 ábrán. Az egydimenziós paraméterterben az egyre későbbi generációk során a populáció jobb megoldást adó elemei a kritériumfüggvény minimumhelyei közelében sűrűsödnek.

Egy populáció elemeinek tulajdonságait valamilyen módon reprezentálni kell. E reprezentáció – megint csak biológiai szóhasználatlálva – kromoszómákkal történik, amely jelen esetben olyan bitfüzéreket – stringeket – jelent, amelyekben minden egyes bit egy tulajdonságot reprezentál. Ha a string egy adott pozícióján 1 található, az adott elem a kérdéses tulajdonsággal rendelkezik, míg a 0 érték e tulajdonság hiányát jelenti. A megoldások egy populációja tehát a kromoszómák, bináris stringek egy populációját jelenti, és az algoritmus lényegében az egymást követő string-populációk fejlődése során jut el a megfelelő megoldásig. A megoldási tér (paraméter tér) minden egyes pontjának egy kromoszóma felel meg.

Az eddigiek alapján összefoglalhatjuk a genetikus algoritmusok fő jellemzőit:

- a paraméterek optimalizálása során nem az egyes paramétereket próbálja megtalálni, hanem az egész paraméterkészlettel (általában annak bináris kódolású változatával) dolgozik egyszerre,
- a keresés során nem egyetlen – az optimális – paramétert keresi, hanem a megoldások egész halmazát adja meg; ezek mindegyike jó megoldás a kritériumfüggvény szempontjából, bár nem egyformán jók,
- csak a kritériumfüggvény egyes értékeit használja, de a kritériumfüggvényről semmit sem tételez fel (pl. unimodalitás, folytonosság, differenciálhatóság), hiszen nem alkalmaz deriválást (a véletlen keresés módszerre hasonlóan ebből a szempontból is),
- valószínűségi átmenetekkel dolgozik.

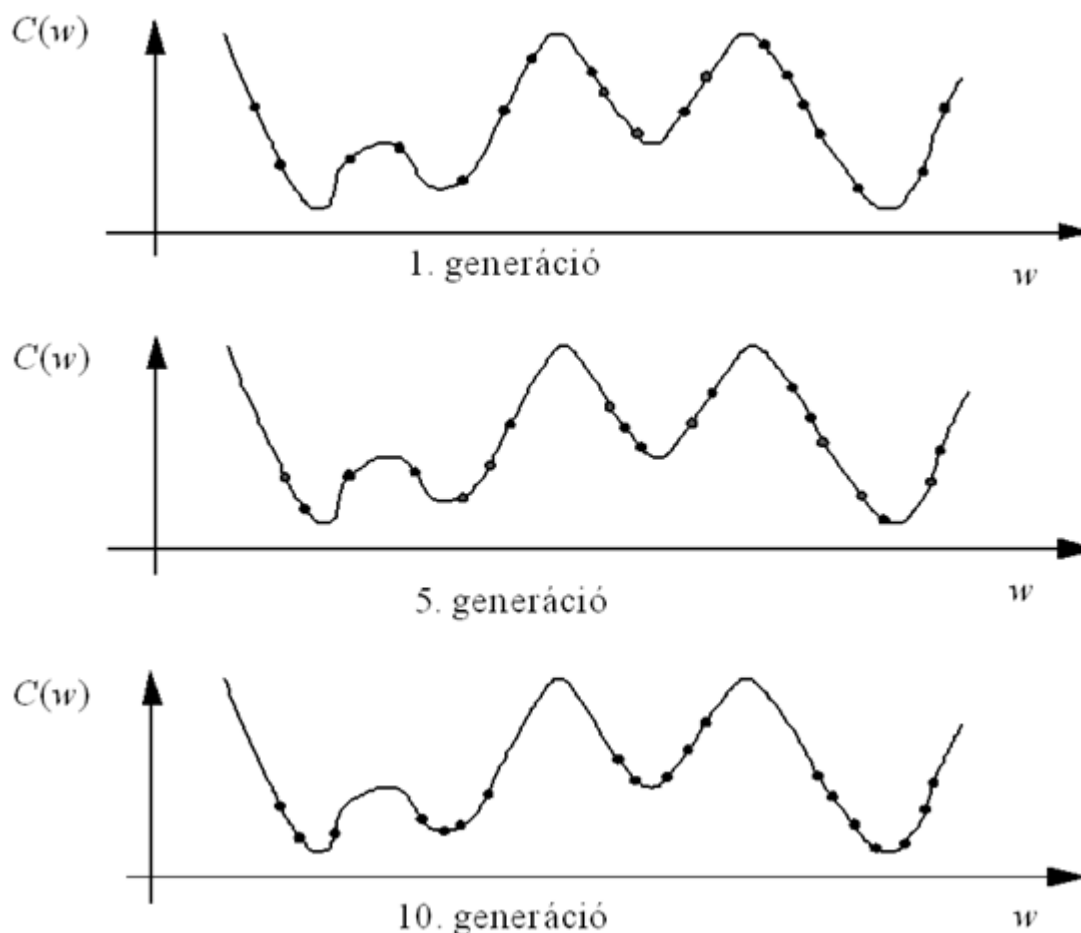
Hogy az egymást követő populációk egyre jobb tulajdonságú stringeket tartsanak, a stringek valamilyen kiértékelésére van szükség. Minden stringhez hozzárendelhetünk egy "jóság" (*fitness*) értéket, ami lényegében annak mértéke, hogy egy adott string által képviselt megoldás milyen kritériumfüggvény-értéket eredményez. Minél kisebb a kritériumfüggvény-érték, annál jobb a megoldás, tehát a fitness érték annál nagyobb.

A genetikus algoritmus a kromoszómák generációival dolgozik. A megfelelő működést az jelenti, ha az újabb generációk átlagos jósága (fitness-értéke) egyre nagyobb, de legalábbis nem csökken. Tehát az egyes lépések során a jó kromoszómák által hordozott információt kell tovább örökíteni a következő generációra.

A generációk közötti átmenet valószínűségi operátorokkal történik. A legegyszerűbb genetikus algoritmus a következő alap-operátorokkal rendelkezik:

- reprodukció,
- keresztezés,
- mutáció.

2.16. ábra - A genetikus algoritmus szélsőérték-kereső mechanizmusa



A **reprodukció** (*reproduction*) az a folyamat, amikor egy string a következő generáció részeként is megjelenik, vagyis egy megoldás a következő generáció megoldás-halmazának is része marad. Egy adott stringnél a reprodukció, vagyis a túlélés bekövetkezésének valószínűsége a string jóságával van kapcsolatban. Minél nagyobb egy kromoszóma jósága az egy populációba tartozó kromoszómák átlagos jóságához viszonyítva, annál nagyobb számban fog reprodukálódni a kromoszóma, vagyis annál nagyobb számban fog előfordulni a következő generációban. A reprodukciónak ezt a módját arányos kiválasztási mechanizmusnak nevezzük.

A **keresztkezésnél** (*crossover*) egy generáció két kromoszómája – a szülő kromoszómák – a tulajdonságaik keresztezése révén hoznak létre új tulajdonságú utódokat. Ilyenkor a szülők bitmintái kereszteződnek új bitkombinációkat hozva létre. Az utódok bitkombinációi részben az egyik szülő, részben a másik szülő bitkombinációit fogják tartalmazni. A keresztezés operátor legegyszerűbb formája az ún. egyponthos keresztezés, amikor az adott generáció bitfűzérei páronként úgy hoznak létre utódokat, hogy egy bitfűzér-pár mindkét tagja egy véletlenszerűen kiválasztott pontnál szétválik, és az így létrejövő részstringek keresztbe kapcsolódva hoznak létre új stringeket. A folyamatot a 2.17 ábrán követhetjük. Megjegyezzük, hogy a szülő kromoszómákból az utód kromoszómák létrehozásánál a keresztezés a legegyszerűbb operáció. Bár a természetben nem található erre minta, genetikus algoritmusoknál a szülő kromoszómák komplex összekeverése is lehetséges. Ezen komplex operációkat *rekombinációnak* hívják, amelyek egy speciális, egyszerű esete a keresztezés [Rad93].

A **mutáció** (*mutation*) során egy string egy bite véletlenszerűen, általában igen kis valószínűséggel megváltozik. A mutáció következtében teljesen új bitkombinációk jöhetnek létre, így megakadályozhatják a további, fejlődésre képtelen, egyöntetű populáció létrejöttét. Az új bitkombinációk a megoldási tér olyan területeit is feltérképezhetik, amelyekre az eddigi populációk nem terjedtek ki.

Fentiek alapján az algoritmus a következő lépésekből áll:

1. egy kromoszóma populáció véletlen generálása,
2. a populáció minden kromoszómájának kiértékelése (a fitness értékek meghatározása),

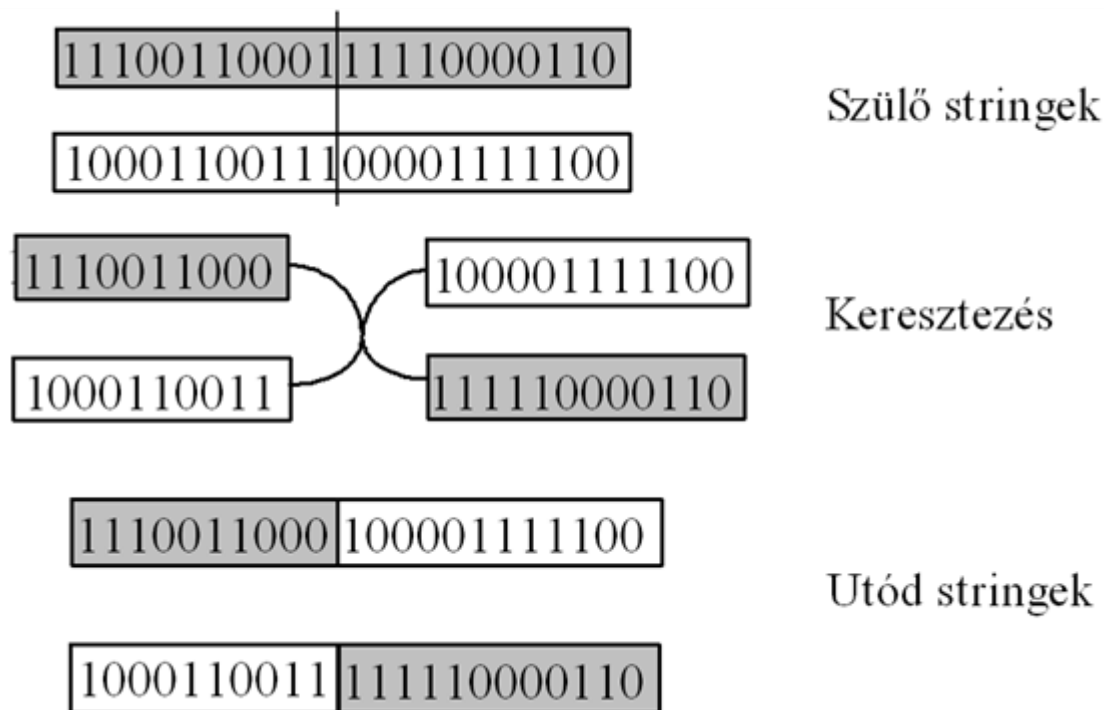
3. új kromoszómák létrehozása a fenti három operátorral,
4. az új kromoszómák kiértékelése,
5. az új populáció (a következő generáció) létrehozása azáltal, hogy a régi kromoszómák közül a leggyengébbeket az újak közül a legjobbakkal váltjuk fel,
6. leállás: ha az eredmény elég jó, nem folytatjuk az eljárást.

A három operátor hatását az alábbi egyszerű példán követhetjük.

2.4 Példa

Tekintsünk egy 4 kromoszómából álló populációt, melyben az egyes kromoszómák (k_i) 10 bitesek. Legyen az egyes bitstringek jósága (f_i , $i = 1, 2, 3, 4$) rendre 0,9; 0,6; 0,6 és 0,3. Ekkor a populáció átlagos jósága $\bar{f} = 0,6$ az egyes kromoszómák jóságainak az átlagos jósághoz viszonyított értékei ($f_i / \bar{f}$) pedig rendre 1,5; 1,0; 1,0 és 0,5-re adódnak. Ha a kiválasztás ezen értékekkel arányos valószínűség szerint történik, akkor az egyes stringek reprodukciójának valószínűségei ($p_i = (1/n) \cdot (f_i / \bar{f})$, $n=4$) 0,375; 0,25; 0,25 és 0,125.

2.17. ábra - A keresztezés folyamata.



A szelekció – arányos kiválasztási mechanizmust alkalmazva – ezek után a következőképpen zajlik. Határozzuk meg az egyes kromoszómák relatív jóságait – a p_i értékeket – a populációt jellemző összeg-jósághoz viszonyítva. Osszuk fel a $[0,1]$ intervallumot ezen relatív jóságoknak megfelelő intervallumokra. Generáljunk a $[0,1]$ intervallumban egyenletes eloszlású véletlen számokat és attól függően reprodukáljuk az egyes kromoszómákat, hogy a véletlen szám a $[0,1]$ intervallum mely részintervallumába esik.

Mivel a részintervallum-hosszak arányosak az egyes kromoszómák relatív jóságával, a reprodukció valószínűsége is arányos lesz ezekkel az értékekkel, vagyis az arányos kiválasztási mechanizmust valósítottuk meg és az egyes kromoszómák p_i -vel arányos számban reprodukálódnak. A szelekciót követően a kromoszómák átlagos jósága növekedett: $\bar{f} = 0,75$.

A kromoszómák alakulását az egyes operátorok alkalmazása után a 2.1 táblázat mutatja. Ugyancsak a táblázatból látható a keresztezés (egy pontos: az ötödik és a hatodik bit között) és a mutáció hatása. A táblázatból az is látható, hogy a keresztezés hatására az eddigieknél jobb képességű egyed ($P_3: f_1=1$) keletkezhet, de az is, hogy a keresztezés a populáció $\bar{f}$ lagos jóság $\bar{f}$ nak stagnálását, a mutáció pedig az átlagos jóság stagnálását ill. csökkenését is eredményezheti ($\bar{f}=0,75$ ill. $\bar{f}=0,6$).

5.2.3. A szkéma elmélet

A genetikus algoritmusok valódi teljesítőképessége jobban megérthető, ha bevezetjük a *szkéma* vagy *hasonló mintázat* fogalmát [Hol75]. Ha két vagy több string hasonló képességekkel rendelkezik, akkor a stringek egy vagy több pozíciójában egyezés található. A stringek közötti lényeges hasonlóságok kezelésére vezethető be a hasonló mintázat vagy szkéma elnevezés, amely egy olyan részstringet jelöl, amelyben a 0 és 1 értékű biteken kívül don't care, közömbös bitek is lehetnek. Pl. a 8 hosszúságú $1*0*01**$ szkéma mindazon stringeket képviselni fogja, amelyekben 1 található az első és a hatodik pozícióban, 0 a harmadikban és az ötödikben. A szkémák hossza megegyezhet a string hosszával, de legtöbbször annál kisebb. A szkéma fogalmának bevezetésével lehetőség nyílik annak bemutatására, hogy generációról generációra a jó tulajdonságú stringek száma – vagyis azon stringek száma, amelyekben jó tulajdonságú szkéma található – az egyes populációkban hogyan növekszik.

2.1. táblázat - A kromoszómák alakulása az egyes operációk után

Kromoszóma	Jóság	Populáció
1 1 1 1 1 1 0 1 1	0,9	Kiinduló (P1)
0 1 1 0 1 0 1 0 1 1	0,6	
1 0 0 0 0 1 1 1 1 1	0,6	
0 0 0 0 0 1 1 1 0 0	0,3	
1 1 1 1 1 1 1 0 1 1	0,9	Szelekció után (P2)
1 1 1 1 1 1 1 0 1 1	0,9	
0 1 1 0 1 0 1 0 1 1	0,6	
1 0 0 0 0 1 1 1 1 1	0,6	
1 1 1 1 1 1 1 1 1 1	1,0	Keresztezés után (P3)
1 1 1 1 1 1 1 0 1 1	0,9	
0 1 1 0 1 0 1 0 1 1	0,6	
1 0 0 0 0 1 1 0 1 1	0,5	
0 1 1 1 1 1 1 0 1 1	0,9	Mutáció után (P4)
1 1 1 1 1 1 1 0 1 1	0,6	
0 1 1 0 1 1 1 0 1 1	0,6	
1 0 0 0 0 1 1 0 1 1	0,3	

Vizsgáljuk meg, hogyan hatnak az egyes operátorok egy S szkéma előfordulási gyakoriságára a generációk függvényében. Ehhez vezessük be a következő jelöléseket: jelölje $m(S,k)$ azon stringek számát, amelyek a k -adik generációban tartalmazzák az S szkémát, $o(S)$ a szkéma *rendjét* (order), ami a szkémán belüli rögzített értékű bitek száma és $\delta(S)$ a szkéma *meghatározó hosszát* (definitive length), ami az első és utolsó rögzített bit

pozíciói közötti távolság. Pl. az előző szkéma (1*0*01**) estében $o(S)=4$, $\delta(S)=5$. A reprodukció során egy kromoszóma annál nagyobb valószínűséggel kerül kiválasztásra, minél nagyobb az adott kromoszóma jósága az egész populáció kromoszómáinak átlagos jóságához viszonyítva.

Ha f_i az i -edik kromoszóma jóságát jelöli, a reprodukció során egy stringet (kromoszómát)

$$p_i = \frac{f_i}{\sum_j f_j} \quad (2.136)$$

valószínűséggel választunk ki. Ekkora valószínűséggel lesz tehát az i -edik kromoszóma eleme a következő generációnak. Ha kiválasztunk n kromoszómát a régi populációból, akkor az n elemű új populációban egy S szkémát tartalmazó stringek számát a következő összefüggés adja meg:

$$m(S, k+1) = m(S, k) n \frac{f(S)}{\sum_j f_j} = m(S, k) \frac{f(S)}{\bar{f}} \quad (2.137)$$

mivel $\bar{f} = \frac{1}{n} \sum_j f_j$ az n elemű új populáció átlagos jósága. Itt $f(S)$ az S szkéma

jóságát, vagyis azon stringek átlagos jóságát jelöli, melyekben az S szkéma megtalálható. Ez azt jelenti, hogy azok a szkémák, amelyek jósága nagyobb, mint a populáció átlagos jósága, az új generáció kromoszómáiban növekvő számban lesznek megtalálhatók, míg az átlagosnál gyengébb képességű szkémák jelenléte az új generációban csökken. Ha $f(S)=(1+c)\bar{f}$, ahol c konstans, akkor k generációt követően az S szkémát tartalmazó kromoszómák száma:

$$m(S, k) = m(S, 0) (1+c)^k, \quad (2.138)$$

vagyis a szkémák túlélése a generációk függvényében exponenciálisan nő ($c>0$) vagy csökken ($c<0$).

A szkémák előfordulási gyakoriságát befolyásolja a további két operátor is. Mind a kereszteződés, mind a mutáció következtében egy szkémát alkotó bitstring megváltozhat, a szkéma széteshet, aminek következtében egyes kromoszómákból az addig meglévő szkéma eltűnik. Egy szkéma túléli a kereszteződést, ha a kereszteződési pont kívül esik a szkéma bitszoportján. Ha l a kromoszóma hossza és P_c a keresztezés bekövetkezésének valószínűsége, az egy pontos keresztezésnél a túlélési valószínűség:

$$p_{s,c} \geq 1 - P_c \frac{\delta(S)}{l-1}. \quad (2.139)$$

A mutáció is elronthat egy szkémát azáltal, hogy a szkéma egy rögzített bitjét megváltoztatja. Ha P_m egy bit véletlen megváltoztatásának valószínűsége, akkor a túlélési valószínűség:

$$p_{s,m} = (1 - P_m)^{o(S)} \approx 1 - o(S) P_m \quad (2.140)$$

mivel $P_m \ll 1$.

Feltételezve, hogy a keresztezés és a mutáció azon hatása, hogy egy szkémát elrontanak egymástól független, valamint elhanyagolva a $P_m P_c$ -t tartalmazó tagot, a három alap-operátor hatását figyelembe véve egy adott S szkéma előfordulása a következő generációban az alábbi összefüggéssel adható meg:

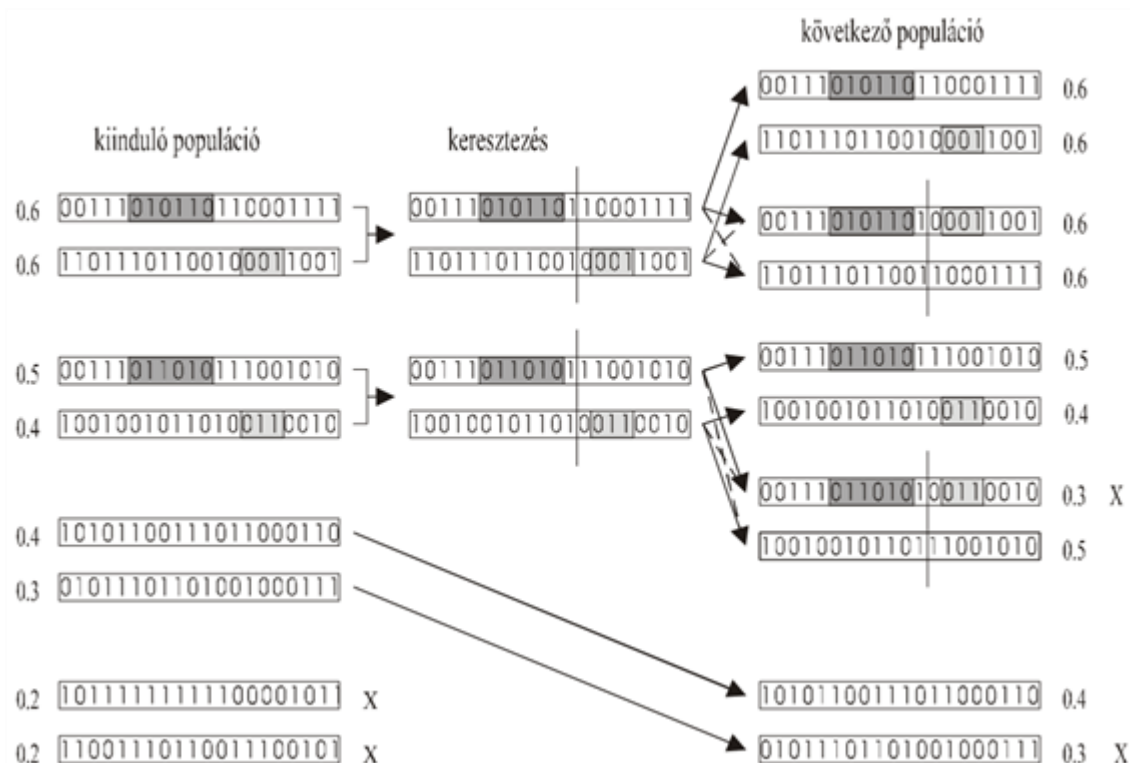
$$m(S, k+1) \geq m(S, k) \frac{f(S)}{\bar{f}} \left[1 - P_c \frac{\delta(S)}{l-1} - o(S) P_m \right]. \quad (2.141)$$

A (2.141) összefüggést, ami Holland szkéma elméletének eredménye, a genetikus algoritmus alaptételének nevezzük.

A mutáció, illetve a keresztezés hatására azon szkémák esnek szét nagyobb valószínűséggel, amelyek rendje, illetve meghatározó hosszúsága nagyobb. Ha egy szkémát tartalmazó kromoszóma jósága az átlag jósnál nagyobb és a szkéma rövid, továbbá, ha szkéma rendje is kicsi, akkor a szkémát tartalmazó kromoszómák száma a generációk függvényében közel exponenciálisan nő.

A reprodukció és a keresztezés operátorok hatása az evolúció során a 2.18 ábrán követhető. Az ábrán eltérő árnyalatú bitsorozatok jelölik a szkémákat. Az egyes bitsorozatok mellé írt értékek a stringek jóságát jelentik. A legkevésbé megfelelő stringek – amelyek jósága a legkisebb – nem kerülnek át a következő generációba; az ábrán X jelöli ezen kromoszómákat. Az ábra azt is illusztrálja, hogy egy stringen belül több szkéma is előfordulhat. Ha több, kedvező tulajdonságokat reprezentáló szkéma a keresztezés hatására egy utód stringben együtt jelenik meg, lehetséges, hogy a keresztezés hatására a szülő stringeknél nagyobb jóságú utód jön létre.

2.18. ábra - A szkémák megjelenése az egyes generációk populációiban



A genetikus algoritmusok alkalmazása különösen ott célszerű, ahol a keresési tér nagy, komplex és kevésbé ismert, illetve, ahol a kritériumfüggvény jellege olyan, hogy más, pl. gradiens alapú eljárások nem alkalmazhatók. A keresési tér mérete ugyanakkor befolyásolja a genetikus algoritmus számításigényét, hiszen nyilvánvaló, hogy nagyobb keresési tér több elemű populációt igényel.

Tipikus feladatoknál a populáció elemszáma néhány száztól több ezerig is terjedhet. A viszonylag nagy elemszám számos területen gátja a genetikus algoritmusok alkalmazásának. A populáció elemszámának meghatározásán túl fontos kérdés az algoritmus egyéb paramétereinek megválasztása. Ilyen további paraméterek pl. a keresztezés típusa (egypontos vagy többpontos keresztezés) és valószínűsége P_c , a mutáció valószínűsége P_m , stb. A paraméterek megválasztására általános eredmények jelenleg nincsenek. A paraméterválasztás alkalmazástól függően, tapasztalati úton történik. Ugyancsak alkalmazástól függ, hogy hogyan képezzük le egy adott problémánál a feladat paramétereit számfüzérekké, kromoszómákká.

Fontos kérdés még az algoritmus leállítása. A leállítás megtörténhet adott számú generáció után, illetve, ha az eredmény már elég jó, vagyis az utolsó generáció kromoszómáinak átlagos jósága egy adott értéknél nagyobb. Az algoritmust azonban le kell állítani akkor is, ha egy populáció stringjei nagymértékű hasonlóságot, homogenitást mutatnak. Ebben az esetben – függetlenül a populáció átlagos jóságától – a kromoszómák jósága a további generációk során már nem fog javulni, az algoritmus megreked.

A túl korai konvergenciát eredményező egyöntetű populáció kialakulásának esélyét a reprodukció előbb említett módja – vagyis amikor egy kromoszóma továbbélése annak jóságával arányos – növeli. Ennek megakadályozására számos módosított eljárást dolgoztak ki, melyek közös jellemzője, hogy nagyobb esélyt adnak a rossz stringek továbbélésének. E módszerek részletes ismertetésére itt nincs módunk, az érdeklődő olvasó a részleteket az irodalomban pl. [Gre87], [Gal89], [Mon89] megtalálja.

Feladatok

2.1 Mutassa meg, hogy az N -dimenziós térben a lineáris szeparáló függvények VC-dimenziója $h=N+1$.

2.2 Adjon szemléletes példát, amely azt mutatja, hogy az $y = \sin(ax)$, $a > 0$ függvények VC-dimenziója végtelen.

2.3 Tekintsük a négyszögletes indikátorfüggvények osztályát, ahol egy indikátorfüggvényt a következőképpen definiálunk:

$$f(\mathbf{x}, \mathbf{c}, \mathbf{d}) = 1 \quad \text{akkor és csak akkor, ha } |x_i - c_i| \leq d_i \quad i = 1, 2, \dots, N$$

ahol $\mathbf{c}$ egy középpont-vektor, $\mathbf{d}$ pedig a koordináta tengelyekkel párhuzamos elhelyezkedésű négyszög szélességvektora. Mutassa meg, hogy ennek a függvényosztálynak a VC-dimenziója $h=2N$.

2.4 Mutassa meg, hogy a legmeredekebb lejtő módszerrel dolgozó szélsőérték-kereső eljárás konvergens, ha $0 < \mu < \frac{1}{\lambda_{\max}}$. Van-e a konvergenciasebesség szempontjából optimális μ ? Ha igen, határozza meg.

2.5 Az LMS eljárás a pillanatnyi gradiens alapján dolgozik. Mutassa meg, hogy az LMS eljárás eredménye torzítatlan becslése a legmeredekebb lejtő eljárással kapott eredménynek, vagyis a súlyvektor várható értéke a Wiener-Hopf megoldáshoz tart.

2.6 Az LMS eljárásnál a bemenet autokorrelációs mátrixa a következő:

$$\mathbf{R} = \begin{bmatrix} 1 & 0,5 \\ 0,5 & 1 \end{bmatrix}$$

Határozza meg a tanulási tényező, μ konvergenciatartományát.

2.7 A következő regularizált kritériumfüggvényt alkalmazza:

$$C_r(\mathbf{w}) = \sum_k (d^{(k)} - y^{(k)})^2 + \lambda \sum_i w_i^2$$

ahol $d^{(k)}$ a kívánt válasz és $y^{(k)}$ a kimenet a k -adik tanítóponton $k=1, \dots, P$. Határozza meg a pillanatnyi gradiens alapú tanítási összefüggést, ha egy $y = \text{sgm}(\mathbf{w}^T \mathbf{x})$ kapcsolatot megvalósító rendszer paramétervektorát kell tanítania.

2.8 Hogyan befolyásolja az LMS eljárás konvergencia-sebességét a $\lambda_{\max} / \lambda_{\min}$ arány, ha $\lambda_{\max}$ és $\lambda_{\min}$ a bemeneti jel autokorrelációs mátrixának legnagyobb illetve legkisebb sajátértéke?

2.9 Milyen esetben és hogyan alkalmazható a Karhunen-Loève transzformáció az LMS eljárás konvergencia-sebességének növelésére?

2.10 A genetikus algoritmus alaptétele ((2.141) egyenlet) megadja az S szkémát tartalmazó kromoszómák számának alakulását a generációk függvényében. Az összefüggés arányos kiválasztási mechanizmust alkalmazó reprodukció mellett származtatható. Definiáljon más kiválasztási mechanizmust és határozza meg a megfelelő módosított alaptételt.

2.11 Genetikus algoritmus szimulációjával határozza meg az $f(x) = x + |\sin 32x|$ függvény maximális értékét, ha $x \in [0, \pi]$.

3. fejezet - Az elemi neuron

A fejezet célja, hogy részletesen bemutassa a leggyakrabban alkalmazott, ellenőrzött tanítással tanítható hálózatok alapelemét, az elemi neuront, valamint a neuron szabad paramétereinek meghatározását eredményező tanuló eljárásokat. Az itt bemutatásra kerülő elemi neuron a természetes (biológiai) neuron, az idegsejt bizonyos tulajdonságait megragadó modelljének is tekinthető.

Amikor elemi neuronról beszélünk, valójában két neuron-modellről van szó, melyek annak ellenére, hogy a felépítésük gyakorlatilag azonos, mind tanításukban, mind működésükben különböznek. A fejezet a közös felépítés ismertetésén túl bemutatja a kétféle tanuló eljárást, és azt is, hogy a kétfajta neuron mennyiben rendelkezik eltérő képességekkel.

Mindkét neuronmodell a 2. fejezetben bemutatott, statikus, memória nélküli

$$y = f(\mathbf{w}^T \mathbf{x}) \quad (3.1)$$

nemlineáris leképezést megvalósító hálózat, amely először a bemeneteire kerülő jelek súlyozott összegét képezi, majd a súlyozott összegre egy nemlineáris függvényt alkalmazva állítja elő a skalár kimenetet. A súlyozó együtthatók vektora, $\mathbf{w}$ képezi a neuron szabadon beállítható paramétereit, mely paraméterek megfelelő megválasztásával, beállításával módosíthatjuk a neuron által megvalósított leképezést.

A kétféle elemi neuront szinte egyidőben javasolták, mint neuron modelleket: Frank Rosenblatt 1956-ban javasolta [Ros58] a róla elnevezett Rosenblatt perceptront, míg Bernard Widrow 1959-ben [Wid60] az adaline-t (adaline= adaptive linear neuron).

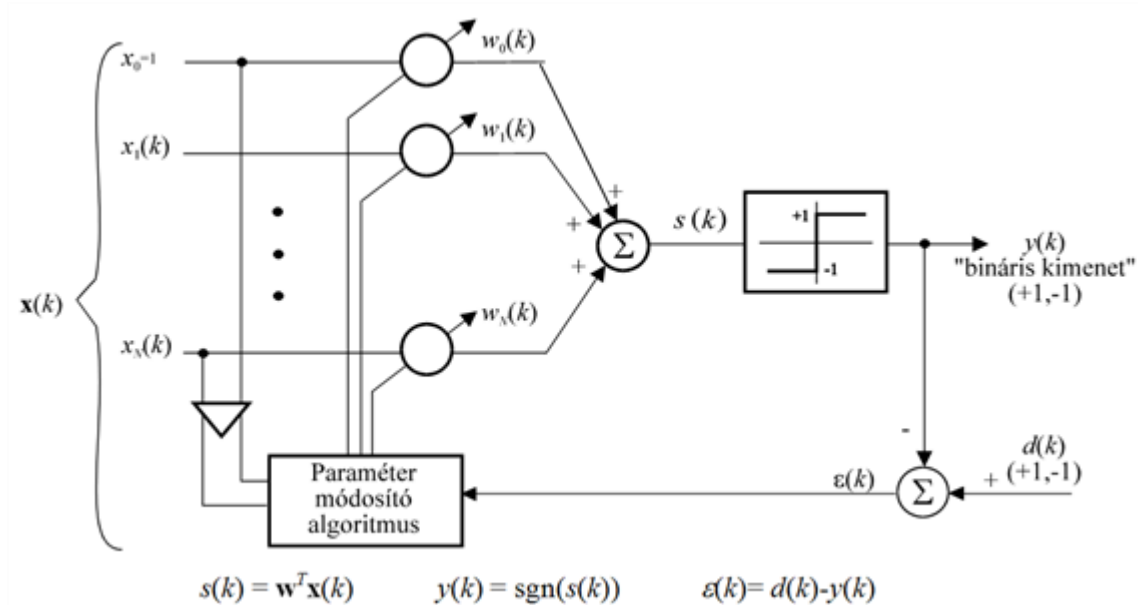
A perceptron egyike a legelső és a legegyszerűbb mesterséges neurális hálózatoknak. Az előrecsatolt, többrétegű hálózatok olyan speciális változatának tekinthető, amely egyetlen rétegből és azon belül egy vagy több processzálo elemből áll. Mind a Rosenblatt perceptron, mind az adaline valójában egy egyrétegű, egy processzálo elemből álló előrecsatolt hálózat.

1. A Rosenblatt perceptron

A Rosenblatt perceptron vagy egyszerű perceptron egy olyan hálózat, amely képes arra, hogy – megfelelő beállítás, tanítás után – két lineárisan szeparálható bemeneti mintahalmazt szétválasszon. A lineáris szeparálhatóság azt jelenti, hogy a bemeneti mintateret egy síkkal (hipersíkkal) két diszjunkt tartományra tudjuk bontani úgy, hogy a két tartomány eltérő osztályba tartozó bemeneti mintapontokat tartalmazzon. Az egyszerű perceptron tehát kétszáltyos esetekben lineáris osztályozási feladatok ellátására alkalmas. A későbbiekben az egyszerű perceptront továbbfejlesztették több-elemű, ill. többrétegű hálózatokká (multilayer perceptron, MLP), amelyek képességei az egyszerű perceptron képességeit messze felülmúlják.

Az egyszerű perceptron felépítéséből (3.1 ábra) látható, hogy ez egy lineáris kombinációt megvalósító hálózat, amelynek a kimenetén egy küszöbfüggvény-nemlinearitás szerepel, vagyis valójában egyetlen memória nélküli processzálo elem.

3.1. ábra - Az egyszerű perceptron felépítése a hibaképzéssel és a paramétermódosítással



A küszöbfüggvény a lineáris kombináció eredményeként kapott súlyozott összeg (s) előjelét szolgáltatja kimeneti jelként.

$$y = f\left(\sum_{i=0}^N w_i x_i\right) = \text{sgn}\left(\sum_{i=0}^N w_i x_i\right) = \text{sgn}(s), \quad (3.2)$$

ahol $\text{sgn}(\cdot)$ az előjelfüggvény (szignum függvény).

A perceptron bemenetére az $\mathbf{x}$ ($N+1$)-dimenziós vektor kerül, ahol az $x_1, \dots, x_N$ bemeneti komponensek jelentik a tényleges bemenőjelet, x_0 pedig az ún. eltolás (*bias*) értéket. A bias szerepe azt biztosítani, hogy a kimenet előjelváltása ne feltétlenül a nulla bemeneti vektornál, hanem a kívánt leképezésnek megfelelő bemeneteknél történjen. A tényleges bemenet dimenziója tehát N , az eltolásérték figyelembevételével kapott $\mathbf{x}$ dimenziója pedig $N+1$.

A tanítás során a megfelelő elválasztó felület meghatározása a cél, vagyis azon $\mathbf{w}$ súlyvektor meghatározása, kialakítása, amely mellett az egyik osztálybeli bemeneti adatokra a lineáris kombináció eredménye pozitív, míg a másik osztálybeli pontokra negatív. Az elválasztó felületet azon bemeneti pontok határozzák meg, ahol $s = \mathbf{w}^T \mathbf{x} = 0$. Az elválasztó felület a bemeneti mintatérben tehát egy, az origón átmenő, a súlyvektorra ($\mathbf{w}$) merőleges sík (hipersík).

Felmerül a kérdés, hogy mi történik, ha a bemeneti mintapontok nem választhatók szét lineáris felülettel. Ebben az esetben nyilvánvaló, hogy semmilyen tanító eljárással sem kaphatunk olyan hálózatot, amely az osztályozási feladatot tökéletesen megoldja. Ilyenkor a feladat lineárisan nem szeparálható, s így a perceptron az ilyen feladatok megoldására alkalmatlan. Természetesen elképzelhető egy olyan megoldás is, amelynél az elválasztó hipersík a mintapontok többségét helyesen választja szét, de – a bemeneti adatok elhelyezkedésétől függően – lesznek olyan mintapontok, melyekre a hálózat rossz választ ad. Ebben az esetben nyilván az a célunk, hogy a hibásan osztályozott adatok számát minimalizáljuk.

A továbbiakban a hálózat működését úgy vizsgáljuk, hogy a bemenetről feltételezzük a lineáris szeparálhatóságot. E feltételezés mellett megmutatjuk, hogy a perceptron a lineáris szeparáló felületet véges számú tanuló lépés alatt megtalálja.

1.1. A perceptron tanulása

Vezessük be a következő jelöléseket. Jelölje $\mathcal{X}^{(1)}$ azon bemeneti mintapontok halmazát, amelyek az (1) osztályba, $\mathcal{X}^{(2)}$ pedig azon bemeneti mintapontok halmazát, amelynek elemei a (2) osztályba tartoznak, és tételezzük fel, hogy mind $\mathcal{X}^{(1)}$, mind $\mathcal{X}^{(2)}$ véges számú vektort tartalmaz. Jelöljön továbbá $\mathbf{w}^*$ egy olyan súlyvektort, amely mellett a hálózat jól osztályoz, vagyis $\mathbf{w}^{*T} \mathbf{x} > 0$, ha $\mathbf{x} \in \mathcal{X}^{(1)}$ és $\mathbf{w}^{*T} \mathbf{x} < 0$, ha $\mathbf{x} \in \mathcal{X}^{(2)}$.

A hálózat tanítását a következőképpen végezzük: egyenként vegyük sorra az összes bemeneti mintapontot, határozzuk meg mindegyik lépésben a hálózat választát, és az alábbi összefüggés szerint módosítsuk a súlyvektort:

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \alpha (d(k) - y(k)) \mathbf{x}(k) = \mathbf{w}(k-1) + \alpha \varepsilon(k) \mathbf{x}(k) \quad (3.3)$$

A súlymódosítás (3.3) összefüggése formailag megegyezik az előző fejezetben bemutatott LMS algoritmussal, ha az itt szereplő α -t megfeleltetjük az LMS algoritmus μ tanulási tényezőjének. Az egyezés azonban csak formai, ahogy ez a következőkből kiderül. Amennyiben mind $d(k)$, mind $y(k)$ csak két értéket (± 1 vagy 0 és 1) vehet fel, és α értékét 1-re választjuk, hibás válasz esetén $\varepsilon(k)$ abszolút értéke is csak két lehetséges konstans érték (2, ill. 1) egyike lehet, míg a hiba előjele a tényleges értékektől függően változik; vagyis valójában a pillanatnyi bemenőjel konstans-szorosával történik a súlykorrekció:

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \alpha(k) \mathbf{x}(k), \quad (3.4)$$

ahol $\alpha(k)$ értéke ± 2 , ill. ± 1 lehet.

A súlymódosítás ez utóbbi formája mutatja, hogy a formai hasonlóság ellenére itt valóban nem az LMS algoritmussal van dolgunk, vagyis a súlymódosítás nem egy hibafelület negatív gradienseinek irányában történik, hanem a súlyvektort egyszerűen a helytelenül osztályozott bemenővektorral arányosan módosítjuk. Ez a módosítás szemléletesen azt jelenti, hogy a súlyvektort a hibásan osztályozott bemeneti vektor irányában (vagy azzal ellentétes irányban) elforgatjuk, s így a módosítás után a skalár szorzat a kívánt értékhez közelít.

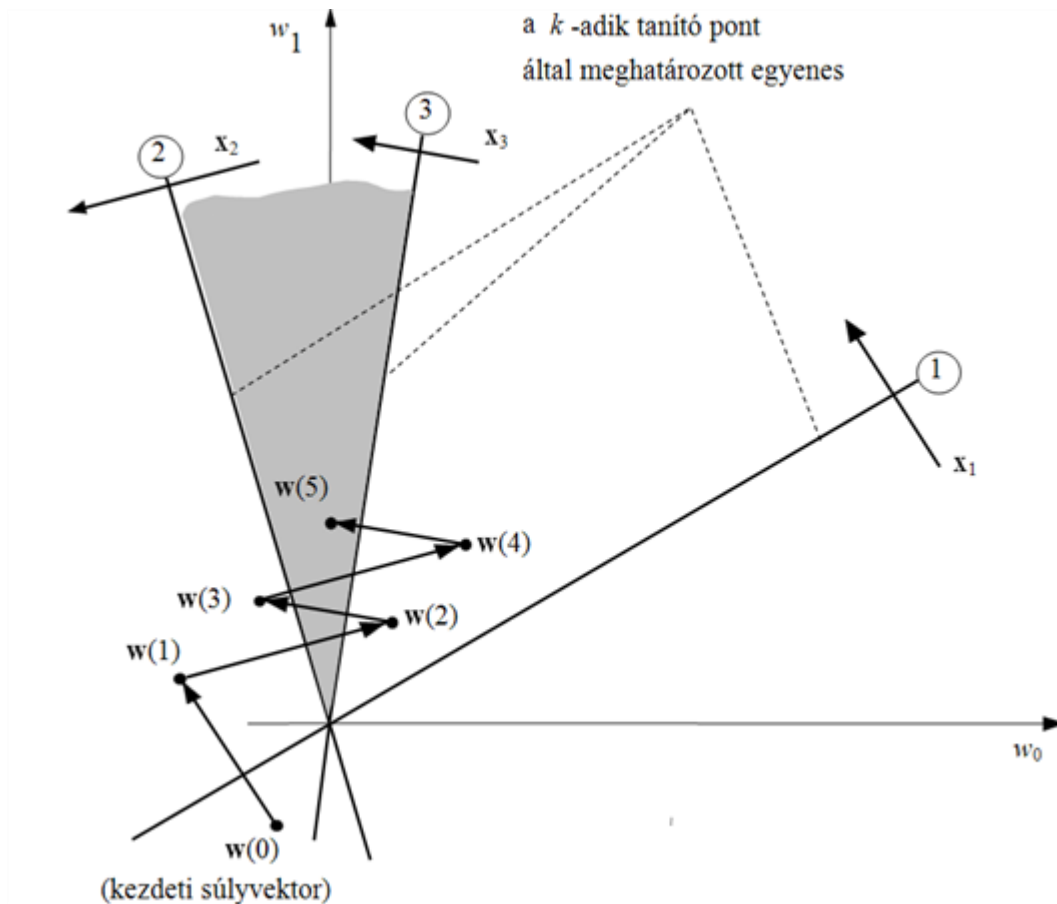
A tanító lépés során a súlyvektort olyan módon változtatjuk, hogy a módosított súlyvektorral a hálózat válasza a helyes értékhez közelítsen. A tanítás hatását a súlyvektorok terében a 3.2 ábrán követhetjük nyomon. A súlyvektorok terében minden bemenővektor egy hipersíkot (kétdimenziós súlytérben egy egyenest) határoz meg. Az egyenes pontjai egyben azon súlyvektorokat is megadják, amelyek mellett a bemenővektor és a súlyvektor skalár szorzata zérus, vagyis az egyenes merőleges a bemeneti mintavektorra. (Másképp fogalmazva a mintavektort a súlytérben olyan hipersíkkal reprezentáljuk, amelynek normálvektora épp a mintavektor.) A hipersík (egyenes) negatív oldalán elhelyezkedő súlyvektorok esetén a hálózat kimenete -1 lesz, míg ha a súlyvektor az egyenes pozitív oldalán található, akkor a hálózat válasza +1.

Súlymódosításra akkor van szükség, ha az aktuális súlyvektor egy tanítópont által meghatározott hipersíknak (egyenesnek) nem a tanítópont osztályának megfelelő oldalán helyezkedik el, vagyis a skalár szorzat nem megfelelő előjelű. Ilyenkor a súlyvektort olyan módon változtatjuk, hogy a skalár szorzat a módosított súlyvektorral megfelelő előjelű legyen – a súlyvektor a hipersík megfelelő oldalára kerüljön –, vagy legalábbis a súlyvektort a helyes irányba mozdítsuk.

Az ábra azt a feltételezett esetet mutatja, amikor három tanítópontunk van, amelyek közül $\mathbf{x}_1$ és $\mathbf{x}_3$ $\mathbb{X}^{(1)}$ elemei, míg $\mathbf{x}_2$ $\mathbb{X}^{(2)}$ eleme, vagyis $\mathbf{w}^{*T} \mathbf{x}_i > 0$, ha $i=1$ és 3, ill. $\mathbf{w}^{*T} \mathbf{x}_i < 0$, ha $i=2$. A $\mathbf{w}^*$ megoldásvektornak tehát a súlytérben az $\mathbf{x}_1$ és $\mathbf{x}_3$ tanító pontokat reprezentáló egyenesek (1) és (3) pozitív oldalán, míg az $\mathbf{x}_2$ -t reprezentáló egyenes (2) negatív oldalán kell elhelyezkednie. Ezeket a feltételeket a súlytérben árnyékoltan jelölt területet pontjai elégíti ki, tehát a megoldásvektornak ezen területen belül kell lennie.

A súlyvektor lépésenkénti módosítását az ábrán követhetjük. $\mathbf{x}_2$ esetében pl. módosításra van szükség, hiszen $\mathbf{w}^{(1)T} \mathbf{x}_2 > 0$, holott $\mathbf{x}_2 \in \mathbb{X}^{(2)}$, tehát negatív eredményt kellett volna kapnunk. A súlyvektor értékét $\mathbf{x}_2$ -vel arányosan csökkentenünk kell. E módosítás következtében létrejött $\mathbf{w}(2)$ súlyvektor viszont rosszul osztályozza $\mathbf{x}_3$ -at, tehát ismételt módosításra van szükség, ezúttal a súlyvektorhoz egy, az $\mathbf{x}_3$ -mal arányos vektort kell hozzáadnunk. Az eljárást így kell folytatnunk mindaddig, míg az összes tanító vektorral megfelelő előjelű skalár szorzatot nem kapunk, vagyis, amíg a megoldás-tartományon belülre nem kerül a súlyvektor. Az ábrán ez $\mathbf{w}(5)$ -nél következik be.

3.2. ábra - Súlymódosítás a perceptron tanításánál



A továbbiakban megmutatjuk, hogy a perceptron a lineáris szeparáló felületet véges számú tanuló lépés alatt megtalálja.

A bizonyítás azon alapul, hogy $\|w^{(k)}\|^2$ növekedése két korlát közé szorítható: egyrészt $\|w^{(k)}\|^2 < \alpha^2 kM$, másrészt $\|w^{(k)}\|^2 \geq \frac{\alpha^2 k^2 b^2}{\|w^*\|^2}$, ahol M a bemeneti mintavektorok normanégyszetének felső korlátja: $\|x_i\|^2 \leq M$, b pedig a megoldás-súlyvektorra vonatkozó

$$w^{*T} x_i \geq b > 0, \text{ ha } x_i \in X^{(1)}$$

és

$$w^{*T} x_i \leq -b < 0, \text{ ha } x_i \in X^{(2)}$$

feltételek kis pozitív konstansa.

1.2. A perceptron tanulás konvergenciája

Az egyszerűbb tárgyalás kedvéért a kiinduló tanítópontokból alakítsunk ki egy módosított mintahalmazt, $X^{(*)}$ -ot, ami az $X^{(1)}$ -beli pontokból és az $X^{(2)}$ -beli pontok mínusz egyszereséből áll. Ez azt jelenti, hogy $\forall x \in X^{(*)}$ -ra $w^{*T} x \geq b > 0$ kell teljesülnön, vagyis a módosított tanítóhalmaz minden elemére igaz, hogy a megoldás súlyvektor által meghatározott sík pozitív felén fog elhelyezkedni.

Tanító lépésre akkor van szükség, ha a pillanatnyi súlyvektor a bemenetre kerülő mintavektort nem megfelelő osztályba sorolja, vagyis, ha bármely $x \in X^{(*)}$ -ra a pillanatnyi súlyvektorral végzett skalár szorzat $w^{(k-1)T} x \leq 0$. A módosított mintahalmazzal végzett tanítás esetén

$$w^{(k)} = w^{(k-1)} + \alpha x^{(k)}, \quad (3.5)$$

ahol $\mathbf{x}(k)$ tehát a k -adik tanító lépésben a perceptronra kerülő olyan minta, amit az aktuális súlyvektor mellett a perceptron rosszul osztályoz. Minden korrekció $\alpha > 0$ ben a pillanatnyi (módosított) bemenővektor konstansszorosát adjuk a súlyvektorhoz, ahol most már

A módosított tanító pontok felhasználása során képezzünk egy képzeletbeli listát, amelybe csak azok a tanító pontok tartoznak, amelyeket az adott lépésben a hálózat hibásan osztályoz. Ez azt jelenti, hogy ha $\mathbf{w}(k-1)$ jelöli a k -adik lépésben az aktuális súlyvektort és $\mathbf{x}(k)$ a képzeletbeli listának azt az elemét, amit a k -adik lépésben veszünk elő, akkor $\mathbf{w}^T(k-1)\mathbf{x}(k) \leq 0$ adódik, tehát súlymódosításra van szükség. A tanítás során a listában az eredeti mintahalmaz minden eleme akárhányszor előfordulhat.

Kövessük végig a súlyvektor alakulását feltételezve, hogy a kiinduló érték $\mathbf{w}^{(0)} = \mathbf{0}$. Az első, a második, ill. a k -adik lépésben kialakuló súlyvektor ennek megfelelően:

$$\begin{aligned}\mathbf{w}^{(1)} &= \alpha \mathbf{x}^{(1)} \\ \mathbf{w}^{(2)} &= \mathbf{w}^{(1)} + \alpha \mathbf{x}^{(2)} = \alpha [\mathbf{x}^{(1)} + \mathbf{x}^{(2)}] \\ &\vdots \\ \mathbf{w}^{(k)} &= \mathbf{w}^{(k-1)} + \alpha \mathbf{x}^{(k)} = \alpha \sum_{i=1}^k \mathbf{x}^{(i)}\end{aligned}\quad (3.6)$$

Képezzük a $\mathbf{w}^T(k)\mathbf{w}^*$ skalár szorzatot:

$$\mathbf{w}^T(k)\mathbf{w}^* = \alpha [\mathbf{x}^{(1)T}\mathbf{w}^* + \mathbf{x}^{(2)T}\mathbf{w}^* + \dots + \mathbf{x}^{(k)T}\mathbf{w}^*] \geq \alpha kb, \quad (3.7)$$

Felhasználva a Schwartz egyenlőtlenséget

$$\|\mathbf{w}^{(k)}\|^2 \|\mathbf{w}^*\|^2 \geq (\mathbf{w}^T(k)\mathbf{w}^*)^2 \geq \alpha^2 k^2 b^2 \quad (3.8)$$

amiből adódik, hogy:

$$\|\mathbf{w}^{(k)}\|^2 \geq \frac{\alpha^2 k^2 b^2}{\|\mathbf{w}^*\|^2} \quad (3.9)$$

vagyis, a súlyvektor normanégyzete legalább a lépésszám négyzetével arányosan nő.

A súlyvektor normanégyzetére ugyanakkor felső korlát is állítható, ugyanis

$$\mathbf{w}^{(k)} = \mathbf{w}^{(k-1)} + \alpha \mathbf{x}^{(k)}, \quad (3.10)$$

így felírható, hogy

$$\|\mathbf{w}^{(k)}\|^2 = \|\mathbf{w}^{(k-1)}\|^2 + 2\alpha (\mathbf{w}^{(k-1)T}\mathbf{x}^{(k)}) + \alpha^2 \|\mathbf{x}^{(k)}\|^2. \quad (3.11)$$

Mivel tanító lépésre azért van szükség, mert $\mathbf{w}^T(k-1)\mathbf{x}^{(k)} \leq 0$, ebből következik, hogy

$$\|\mathbf{w}^{(k)}\|^2 - \|\mathbf{w}^{(k-1)}\|^2 \leq \alpha^2 \|\mathbf{x}^{(k)}\|^2. \quad (3.12)$$

Amennyiben a kezdeti súlyvektor hossza nulla, vagyis $\mathbf{w}^{(0)} = \mathbf{0}$, a fentiekből adódik, hogy

$$\|\mathbf{w}^{(k)}\|^2 \leq \alpha^2 \sum_{i=1}^k \|\mathbf{x}^{(i)}\|^2 \leq \alpha^2 k M, \quad (3.13)$$

feltéve, hogy kiinduló megállapításunk, miszerint az összes tanító vektor normanégyzete felülről korlátos, igaz. Ez azt jelenti, hogy a vektor normanégyzete nem nőhet a lépésszámmal arányosnál gyorsabban. A két korlát összevetéséből következik, hogy véges számú lépés után az eljárásnak le kell állnia.

A tanító lépések számának felső korlátja, $k_{\max}$ arányos a bemeneti vektor dimenziójával, de nem függ a tanítópontok számától. Természetesen egy tényleges tanító eljárásnál a konvergencia ideje függ a tanítópontok számától, hiszen minél több tanítópontunk van, annál több pontra kell elvégezni az ellenőrzést, hogy vajon a pillanatnyi súlyvektor mellett a hálózat már helyesen működik-e. Azt is látni kell, hogy a konvergencia ténye nem függ értékeitől, hiszen csupán annyit kötöttünk ki, hogy legyen $\alpha > 0$. A konvergencia sebességét azonban már befolyásolja megválasztása. Így a perceptron tanulása különböző változatai alakultak ki, melyek megválasztásában térnek el. Az eddig bemutatott tanuláshoz előre rögzített konstans, ezért ezt szokás fix növekményes tanulási szabálynak is nevezni, szemben a lépésenként változó α -t használó módszerekkel. Minthogy elvi különbség ezen módszerek között nincs, a változó növekményes eljárásokat nem részletezzük. A különböző perceptron tanulási változatok részletes bemutatása megtalálható pl. a következő irodalmakban: [Nil65], [Has95], [Vee95].

1.3. A perceptron kapacitása

Mint a neurális hálózatok többsége, a perceptron is felfogható, mint egy asszociatív memória. A hálózatnak bizonyos bemenetekre kell jó válaszokat adnia, asszociálnia, amelyet úgy érünk el, hogy a hálózat súlyainak meghatározásához tanítópontokat használunk fel. Valójában tehát a tanítópontok felhasználása úgy is felfogható, mint egy tárolási folyamat, ahol a tanítópontokat valamilyen módon a hálózat súlyaiban tároljuk. Felmerül a kérdés, hogy hány tanítópont tárolható a hálózatban úgy, hogy a hálózat a megkívánt működést mutassa, illetve általában milyen válaszokat ad a hálózat egy megtanított esetben. Erre a kérdésre ad feleletet a hálózatok kapacitása. Az alábbiakban a perceptron kapacitását vizsgáljuk [Cov65].

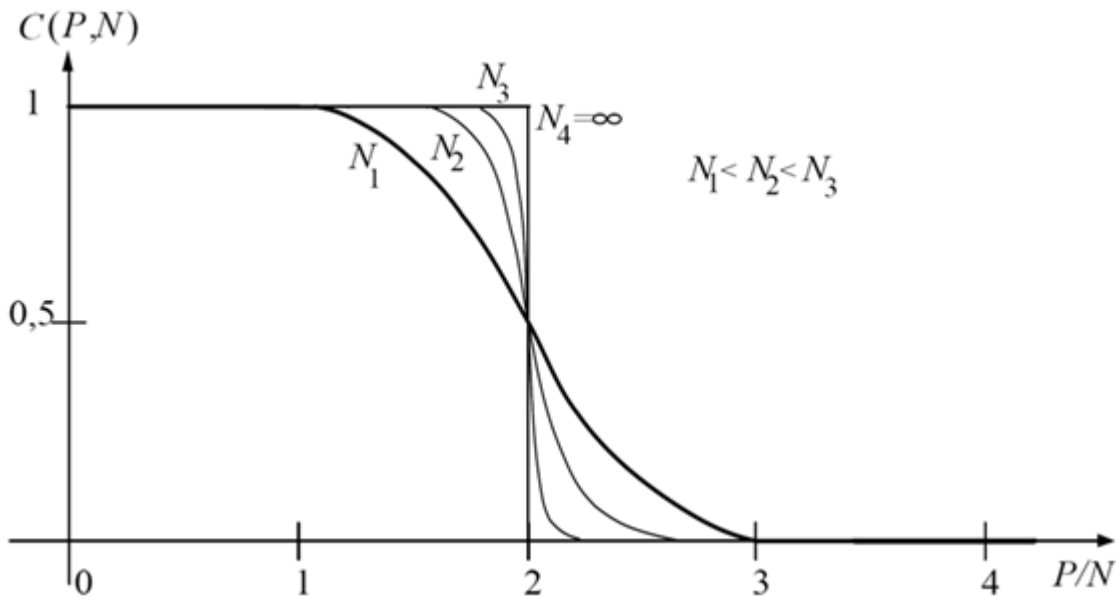
A perceptront eddig úgy mutattuk be, mint egy olyan egyszerű hálózatot, amely az N -dimenziós mintatérre egy $(N-1)$ -dimenziós hipersíkkal két tartományra tudja bontani. A perceptron szeparáló képességével kapcsolatban a következő kérdés merül fel: hány, két különböző osztályból származó pontot helyezhetünk el véletlenszerűen egy N -dimenziós mintatérben akkor, ha azt kívánjuk, hogy a két osztály pontjai egy hipersíkkal szétválaszthatók legyenek. Másképp fogalmazva: N -dimenziós bemeneti pontok esetén, P véletlenszerűen kiválasztott pont mellett 2^P kétosztályos szeparálás – vagyis amikor a pontok egy részéhez $+1$ -et, a többihez -1 -et rendelünk – lehetséges; kérdés, hogy a 2^P lehetőség közül hány olyan eset van, amikor a két osztályba tartozó pontok elválasztása hipersíkkal megtörténhet, tehát amikor a szeparálási feladatot egy perceptron meg tudja oldani.

Jelöljük ezen esetek számát $L(P,N)$ -nel. Defináljuk a perceptron kapacitását úgy, mint a lineárisan szeparálható kétosztályos esetek számának és az összes kétosztályos eset számának arányát, vagyis $C(P,N)=L(P,N)/2^P$. Megmutatható, hogy a kapacitás a következő összefüggéssel adható meg:

$$C(P, N) = \begin{cases} 1 & \text{ha } P \leq N \\ \frac{2}{2^P} \sum_{i=0}^{N-1} \binom{P-1}{i} & \text{ha } P > N \end{cases} \quad (3.14)$$

A kapacitás meghatározásának részletes bemutatásától eltekintünk (az érdeklődő olvasó a részletes származtatást például a [Her91] irodalomban megtalálja). A kapacitás alakulását a $P/(N+1)$ arány függvényében a 3.3 ábrán mutatjuk be.

3.3. ábra - A perceptron kapacitásának alakulása



Látható, hogy ha N elég nagy, akkor $P < 2N$ esetében gyakorlatilag az összes kétszáltyos szeparálás lineáris szeparálás, míg ezen érték fölött lineáris szeparálás egyre kevésbé lehetséges: ha $P/N \gg 2$ a lineárisan szeparálható esetek száma nullához tart. Az is látható, hogy, ha $P \leq N$, akkor $C(P, N) = 1$, vagyis, ha a mintapontok száma nem nagyobb, mint a bemeneti tér dimenziója és a pontok általános elhelyezkedésük, a lineáris szeparálás mindig lehetséges, bármilyen módon is rendeljük a pontokat a két osztályhoz. A pontok akkor és csak akkor általános elhelyezkedésűek, ha (*) $P > N$ és a P pontból nem tudunk kiválasztani $N+1$ olyan pontot, melyek egy $(N-1)$ -dimenziós hipersíkon helyezkednek el; illetve, ha (**) $P \leq N$ és a P pont nem helyezkedik el egy $N-2$ dimenziós hipersíkon.

A perceptron kapacitásra vonatkozó eredmény alapján látható, hogy ha a tanítópontok száma a pontok dimenziójához képest elegendően nagy, akkor a mintapontok lineárisan nem lesznek szeparálhatók. A gyakorlati problémák döntő többsége lineárisan nem szeparálható feladat, vagyis az egyszerű perceptronnal nem megoldható.

A perceptron kapacitás meghatározásának a jelentősége az, hogy ezáltal megmutatható, hogy a bemeneti szeparálandó pontok számához viszonyítva kellően nagyra választva a bemeneti tér dimenzióját – ami azt is jelenti egyben, hogy kellően nagyra választva a perceptron súlyainak számát – a kétszáltyos feladatok lineárisan szeparálható feladatok lesznek. Ezt az eredményt bizonyos hálózatok konstrukciójánál fel is használhatjuk (ld. pl. a bázisfüggvényes hálózatokat, melyeket a 6. fejezet mutat be).

2. Az adaline

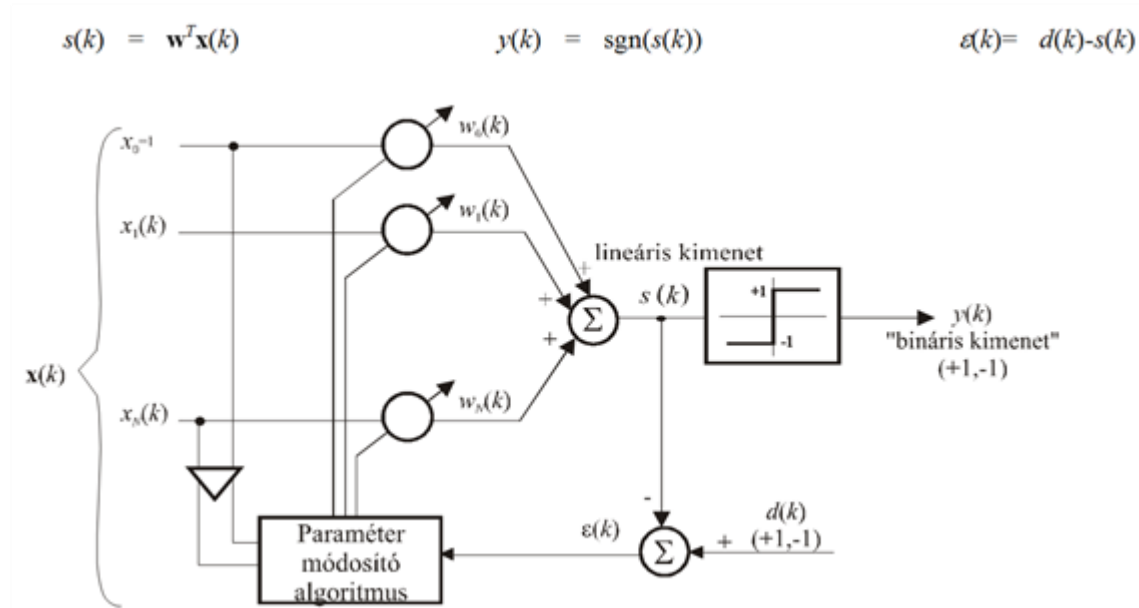
A küszöbfüggvényt alkalmazó perceptronnal kívül az előbbi hálózat különböző változatai is elterjedtek. Ezen változatokban a különbség a kimeneti hiba értelmezésében, illetve a kimeneti nemlinearitásban és ebből következően a tanulási eljárásban van. A perceptronnal lényegében megegyező felépítésű hálózat az *adaline*, amely csaknem egyidős a perceptronnal és amelyet Bernard Widrow és munkatársai javasoltak [Wid60]. A két hálózat közötti alapvető különbség a hiba értelmezésében és a háló tanításában van. Míg a Rosenblatt-féle perceptronnál a hálózat tényleges kimenetét hasonlítjuk össze a kívánt válasszal, addig az *adaline*-nál a lineáris kombináció eredményét, tehát egy lineáris hálózatrész kimenetét vetjük össze egy kívánt jellel (ld. 3.4 ábra). (A hálózat elnevezése is ezt tükrözi: *adaline* = *adaptive linear element*, vagy *adaptive linear neuron*.) A hálózat tényleges kimenete azonban itt is egy küszöbfüggvény nemlinearitás után kapható.

2.1. Az adaline tanítása

Mivel a hiba szempontjából a lineáris kimenetet tekintjük az *adaline* kimenetének, a kimenet a paraméterek lineáris függvénye és négyzetes hiba estén a kritériumfüggvény kvadrátikus felületet jelent. Az *adaline* tanításánál tehát alkalmazhatók a 2.5.1. alfejezetben bemutatott eljárások. Az *adaline* súlyvektorának optimális értékét tehát akár analitikusan, a Wiener-Hopf egyenlet alapján, akár gradiens alapú iteratív eljárással

meghatározhatjuk. Az iteratív gradiens alapú eljárások közül is leggyakrabban az LMS algoritmust alkalmazzuk, annak nagyfokú egyszerűsége miatt.

3.4. ábra - Az adaline felépítése a hibaképzéssel és a paramétermódosítással



A Wiener-Hopf egyenlet a négyzetes hiba várható értékének minimumához tartozó paraméter-vektort eredményezte, az LMS algoritmról pedig láttuk, hogy olyan iteratív eljárás, amely mindig a pillanatnyi négyzetes hiba csökkentésének irányában módosítja az aktuális paramétervektort. Azt is láttuk, hogy a hiba várható értékét csak becsülni tudjuk, mivel a mintapontok eloszlását általában nem ismerjük. A mintapontok eloszlásának ismerete nélkül kritériumfüggvénynek a mintapontokban meghatározható négyzetes hiba átlagát vagy összegét tekinthetjük:

$$C(\mathbf{w}) = \frac{1}{P} \sum_{i=1}^P (d_i - f(\mathbf{w}, \mathbf{x}_i))^2 \quad (3.15)$$

Adaline-nál a lineáris kapcsolat miatt $f(\mathbf{w}, \mathbf{x}) = \mathbf{w}^T \mathbf{x}$. Az átlagos négyzetes hiba ilyenkor felírható az alábbi mátrixos formában is:

$$C(\mathbf{w}) = (\mathbf{d} - \mathbf{X}\mathbf{w})^T (\mathbf{d} - \mathbf{X}\mathbf{w}) \quad (3.16)$$

ahol $\mathbf{d}$ a tanítópontokbeli kívánt válaszokból képezett P elemű oszlopvektor, $\mathbf{X}$ a bemeneti vektorokból képezett mátrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_P^T \end{bmatrix}, \quad (3.17)$$

$\mathbf{w}$ pedig a keresett paramétervektor. A megoldásvektor ideális esetben azt biztosítja, hogy

$$\mathbf{d} = \mathbf{X}\mathbf{w}, \quad (3.18)$$

vagyis egy lineáris egyenletrendszer kapunk. A lineáris egyenletrendszer megoldása

$$\mathbf{w} = \mathbf{X}^{-1} \mathbf{d}, \quad (3.19)$$

amennyiben $\mathbf{X}$ inverze létezik. $\mathbf{X}^{-1}$ létezésének szükséges feltétele, hogy $\mathbf{X}$ kvadratikusan legyen. (3.17) alapján látható, hogy $\mathbf{X}$ P sorból és $N+1$ oszlopból áll. A sorok száma az egyenletek számát, az oszlopok száma az ismeretlenek számát adja meg. Inverz csak akkor létezhet, ha $P=N+1$, sőt még így is csak akkor, ha $\mathbf{X}$ rangja teljes, vagyis $N+1$. Ez akkor áll elő, ha pont annyi tanítópontunk van, mint ahány ismeretlen súly található a

hálózatban továbbá, ha a bemeneti vektorok lineárisan függetlenek egymástól. Ha az inverz nem létezik, a pszeudo- vagy Moore-Penrose inverz alkalmazható, amikor is a megoldásvektort az alábbi formában kapjuk:

$$\mathbf{w}^* = \mathbf{X}^T \mathbf{d} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{d} \quad (3.20)$$

Vegyük észre, hogy ugyanerre a megoldásra jutunk, ha a (3.16) összefüggés gradiensének nulla értékét biztosító paramétervektort határozzuk meg:

$$\frac{\partial C(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{X}^T \mathbf{d} + 2\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{0} \quad (3.21)$$

Ugyanis ekkor eredményként a normál egyenletnek nevezett összefüggést kapjuk:

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{d}, \quad (3.22)$$

melyből a megoldásvektor a keresett paraméter LS becslése, amely megegyezik (3.20)-szal:

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{d} \quad (3.23)$$

Vegyük észre azt is, hogy a (3.23) összefüggés megfeleltethető a Wiener-Hopf egyenletnek, mivel $\mathbf{X}^T \mathbf{X}$ az $\mathbf{R} = E\{\mathbf{x}\mathbf{x}^T\}$ autokorrelációs mátrix, $\mathbf{X}\mathbf{d}$ pedig a $\mathbf{p} = E\{\mathbf{x}\mathbf{d}\}$ keresztkorrelációs vektor becslésének tekinthető.

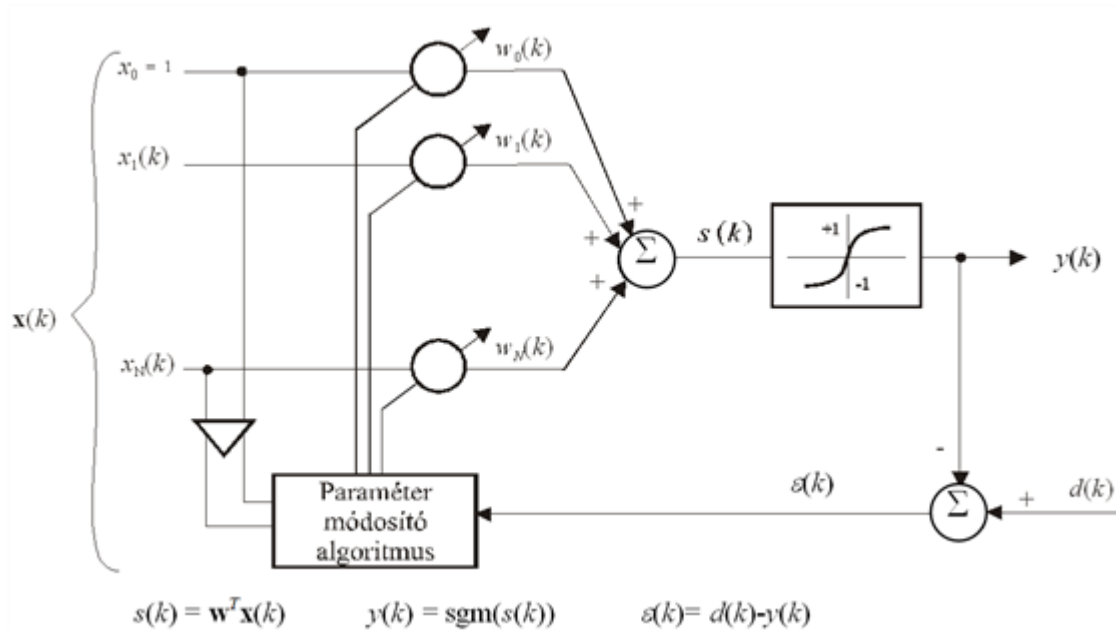
A két elemi neuron összehasonlításánál meg kell állapítanunk, hogy a felépítésük azonossága mellett több szempontból is jelentős különbségeket mutatnak. A perceptron és az adaline tanítása alapvetően eltérő még akkor is, ha a perceptron tanulás (3.2) összefüggéssel megadott iteratív eljárása formailag megegyezik az LMS algoritmussal. A perceptron tanulás egy hibakorrekciós eljárás, ahol – lineárisan szeparálható mintapontok esetén – a konvergencia bármely $\alpha > 0$ mellett biztosított, míg az LMS algoritmus egy pillanatnyi gradiensen alapuló gradiens eljárás. Ennek konvergenciáját már szigorúbb feltétel biztosítja, hiszen itt a tanulási tényezőre a (2.69) összefüggésben megadott felső korlátnak is érvényesnek kell lennie.

A hiba értelmezésénél, illetve a tanításnál lévő különbségek mellett a számítási képességet tekintve is különbözik az adaline a Rosenblatt perceptrontól. Az adaline bármely feladat megoldásánál használható, és minden esetben megoldást ad. Tehát nem kell kikötnünk, hogy a megoldandó feladat lineárisan szeparálható legyen. Ugyanakkor azt is látni kell, hogy egy általános feladatnál semmi biztosítékunk nincs arra, hogy a mintapontok által képviselt leképezést az adaline hibátlanul meg tudja tanulni. A tanuló eljárás mindössze annyit garantál, hogy megoldásként a kvadratis hiba felület minimumpontjához tartozó paramétervektort kapjuk. Ebből azonban még nem következik, hogy az optimumnál a négyzetes (átlagos négyzetes) hiba nulla is lesz.

3. Egy processzáló elem szigmoid kimeneti nemlinearitással

Az előzőekben bemutatott két elemi neuron ugrásfüggvényt alkalmazott kimeneti nemlinearitásként. Az ugrásfüggvény következménye, hogy a kimenet nemlineáris függvénye a súlyvektornak, vagyis ha az elemi neuron kimeneténél értelmezzük a négyzetes kritériumfüggvényt, akkor nem kapunk kvadratis hiba felületet. Sőt az ugrásfüggvény miatt a kritériumfüggvény nem is lesz mindenhol differenciálható. A differenciálhatóság és a négyzetes hiba felület hiánya a perceptronnál nem okozott gondot, hiszen ott nem gradiens alapú tanuló eljárást alkalmaztunk. Az adaline-nál a nem folytonos nemlinearitás hatását úgy védtük ki, hogy a hibát nem az igazi kimeneten, hanem a lineáris összegző kimenetén értelmeztük. A későbbiekben látni fogjuk, hogy egyik megoldás sem kedvező, ha ezeket az elemi neuronokat összetett neuronhálónál szeretnénk használni, ahol a neuronháló több, rétegekbe szervezett neuronokból állnak. Ilyen esetekben a perceptron tanulást azért nem alkalmazhatjuk, mert az csak olyan neuronok tanítására jó, ahol a neuron kimenetén tudjuk értelmezni a hibát. A pillanatnyi gradiens alapú LMS algoritmus alkalmazásának pedig a rejtett réteg neuronjainak kimenetén található nem differenciálható nemlinearitás az akadály.

3.5. ábra - Sigmoid kimeneti nemlinearitással felépülő elemi neuron a hibaképzéssel és a paramétermódosítással



A probléma megoldását jelentheti, ha az ugrásfüggvényt helyettesítjük egy olyan függvénnyel, amely folytonos, differenciálható és az ugrásfüggvény közelítésének tekinthető. A folytonos, differenciálható nemlinearitással felépített elemi neuron biztosítja, hogy a kimenet a neuron szabad paramétereinek folytonos, differenciálható függvénye legyen, és így a paraméterek meghatározásánál – a neuron tanításánál – a gradiens eljárások alkalmazhatók legyenek.

Az így, az előzőekben bemutatott elemi neuron kismértékű módosításával létrejött neuront a 3.5 ábra mutatja. A neuron most is egy súlyozott összegzőből és az azt követő nemlinearitásból áll, a nemlinearitás azonban az előzőekben alkalmazott szignum függvény helyett a szigmoid függvény lesz. A szigmoid függvény folytonos, differenciálható, monoton növekvő, korlátos, telítődő függvény. Sigmoid függvénynek sokféle konkrét függvény alkalmazható. A két leggyakrabban alkalmazott szigmoid függvényt – a tangens hiperbolikus függvényt és a logisztikus függvényt – az 1. fejezetben, az 1.3 (c) és (d) ábrákon mutattuk be.

A folytonos, differenciálható be-kimeneti leképezés miatt erre a neuronra már alkalmazhatók a gradiens eljárások, akkor is, ha a hibát nem a lineáris rész, hanem a teljes hálózat kimenetén értelmezzük.

$$\varepsilon(k) = d(k) - y(k) = d(k) - \text{sgm}(s(k)) = d(k) - \text{sgm}(\mathbf{w}^T(k) \mathbf{x}(k)), \quad (3.24)$$

Ha a négyzetes hiba pillanatnyi gradiense

$$\frac{\partial \varepsilon^2}{\partial \mathbf{w}} = -2\varepsilon \text{sgm}'(s) \mathbf{x}, \quad (3.25)$$

alapján módosítjuk a súlyokat, akkor az LMS eljárás megfelelőjére jutunk. Itt $\text{sgm}'(s)$ a kimeneti nemlinearitás deriváltját jelöli.

A súlymódosítás a gradiens eljárásnak megfelelően a következőre adódik:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu(k) \varepsilon(k) \text{sgm}'(s(k)) \mathbf{x}(k) = \mathbf{w}(k) + 2\mu(k) \delta(k) \mathbf{x}(k) \quad (3.26)$$

A kapott összefüggés annyiban különbözik az LMS algoritmustól, hogy itt az ε kimeneti hiba helyett egy $\delta = \varepsilon \text{sgm}'(s)$ "származtatott hiba" szerepel. (Megjegyezzük, hogy ezt a tanítási szabályt szokás ezért **delta szabálynak** (delta rule) is nevezni.)

A logisztikus illetve a tangens hiperbolikus függvények nemlineáris transzfer függvényként (aktivációs függvényként) való alkalmazása azzal az előnnyel is jár, hogy a deriváltjuk könnyen számítható. Ugyanis:

- a logisztikus függvény és a deriváltja:

$$y = \text{sgm}(s) = \frac{1}{1+e^{-s}}, \quad y' = \text{sgm}'(s) = y(1-y) \quad (3.27)$$

- míg a tangens hiperbolikus függvény és a deriváltja:

$$y = \tanh(s) = \frac{1-e^{-2s}}{1+e^{-2s}}, \quad y' = \tanh'(s) = 1-y^2 \quad (3.28)$$

Az ugrásfüggvény nemlinearitás szigmoid függvényre való kicserélése – bármennyire is kismértékű módosításnak tűnik – fontos következményekkel jár. Ez biztosítja, hogy a tanítás gradiens eljárással lehetséges legyen, akkor is, ha a hibát az elemi neuron nemlineáris kimenetén értelmezzük, és ez teszi lehetővé, hogy az így létrejött elemi neuronból, mint építőelemből létrehozott összetett, többrétegű neuronháló szintén tanítható legyen gradiens alapú eljárással. A szigmoid függvény alkalmazása a neuron számítási képességét is megváltoztatja, azonban annak önmagában nincs jelentősége hogy egy neuron milyen leképezést tud megvalósítani. Ennél sokkal fontosabb, hogy a felhasználásával felépített többrétegű neuronháló milyen képességekkel rendelkeznek. Ebben a tekintetben a legfontosabb eredményeket az 1. fejezetben mutattuk be: egy szigmoid nemlinearitást alkalmazó neuronokból felépített többrétegű, előrecsatolt háló univerzális osztályozó, illetve univerzális approximátor. Az ebben a fejezetben bemutatott elemi neuronok közül ezért a neuronhálók konstrukciójában a legfontosabb szerepet ez az elemi neuron játssza.

4. További elemi neuronok

Az 1. fejezetben láttuk, hogy az ún. bázisfüggvényes hálózatoknál a bemeneti rétegben speciális neuronokat használunk, melyek nem végeznek összegzést, hanem a bemenetükre kerülő vektor valamilyen nemlineáris leképezését valósítják meg. Ezek a neuronok tehát egy vektor-skálár függvényt implementálnak, és általában nincsenek olyan szabad paramétereik, melyeket ellenőrzött tanítással határozzunk meg. (Ez alól van kivétel, amellyel a bázisfüggvényes hálózatokat bemutató 5. fejezetben foglalkozunk.)

Az elemi neuronok között külön meg kell említeni azokat a lineáris neuronokat, melyek felépítése megegyezik a perceptron lineáris részével, tehát melyek szintén lineáris súlyozott összegzést valósítanak meg, de ahol a súlyok kialakítása az előbbieken bemutatott eljárástól jelentősen különbözik. Több olyan nemellenőrzött tanítású hálóarchitektúra is létezik, melyek lineáris neuronokból épülnek fel, és melyek tanítása a Hebb-szabály szerint történik. A Hebb-szabály neuro-biológiai megfigyeléseken alapul. Megfogalmazása Donald Hebb nevéhez fűződik [Hebb49].

Egy lineáris neuron w_i súlyának módosítása a Hebb-szabály szerint a következő:

$$\Delta w_i = \mu x_i y, \quad (3.29)$$

ahol x_i a neuron adott súlyhoz kapcsolódó bemenete, y a neuron kimenete és μ egy pozitív konstans tanulási tényező. A Hebb-szabály azt mondja ki, hogy a súlymódosítás mértéke arányos a súly által összekapcsolt bemenet és a neuron kimenetének szorzatával, és azt fejezi ki, hogy ha egy bemenet szerepe a kimenet előállításában nagy, akkor ezt a szerepet növelni kell. A Hebb-szabály – amelynek alkalmazásával érdekes és fontos tulajdonságú neuronhálók konstruálhatók – azzal a következménnyel jár, hogy ismételt alkalmazásával a neuron súlyai minden határon túl tudnak nőni. Ezért önmagában ritkán alkalmazzák, valamilyen korlátozó hatás, normalizálás beépítése szükséges. A különböző normalizáló eljárásokat az egyes nemellenőrzött tanulású hálózatoknál, a 10. fejezetben mutatjuk be.

Feladatok

3.1 Mutassa meg, hogy a perceptron tanulás akkor is konvergens, ha nem konstans, hanem olyan α tanulási tényezőt alkalmazunk, mely minden tanító lépésben biztosítja, hogy a korrekciót követően az adott mintapont osztályozása helyes lesz.

3.2 A perceptron tanulás (3.3) és az LMS eljárás összefüggéseinek formai hasonlósága alapján megállapítható, hogy $\alpha = 2\mu$ választással a két eljárás formailag azonos. Igaz-e, hogy ezzel a választással a két eljárás ekvivalens? Igaz-e hogy μ konvergenciatartománya alapján megadható α konvergenciatartománya? Adja meg, hogy α -t milyen tartományon belül kell megválasztania, hogy a perceptron tanulás konvergens legyen.

3.3 A perceptron kapacitására vonatkozó összefüggés szerint a lineáris szeparálhatóságot a $P/N=1$ arány biztosítja. Ha a rendelkezésre álló tanítópontokra $P/N > 1$ igaz, hogyan transzformálhatja úgy a mintapontokat, hogy a lineáris szeparálhatóság feltétele teljesüljön?

3.4 Képes-e egy kétbemenetű perceptron a logikai ekvivalencia függvényt megtanulni?

3.5 A kétváltozós logikai függvények közül melyek megtanulására képes egy perceptron? És melyek megtanulására egy adaline?

3.6 Adja meg az adaline súlyvektorának LS becslését. Mit tehet, ha a szükséges mátrix invertálásnál szingularitási problémával találkozunk?

3.7 Egy szigmoid kimeneti nemlinearitást alkalmazó elemi neuront tanítunk gradiens alapú, legmeredekebb lejtő eljárással. Milyen hibafelületet kapunk? Garántáltan konvergál-e az eljárás a legkisebb négyzetes hibát jelentő globális minimumhoz, és ha igen, mi ennek a feltétele?

3.8 Egy nemlineáris elemi neuron kimenetén a következő aktivációs függvényt alkalmazzuk:

$$y = \text{sgn}(s) = \begin{cases} \sin(s) & \text{ha } -\pi/2 \leq s \leq \pi/2 \\ -1 & \text{ha } s < -\pi/2 \\ +1 & \text{ha } s > \pi/2 \end{cases}$$

Határozza meg a neuron tanulási szabályát.

4. fejezet - A többrétegű perceptron (MLP)

A többrétegű perceptron (*multi-layer perceptron*, *MLP*) a gyakorlati feladatok megoldásánál talán a leggyakrabban alkalmazott hálózat-architektúra. Az MLP rétegekbe szervezett neuronokból áll, ahol annak biztosítására, hogy a hálózat kimenete a súlyok folytonos, differenciálható függvénye legyen, a neuronok differenciálható kimeneti nemlinearitással rendelkeznek.

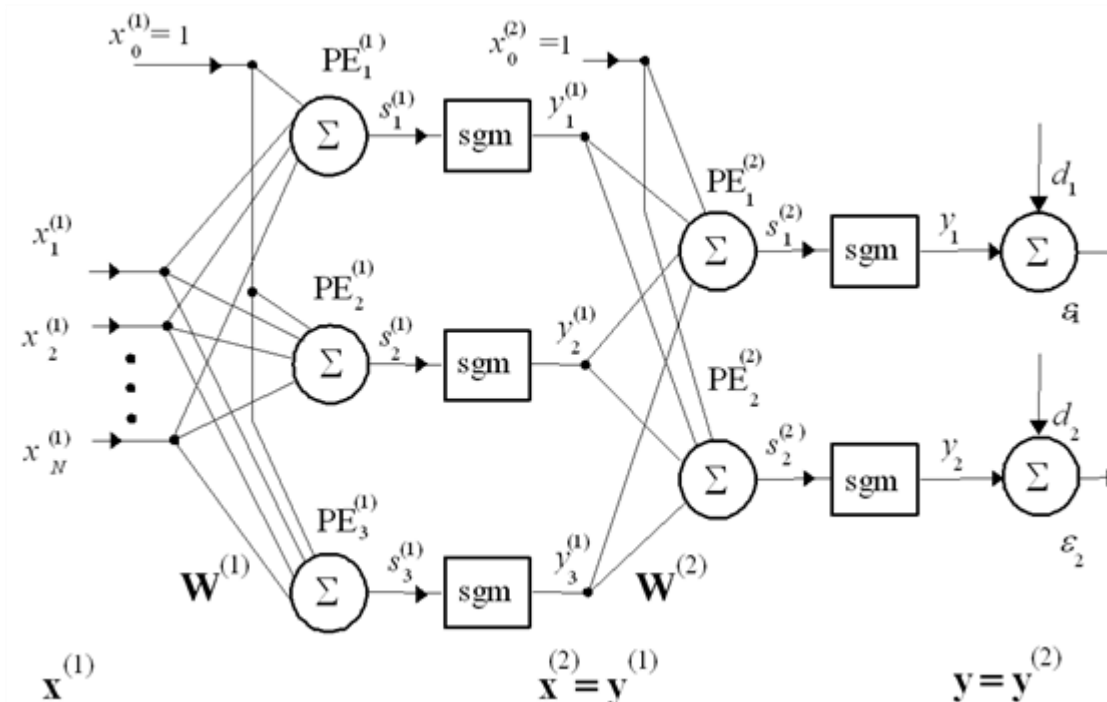
A fejezet bemutatja a hálózat felépítését, részletesen ismerteti a háló tanításánál szinte kizárólagosan alkalmazott **hibavisszaterjesztés** (*back-propagation*) algoritmust, majd a háló konstrukciójánál felmerülő elméleti és gyakorlati kérdéseket vizsgálja.

1. Az MLP felépítése

A többrétegű hálózatok felépítése a 4.1 ábrán követhető. Az ábra egy két aktív réteget tartalmazó hálózatot mutat, amelyben az első aktív rétegben – a rejtett rétegben – három, a második aktív rétegben – jelen esetben a kimeneti rétegben – két processzáló elem található. A hálózat tehát egy többrétegű előrecsatolt hálózat. Az 1. fejezetben láttuk, hogy egy megfelelő méretű, többrétegű, szigmoid nemlinearitásokat tartalmazó hálózat – a súlyvektorok kialakítása után – tetszőleges folytonos nemlineáris függvény tetszőleges pontosságú approximációjára képes. A súlyvektorok meghatározása összetartozó $\{\mathbf{x}_i, \mathbf{d}_i\}_{i=1}^p$ tanítópont-párok felhasználásával ellenőrzött tanuló eljárással történik.

A háló bemutatásához vezessük be a következő jelöléseket: egy L információ-feldolgozást végző rétegből álló MLP-nél az egyes rétegeket felső indexszel ($l=1, 2, \dots, L$) különböztetjük meg. A rétegen belüli processzáló elemekhez (PE) az i indexet rendeljük, míg j a PE bemeneteit megkülönböztető index jelölésére szolgál – ez lesz egyben a megfelelő súlyvektor komponenseinek indexe is. (Pl. $PE_i^{(l)}$ az l -edik réteg i -edik processzáló elemét, neuronját jelöli, $\mathbf{w}_i^{(l)}$ ennek a neuronnak a súlyvektorát, míg $\mathbf{w}_j^{(l)}$ ugyanezen súlyvektornak az j -edik komponensét jelöli.) A neuronok bemeneteire kerülő jeleket $\mathbf{x}$ -szel, a kimeneti jeleket $\mathbf{y}$ -nal jelöljük, szintén alkalmazva a réteg- és a rétegen belüli indexeket. Így pl. $x_j^{(l)}$ az l -edik neuronréteg i -edik processzáló elemének j -edik bemenetére kerülő jelet jelöli. A kimeneteknél megkülönböztetjük a súlyozott összegző kimenetét (lineáris kimenet) a neuron nemlineáris kimenetétől. A lineáris kimenetet az eddigi jelöléseknek megfelelően s -sel jelöljük. Egy MLP tetszőleges számú rejtett réteget használhat, és mint ahogy a hálók képességeit tárgyaló 1. fejezetben láttuk, ahhoz, hogy univerzális approximátor képességgel rendelkezzen, legalább egy szigmoid nemlinearitással rendelkező rejtett réteget kell tartalmaznia. A kimeneti réteg lehet lineáris, de lehet nemlineáris is. Az MLP bemutatásánál az általánosabb tárgyalás érdekében feltételezzük, hogy nemlineáris kimeneti réteget használunk.

4.1. ábra - A többrétegű perceptron felépítése a hibaszámítással



A fenti felépítésnek megfelelően – összhangban az (1.6.) összefüggéssel – az MLP bemenet-kimenet leképezése – ha mindegyik nemlineáris rétegben azonos a nemlinearitás – a következő:

$$\mathbf{y} = \mathbf{f} \left\{ \mathbf{W}^{(L)} \mathbf{f} \left\{ \mathbf{W}^{(L-1)} \dots \mathbf{f} \left(\mathbf{W}^{(1)} \mathbf{x} \right) \right\} \right\}, \quad (4.1)$$

ahol $\mathbf{W}^{(l)}$ az l -edik réteg neuronjainak súlyvektorait összefogó mátrix, $\mathbf{f}(\cdot)$ pedig az egy réteghez tartozó neuronok kimenetein értelmezendő szigmoid függvényeket jelöli. Egy MLP, még a legegyszerűbb esetben is, amikor csak egy rejtett réteget tartalmaz, paramétereiben nemlineáris leképezést valósít meg.

Az MLP tanítása ellenőrzött tanítás, ahol a hálózat kimenetén értelmezett hiba felhasználásával határozhatjuk meg a kritériumfüggvény vagy kockázat paraméterfüggését. Bár az MLP-nél is szinte kizárólag négyzetes hibafüggvényt szoktak alkalmazni, a nemlineáris paraméterfüggés miatt a hibafelület nem lesz kvadratikuss. Mindössze abban lehetünk biztosak, hogy a hibafelület a tanítandó paraméterek folytonos, differenciálható függvénye, ami a gradiens alapú tanuló algoritmusok alkalmazását lehetővé teszi.

A következőkben az MLP-tanításánál leggyakrabban alkalmazott hibavisszaterjesztéses vagy back-propagation eljárást fogjuk bemutatni, ami szintén gradiens alapú tanító eljárás.

2. Az MLP tanítása, a hibavisszaterjesztéses algoritmus

A hibavisszaterjesztéses algoritmust a 4.1 ábrán látható MLP architektúra segítségével fogjuk származtatni. Az algoritmus egy pillanatnyi gradiensén alapuló iteratív tanuló eljárás, melynek származtatásánál az elemi neuron tanításánál alkalmazott utat fogjuk követni. Minthogy iteratív eljárásról van szó, az egyes súlyok, illetve súlyvektorok lépésenkénti módosító összefüggéseit írjuk fel. A már bevezetett jelöléseken túl k szolgál a diszkrét lépésindex jelölésére. Így $\mathbf{w}_i^{(l)}(k)$ az l -edik réteg, i -edik processzáló elemének a k -adik időpillanatban érvényes súlyvektorát, $w_{ij}^{(l)}(k)$ pedig értelemszerűen ugyanezen súlyvektornak a j -edik komponensét jelöli.

A hálózat tanítását két (aktív) rétegű hálózaton mutatjuk be, de az eredmények természetes módon kiterjeszthetők tetszőleges rétegszámra is.

Először írjuk fel a háló kimeneti négyzetes hibáját a k -adik lépésben.

$$\varepsilon^2(k) = \varepsilon_1^2(k) + \varepsilon_2^2(k) = (y_1(k) - d_1(k))^2 + (y_2(k) - d_2(k))^2. \quad (4.2)$$

A súlymódosításhoz a megfelelő súlyok szerinti pillanatnyi deriváltakat

$$\nabla_y^{(l)}(k) = \frac{\partial \varepsilon^{(l)}(k)}{\partial w_y^{(l)}(k)} \quad (4.3)$$

illetve a súlyvektorok szerinti pillanatnyi gradienseket

$$\nabla_i^{(l)}(k) = \frac{\partial \varepsilon^{(l)}(k)}{\partial w_i^{(l)}(k)} \quad (4.4)$$

kell kiszámítani. A továbbiakban a (k) lépésindexet csak ott fogjuk jelölni, ahol ez az érthetőség miatt szükséges.

A *kimeneti rétegnél* ($l=L=2$) a pillanatnyi derivált (gradiens) megegyezik a szigmoid nemlinearitással rendelkező elemi neuronnál meghatározott deriválttal (gradienssel), hiszen a kimeneti réteg processzálo elemeinek működése semmiben sem tér el az egyetlen, szigmoid kimeneti nemlinearitással rendelkező PE működésétől.

$$\frac{\partial \varepsilon^{(2)}}{\partial w_y^{(2)}} = -2\varepsilon_i \text{sgm}'(s_i^{(2)}) x_j^{(2)} = -2\delta_i^{(2)} x_j^{(2)}, \quad (4.5)$$

illetve vektoros formában írva:

$$\frac{\partial \varepsilon^{(2)}}{\partial \mathbf{w}_i^{(2)}} = -2\varepsilon_i \text{sgm}'(s_i^{(2)}) \mathbf{x}^{(2)} = -2\delta_i^{(2)} \mathbf{x}^{(2)} \quad (4.6)$$

A súlymódosítás tehát

$$\mathbf{w}_i^{(2)}(k+1) = \mathbf{w}_i^{(2)}(k) + 2\mu \varepsilon_i(k) \text{sgm}'(s_i^{(2)}(k)) \mathbf{x}^{(2)}(k) = \mathbf{w}_i^{(2)}(k) + 2\mu \delta_i^{(2)}(k) \mathbf{x}^{(2)}(k) \quad (4.7)$$

A *rejtett réteg* processzálo elemeinél a fenti összefüggések közvetlenül nem alkalmazhatók, mivel nem ismerjük az egyes processzálo elemek kimenetén fellépő hibát. A lánc-szabály alkalmazásával azonban a deriváltakat itt is meg tudjuk határozni, hiszen a rejtett réteg processzálo elemeinek súlytényezői befolyásolják ezen processzálo elemek lineáris (s) és nemlineáris (y) kimeneteit, továbbá ezen kimeneteken keresztül a későbbi rétegek kimeneteit is. Tehát a parciális deriváltak lépésenként számíthatók.

$$\begin{aligned} \frac{\partial \varepsilon^{(2)}}{\partial w_y^{(2)}} &= \frac{\partial \varepsilon^{(2)}}{\partial s_i^{(2)}} \frac{\partial s_i^{(2)}}{\partial w_y^{(2)}} = \frac{\partial (\varepsilon_1^{(2)} + \varepsilon_2^{(2)})}{\partial s_i^{(2)}} \frac{\partial s_i^{(2)}}{\partial w_y^{(2)}} = \frac{\partial \varepsilon_1^{(2)}}{\partial s_i^{(2)}} \frac{\partial s_i^{(2)}}{\partial w_y^{(2)}} + \frac{\partial \varepsilon_2^{(2)}}{\partial s_i^{(2)}} \frac{\partial s_i^{(2)}}{\partial w_y^{(2)}} \\ &= 2\varepsilon_1 \frac{\partial \varepsilon_1}{\partial s_i^{(2)}} \frac{\partial s_i^{(2)}}{\partial w_y^{(2)}} + 2\varepsilon_2 \frac{\partial \varepsilon_2}{\partial s_i^{(2)}} \frac{\partial s_i^{(2)}}{\partial w_y^{(2)}} \end{aligned} \quad (4.8)$$

Ismét a lánc-szabályt alkalmazva a hibakomponensek parciális deriváltjai meghatározhatók.

ε_1 -re bemutatta:

$$\frac{\partial \varepsilon_1}{\partial s_i^{(2)}} = \frac{\partial \varepsilon_1}{\partial s_1^{(2)}} \frac{\partial s_1^{(2)}}{\partial y_i^{(1)}} \frac{\partial y_i^{(1)}}{\partial s_i^{(2)}} = -\text{sgm}'(s_1^{(2)}) w_{1i}^{(2)} \text{sgm}'(s_i^{(1)}) \quad (4.9)$$

majd az ε_2 -re hasonlóan nyerhető deriváltat felhasználva a gradiens a következőre adódik:

$$\begin{aligned} \frac{\partial \varepsilon^{(2)}}{\partial w_y^{(2)}} &= -2\varepsilon_1 \text{sgm}'(s_1^{(2)}) w_{1i}^{(2)} \text{sgm}'(s_i^{(1)}) \frac{\partial s_i^{(2)}}{\partial w_y^{(2)}} - 2\varepsilon_2 \text{sgm}'(s_2^{(2)}) w_{2i}^{(2)} \text{sgm}'(s_i^{(1)}) \frac{\partial s_i^{(2)}}{\partial w_y^{(2)}} \\ &= -[2\delta_1^{(2)} w_{1i}^{(2)} + 2\delta_2^{(2)} w_{2i}^{(2)}] \text{sgm}'(s_i^{(1)}) x_j^{(1)} = -2\delta_i^{(1)} x_j^{(1)}, \end{aligned} \quad (4.10)$$

és így a súlymódosítás

$$\begin{aligned} \mathbf{w}_i^{(1)}(k+1) &= \mathbf{w}_i^{(1)}(k) + 2\mu [\delta_1^{(2)}(k) w_{1i}^{(2)}(k) + \delta_2^{(2)}(k) w_{2i}^{(2)}(k)] \text{sgm}'(s_i^{(1)}(k)) \mathbf{x}^{(0)}(k) \\ &= \mathbf{w}_i^{(1)}(k) + 2\mu \delta_i^{(1)}(k) \mathbf{x}^{(0)}(k), \end{aligned} \quad (4.11)$$

ahol

$$\delta_i^{(1)} = \left\{ \delta_1^{(2)} w_{1i}^{(2)} + \delta_2^{(2)} w_{2i}^{(2)} \right\} \text{sgm}' \left\{ s_i^{(1)} \right\} \quad (4.12)$$

az ún. "visszaterjesztett hiba". A súlymódosítás tehát itt is az LMS algoritmussal formailag megegyező eljárással történik, a hiba helyén azonban súlyozott, "visszaterjesztett hiba" szerepel. A súlyozó együtthatók megegyeznek az adott hálózat-részben az előreccsatolásnál szereplő súlytényezőkkel. A súlymódosítás ennek alapján tetszőleges rejtett rétegre megadható:

$$\begin{aligned} \mathbf{w}_i^{(l)}(k+1) &= \mathbf{w}_i^{(l)}(k) + 2\mu \left(\sum_{r=1}^{N_{l+1}} \delta_r^{(l+1)}(k) w_{ri}^{(l+1)}(k) \right) \text{sgm}' \left\{ s_i^{(l)}(k) \right\} \mathbf{x}^{(l)}(k) \\ &= \mathbf{w}_i^{(l)}(k) + 2\mu \delta_i^{(l)}(k) \mathbf{x}^{(l)}(k) \end{aligned} \quad (4.13)$$

ahol a "visszaterjesztett hiba":

$$\delta_i^{(l)}(k) = \left(\sum_{r=1}^{N_{l+1}} \delta_r^{(l+1)}(k) w_{ri}^{(l+1)}(k) \right) \text{sgm}' \left\{ s_i^{(l)}(k) \right\} \quad (4.14)$$

és N_{l+1} az $(l+1)$ -edik réteg bemeneteinek száma.

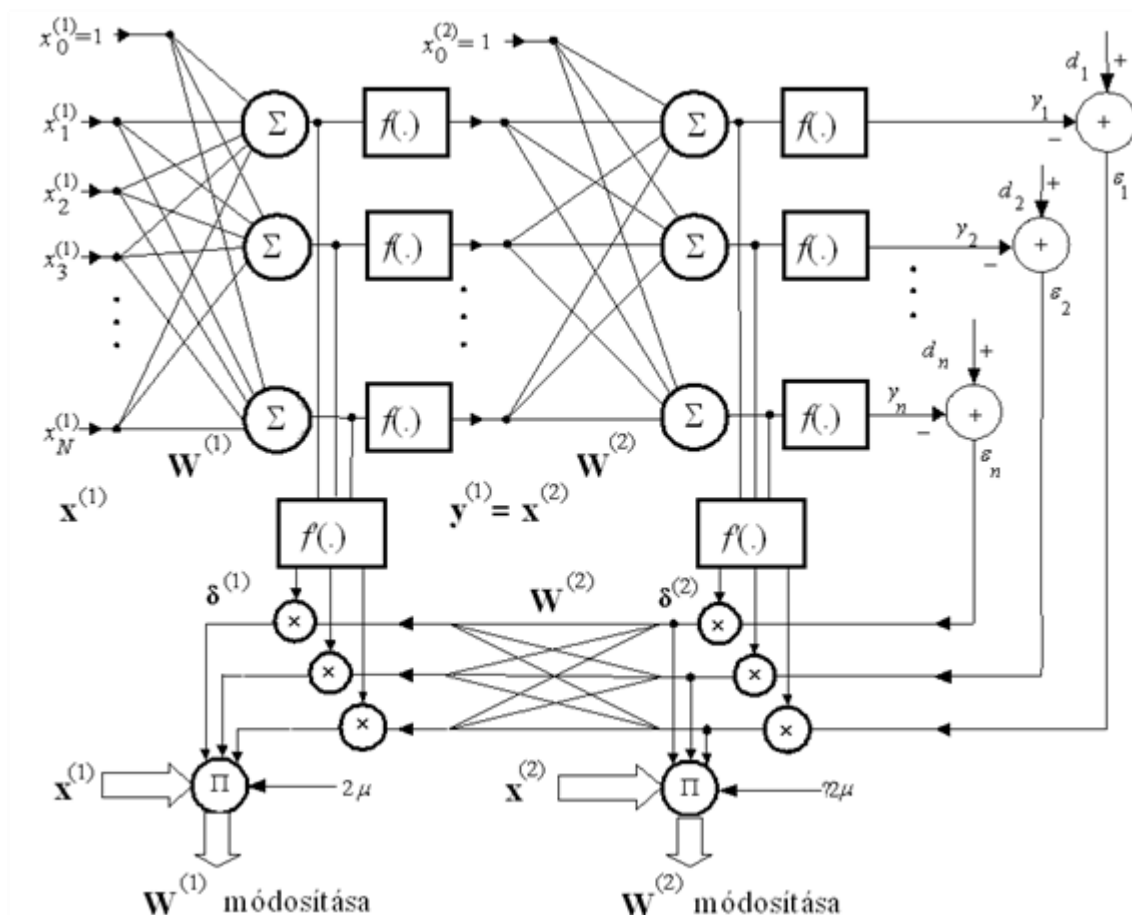
Ha az l -edik réteg összes processzáló elemének súlyvektorait egy, az (1.6.) összefüggésben definiált $\mathbf{W}^{(l)}$ mátrixba fogjuk össze, ahol tehát a mátrix i -edik sora az i -edik processzáló elem súlyvektora, akkor az l -edik réteg összes súlyvektorának módosítása tömören az alábbi formában adható meg:

$$\mathbf{W}^{(l)}(k+1) = \mathbf{W}^{(l)}(k) + 2\mu \delta^{(l)} \mathbf{x}^{(l)r}(k). \quad (4.15)$$

Itt $\delta^{(l)}$ a (4.14) összefüggéssel megadott $\delta_i^{(l)}$ visszaterjesztett "hibákból" képezett oszlopvektor. A súlymódosítást is tükröző hálózatarchitektúra két aktív rétegű hálózathál a 4.2 ábrán látható. (Az ábrán a kimeneti nemlinearitást jelölő sgm helyett az általánosabb $f(\cdot)$ jelölés szerepel.)

Az előbbieken bemutattuk a hibavisszaterjesztéses hálózat alapvető felépítését, működését. Az 1. fejezet approximációval foglalkozó részei pedig annak matematikai hátterét foglalták össze, hogy egy többretegű, előreccsatolt hálózat egyáltalán milyen képességekkel rendelkezik. A hálózatok elvi alapjainak ismerete azonban még nem elegendő ahhoz, hogy e hálózatokat hatékonyan alkalmazzunk is tudjuk különböző gyakorlati feladatok megoldására. A következőkben azokkal a kérdésekkel foglalkozunk, melyek az elvi háttér birtokában most már azt is biztosítják, hogy a hálót hatékonyan tudjuk alkalmazni gyakorlati feladatok megoldására.

4.2. ábra - Az MLP tanítása back-propagation algoritmussal



3. Az MLP konstrukciójának általános kérdései

Az MLP gyakorlati alkalmazásánál – azon túl, hogy meg kell találnunk az adott feladat olyan megfogalmazását, amely alkalmassá teszi, hogy a megoldást neurális hálózat segítségével keressük – számos kérdés merül fel. Ezek a kérdések a konkrét hálózatok konstruálásával, ill. tanításával, stb. kapcsolatosak. Az itt tárgyalni kívánt kérdések valójában mind a gyakorlati alkalmazások kapcsán merülnek föl, a válaszok azonban sokszor komoly elméleti vizsgálat eredményeként születnek. Ugyanakkor az elmélet számos a gyakorlati alkalmazással összefüggő kérdésre még nem ad közvetlenül felhasználható választ. Minthogy a kérdésekre mindenképpen választ kell adjunk, ilyenkor a praktikus megközelítés marad.

A megválaszolendő legfontosabb kérdések a következők:

- mekkora (hány réteg, rétegenként hány processzáló elem) hálózatot válasszunk,
- hogyan válasszuk meg a tanulási tényező, μ értékét,
- milyen kezdeti súlyértékeket állítsunk be,
- hogyan válasszuk meg a tanító és a tesztelő minta készletet,
- hogyan használjuk fel a tanító pontokat, milyen gyakorisággal módosítsuk a hálózat súlyait,
- meddig tanítsuk a hálózatot, stb?

Ezekre és hasonló kérdésekre, melyeket konkrét alkalmazásoknál meg kell válaszolni, minden szempontból kielégítő általános válaszok jelenleg még nincsenek. Nincsenek, vagy csak részben vannak olyan bizonyított eredmények, amelyek adott alkalmazáshoz megadnák, hogy hogyan kell megválasztani a szükséges hálózat méretét. De általában az előbb feltett többi kérdésre is csak tapasztalati válaszok adhatók. Ez az oka annak, hogy az MLP sikeres alkalmazásához jelenleg meglehetősen sok tapasztalat szükséges, amit csak úgy szerezhethünk meg, ha számos, jellegében eltérő feladat megoldására vállalkozunk.

A következőkben az MLP és a back-propagation tanítási eljárással kapcsolatos, az előbbieken felsorolt néhány gyakorlati szempontot vesszük sorra. A neurális hálózatok alkalmazásánál felmerülő általános kérdésekkel részletesebben a 13. fejezetben foglalkozunk.

3.1. A hálózat méretének megválasztása

A hálózat méretének megválasztása a rétegek számának és az egyes rétegeken belül a neuronok számának meghatározását jelenti. Az elméleti eredmények szerint adott feladat megoldásához – amennyiben az nemlineáris függvényapproximációt igényel – legalább háromrétegű – tehát két tanítható aktív réteggel rendelkező – hálózatra van szükség. A rétegek számának növelése – három, esetleg még több tanítható réteg alkalmazása – azonban megkönnyítheti a feladat megoldását, illetve rétegenként kevesebb processzáló elem felhasználása is elegendőnek bizonyulhat.

A hálózat méretének meghatározására – a rejtett neuronok számának becslésére – évek óta komoly erőfeszítések történnek. A nem kevés matematikai eredmény, pl. [Kür92], [Mha94], [Mai99], [Sca98] felső becslést tud adni a rejtett neuronok számára. A kapott eredmények azonban túl pesszimisták. [Kür92] például a két rejtett réteggel dolgozó MLP-k komplexitására ad meg korlátokat. E szerint adott hibájú approximációhoz az első rejtett rétegben $N_1 = M(l+1)$, a másodikban pedig $N_2 = l^2(l+1)^n$ a neuronok elégséges száma, ha $N \geq 2$ a bemenet dimenziója, és $l \geq 2N+1$. Ez az eredmény egyrészt jól illusztrálja az ún. „dimenzió átka” (*curse of dimensionality*) problémát, ami röviden azt jelenti, hogy a dimenzió növelésével a komplexitás exponenciálisan nő, másrészt azt mutatja, hogy pl. egy 10-dimenziós problémát már nem tudnánk megoldani a szükséges óriási neuron-szám miatt, ami a gyakorlatnak teljes mértékben ellentmond. Vera Kůrková eredménye ugyanakkor konstruktív, szemben Vitalij Maiorov és Allan Pinkus [Mai99] eredményével, amely bár jóval kisebb háló szükségességét mondja ki, de a konstrukció módjáról nem mond semmit.

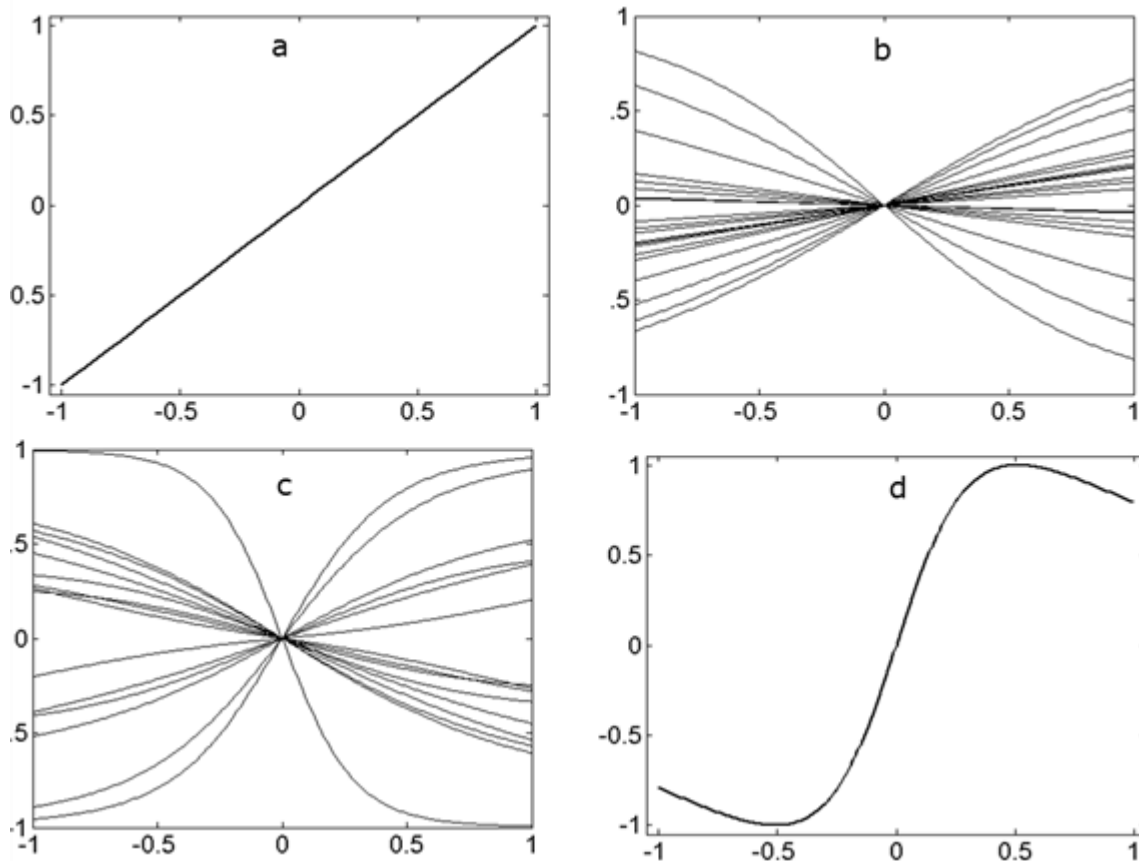
E felső becslések tehát a praktikus szükséges neuronszámnál jóval nagyobb értékeket adnak, így az eredményeknek elsősorban elméleti jelentőségük van, a gyakorlatban még nem alkalmazhatók. Ezeknek a pesszimista elméleti eredményeknek – legalábbis részben – az az oka, hogy az eredmények túl általánosak, mivel olyan korlátokat adnak meg, melyek bármilyen feladatra érvényesek, köztük elfajult, a gyakorlatban igen ritkán előforduló feladatokra is. Konkrét esetekben a neuronháló konstrukciójánál ezért a processzáló elemek számának meghatározására jelenleg csak a tapasztalatokra támaszkodhatunk.

A gyakorlatban a hálózat méretének meghatározásánál kétféle úton járhatunk. Egyrészt kiindulhatunk egy nagyobb hálózattól, amiről bizonyos tapasztalat birtokában nagy valószínűséggel állíthatjuk, hogy a feladat megoldásához elegendő lesz, majd megkísérelhetjük a hálózatban megmutatkozó redundancia minimális értékre szorítását. Másrészt kisebb méretű hálózattól kiindulva is megpróbálhatjuk a feladatot megoldani, és amennyiben ez nem sikerül, fokozatosan bővíthetjük a hálózatot újabb processzáló elemekkel, ill. rétegekkel.

A redundancia csökkentése azt jelenti, hogy megpróbáljuk megkeresni a felesleges súlyokat, processzáló elemeket, esetleg rétegeket, majd ezeket a hálózattól kimetszve a maradék, egyszerűsített hálózattal oldjuk meg a feladatot. Felesleges súlyoknak, processzáló elemeknek, esetleg rétegeknek azok a hálózatelemek tekinthetők, melyek kihagyásával a feladat megoldható, tehát amelyek vagy nem vesznek részt a kimenet előállításában (pl. egy súly értéke vagy egy processzáló elem kimenete a tanítás során végig nulla, esetleg konstans) vagy amelyek szerepét más hálózatelem is betöltheti, így a hálózat képességeinek redukciója nélkül elhagyhatók.

Ilyen esetre példa, ha két súlyérték vagy processzáló elem kimenete együtt változik a tanítás során, vagy az egyik kimenet a másik konstans-szorosa. Ezekre az eshetőségekre mutat példát a 4.3. ábra, amely a tanító minták függvényében ábrázolja az egyes processzáló elemek kimenő jelének alakulását egy nemlineáris leképzést megvalósító négyrétegű (két rejtett rétegű) hálózatonál.

4.3. ábra - A hálózat processzáló elemeinek kimenete a bemenet függvényében



A 4.3. (a) ábra a bemeneti réteg "processzáló elemei"-nek kimenetét mutatja. Minthogy ez a réteg csak erősítési, jelszétosztási feladatot tölt be, nyilvánvaló, hogy a bemeneti "processzáló elemek" egységnyi átvitelt valósítanak meg. A 4.3. (b), (c) és (d) ábrák rendre a második, harmadik és negyedik, egyben kimeneti réteg processzáló elemeinek kimenetét adják meg. Jól látható, hogy mind a második, mind a harmadik rétegben vannak olyan processzáló elemek, melyek a teljes működési tartományban csaknem nulla kimenetet produkálnak, illetve olyan processzáló elem-párok, amelyek kimenete a teljes tartományban együtt mozog, így ezen processzáló elemek, illetve az együttmozgó processzáló elemek egyikének jelenléte felesleges, a hálózathoz kimetszhetők.

A hálózatok méretének bizonyos összeköttetések kimetszésével (*pruning*) történő csökkentésére egyéb módszerek is alkalmazhatók. Ezek a módszerek adott súlyok kimeneti hibára gyakorolt hatása alapján végzik a kimetszéseket. Ha egy súly értékét nullára változtatjuk – ezáltal hatását kiiktatjuk –, akkor a kimeneti hiba változása alapján eldönthető, hogy az adott súly lényeges-e a hálózat helyes működése szempontjából vagy nem. Ha a hibanövekedés túl nagy, a kérdéses súlyra (a megfelelő összeköttetésre) szükség van, ellenkező esetben az összeköttetés megszüntethető, kimetszhető.

E módszerek két csoportba sorolhatók:

- az egyik csoportba tartozó eljárások a kimeneti hiba egyes súlyok szerinti érzékenységeinek becslésén alapulnak: a hálózathoz a legkisebb érzékenységgű súlyok metszhetők ki,
- a másik csoportba tartozó módszereknél a kritériumfüggvényhez egy újabb ún. büntető tagot adunk, ezáltal egyfajta regularizációt alkalmazunk.

Az érzékenység-becslés alapján dolgozó eljárások a hibafelület súlyok szerinti Taylor-soros közelítéséből indulnak ki. Másodfokú közelítést alkalmazva szükségük van a hibafelület másodrendű deriváltjait tartalmazó Hesse-féle $\mathbf{H}$ mátrixra. Az OBD (*Optimal Brain Damage*) [LeC90] feltételezi, hogy a Hesse mátrix diagonális, amely általában nem realiztikus feltételezés, az OBS (*Optimal Brain Surgeon*) [Has92] e feltételezés nélkül dolgozik, így a hibafelületről pontosabb információt használ fel és egyrészt nagyobb mértékű méret-csökkentést eredményez, másrészt ezt valóban a legkevésbé fontos súlyok elhagyásával teszi.

A büntető tagot alkalmazó módszereknél a tanulás során a hálózat súlyainak csökkentésére törekszünk. Ennek megfelelően a büntető tag lehet pl. a hálózat súlyainak négyzetes összege vagy a súlyok abszolút értékének az összege, ami azt jelenti, hogy a kritériumfüggvény minimalizálása során a megfelelő leképezés megtanulásán túl a minél kisebb súlyok elérése a cél. A megtanított hálózatnál a nullához közeli súlyok elhagyásával a hálózat mérete csökkenthető. A módosított kritériumfüggvény lehet pl. a következő [Ish96]:

$$C_r(\mathbf{w}) = C(\mathbf{w}) + \lambda \sum_{i,j} |w_{ij}| \quad (4.16)$$

ahol $C(\mathbf{w}) = \sum_k \sum_{j=1}^{n_k} \left\{ d_j^{(k)} - y_j^{(k)} \right\}^2$ a hálózat átlagos négyzetes hibája az összes kimenetre és az összes tanító pontra végezve az átlagolást, a második tag a büntető tag, amely a minél kisebb értékű súlyok elérését segíti elő, λ pedig a büntető tag relatív súlyát meghatározó együttható. A súlymódosítás a deriválás elvégzése után:

$$\Delta w_{ij} = \mu \left(-\frac{\partial C}{\partial w_{ij}} \right) - \mu \lambda \operatorname{sgn}(w_{ij}) \quad (4.17)$$

vagyis a súly abszolút értéke minden tanító lépésben konstans értékkel csökken. Az ilyen tanulási eljárást felejtő tanulásnak (*weight decay*) is nevezik, hiszen minden egyes tanító lépésben egy súlyfelejtő tagot is alkalmazunk. A felejtő tanulás hátránya, hogy a felejtő tag minden súlynál megjelenik, azoknál is, melyek fontosak a hálózat megfelelő működéséhez, így várhatóan nagyobb hibát eredményez, mint a “sima” back-propagation eljárás. E hatás mérsékelhető, ha a felejtés csak azon súlyokra vonatkozik, melyek abszolút értéke egy adott küszöböt (Θ) nem halad meg. A módosított felejtő tanulásnak megfelelő kritériumfüggvény:

$$C_r(\mathbf{w}) = C(\mathbf{w}) + \lambda \sum_{|w_{ij}| \geq \Theta} |w_{ij}| \quad (4.18)$$

A konstans mértékű felejtéssel szemben szokás exponenciális jellegű felejtést is alkalmazni. Ez utóbbi a súlyok négyzetösszegével arányos büntető tag mellett nyerhető [Pla86]:

$$C_r(\mathbf{w}) = C(\mathbf{w}) + \lambda \sum_{i,j} w_{ij}^2 \quad (4.19)$$

A (4.16) és (4.18) összefüggések konkrét példái a regularizációs összefüggéseknek (ld. 2.1. alfejezet), ahol a regularizációs eljárással a négyzetes hiba csökkentése mellett egy járulékos feltétel teljesítése, a súlyok összegének minimalizálása is elérendő cél.

A redundancia csökkentése annak ellenére hasznos lehet, hogy nagyobb méretű hálózat alkalmazása – a nagyobb szabadságfok következtében – sok esetben segíthet a tanítás gyorsításában. A hálózatban meglévő redundanciák azonban általában a szükséges számítások mennyiségét növelik, így nemcsak a hálózat tanításának, hanem a tanított hálózat működésének is csökkentik a sebességét.

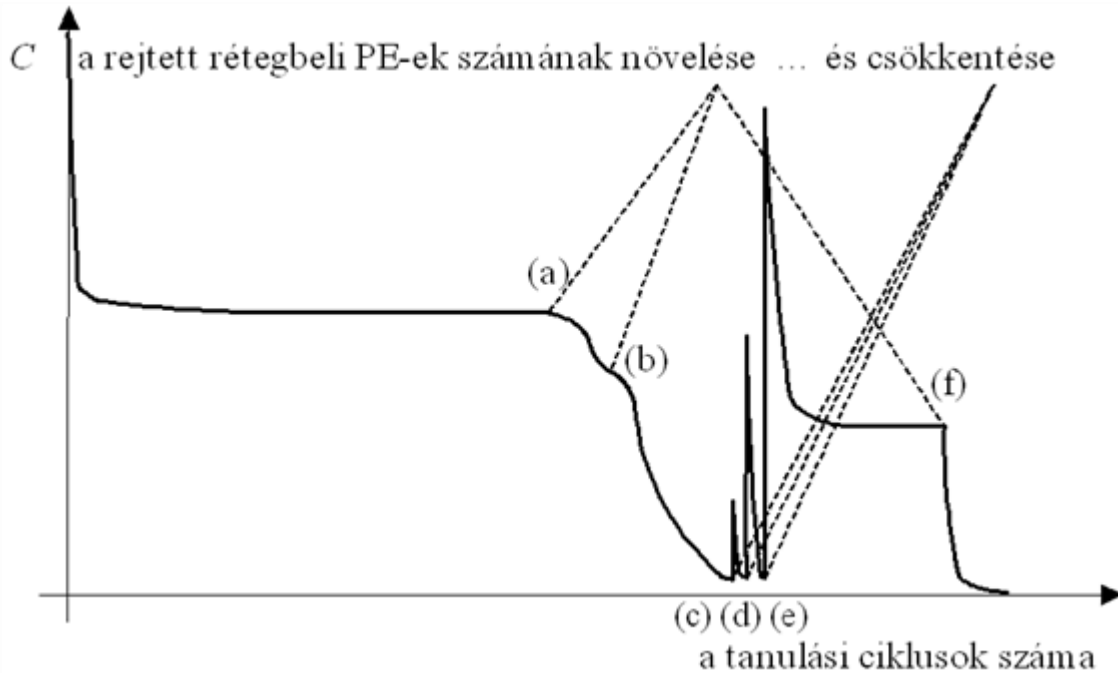
A súlyok nagyságának (a súlyvektorok hosszának) leszorítását célzó regularizációs tag alkalmazása nem csupán arra szolgál, hogy elősegítse a hálózat egyszerűsítését, hanem a kisebb súlyok önmagukban is hozzájárulhatnak a háló általánosító-képességének javításához. A regularizáció ilyen szerepéről a bázisfüggvényes hálózatok (6. fejezet) és a kernel gépek (7. fejezet) bemutatásánál részletesen is lesz szó.

A hálózat bővítésére akkor lehet szükség, ha adott méretű hálózatnál a tanítás hatására már nem csökken a hiba. Újabb processzáló elem beiktatásával a hálózat ismét taníthatóvá válhat, vagyis a kimeneti hibát tovább csökkenthetjük. A folyamat az 4.4 ábrán követhető. Látható, hogy a processzáló elemek számának növelése (az ábrán az (a) és (b) pontok) a hálózat tovább-taníthatóságát, vagyis a hiba további csökkenését eredményezheti. A már kis hibájú hálózatnál ezután kísérletet tehetünk a hálózat egyszerűsítésére. A processzáló elemek számának csökkentése (az ábrán a (c), (d) és (e) pontok) nyilvánvalóan a hiba hirtelen megnövekedését eredményezi, ami azonban tovább-tanítással ismét gyorsan lecsökkenthető, ha az egyszerűbb hálózat is képes a feladat megoldására ((c) és (d)). Ha azonban az így kapott egyszerűbb hálózat már nem alkalmas a feladat megfelelő megoldására (e), a hiba adott határ alá nem szorítható, a hálózat nem tanítható tovább. Jobb eredmény tehát ismét csak akkor nyerhető, ha újra növeljük a hálózat méretét (f).

Megjegyezzük, hogy az alulról felfelé építkező eljárás, az egyszerűbbtől a bonyolultabb irányában történő hálózat módosítás és az ellenkező irányú, bonyolulttól az egyszerűbb felé történő módosítás általában eltérő hálózat-felépítést és hálózat-méretet eredményez.

A megfelelő méretű hálózat megkeresésének más módszerei is vannak. Alkalmazhatjuk pl. a genetikus algoritmusokat is a feladat szempontjából optimális hálózat-struktúra meghatározására. Ebben az esetben a genetikus algoritmus populációi különböző méretű hálózatokból épülnek fel, és az egyes generációk a feladat szempontjából egyre jobb képességekkel rendelkező hálózatokat fognak tartalmazni. Bármelyik megoldást is választjuk több, különböző méretű hálózat tanítását kell elvégezni, ami a megfelelő méretű hálózat megtalálását és a feladat megoldását meglehetősen lassúvá teszi.

4.4. ábra - A hálózat taníthatóságának alakulása változó hálózat-méret mellett



3.2. A tanulási tényező és a súlyok kezdeti értékeinek megválasztása

Már az elemi neuronnál láttuk, hogy a tanulási tényező – amit szokás bátorsági tényezőnek is hívni, hiszen az a szerepe, hogy megmondja milyen bátran mehetünk a negatív gradiens irányában – megválasztása alapvetően befolyásolja a tanító eljárás alakulását. Sőt, azt is láttuk, hogy nem megfelelően megválasztott tanulási tényező mellett az eljárás divergens lesz: a hiba nem csökkenni, hanem nőni fog. A bátorsági tényező megfelelő megválasztása tehát az MLP-nél is fontos. Ugyancsak fontos kérdés, hogy adott hálózat tanításánál milyen kezdeti súlyértékekből induljunk ki, hiszen ez is döntően befolyásolja a tanulás alakulását.

3.2.1. A tanulási tényező, μ megválasztása

A μ tanulási tényező megválasztására jelenleg nincs egyértelműen javasolható egyszerű módszer. Mint a 2. fejezetben láttuk, egy lineáris processzáló elem esetén meghatározható μ azon értéktartománya, amelyen belül a konvergencia biztosított, sőt az optimális konvergencia-sebességet biztosító μ is megadható. Az optimális konvergencia-sebességet, vagy legalább a konvergenciát biztosító μ meghatározása többrétegű hálózatnál jóval nehezebb, hiszen itt nem biztosítható a kvadratikusan csökkenő hibafüggvény, így optimális μ -ról nem is beszélhetünk, sokkal inkább olyan μ megválasztási stratégiáról, ahol nem egy rögzített értékű tanulási tényezőt alkalmazunk, hanem a μ értékét lépésről-lépésre változtatjuk. Az is elképzelhető, hogy processzáló elemenként, vagy legalább rétegenként eltérő μ megválasztása indokolt. A legtöbb esetben μ értékét tapasztalati úton határozzák meg általános tendenciák felhasználásával. Kis μ egy minimumhely pontosabb megtalálását, de lassú konvergenciát eredményez, míg nagy μ mellett a konvergencia kezdetben gyorsabb lehet, de a minimumhely megközelítése nehezebb, sőt bizonyos értéknél nagyobb μ mellett a tanítási eljárás konvergenciája sem biztosított, a tanulás alatt a hálózat "elszállhat".

A gyorsabb kezdeti konvergencia és a minimumhely megfelelő pontosságú megközelítése érhető el változó, lépésfüggő μ alkalmazásával. Ebben az esetben valamely kezdeti, viszonylag nagy értékből kiindulva valahány lépésként csökkentjük μ -t. A csökkentés menetét – mint azt a sztochasztikus szélsőérték-kereső eljárásoknál

(2.5.2. rész) említettük – leginkább a sztochasztikus approximáció konvergenciáját biztosító együtthatósorozat csökkentésének menete szerint szokás megválasztani. E szerint a következő követelményeket teljesítő $\mu(k)$ sorozat alkalmazható:

$$\sum_{k=1}^{\infty} \mu(k) = \infty, \quad \sum_{k=1}^{\infty} \mu^2(k) < \infty. \quad (4.20)$$

Ebben az összefüggésben k nem feltétlenül a lépésindexet jelöli, μ csökkentésére rendszerint csak néhány lépésenként kerül sor. Láttuk, hogy a fenti feltételeknek eleget tesz pl. a $\mu(k) = \mu_0 / k$ sorozat. A kezdeti érték (μ_0) megválasztására tapasztalati összefüggés, hogy minél nagyobb a hálózat és minél több az egyes neuronok bemeneteinek száma, annál kisebbre kell választani.

A tanulás konvergenciájának sebessége növelhető adaptív μ választással is. A szokásos eljárás: μ változtatását attól tesszük függővé, hogy a súlymódosítások csökkentik-e a kritériumfüggvény (kockázat) értékét. Ha a hiba nem csökken, akkor μ értéke túl nagy, csökkenteni kell. Ezzel szemben, ha több egymást követő tanító lépés során a hiba folyamatosan csökken, akkor lehetséges, hogy túl óvatosan választottuk meg μ -t, valószínűleg nagyobb érték mellett is biztosított a hiba csökkenése, érdemes tehát nagyobb μ -vel megkísérlni a tanítást. Az irodalomban különböző módosítási stratégiákat találunk, amelyeknél μ növelésének vagy csökkentésének mértéke, illetve a módosítás feltétele különböző. Kis kezdő μ esetén, ha a tanító lépések során a hiba csökken, növeljük a tanulási tényezőt exponenciálisan, míg ha a hiba jelentősen nő, csökkentjük ugrásszerűen [Vog88]. A növelés feltételeként szabható, hogy a hibafelületen egy (lokális) minimumhelyhez képest éppen hol tartunk. Ha a hiba a módosítások során csökken, miközben a gradiens előjele nem változik, növeljük μ értékét. Ezzel szemben, ha a gradiens előjele az egymást követő súlymódosításoknál gyakran változik, ez azt jelenti, hogy egy minimumhely körül ugrál a pillanatnyi megoldásunk, tehát μ értékét csökkenteni kell [Cha87].

Az adaptív μ alkalmazása egzaktabb alapokon is lehetséges, ha figyelembe vesszük a hibafelület alakját. Láttuk, hogy kvadrátikus hibafelületnél μ optimális értéke a felület alakjától függ. Többretegű perceptron esetében a hibafelületen kiterjedt lapos tartományok, továbbá ugrásszerűen változó, hirtelen meredek szakaszok is lehetnek, így attól függően, hogy a hibafelület mely részén mozgunk más és más μ értékre lenne szükség. Az adott lépésben optimális μ meghatározásához azonban az adott hibafelület-rész alakjáról kell, hogy ismeretekkel rendelkezünk. Dinamikus tanulási tényező optimalizálásra ad eljárást [Yu95], amelynél a hibafelületnek a tanulási tényező szerinti első és második deriváltjára vonatkozó információ segítségével lehetséges gyorsabb konvergenciát biztosító μ megválasztása. (A tanulási tényező megválasztásának kérdésére a későbbiekben a *hibavisszaterjesztési algoritmus változatai* c. részben még visszatérünk.)

3.2.2. A súlyok kezdeti értékeinek meghatározása

A súlyok kezdeti értékeinek beállítására – bár természetesen ez is befolyásolja a tanulás konvergenciájának gyorsaságát – jelenleg szintén nincs matematikailag megfogalmazható összefüggés. A kezdeti súlyvektor a hibafelületen a kezdeti pont helyzetét határozza meg, így minél messzebb van ez a pont a megoldás helyétől, ill. minél bonyolultabb a kezdeti pont és a megoldás között a kritériumfelület, annál lassabban tanul a hálózat. Amennyiben valamilyen a priori ismeret rendelkezésünkre áll a hibafelület alakjáról, elhelyezkedéséről, akkor ezt célszerű figyelembe venni a kezdeti értékek megválasztásánál, hiszen így jelentősen gyorsíthatjuk a tanulást. A priori ismeret hiányában a véletlenszerű súlybeállítás a leginkább megfelelő: az egyes súlyokat egy egyenletes eloszlású valószínűségi változó különböző értékeire választhatjuk. A véletlen kezdeti értékek a szimmetriák elkerülését biztosíthatják, megakadályozva, hogy különböző neuronok hasonló leképezést valósítsanak meg és így nemkívánt redundancia jelenjen meg a hálózatban. A véletlen tartomány nagysága befolyásolhatja a konvergencia sebességét, azonban általában itt is csak tapasztalati összefüggés állítható fel: minél nagyobb a hálózat, annál kisebb véletlen értékek választása célszerű. A háló méretétől nem független, viszonylag kis értékek megóvhatnak attól, hogy az egyes neuronok a tanítás túl korai szakaszában az aktivációs függvény telítési szakaszára kerüljenek.

A hálósúlyok kezdeti értékének megállapításánál azt is figyelembe kell venni, hogy az egyes neuronok kimenetein telítődő nemlineáris függvényt, szigmoid függvényt alkalmazunk. Mind a szigmoid nemlinearitással dolgozó elemi neuron tanításánál, mind az MLP tanításánál láttuk, hogy a súlymódosítás összefüggésében a nemlinearitás deriváltja, $\text{sgm}'(\cdot)$ is szerepel, amely, ha a telítési szakaszon vagyunk, nagyon kicsi is lehet. Törekedni kell tehát arra, hogy a kezdeti értékek beállításánál minél több neuron az „aktív” tartományában működjön, vagyis a neuronok lineáris kimenetei a szigmoid függvények meredek szakaszára essenek. Mint láttuk, ebben a kis kezdeti súlyok segíthetnek. Derrick Nguyen és Bernard Widrow azonban hatékonyabb módszert javasoltak [Ngu90].

A Nguyen-Widrow eljárás olyan módon próbálja a kezdeti súlyokat beállítani, hogy a rejtett réteg neuronjainak aktív tartományai a bemeneti térben lehetőleg egyenletesen legyenek szétosztva, vagyis az egyes neuronok a bemeneti tér más-más tartományában legyenek aktívak és lehetőleg ne legyen olyan bemeneti tartomány, ahol egyetlen neuron sem aktív. Ennek érdekében egyrészt a súlyok célszerű „hosszára” adnak egy összefüggést, másrészt az egyes neuronok eltolásértékének (w_0) megfelelő megválasztásával biztosítják az aktív tartományok kellő szétosztását a bemeneti térben. Az eljárás gyakorlati alkalmazása azt mutatja, hogy a Nguyen-Widrow kezdeti-érték beállítással a konvergencia jelentősen, akár nagyságrendekkel is gyorsítható.

3.3. A tanító pontok felhasználása, a háló minősítése

A hálók tanításánál a tanítópontokat általában többször, ismételten felhasználjuk, és a tanítást addig folytatjuk, amíg az eljárás vagy magától le nem áll, vagy megfelelően kis hibát el nem érünk. Bár elvileg a gradiens eljárások egy minimumpont elérésekor önmaguktól leállnak, a pillanatnyi gradiens alapú eljárásoknál semmi sem garantálja, hogy nulla gradiens egyáltalán elő fog fordulni. Így legtöbbször a tanulás leállításáról külön gondoskodni kell. Általában előre nem határozható meg a hálózat hibájának alakulása a tanító lépések függvényében, így azt sem lehet megmondani, hogy megfelelően kis hiba eléréséhez hány tanító lépés szükséges. A hálózat "jóságának" kiértékelése, minősítése amúgy is számos elméleti és gyakorlati problémát vet fel. Nem csupán a tanító lépések, hanem a tanító pontok számának meghatározása is kérdéses. (A tanító pontok számának kérdésére ennek a résznek a végén még röviden kitérünk.)

3.3.1. A tanító pontok felhasználása

További lehetőségeket jelent, hogy a súlymódosításokat pontonként, vagy kötegelt (*batch*) eljárás szerint, csak a teljes tanító készlet (*epoch*) felhasználása után végezzük. Ezek az eljárások a súlykorrekció gyakoriságában térnek el. Az első esetben minden tanító pont alkalmazásánál meghatározzuk a kimeneti hibát és az ebből számítható korrekciós értékkel lépésenként el is végezzük a súlymódosítást. A batch eljárásnál a teljes tanító készlet összes mintáját egyenként, egymás után ráadva a hálózatra minden esetben kiszámítjuk a hibát, de az egyes tanító pontokhoz tartozó hibával nem végzünk korrekciót, hanem az egyedi korrekciókat összegezzük és az akkumulált módosítást végezzük csak el.

Az első esetben tehát a súlymódosítások száma megegyezik a tanító pontok számának és a tanítási ciklusok számának szorzatával, míg a batch eljárásnál csak ciklusonként korrigálunk, ahol egy ciklus a teljes tanító készlet egyszeri felhasználását jelenti. Természetesen a batch eljárásnál nem feltétlenül kell a teljes tanító készletet felhasználni mielőtt egy súlymódosításra sor kerülne. Így valójában a pontonkénti súlymódosítás és a teljes tanító készletenkénti súlymódosítás között bármely eset alkalmazható, amikor a súlymódosítást a pontonkénti esetenél ritkábban, de az epochonkénti esetenél gyakrabban végezzük el.

A két szélső eset nemcsak a korrekciók számában tér el, hanem jellegében is. A mintánkénti tanítás – a pillanatnyi gradiens keresztül – mintegy zajt visz a rendszerbe, hiszen a pillanatnyi hibaértékek által meghatározott hibafelületen mozgunk, a batch eljárás ezzel szemben átlagoló jellegű. Megjegyezzük, hogy a mintánkénti tanítás nagyon kis μ -vel a batch eljárással ekvivalens megoldásra vezet, ami nem meglepő, hiszen ilyenkor lépésenként olyan kismértékű módosítást hajtunk végre, hogy az átlagoló jelleg megmarad. A mintánkénti tanításból adódó "zaj" sztochasztikus elemet visz a tanításba, ezáltal segíthet a lokális minimumokba való beragadás elkerülésében. Sok esetben tudatosan alkalmazunk zajt a hálózatban – a súlyokhoz vagy a bemeneti mintákhoz véletlenszerű értékeket adunk – így kíséreljük meg a hálózatot egy lokális minimumból kiugratni (ld. még sztochasztikus szélsőérték kereső eljárások, 2.5.2. rész).

3.3.2. A háló minősítése

További kérdés, hogy egyáltalán hogyan minősítsünk egy hálózatot. Minősítésre több okból is szükségünk van. Egyrészt ennek alapján dönthetjük el, hogy a háló kellően megtanulta-e a feladatot, leállhatunk-e már a tanulással, másrészt ez biztosítja, hogy tudunk arra vonatkozóan mondani valamit, hogy mennyire sikerült a feladatot egyáltalán megtanítani.

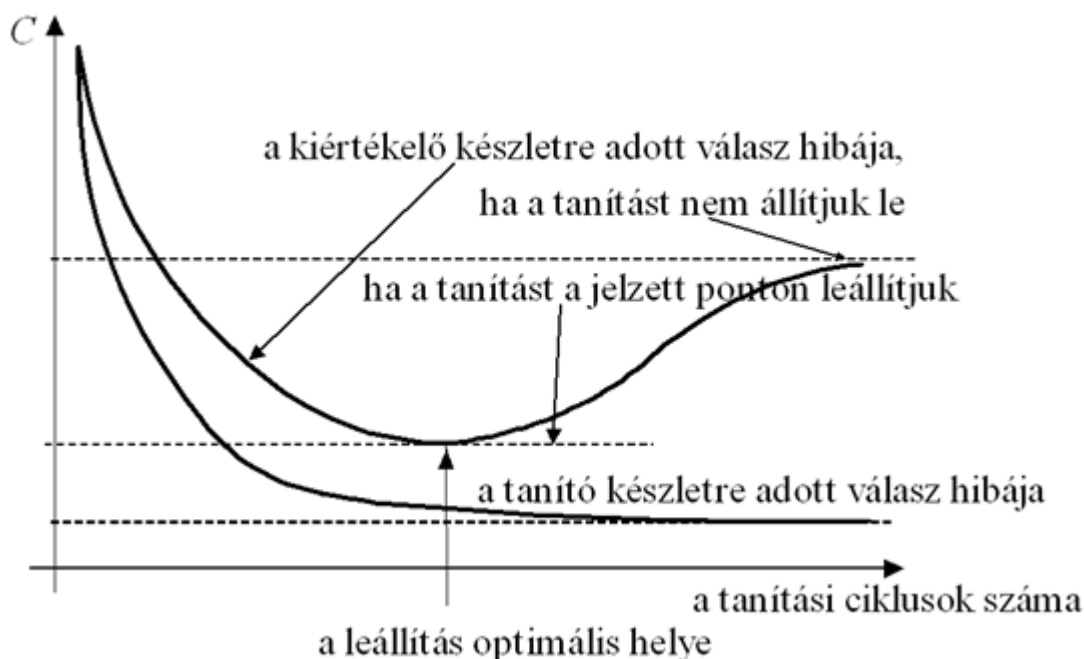
A tanulással foglalkozó 2. fejezetben láttuk, hogy a minősítés elvben a kockázat meghatározásával lenne lehetséges. Mivel azonban ehhez a mintapontok eloszlását ismerni kellene, ez az út nem járható. Azt is láttuk, hogy a valódi kockázat – megfelelő feltételek fennállta esetén – a tapasztalati kockázat alapján is becsülhető (ERM elv). A feltételeket a statisztikus tanuláselmélet keretében fogalmazták meg, amely elmélet a valódi kockázatra felső korlátokat is származtatott.

Az MLP minősítésénél az ERM elv és a felső korlátok nem alkalmazhatók. A felső korlátok alkalmazása a háló által megvalósítható függvényosztály VC-dimenziójának ismeretét és azt kívánja, hogy a VC-dimenzió véges legyen. MLP esetén a VC-dimenzió általában nem határozható meg, ugyanakkor az megmutatható, hogy a VC-dimenzió akár végtelen is lehet. Végtelen VC-dimenzió mellett a felső korlátokat megadó (2.31) és (2.33) összefüggések értelmezhetetlenek. A többrétegű perceptronok „jóságának” meghatározására azonban a felső korlátok még akkor sem lennének alkalmazhatók, ha a VC-dimenzió véges lenne, ugyanis a felső korlátok általában túl pesszimisták, konkrét gyakorlati feladat megoldásánál a hálók minősítésére nem használhatók.

A tanítás leállítása érdekében történő minősítés a következőképpen történhet. Adott tanító készlet mellett szükségünk van egy ún. kiértékelő mintakészletre (*validation set*) is. A hálózat tanítására csak a tanító készlet mintáit használjuk, míg az adott számú tanító lépésen átesett hálózat minősítése a kiértékelő készletre adott válaszok alapján lehetséges. A kiértékelő készlet olyan be- és kimeneti mintapárokat tartalmaz, amelyek a megoldandó feladatból származnak, és a hálózat normál működési tartományába esnek, de amelyeket a tanításra nem használtunk fel. Az MLP hálózatok általánosító, interpolációs képessége következtében azonban azt várjuk, hogy a megtanított hálózat ezekre, a még sohasem látott mintákra is jó válaszokat szolgáltatson. Amennyiben csak a tanítópontokra adott válaszok alapján értékelünk, *túltaníthatjuk* a hálózatot. A kiértékelő készlet szerepe éppen az, hogy a túltanítást megakadályozzuk. A tanító eljárás során, bizonyos számú tanító lépést követően a validációs készletre adott válasz hibája alapján dönthetjük el, hogy a háló már kellő mértékben megtanulta-e a feladatot. A minősítő készlet mintáit tehát szintén a tanítás során, de nem a háló tanítására használjuk fel. Szokás ezért a validációs készletet is a tanító készlet részének tekinteni.

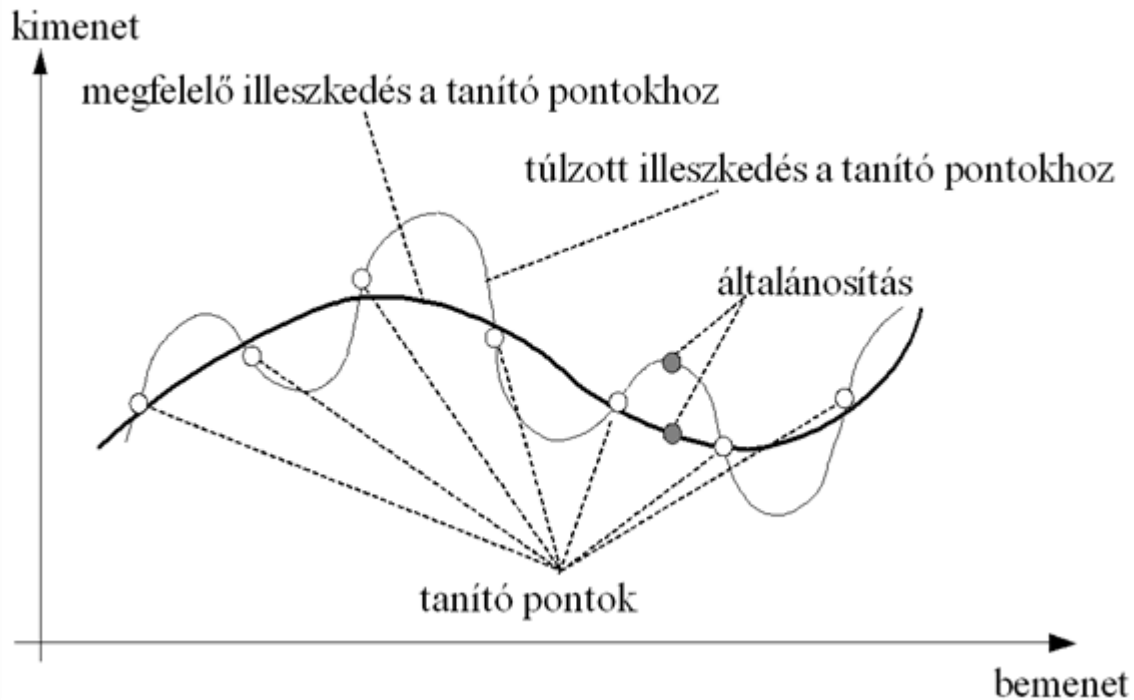
Túltanítás (*overtraining*) akkor lép fel, ha a tanító készlet mintáira már nagyon kis hibájú válaszokat kapunk, miközben a kiértékelő készletre egyre nagyobb hibával válaszol a hálózat. Ez azért következhet be, mert a hálózat válaszai túlzottan illeszkednek a véges számú tanító pont által megszabott leképezéshez – a tanító pontokban akár tetszőlegesen kis hiba is biztosítható –, miközben a közbenső pontokra adott válaszok jelentősen eltérhetnek a megfelelő kívánt válaszoktól. A túltanulás esetét mutatja a 4.5 ábra tanulási görbéje.

4.5. ábra - Tanulási görbe: a korai leállítás szerepe a túltanulás elkerülésében



Az ábra alapján látható, hogy a tanítást a megjelölt helyen – vagyis amikor a tanító mintákra adott válasz hibája ugyan még csökken, de a kiértékelő (validációs) mintákra már növekedni kezd – leállítva, a hálózat működése jobban közelít a kívánt leképezéshez. Ezt a leállítási stratégiát korai leállításnak nevezzük. A túltanulás következtében beálló tanítópontokhoz való túlzott illeszkedés (*overfitting*) hatását a leképezésre a 4.6. ábra illusztrálja.

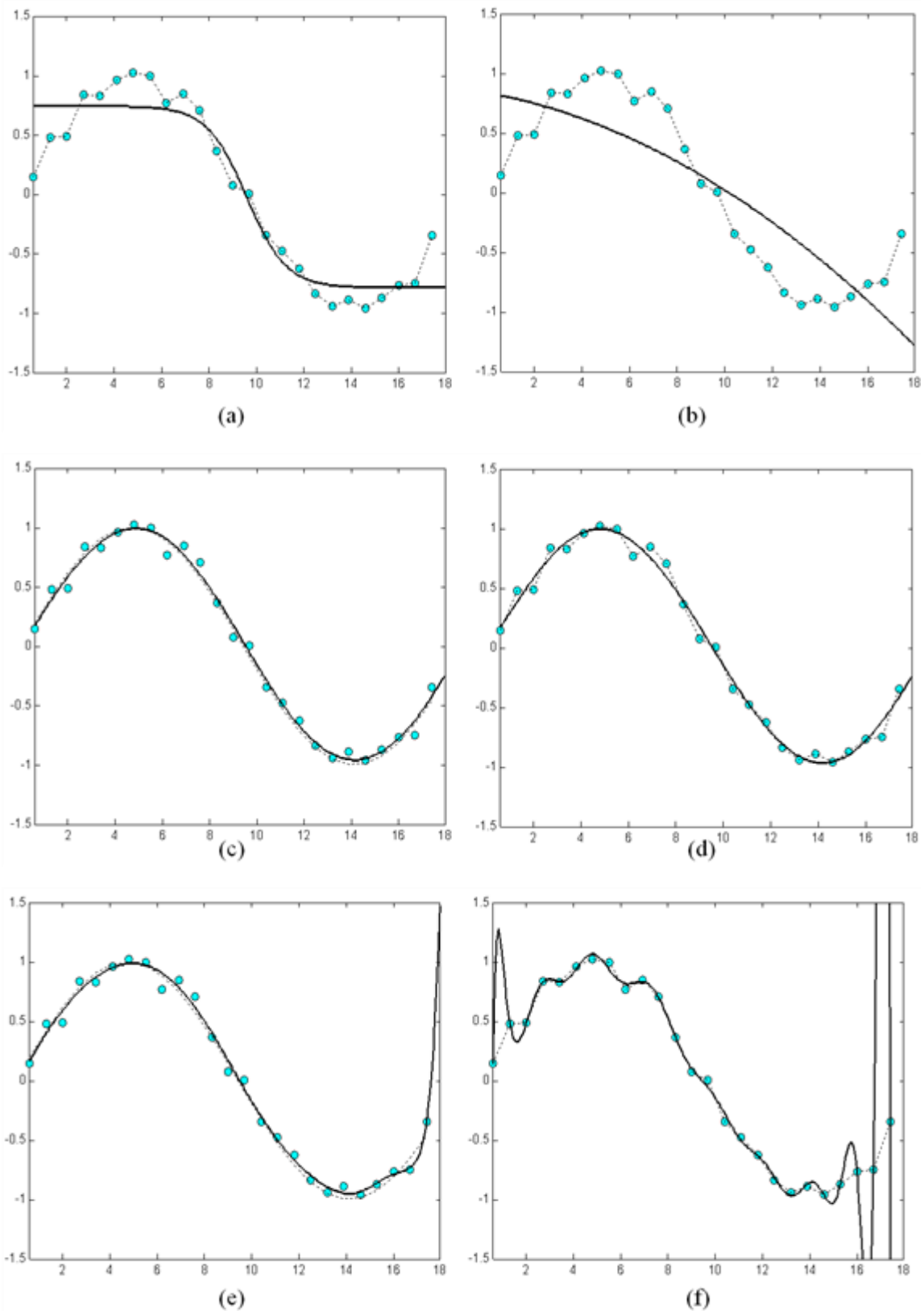
4.6. ábra - A hálózat leképezésének túlzott illeszkedése a tanító pontokhoz



A tanító pontokhoz való túlzott illeszkedés a hálózat általánosító-képességét rontja. Túltanulás elsősorban akkor következhet be, ha a hálózat mérete – a hálózat szabadságfoka – a tanító pontok számához viszonyítva túl nagy. A helyzet hasonló ahhoz az esethez, amikor polinomiális regressziónál kevés mintapontra akarunk nagyfokszámú polinomot illeszteni. A fokszám csökkentésével – esetleg a mintapontokbeli tökéletes illeszkedés rovására – a pontsorozat általános trendjének követése inkább lehetséges. A 4.7 ábra arra ad példát, hogy egy egyszerű függvény approximációjánál a szabad paraméterek számának növelése milyen mértékben növelheti a túltanulás esélyét. Az ábrán a szinusz függvény egy periódusát közelítjük polinomiális approximációt és MLP-t alkalmazva. A 4.7 (c) és (e) ábrák különböző komplexitású MLP-k eredményei: az első egy rejtett neuront, a második ötöt, a harmadik húszat használ. A (b), (d) és (f) ábrák hasonlóan egyre nagyobb komplexitású polinomiális közelítés eredményei, rendre másod-, ötöd- és huszadfokú polinommal történt approximációk. Látható, hogy miközben az MLP-nél a szabad paraméterek számának sokszorosára növelése alig rontott az approximáció minőségén, a polinomiális approximációnál ez a hatás igen jelentős. A polinomiális approximáció sokkal érzékenyebb a paraméterek számának megfelelő megválasztására. A túl nagy fokszám jelentős túlilleszkedést eredményezett. Az ábra ugyanakkor azt is mutatja, hogy túl kevés szabad paraméter mellett mindkét eszköz esetében nagy lesz a hiba, ami természetes is, hiszen ilyenkor az approximáló rendszer képessége nem elegendő a feladat megoldására.

Miközben általánosan igaz, hogy a túl sok szabad paraméter túltanulásra és túlilleszkedésre vezet, az MLP hálózatoknál viszonylag tág határok között megválaszthatjuk a háló méretét anélkül, hogy jelentős mértékben túltanulást tapasztalnánk. Ez, a mindenképpen előnyös tulajdonság egyfajta implicit regularizációként értelmezhető, hiszen anélkül, hogy bármilyen simasági mellékfeltételt alkalmaztunk volna, a háló a tanítópontokra illeszthető végtelen sok lehetőség közül a simább válaszokat előnyben részesíti. Ez a tulajdonság a polinomiális approximációra már nem mondható el.

4.7. ábra - A szabad paraméterek számának hatása a túltanulás hajlamra MLP: (a), (c), (e) polinomiális approximáció (b), (d), (f)



A tanító és a kiértékelő mintakészleteken kívül szokás még egy ún. minősítő (*test*) készletet is definiálni, amely szintén az adott problémából származó összetartozó be- kimeneti mintapárok halmaza, és amely mintapárokat a tanításnál egyáltalán nem használunk fel. A minősítő készlet a megtanított háló végső értékelésére, a háló általánosító-képességének becslésére használható. A rendelkezésre álló mintapontokat egy problémánál ezért három részre, tanító-, kiértékelő- és tesztelő készletre kell bontanunk. Kellően nagyszámú adat mellett ez nem okoz nehézséget. Ha azonban kevés adat áll rendelkezésünkre (ez számos gyakorlati feladatnál előfordulhat

vagy az adatok beszerzésének nagy költsége, vagy az adatoknak a probléma természetéből adódó kis száma miatt), a három diszjunkt készletre bontás azt eredményezheti, hogy az egyes részfeladatok elvégzéséhez túl kevés mintapontunk marad. Így, ha viszonylag sok pontot használunk fel a tanításra, és kevés marad a megtanított hálózat minősítésére, nem lehetünk biztosak a megtanított hálózat képességeit illetően.

Ezzel szemben, ha a minősítésre hagyunk több mintapontot és ezért kevés tanító pontunk lesz, csak azt konstatálhatjuk – bár ezt kellő biztonsággal tehetjük –, hogy a hálónk meglehetősen gyengén tanulta meg a feladatot. Nem közömbös tehát, hogy a mintakészlet mekkora hányadát használjuk tanításra és mekkorát minősítésre.

A megfelelő arányok meghatározására számos elméleti és gyakorlati eredmény született. A mintahalmaznak a korai leállításhoz szükséges két részre bontásával foglalkozik pl. [Guy96] és [Kea96]. Ezek az eredmények a VC-dimenzió alapján meghatározható korlátok felhasználásával adnak javaslatot a mintapontok tanító és kiértékelő halmazokra bontásához. Az ezzel kapcsolatos nehézségeket az előbbiekben említettük, itt azonban a helyzet kedvezőbb, hiszen nem a tényleges kockázatnak a gyakorlati alkalmazás szempontjából is használható becslése a cél, hanem csupán a mintakészlet megfelelő bontása. A különböző megközelítések alapján született eredményeket itt nincs módunk ismertetni, csupán egy megközelítés gyakorlatban jól alkalmazható eredményeit foglaljuk össze röviden.

Amari és munkatársai a tanítópontok két részre történő bontását vizsgálták azzal a céllal, hogy a túltanulás korai leállítással történő elkerüléséhez a megfelelő tanító/kiértékelő arányt meghatározzák [Ama97]. A megfelelő arány függ a háló komplexitásától, a háló szabad paramétereinek számától is. Ha a tanító pontok száma P kisebb vagy közel egyenlő a háló szabad paramétereinek számával (m), akkor nagy a túltanulás veszélye, hiszen ilyenkor a háló elvileg is képes az összes tanítópontot megtanulni, memorizálni. Ha ugyanekkor még kevés pontunk van – ahogy ezt az előbb már említettük – akkor a pontkészlet két készletre történő hasítása nem oldja meg a problémát: túl kevés mintapont marad a háló tanítására. Ekkor a túltanulás elkerülésére a 2. fejezetben említett regularizációs technikát alkalmazhatjuk. Megmutatható ugyanis, hogy a korai leállítás és a megfelelő mellékfeltétellel végzett regularizáció hasonló eredményre vezet [Bis95].

Ha a súlyok számához képest nagyon sok mintapontunk van ($P > 30m$), akkor megmutatható, hogy korai leállításra nincs szükség, helyesebb, ha a rendelkezésre álló pontokat mind tanításra használjuk. A két szélsőség között ($P < 30m$, de $P > m$) játszik fontos szerepet a korai leállítási technika. Ehhez viszont valóban szükséges a mintapontok tanító és értékelő készletre való bontása. Amari eredményei azt mutatják, hogy – feltéve, hogy m elegendően nagy – a legjobb stratégia, ha mindössze $P/\sqrt{2m}$ pontot használunk kiértékelésre, a többit pedig tanításra. Ha pl. a szabad paraméterek száma $m=100$, akkor a pontok 7%-át kell kiértékelésre használnunk és a maradék 93%-ot tanításra.

Kevés pont esetén az előbbi indokok miatt a végső minősítés céljait szolgáló teszt készlet sem különíthető el a mintakészletből. Ilyenkor a már említett (ld. 2. fejezet) kereszt kiértékelési (*cross validation*) eljárást alkalmazhatjuk (pl. [Sto78]).

A háló mérete, a felhasznált tanító pontok száma és a háló hibája a teszt készleten, vagyis a háló általánosító-képessége nem függetlenek egymástól. Az eddigiekben vizsgált kérdések tehát valójában csak egymással összefüggésben tárgyalhatók. Az alábbiakban e kérdéshez kapcsolódó két általános alaperedményt említünk meg, melyek a hálók osztályozó, illetve függvényapproximációs képességeikre vonatkoznak.

Az első általános eredmény, amely Eric Baum és David Haussler nevéhez fűződik [Bau89], osztályozós hálókra vonatkozik. Arra ad választ, hogy mekkora tanító készlet szükséges adott méretű hálózat megtanításához, ha a megtanított háló általánosító-képességéről valamilyen állítást is szeretnénk megfogalmazni.

Legyen egy M processzáló elemből álló, m súllyal rendelkező hálónk. Legyen $0 < \varepsilon \leq \frac{1}{8}$. Bizonyítható, hogy ha $P \geq O\left(\frac{m}{\varepsilon} \log \frac{M}{\varepsilon}\right)$ tetszőleges eloszlású mintát tanítunk a hálónak, és a tanító minták legalább $\left(1 - \frac{\varepsilon}{2}\right)$ -ed részét a megtanított háló helyesen osztályozza, akkor ugyanebből a feladatból származó, de a tanításnál fel nem használt tesztelő minták legalább $(1 - \varepsilon)$ -od részét is 1 valószínűséggel helyesen fogja osztályozni a háló.

Ha eltekintünk a logaritmikus tényezőtől, akkor a szükséges tanítópontok száma, a súlyok (szabad paraméterek) száma és az osztályozási hibára jellemző konstans ε között az alábbi közelítő kapcsolat áll fenn: $P \geq \frac{m}{\varepsilon}$, ami

megegyezik a Widrow által megfogalmazott ökölszabállyal. Ez azt jelenti, hogy pl. 10 %-os általánosító hiba eléréséhez a tanító pontok számát legalább a háló szabad paraméterei számának tízszeresére kell választani.

Függvényapproximációs feladatokra Barron jutott az előzőhöz hasonló jellegű eredményre [Bar93]. Az eredmény többrétegű perceptron approximációs tulajdonsága, az approximálandó függvény jellemzői és a háló mérete között ad kapcsolatot. Mint az előzőekben láttuk az MLP hálózatok egyetlen szigmoid nemlinearitását rejtett réteggel és lineáris kimeneti réteggel univerzális approximátorok, amennyiben a rejtett rétegben elegendő neuron található. Egy konkrét háló és approximálandó függvény esetében a háló approximációs képessége, általánosítása azzal az eltéréssel jellemezhető, mely az approximálandó $f(\mathbf{x})$ és a háló által megvalósított approximáló $\hat{f}(\mathbf{x})$ függvény között van. Négyzetes értelemben véve az eltérést (négyzetes veszteségfüggvényt alkalmazva) az approximáció kockázatának (ld. 3. fejezet) egy felső korlátja határozható meg. Barron eredménye szerint:

$$R \leq O\left(\frac{C_f^2}{M}\right) + O\left(\frac{MN}{P} \log P\right), \quad (4.21)$$

ahol C_f az approximálandó függvény regularitására, "változékonyságára" jellemző mennyiség, M a háló rejtett rétegbeli neuronjainak száma, N a bemenet dimenziója és P a tanító pontok száma. Az összefüggés mutatja a háló méretének az általánosításra, az approximáció jószágára vonatkozó ellentmondásos hatását. Az előző megfontolásoknál már utaltunk rá, hogy ha a rejtett rétegbeli neuronszám túl kicsi, a háló nem képes a megfelelő változékonyságú függvény kellően kis hibájú közelítésére. Ugyanakkor a tanítópontok számához viszonyított túl nagy háló túlilleszkedésre hajlamos, így az általánosítási hiba ezért lehet nagy. A korlát két tagja ezt a két ellentétes hatást fejezi ki.

3.4. A hibavisszaterjesztéses algoritmus változatai

A hibavisszaterjesztéses algoritmusnak számos változatát dolgozták ki. A változatok célja egyrészt a konvergencia gyorsítása (pl. momentum módszer), másrészt a hibafelületen jelentkező lokális minimumokba való beragadás elkerülése. A back-propagation eljárás a hibafelület gradiense alapján számol, tehát a tanítás alapjául lényegében bármely gradiens alapú eljárás szóba jöhet. Így alkalmazzák MLP hálózatok tanítására a *konjugált gradiens módszert*, a *Levenberg-Marquart eljárást*, de a rekurzív legkisebb négyzetes hibájú (*recursive least squares, RLS*) [Leu01] és a *Kalman szűrő* alapú eljárás [Sha92], [Iig92], [Sum99] is szokásos. A különböző módszerek eltérő lépésszám alatt érik el a megfelelő kis hibájú megoldást, azonban az egyes lépésekhez szükséges számítások mennyisége is jelentősen eltérhet, így a kisebb lépésszámú eljárások összességében nem feltétlenül eredményeznek kedvezőbb tanulást.

A gradiens alapú eljárásoktól alapvetően eltérnek a véletlen keresési módszerek (véletlen keresés, genetikus algoritmusok), melyeket szintén alkalmazhatunk a megfelelő súlyok megtalálására. Ezen módszerek előnye, hogy globális keresést végeznek, nem igénylik a gradiens meghatározását, így akkor is alkalmazhatók, ha a hibafüggvény nem differenciálható. Mindezek következtében lokális minimumokba való beragadásuk kevésbé valószínű. A "feltételgyengítések" és a lokális minimumokba való beragadási esély csökkentésének ára – különösen nagyméretű hálózatoknál – a jelentős számítási igény. A véletlen keresés (*random search*) bizonyítottan megtalálja a globális minimumot (ld. 2. fejezet), ha a bemeneti súlytér kompakt. E feltétel az MLP súlyaira nem biztosítható, ezért nem garantált, hogy a véletlen kereséssel eljutunk a globális minimumhoz.

Genetikus algoritmus esetén a hálózat súlyaiból kell kialakítanunk a kromoszómákat, a jóságfüggvény szerepét pedig pl. a hibafüggvény (-1)-szerese töltheti be. A kromoszómák kialakításának egy lehetséges módja, hogy a súlyértékek bináris reprezentációiból képezzünk bitfűzéseket, minden egyes bitfűzés egy adott struktúrájú és adott súlyokkal rendelkező hálónak felel meg. Természetesen más reprezentáció is lehetséges. A genetikus algoritmusok egyik erőssége, hogy nem igénylik a gradiens meghatározását hátrány is lehet, amennyiben a gradiens információ rendelkezésünkre áll. Ha tehát a gradiens ismert, célszerű felhasználni, hiszen ez járulékos információ a hibafelületről, így felhasználása a genetikus keresést gyorsíthatja. Az eltérő eljárások kombinálása általában is hasznos lehet. Pl. valamely véletlen keresési eljárással próbálunk először a minimumhely közelébe eljutni, ahonnan már egy gradiens alapú algoritmussal a minimum pontos helye megtalálható.

Az említett tanulási eljárásokkal részletesen a 2. fejezetben foglalkoztunk, így itt csak az ún. *momentum módszert* foglaljuk össze.

3.4.1. Momentum módszer

A momentum módszer olyan heurisztikus eljárás, ahol az egyes lépésekben a súlymódosítás meghatározása két részből áll. Először a gradiens módszernek megfelelően meghatározunk egy súlymódosító értéket, azonban nem ezzel végezzük el a korrekciót, hanem még figyelembe vesszük az előző módosítás hatását is. A súlymódosítás ennek megfelelően:

$$\Delta \mathbf{w}(k+1) = \mu(-\nabla f(k)) + \eta \Delta \mathbf{w}(k) \quad (4.22)$$

ahol η az ún. momentum együttható, amelynek értéke 0 és 1 között kell legyen (gyakori választás a 0,8 körüli érték). Az új tag tehetetlenséget visz az eljárásba, hiszen az adott lépésben a módosítás irányában szerepet kap az előző irány, s így közvetve a régebbi irányok is. A momentum módszer különösen kedvező, ha a hibafelületen keskeny völgy húzódik végig, és a momentum tag nélkül a megoldás felé ezen völgy környezetében túl sok iteráció megtételével haladnánk. A momentum módszer (4.22) összefüggéséből látható, hogy ez formailag megegyezik a konjugált gradiens módszerrel (ld. 2.5.1. alfejezet), ahol szintén a pillanatnyi gradiens és az előző irány eredője irányában történik az elmozdulás. Azonban míg a konjugált gradiens módszer olyan szélsőérték-kereső eljárás, amely kvadratikusan hibafelület esetén bizonyítottan N lépésben konvergál (N -dimenziós hibafelület esetén) és ahol az egyes tagok együtthatói meghatározhatók (ld. (2.78) és (2.86) összefüggések), addig a momentum módszer heurisztikus eljárás, ahol mind μ , mind η megválasztása tapasztalati úton történik.

Az előzőekben utaltunk már a μ megválasztásánál felmerülő nehézségekre. Mivel mind μ , mind η megválasztása a hibafelület függvénye, felmerül a két tényező együttes optimalizálásának igénye is. E megközelítés alapján javasol paraméter megválasztást [Yu97]. Az eljárás alap gondolata, hogy olyan – optimális – μ és η lépésméreteket határozzunk meg, melyek biztosítják, hogy a következő tanító lépés során kapott súlyok a lehető legnagyobb mértékben csökkentsék a hibát. Ehhez a hibafelület μ és η szerinti első és második deriváltjait kell meghatározni, azonban mivel a deriváltak számításánál az eredeti hibavisszaterjesztési eljárás részeredményei felhasználhatók, a járulékos számítások mennyisége nem jelentős. A módszer alkalmazásával – a számítási igényeknek a klasszikus back-propagation eljáráshoz viszonyított 2-3-szoros megnövelése árán – nagyságrendi sebességjavulás érhető el. Eredőben tehát adott hibahatár elérése kevesebb számítással, gyorsabban lehetséges.

A többrétegű, hibavisszaterjesztéssel alapuló tanítással dolgozó hálózatok számos alkalmazási lehetőséggel rendelkeznek. Gyakori alkalmazásukat kedvező tulajdonságaik, bizonyított approximációs képességük és viszonylag egyszerű felépítésük indokolják. Az alkalmazási területekkel és néhány érdekes alkalmazási példával a 7. fejezetben találkozhatunk.

Feladatok

4.1 Oldja meg a XOR problémát egy rejtett rétegű MLP hálózattal. Törekedjen a lehető legegyszerűbb hálózat elérésére. Rajzolja fel a feladat hibafelületét.

4.2 Mutassa meg, hogy a XOR probléma megoldható egy két neuronból (egy rejtett és egy kimeneti neuron) álló hálózattal. Határozza meg a háló súlyait. Rajzolja fel a háló által megvalósított elválasztó felületet.

4.3 Vizsgálja meg a kezdeti súlyértékek megválasztásának hatását a tanító eljárás konvergencia-sebességére. Válasszon nagyságrendileg eltérő kezdeti súlyértékeket. A vizsgálatot különböző feladatoknál (egydimenziós bemenet, pl. $y = \sin(x)$, kétdimenziós bemenet, pl. XOR, és egy sokdimenziós osztályozási probléma, pl. karakterfelismerés 10x14 pixeles bináris képek alapján) végezze el.

4.4 Konstruáljon egy, az $y = \sin(x)$ függvényt approximáló MLP-t. Törekedjen a minél egyszerűbb hálózatra. Milyen aktivációs függvényt célszerű választania a kimeneti réteg neuronjánál: logisztikus, tangens hiperbolicus vagy lineáris függvényt?

4.5 Konstruáljon MLP-t, amely megoldja a két egymásba tekeredő spirál szétválasztását. Vizsgálja meg, hogy hogyan befolyásolja a tanítás gyorsaságát és a háló méretét, ha egy, illetve ha két rejtett réteget alkalmaz.

4.6 Oldja meg a paritásproblémát (N -bites bemeneti minták páros és páratlan paritású csoportokra való szétválasztása) MLP hálózattal. Adjon magyarázatot, hogy miért "nehéz" ez az egyszerű feladat egy MLP-nek.

4.7 Vizsgálja meg a momentum módszer hatását a konvergencia-sebességre. Oldja meg a 4.5 feladatot, de most alkalmazzon momentum tagot is a tanításnál. Kísérletileg elemezze, hogy milyen irányban és mértékben befolyásolja a tanulási tényező és a momentum faktor a tanulás sebességét.

4.8 Konstruáljon egy, az $y = x^2 / (1 + 4x^2)$ függvényt approximáló MLP-t. Vizsgálja meg, hogy a háló méretének nagyfokú változtatása milyen hatással van az eredményre. Próbálkozzon néhány (5-6) és néhányszor tíz (30-50) rejtett neuronnal.

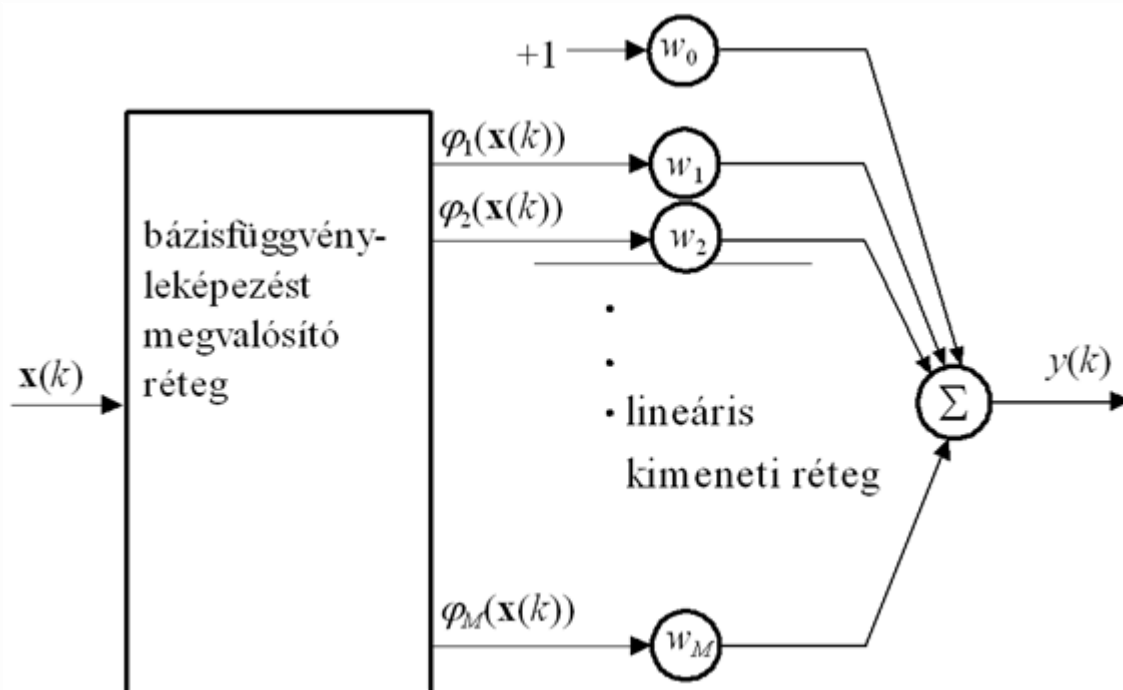
4.9 Alkalmazzon az előző feladatra egy súlyminimalizáló regularizációs tanító eljárást a néhányszor tíz neuronnal felépített esetre. Próbálja a háló méretét ennek segítségével redukálni.

5. fejezet - Bázisfüggvényes hálózatok

Az előzőekben bemutatott MLP hálózatok számos előnyös tulajdonságuk és széleskörű alkalmazásuk ellenére hátrányokkal is rendelkeznek. A hátrányok között első helyen kell megemlíteni, hogy az MLP hálózatok paramétereikben nemlineáris leképezést valósítanak meg. Ennek következménye, hogy az MLP hálózatok kritériumfelülete még a szokásosan alkalmazott négyzetes hibafüggvény mellett sem kvadratikussá, a hibafelület lokális minimumokkal rendelkezhet. A lokális minimumok lehetősége miatt a gradiens alapú hibavisszaterjesztési tanító eljárás (és bármely más gradiens alapú eljárás) nem garantálja, hogy a globális optimumot elérjük. További hátrányuk a tanító eljárás viszonylagos lassúsága, ami különösen nagyméretű hálózatoknál és elsősorban olyan alkalmazásoknál jelent komoly nehézséget, ahol nem csupán off-line tanulásra lehet szükség, hanem ahol a hálózat működését folyamatosan a változó környezethez kell igazítani. A lassú tanulás fő oka a több tanítható réteg és az ebből következő hibavisszaterjesztés szükségessége. A tanítás gyorsítható, ha a tanítható rétegek számát csökkenteni tudjuk.

Az 1. fejezetben összefoglaltuk azokat az elméleti eredményeket, amelyek megmutatták, hogy minimum két aktív rétegre szükség van ahhoz, hogy a hálózat megfelelő függvény-approximációs képességekkel rendelkezzen. A következőkben olyan hálózatokkal fogunk foglalkozni, amelyekben szintén két aktív réteg található, az egyik aktív réteg szerepe azonban jelentősen különbözik az eddig tárgyalt MLP hálózatoknál betöltött szereptől. A szokásos tanító eljárással csak a hálózat kimeneti rétegét tanítjuk, lehetővé téve így a gyorsabb tanulást. Ennek ellenére e hálózatok lényegében hasonló képességeket mutatnak, mint a többrétegű perceptron (MLP) hálózatok.

5.1. ábra - A bázisfüggvényes hálózatok általános felépítése



A családba tartozó hálózatok alapvető felépítése – bár a hálózatok eltérő eredetűek, és egymástól függetlenül jöttek létre – hasonló. A hálózatok két rétegből állnak (5.1 ábra): egy nemlineáris leképezést megvalósító rétegből (amely a család egyes tagjainál rögzített, más esetben ez is tanítható, azonban a tanítás itt egyes hálónál felügyelet nélküli, más hálónál felügyelt eljárással történik) és egy felügyelt tanítással tanítható lineáris rétegből. E hálózatok tehát megtestesítik a 2. fejezetben bemutatott esetet, amikor a hálózat által modellezett rendszer két egymást követő leképezést valósít meg: egy nemlineáris $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$ és egy lineáris $\boldsymbol{\varphi}(\mathbf{x}) \rightarrow y$ leképezést.

A család minden tagjánál a kimeneti réteg, amely lineáris súlyozott összegzést végez, ellenőrzött tanítású, ahol a súlyok meghatározása a szokásos LMS algoritmussal történhet. A nemlineáris leképezést az első réteg valósítja meg. Ennek felépítése – bár az egyes hálótípusoknál jelentős különbségek találhatók – abból a szempontból minden bázisfüggvényes hálónál hasonló, hogy a bemenet nemlineáris transzformációját mindegyik típus bázisfüggvények alkalmazásával végzi. Ezért az egész hálózatosztályt szokás bázisfüggvényes hálózatosztálynak nevezni. A bázisfüggvényes hálózatok által megvalósított leképezés általános formája egykimenetű hálózatot feltételezve:

$$y = \sum_{i=1}^M w_i \varphi_i(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) \quad (5.1)$$

A bázisfüggvényes hálózatok konkrét típusai a $\{\varphi_i(\mathbf{x})\}_{i=1,\dots,M}$ bázisfüggvényekben térnek el egymástól. A hálók konstrukciójánál olyan bázisfüggvények alkalmazása jöhet szóba, melyekkel valamilyen általános képesség megvalósítható. Az 1. fejezetben láttuk, hogy a bázisfüggvényes hálóarchitektúra is működhet univerzális approximátorként, vagyis, amikor egy megfelelő méretű és megfelelő tulajdonságú bázisfüggvényekkel dolgozó háló tetszőleges pontossággal közelíthet egy folytonos függvényt (1.6 tétel).

A bázisfüggvényes hálócsaládba tartozó leggyakrabban alkalmazott hálótípus az RBF háló, ahol radiálisan szimmetrikus bázisfüggvényeket alkalmazunk. Ebbe a függvénycsaládba tartoznak pl. a Gauss függvények vagy a B-spline függvények is. A radiális függvényeken kívül további függvények (pl. wavelet [Zha92], [Zha97]) alkalmazásával is születtek bázisfüggvényes hálók.

A bázisfüggvényes hálókön belül megkülönböztethetünk paraméteres bázisfüggvényekkel, illetve rögzített bázisfüggvényekkel dolgozó hálókat. Az előbbieknél a bázisfüggvények paramétereit szintén a tanuló eljárás részeként kell meghatározni. E hálókra az is jellemző, hogy az általuk megvalósított leképezés a bázisfüggvények paramétereire nézve már nem lineáris. A bázisfüggvények paramétereinek meghatározása lehetséges ellenőrzött tanítással, de egyes esetekben nemellenőrzött tanítás is alkalmazható, sőt van olyan eset is, amikor a bázisfüggvények paramétereinek meghatározása a mintapontoktól függetlenül történik.

A bázisfüggvényes hálózatok más megközelítésben az egyszerű perceptron kiterjesztéseiként is vizsgálhatók. Láttuk, hogy a perceptron a bemeneti minták lineáris kétosztályos szeparálására képes. A gyakorlati feladatok döntő részénél azonban a lineáris szeparálás nem elegendő: a mintákat csak nemlineáris felülettel tudjuk szétválasztani. A perceptron működése kiterjeszthető, ha a bemenetere kerülő mintákat előbb nemlineáris transzformációnak vetjük alá azzal a céllal, hogy a transzformáció következtében létrejövő mintapontok már lineárisan szeparálhatók legyenek, tehát a transzformált tartományban a feladat perceptronnal megoldható legyen. Ha e transzformáció során az eredetileg N -dimenziós mintateret egy $(M > N)$ -dimenziós tartományra képezzük le, akkor a mintapontok számának a dimenzióhoz viszonyított arányát csökkentjük, ami a perceptron kapacitásánál (3.1.3. alfejezet) bemutatottak szerint növeli a lineáris szeparálhatóság esélyét. Kellően nagymértékű dimenzió-növelés mindenképpen biztosítja a lineáris szeparálhatóságot [Cov65].

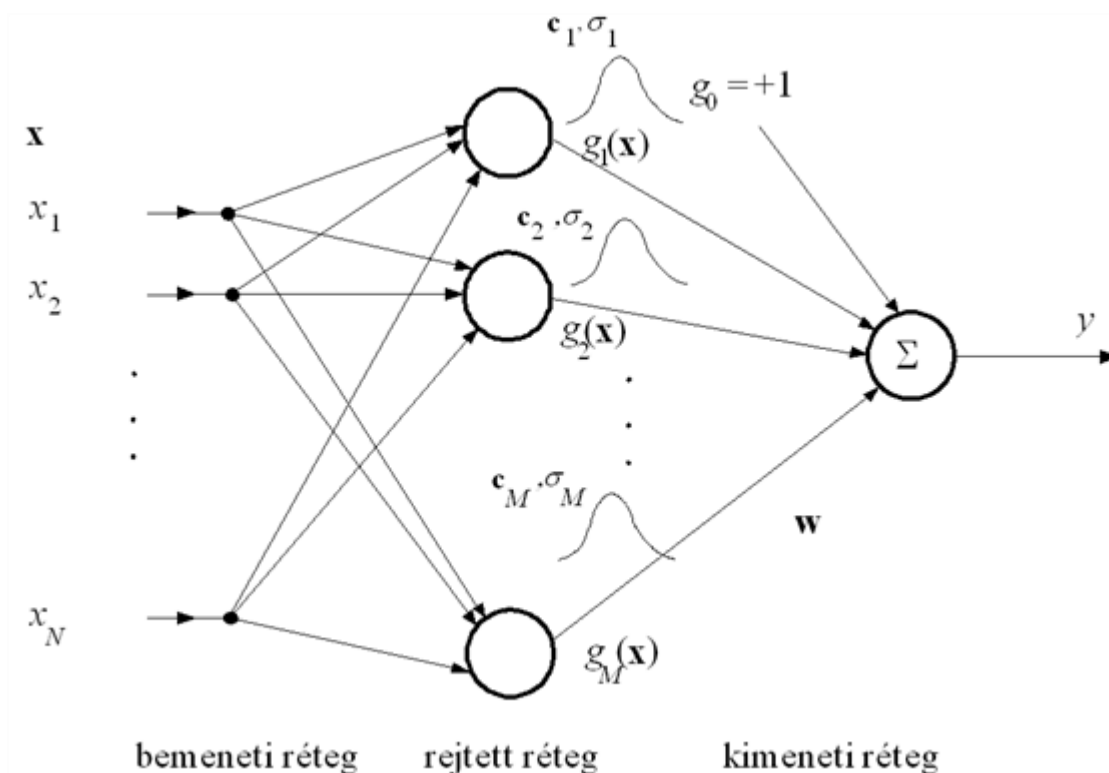
A bázisfüggvényes hálózatoknál – amennyiben csak a kimeneti réteg paramétereit tanítjuk – a tanulás folyamata gyorsabb, mint az MLP hálózatoknál. A nagyobb sebességnek két oka is van: egyrészt a módosítandó súlyok száma általában kisebb, és minthogy nincs szükség hibavisszaterjesztésre, egy tanítási lépés rendszerint kevesebb számítást igényel; másrészt a megfelelő képességű hálózat kialakításához szükséges tanító lépések száma is kisebb. Általánosságban azt mondhatjuk, hogy e hálózatok képességei hasonlóak a többretegű perceptron képességéhez, ezért számos alkalmazásban az MLP hálózatok reális alternatíváit jelentik.

A következőkben a bázisfüggvényes hálózatok két fontos típusával, az RBF és a CMAC hálózatokkal foglalkozunk. Bemutatjuk a hálózatok felépítését, a hálózatok létrehozásának lépéseit, a főbb tulajdonságait, legfontosabb előnyeit és hátrányait.

1. Az RBF (Radiális Bázisfüggvényes) hálózat

Az RBF hálózat olyan két aktív rétegű előrecsatolt hálózat, melyben a rejtett réteg radiális bázisfüggvényekkel dolgozó nemlineáris leképezést valósít meg. A háló felépítése az 5.2 ábrán látható.

5.2. ábra - Az (egyikimenetű) RBF hálózat felépítése



A hálóban az első rétegbeli processzáló elemek valósítják meg a bázisfüggvényeket. Az RBF hálónál ezek gömbszimmetrikus függvények. Az egyes processzáló elemekhez általában két paraméter tartozik, melyek a bázisfüggvények középpont- és szélességparaméterei. A rejtett réteg processzáló elemei minden bemeneti vektorhoz eltérő paraméterekkel rendelkező bázisfüggvény-értékeket rendelnek, a kimeneti processzáló elemek pedig ezen függvény-értékek lineáris kombinációját határozzák meg. Ha $\mathbf{x}$ a hálózat bemeneti vektora, $\mathbf{g}(\cdot)$ jelöli a radiális bázisfüggvényt, $\mathbf{c}_i$ pedig az i -edik processzáló elemhez tartozó függvény középpont-paraméterét, akkor az RBF hálózat által megvalósított leképezés a következő formába írható:

$$y = \sum_i w_i g_i(\mathbf{x}) = \sum_i w_i g(\|\mathbf{x} - \mathbf{c}_i\|) = \mathbf{w}^T \mathbf{g}(\mathbf{x}) \quad (5.1)$$

Ez azt jelenti, hogy a rejtett réteg minden processzáló eleme előbb meghatározza a bemeneti vektornak a processzáló elemhez rendelt középpont-paramétertől való távolságát, majd e távolságérték valamely nemlineáris függvényét számítja ki.

Az RBF hálózat szintén képes függvényapproximációra. Mint az 1.4. szakaszban láttuk (ld. 1.6. tétel), megfelelően megválasztott $g(\cdot)$ függvény és alkalmas $\mathbf{c}_i$ és w_i paraméterek mellett a hálózat tetszőleges folytonos függvény tetszőleges pontosságú approximációjára képes, ha a rejtett réteg mérete, a bázisfüggvények száma megfelelő. Az approximáció valójában úgy történik, hogy a tanítópontokra vagy a tanítópontokból valamilyen módon kialakított ún. klaszterközéppontokra bázisfüggvényeket illesztünk, és ezen bázisfüggvények lineáris kombinációját határozzuk meg. A bázisfüggvények legtöbbször a

$$g_i(\mathbf{x}) = \exp\left[-\|\mathbf{x} - \mathbf{c}_i\|^2 / 2\sigma_i^2\right] \quad (5.2)$$

Gauss függvények, de más bázisfüggvény is választható. Szokásos további függvények pl:

$$g(r) = r \quad (\text{szakaszonkénti lineáris approximáció})$$

$$g(r) = r^3 \quad (\text{köbös approximáció})$$

$$g(r) = (c^2 + r^2)^\beta \quad 0 < \beta < 1 \quad (\beta = 1/2 - \text{nél multikvadratikus függvény})$$

$$g(r) = (c^2 + r^2)^{-\alpha} \quad \alpha > 0 \quad (\alpha = 2 - \text{nél inverz multikvadratikus függvény}),$$

ahol $c > 0$ és $r_i = \|x - c_i\|$, melyben a c_i értékek az alkalmasan megválasztott középpontok.

Az RBF hálók konstrukciója a háló méretének (a rejtett rétegbeli elemek számának), a bázisfüggvények paramétereinek (amennyiben vannak a bázisfüggvényeknek szabad paraméterei) és a kimeneti réteg súlyainak meghatározását jelenti.

A hálózat komplexitása a rejtett rétegbeli processzáló elemek számától, vagyis az eltérő középpontú bázisfüggvények számától függ. A c_i középpontok számának meghatározása azonban nem független a középpontok helyzetének megválasztásától. A két kérdést tehát együttesen kell vizsgálni.

Gauss bázisfüggvények esetén a középpontokon kívül az ún. szélességparaméterekre (σ_i) is szükség van. A σ_i értékekre azonban az approximáció kevésbé érzékeny, ezért ezek széles határok között megválaszthatók. Legtöbb esetben a szélességparaméter minden rejtett processzáló elemre azonos, vagyis $\sigma_i = \sigma \quad \forall i = 1, \dots, P$.

A rejtett rétegbeli processzáló elemek paramétereit és a kimeneti réteg paramétereit eltérő módon választjuk meg, ill. határozzuk meg. A kimeneti réteg paraméterei, a w_i súlyok vagy analitikus úton meghatározhatók, vagy ellenőrzött tanítás során alakíthatók ki. A bázisfüggvény paramétereinek (a c_i és esetenként a σ_i értékeknek) a megválasztására különböző eljárásokat alkalmazhatunk.

A bázisfüggvények számának és a középpont paramétereknek a meghatározása a tanítópontok számától is függ. Kevés tanítópont esetén triviális megoldás lehet, ha minden egyes tanítópont egyben függvényközeppt is lesz, vagyis a rejtett rétegben a processzáló elemek száma megegyezik a tanítópontok számával. A kimeneti lineáris réteg tanítása ilyenkor – ahogy ezt a lineáris elemi neuron, az adaline tanításánál láttuk – egy olyan lineáris egyenletrendszer megoldását jelenti, ahol az egyenletek száma és az ismeretlenek száma megegyezik. Itt tehát analitikus megoldás is nyerhető. Ha az x_i tanítópontokra adott kívánt válaszokat most is d_i , $i=1, 2, \dots, P$ jelöli, akkor a megoldandó egyenlet:

$$Gw = d, \quad (5.3)$$

ahol a G az ún. interpolációs mátrix (a bázisfüggvények által megvalósított leképezések $P \times P$ -s mátrixa) és d a d_i kívánt válaszokból álló oszlopvektor. A kvadratikusan mátrix elemei: $g_{ij} = g_i(x_j)$. Amennyiben G nem szinguláris a megoldásvektor létezik:

$$w^* = G^{-1}d. \quad (5.4)$$

A gyakorlatban (5.4) megoldása nehézségekbe ütközik, ha G közel szinguláris. Ebben az esetben a regularizáció elmélet (ld. 2.1. szakasz) segítségével juthatunk a megoldáshoz. A regularizáció során a G mátrix helyett a $G + \lambda I$ perturbált mátrixot alkalmazzuk, ahol λ a regularizációs együttható, melynek már igen kis értéke mellett is biztosított, hogy a perturbált mátrix invertálható lesz. A regularizáció valójában azt jelenti, hogy a $Gw = d$ összefüggés helyett egy $(G + \lambda I)w = d$ összefüggés megoldását keressük, vagyis olyan súlyokat keresünk, melyekkel a háló leképezése a tanítópontokban sem lesz pontos, tehát ott sem fog a kívánt válaszokkal megegyező kimeneteket előállítani. Az eltérés függ a regularizációs együttható értékétől. A regularizáció azáltal, hogy a tanítópontokban is megenged eltérést a kívánt válaszoktól, valójában egy simább leképezést megvalósító hálót eredményez. A regularizáció és a simaság kapcsolatáról a későbbiekben, a kernel gépeknél (6. fejezet) még lesz szó.

Az analitikus megoldáshoz az összes tanítópont ismeretére szükség van, és a megfelelő mátrix invertálási műveletet el kell végezni. A mátrix invertálás elkerülhető, ha pontonkénti iteratív tanítást (pl. LMS eljárást) alkalmazunk.

Nagyszámú tanítópont mellett minden tanítópont középpontként való felhasználása nem alkalmazható, hiszen ekkor sok rejtett processzáló elemet tartalmazó, igen nagy hálót is nyerhetünk. Ez egyrészt a szükséges számítások mennyiségét nagyon megnövelné, másrészt a sok processzáló elem szükségtelen is.

Amennyiben a tanítópontoknak csak egy részéhez rendelünk bázisfüggvényeket, akkor részint a megfelelő súlyértékeket az előbbieken bemutatott megoldástól némileg eltérő módon nyerjük, részint meg kell adnunk a középpontként felhasználandó tanítópontok kiválasztási eljárását.

Ha a mintapontok száma, P nagyobb, mint a bázisfüggvények száma, vagyis a meghatározandó súlyok száma, M , a $\mathbf{G}$ mátrix nem lesz kvadrátikus. $P > M$ esetben túlhatározott egyenletrendszerrel van dolgunk és analitikus megoldást LS értelemben a pszeudo-inverz meghatározásával a (3.23) összefüggésnek megfelelően nyerhetünk:

$$\mathbf{w}_{LS}^* = \mathbf{G}^T \mathbf{d} = (\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T \mathbf{d} \quad (5.5)$$

Előfordulhat, hogy a $\mathbf{G}^T \mathbf{G}$ mátrix nem invertálható, lehet szinguláris vagy közel szinguláris akár numerikus problémákból adódóan. Ekkor itt is a regularizáció alkalmazható:

$$\mathbf{w}_{LS}^* = \mathbf{G}^T \mathbf{d} = (\mathbf{G}^T \mathbf{G} + \lambda \mathbf{I})^{-1} \mathbf{G}^T \mathbf{d} \quad (5.6)$$

A megoldást iteratív úton most is meghatározhatjuk az LMS algoritmus alkalmazásával. Túlhatározott egyenletrendszer esetén a pontonkénti tanulásnál azonban nem kapunk egyértelmű megoldást. Meghatározható egy olyan tartomány, melyen belül kell legyen a megoldásvektor. A tartományon belüli konkrét megoldás, azonban függ a tanítópontok sorrendjétől és a tanulási tényezőtől is.

A középpontok kiválasztásának, illetve meghatározásának néhány lehetőségével a következő részben foglalkozunk.

1.1. Középpont kiválasztó eljárások

A középpont kiválasztó eljárások célja minden esetben az, hogy az összes tanítópontnál kevesebb bázisfüggvénnyel oldjuk meg a feladatot és ezen bázisfüggvények középpont paramétereit valamilyen módon a tanítópont-készlethez igazodóan meghatározzuk. A középpont kiválasztó eljárások nagy mintapontszám mellett biztosítják, hogy megvalósítható méretű háléhoz jutunk.

Nagy mintapontszám több esetben is előállhat. Egyrészt rendelkezésünkre állhat nagyon nagyszámú – több tízezer, vagy akár több százezer – tanítópont is, melyek mindegyikét nem lehet és nem is szükséges felhasználnunk. Elegendő egy megfelelő részhalmaz kiválasztása és ennek felhasználásával ún. ritka (*sparse*) megoldás létrehozása. Ebben az esetben a tanító készletünk tulajdonképpen redundáns, így valójában információvesztés nélkül lehet a tanítópontok közül egy részhalmazt kiválasztani: a $\mathcal{C}_k$ középpontokat egyszerűen a tanítópontokból véletlenszerűen kiválasztott K pont képezi.

Az is lehetséges, hogy a tanítópontok egyes csoportjaihoz rendelünk középpontokat. Ekkor valójában a pontok klaszterezését kell elvégeznünk. A tanítópontok csoportosítására, klaszterek képzésére bármilyen klaszter kialakító algoritmus használható.

Nagyszámú mintapont akkor is előállhat, ha a megoldandó probléma sokdimenziós. Ekkor a sokdimenziós bemeneti tér miatt eleve sok tanítópontra van szükség, hogy a feladatot kellően reprezentáló tanítópont-készlet alapján történhessen a háló konstrukciója. (Ezt a sokdimenziós feladatoknál jelentkező nehézséget szokás a dimenzió átkának (*curse of dimensionality*) is nevezni. A kérdéskört általánosabban a 13. fejezetben vizsgáljuk.) Ilyen esetekben a nagy mintapontszám általában nem redundancia következménye, és a középpontok számának csökkentése mindenképpen információvesztéssel jár, a pontokból egy részhalmaz kiválasztása komolyabb megfontolást igényel. A véletlenszerű kiválasztás helyett, a megfelelő pontok kiválasztásánál figyelembe kell vennünk az eredményül kapott háló képességeit is.

A kétféle esetre számos megoldás született. A következőkben előbb egy olyan iteratív eljárást mutatunk be, amely fokozatosan növeli a középpontok számát újabb és újabb tanítópontok középpontként való beiktatásával, amíg megfelelő képességű háléhoz nem jutunk; majd az egyik legegyszerűbb és az RBF hálózatok középpont-paramétereinek meghatározásánál gyakran alkalmazott klaszterező eljárást, a K -átlagképző vagy K -közép (K -means) eljárást ismertetjük röviden.

1.1.1. Az OLS eljárás

Az ortogonális legkisebb négyzetes hibájú (OLS) eljárás [Che91] iteratív módon választ középpontokat úgy, hogy közben figyelembe veszi a háló képességét is. Ez azt jelenti, hogy a háló méretét lépésről lépésre növeljük, újabb és újabb bázisfüggvények bekapcsolásával egészen addig, amíg megfelelő teljesítőképességet el nem érünk.

A módszer bemutatásához induljunk ki a háló által megvalósított leképezésből ((5.1) egyenlet). Ha csak egyetlen bázisfüggvényt alkalmazunk, a szummás kifejezésnek csak egy tagja lesz: a kimenetek egy adott

középpontra illesztett bázisfüggvény súlyozott értékeiként adódnak. Hogy az így kapott egy rejtett elemű „háló” milyen képességekkel rendelkezik, azt a rendelkezésre álló ismert válaszó pontokra adott válaszok alapján állapíthatjuk meg. Az egyetlen súly értéke egy olyan túlhatározott lineáris egyenletrendszer megoldásaként nyerhető (ld. (5.5) összefüggés), melynél a G mátrix egyetlen oszlopot és a rendelkezésre álló P tanítópontnak megfelelő P sort tartalmaz. A kapott LS megoldásnak az összes tanítópontra vonatkozó teljesítőképessége alapján dönthető el, hogy az egy rejtett elemű „háló” megfelelően oldja-e meg a feladatot vagy sem.

A megoldás minősége függ az alkalmazott egyetlen bázisfüggvény elhelyezésétől, vagyis a középpont paramétertől. Hogy az egy bázisfüggvénnyel dolgozó hálók közül a legjobbat kapjuk, bázisfüggvény-középpontként az összes tanítópontot végig kell vizsgálni. Amennyiben az egy bázisfüggvényes hálók legjobbika sem elegendően kis hibájú, több bázisfüggvénnyel kell a feladatot megoldani. Így elvben meghatározhatjuk a legjobb két, három, stb. rejtett neuronos RBF hálót. Az eljárás gyakorlati alkalmazása a hatalmas számításigény miatt nem célszerű vagy nem is lehetséges.

Kisebbszámításigényű az a változat, amikor a hálót oly módon bővítjük fokozatosan, hogy a már kiválasztott középpontokhoz választunk újabb és újabb középpontokat. Így egy adott komplexitású háló középpontjait már nem módosítjuk. A feladat „mindössze” annyi, hogy a még középpontként fel nem használt pontok közül kell minden lépésben az LS értelemben legjobbat kiválasztani, és a már kiválasztottakhoz új bázisfüggvény-középpontként hozzávenni. A háló méretét most is fokozatosan bővítjük egészen addig, amíg a megfelelő teljesítőképességet el nem érjük. A második változat csak szuboptimumot eredményez, és még mindig elfogadhatatlanul nagy számításigényű. Az OLS eljárás a háló fokozatos bővítését egy ortogonalizáló lépés közbeiktatásával az előbbinél jóval hatékonyabban oldja meg.

Az OLS eljárás bemutatásához induljunk ki a háló leképezésének mátrixos alakjából:

$$\mathbf{y} = \mathbf{G}\mathbf{w}, \quad (5.7)$$

ahol a $\mathbf{G}$ mátrix i -edik sora a háló rejtett rétegének válasza az $\mathbf{x}_i$ bemeneti vektorra $\mathbf{g}(\mathbf{x}_i) = [g_1(\mathbf{x}_i), g_2(\mathbf{x}_i), \dots, g_M(\mathbf{x}_i)]^T$, $i=1, 2, \dots, P$; $\mathbf{y} = [y_1, y_2, \dots, y_P]^T$ a tanítópontokra adott válaszokból képezett vektor, és $\mathbf{w}$ a kimeneti réteg súlyvektora. $\mathbf{G}$ tehát felírható, mint egy P sorból álló mátrix:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}(\mathbf{x}_1)^T \\ \mathbf{g}(\mathbf{x}_2)^T \\ \vdots \\ \mathbf{g}(\mathbf{x}_P)^T \end{bmatrix} \quad (5.8)$$

Ugyanakkor $\mathbf{G}$ az alábbi módon is értelmezhető

$$\mathbf{G} = [\mathbf{g}_1, \dots, \mathbf{g}_M], \quad \text{ahol} \quad \mathbf{g}_j = [g_j(\mathbf{x}_1), \dots, g_j(\mathbf{x}_P)]^T \quad j = 1, 2, \dots, M \quad (5.9)$$

Itt $\mathbf{g}_j$ az j -edik bázisfüggvénynek a P tanítópontra adott válaszaiból képezett vektor. E szerint a háló válaszaiból képezett vektor

$$\mathbf{y} = \sum_{j=1}^M w_j \mathbf{g}_j \quad (5.10)$$

formában is értelmezhető. Minthogy a kimenetek vektora a $\mathbf{g}_j$ vektorok súlyozott összegeként állítható elő, ha a $\mathbf{g}_j$ vektorok lineárisan függetlenek, akkor egyben bázist is alkotnak a kimeneti vektorok terében.

Adott súlyvektor mellett azonban a háló válasza nem feltétlenül állítja elő hibátlanul a kívánt válaszokat, tehát:

$$\mathbf{d} = \sum_{j=1}^M w_j \mathbf{g}_j + \mathbf{e} = \mathbf{G}\mathbf{w} + \mathbf{e}, \quad (5.11)$$

ahol $\mathbf{e}$ az egyes tanítópontokban kapott válaszok hibáiból képezett vektor. $\mathbf{G}\mathbf{w}$ $\mathbf{d}$ -nek a $\mathbf{g}_j$ bázisvektorok által kifeszített térre való vetületét adja. Minél kevesebb bázisfüggvényt használunk, annál nagyobb hibával tudjuk $\mathbf{d}$ -t közelíteni a bázisvektorok által kifeszített térben. A hiba azonban nemcsak attól függ, hogy hány bázisvektort használunk, hanem attól is, hogy melyek ezek a bázisvektorok. A bázisvektorok számát a háló

bázisfüggvényeinek száma adja meg, azt pedig, hogy mely bázisvektorokat használjuk, a bázisfüggvények középpont paraméterei határozzák meg. Minél inkább korreláltak a bázisvektorok, annál kisebb hozzájárulást jelentenek a $\mathbf{d}$ kimeneti vektor előállításához. Legkedvezőbb az lenne, ha a bázisvektorok ortogonálisak lennének, ekkor ugyanis mindegyik bázisvektor a többitől független hozzájárulást jelentene, ahol az egyes bázisvektorok hozzájárulásának mértéke egyenként számítható. Az OLS eljárás a $\mathbf{g}_j$ vektorok transzformációja útján egy ortogonális bázisvektor-rendszert hoz létre.

Ennek érdekében dekomponáljuk a $\mathbf{G}$ mátrixot az alábbi módon:

$$\mathbf{G} = \mathbf{B}\mathbf{A}, \quad (5.12)$$

Itt $\mathbf{A}$ egy olyan $M \times M$ -es felső háromszögmátrix, melynek főátlójában csupa egyesek szerepelnek

$$\mathbf{A} = \begin{bmatrix} 1 & a_{12} & a_{13} & \dots & a_{1M} \\ 0 & 1 & a_{23} & \dots & a_{2M} \\ 0 & 0 & \ddots & & \vdots \\ \vdots & & & \ddots & \\ 0 & & & 0 & 1 & a_{M-1,M} \\ 0 & \dots & & 0 & 0 & 1 \end{bmatrix}, \quad (5.13)$$

$\mathbf{B}$ pedig egy $P \times M$ -es ortogonális mátrix. Az ortogonalitás következménye, hogy $\mathbf{B}^T \mathbf{B} = \mathbf{H}$ egy olyan diagonálmátrix, melynek főátlójában a $h_j = \mathbf{b}_j^T \mathbf{b}_j$ értékek találhatók.

Az ortogonális vektorok által kifeszített tér azonos a $\mathbf{g}_j$ bázisvektorok által kifeszített térrel, vagyis a kimenetek vektora, $\mathbf{d}$ előállítható az ortogonális vektorok súlyozott összegeként is:

$$\mathbf{d} = \sum_{j=1}^M v_j \mathbf{b}_j + \mathbf{e} = \mathbf{B}\mathbf{v} + \mathbf{e}, \quad (5.14)$$

ahol a $\mathbf{b}_j$ ortogonális bázisvektorok súlyait a $\mathbf{v}$ vektor komponensei adják. Ennek a súlyvektornak a meghatározása szintén történhet a négyzetes hiba minimalizálásával.

$$\mathbf{v}_{LS}^* = \mathbf{B}^T \mathbf{d} = (\mathbf{B}^T \mathbf{B})^{-1} \mathbf{B}^T \mathbf{d} = \mathbf{H}^{-1} \mathbf{B}^T \mathbf{d} \quad (5.15)$$

Itt azonban az ortogonalitás miatt $\mathbf{H}^{-1}$ sokkal egyszerűbben számítható, így a súlyokat egyenként is meg tudjuk határozni.

$$v_j^* = \frac{\mathbf{b}_j^T \mathbf{d}}{\mathbf{b}_j^T \mathbf{b}_j} \quad j = 1, \dots, M \quad (5.16)$$

Ugyanakkor az is igaz, hogy

$$\mathbf{A}\mathbf{w}_{LS}^* = \mathbf{v}_{LS}^*, \quad (5.17)$$

vagyis, ha ismerjük $\mathbf{A}$ -t és $\mathbf{v}$ -t, a keresett $\mathbf{w}$ súlyvektor is meghatározható.

Az (5.17) összefüggés származtatásához például a klasszikus a Gram-Schmidt ortogonalizáló eljárással juthatunk el. Alkalmazzuk az eljárást $\mathbf{G}$ ortogonalizálására. Az ortogonalizáló eljárás k -adik lépésében olyan $\mathbf{b}_k$ vektort keresünk, amely a megelőző lépésekben meghatározott $k-1$ vektor mindegyikére ortogonális. $\mathbf{B}$ első vektoraként a $\mathbf{g}_j$ vektorok bármelyikét választhatjuk. Legyen ezért

$$\mathbf{b}_1 = \mathbf{g}_1 \quad (5.18)$$

A k -adik vektor ($k=2, \dots, M$) ezek után a következőképpen nyerhető

$$\mathbf{b}_k = \mathbf{g}_k - \sum_{j=1}^{k-1} a_{jk} \mathbf{b}_j, \quad (5.19)$$

ahol

$$a_{jk} = \frac{\mathbf{b}_j^T \mathbf{g}_k}{\mathbf{b}_j^T \mathbf{b}_j} \quad 1 \leq j < k \quad (5.20)$$

Az ortogonalizáló eljárással tehát mind a $\mathbf{b}_j$ ortogonális bázisvektorok, mind az $\mathbf{A}$ mátrix a_{jk} együtthatói meghatározhatók.

Az OLS eljárás célja azonban most nemcsak az, hogy a bázisvektorok mátrixát ortogonalizáljuk, hanem az is, hogy az ortogonális bázisvektorok egy megfelelő részhalmazát kiválasszuk. Addig, amíg a kiinduláskor a $\mathbf{G}$ mátrix maximum annyi oszlopból állt, ahány tanítópontunk van (egy bázisfüggvényes hálónál, ha minden tanítópont egyben bázisfüggvény középpont is, a $\mathbf{G}$ mátrix egy $P \times P$ -s mátrix, ld. (5.3)), most az ortogonális bázisvektorok egy kisebb számú részhalmazát szeretnénk felhasználni. Az eddigi eljárásból az is látszik, hogy az ortogonális bázisvektor-rendszer a kiinduló lépés függvénye, tehát függ attól, hogy $\mathbf{b}_1$ -ként melyik $\mathbf{g}_j$ bázisvektort választjuk. Sőt, ha nem az összes vektort akarjuk felhasználni, akkor az ortogonális bázisvektorok előállítási sorrendje sem közömbös.

A legfontosabb bázisvektorok kiválasztásához vizsgáljuk meg előbb az egyes bázisvektorok szerepét. Mivel $\mathbf{b}_i$ és $\mathbf{b}_j$ ortogonálisak, ha $i \neq j$, a kívánt kimenetek négyzetes értéke felírható az alábbi formában:

$$\mathbf{d}^T \mathbf{d} = \sum_{j=1}^M v_j^2 \mathbf{b}_j^T \mathbf{b}_j + \mathbf{e}^T \mathbf{e} \quad (5.21)$$

Tételezzük föl, hogy a $\mathbf{d}$ kívánt válaszok vektora nulla várható értékű (ha ez nem teljesül, akkor vonjuk ki belőle a várható értékét). Ekkor $\mathbf{d}$ varianciájának a becslése:

$$\frac{1}{P} \mathbf{d}^T \mathbf{d} = \frac{1}{P} \sum_{j=1}^M v_j^2 \mathbf{b}_j^T \mathbf{b}_j + \frac{1}{P} \mathbf{e}^T \mathbf{e} \quad (5.22)$$

Az (5.22) összefüggés első tagja az a variancia-komponens, ami a bázisvektorokkal kifejezhető, míg a második tagja az ezekkel nem kifejezhető rész. Ebből az is látszik, hogy a kimeneti varianciához a j -edik bázisfüggvény hozzájárulása $v_j^2 \mathbf{b}_j^T \mathbf{b}_j / P$. Minél nagyobb arányt képvisel ez az érték a kimeneti varianciában, annál fontosabb a $\mathbf{b}_j$ bázisvektor felhasználása. A j -edik bázisvektornak a kimeneti varianciához való relatív hozzájárulását az

$$[\text{err}_j] = \frac{v_j^2 \mathbf{b}_j^T \mathbf{b}_j}{\mathbf{d}^T \mathbf{d}} \quad (5.23)$$

menyiség fejezi ki. Ez az arány egy egyszerű és hatékony mértéke lehet az egyes bázisvektorok fontosságának. Ennek felhasználásával relative kis számításigénnyel minden lépésben azt az ortogonális bázisvektort és a megfelelő v_j súlyozó együtthatót meg tudjuk meghatározni, melynek a kimeneti varianciához való hozzájárulása a legnagyobb.

Ehhez a Gram-Schmidt eljárást a következő módon alkalmazzuk:

Az első lépésben minden $1 \leq j \leq M$ értékre számítsuk ki a következőket

$$\begin{aligned} \mathbf{b}_1^{(0)} &= \mathbf{g}_1 \\ v_1^{(0)} &= \frac{(\mathbf{b}_1^{(0)})^T \mathbf{d}}{(\mathbf{b}_1^{(0)})^T \mathbf{b}_1^{(0)}} \\ [\text{err}_1]^{(0)} &= \frac{(v_1^{(0)})^2 (\mathbf{b}_1^{(0)})^T \mathbf{b}_1^{(0)}}{\mathbf{d}^T \mathbf{d}} \end{aligned} \quad (5.24)$$

Keressük azt az j értéket mely mellett

$$[\text{err}_j]^{(0)} = \max \{ [\text{err}_j]^{(0)}, \quad 1 \leq j \leq M \} \quad (5.25)$$

és válasszuk ki első bázisvektornak a

$$\mathbf{b}_1 = \mathbf{b}_1^{(0)} = \mathbf{g}_1 \quad (5.26)$$

vektort.

A k -adik lépésben, ha $k \geq 2$, minden $1 \leq j \leq M$, $j \neq j_1, \dots, j \neq j_{k-1}$ értékre számítsuk ki a következőket

$$\begin{aligned} \alpha_i^{(j)} &= \frac{\mathbf{b}_i^T \mathbf{g}_j}{\mathbf{b}_i^T \mathbf{b}_i}, \quad 1 \leq j \leq M \\ \mathbf{b}_i^{(j)} &= \mathbf{g}_j - \sum_{i=1}^{j-1} \alpha_i^{(j)} \mathbf{b}_i \\ v_i^{(j)} &= \frac{(\mathbf{b}_i^{(j)})^T \mathbf{d}}{(\mathbf{b}_i^{(j)})^T \mathbf{b}_i^{(j)}} \\ [\text{err}]_i^{(j)} &= \frac{(v_i^{(j)})^2 (\mathbf{b}_i^{(j)})^T \mathbf{b}_i^{(j)}}{\mathbf{d}^T \mathbf{d}} \end{aligned} \quad (5.27)$$

Keressük azt a j értéket, mely mellett

$$[\text{err}]_i^{(j)} = \max \{ [\text{err}]_i^{(j)}, \quad 1 \leq j \leq M, \quad j \neq j_1, \dots, j \neq j_{k-1} \}, \quad (5.28)$$

és a k -adik bázisvektornak válasszuk a

$$\mathbf{b}_k = \mathbf{b}_i^{(j)} = \mathbf{g}_k - \sum_{i=1}^{k-1} \alpha_i \mathbf{b}_i \quad (5.29)$$

vektort, ahol $\alpha_i = \alpha_i^{(j)}$, $1 \leq i < k$.

Az eljárást az M_s -edik lépésben állítsuk le, ha

$$1 - \sum_{i=1}^{M_s} [\text{err}]_i < \rho, \quad (5.30)$$

ahol $0 < \rho < 1$ egy előre megválasztott küszöbérték.

Az eljárással M_s bázisvektort választottunk ki. Bár az OLS eljárás szintén számításigényes, az eredeti középpont-válogató eljárásnál jóval hatékonyabb, hiszen minden egyes új bázisvektor beiktatása az addigi bázisvektorok változatlanul hagyása mellett történik. Az ortogonális bázisvektorok miatt az is igaz, hogy minden újabb bázisvektor a megelőzőktől független hozzájárulást jelent a kimenetek előállításához és ez a hozzájárulás kiszámítható az éppen aktuális háló teljes kiértékelése nélkül. Az aktuális súlyvektor kiszámításához sem kell minden egyes lépésben a pseudo-inverz számítását elvégezni, hiszen az újonnan kiválasztott bázisvektorok együtthatói egymástól függetlenül számíthatók.

Az eljárás alkalmazásánál fontos a ρ küszöbérték megfelelő megválasztása, mivel ez szabályozza, hogy a végleges háló adott komplexitás mellett milyen pontosságot ér el. Számos alkalmazásban ρ meghatározása maga is az eljárás része. Erre ad egy rendszer-identifikációhoz kapcsolódó példát [Che89].

1.1.2. A K-közép eljárás

A standard K -közép (K -means) klaszter kialakító algoritmus is feltételezi, hogy az összes tanítópont a rendelkezésünkre áll. A K -means algoritmus célja, hogy K olyan klaszter-középpontot ($\mathbf{c}_k$, $k=1,2,\dots,K$) határozzon meg, hogy a tanítópontok és a hozzájuk legközelebb eső klaszter-középpont négyzetes távolságainak összege minimális értéket vegyen fel. Tehát, ha X az $\mathbf{x}_i$, $i=1, 2, \dots, P$ tanítópontok halmazát, $X^{(k)}$ pedig azon halmazt jelöli, melynek pontjai a $\mathbf{c}_k$ ($k=1,2,\dots,K$) középponthez lesznek legközelebb, akkor a következő kifejezés megoldását keressük :

$$\min_{\mathbf{c}_k} \sum_{k=1}^K \sum_{\mathbf{x}_i \in X^{(k)}} \|\mathbf{x}_i - \mathbf{c}_k\|^2 \quad (5.31)$$

Az algoritmus az alábbi lépésekből áll:

1. Inicializálás: véletlenszerűen válasszunk meg K középpontot: $\mathbf{c}_k$, $k=1,2,\dots,K$.

2. Határozzuk meg minden tanítópontra, hogy mely köz $\mathbf{c}_k$ ponthoz van a legközelebb $\mathbf{x}^{(i)}$ gyis alakítsuk ki a K klasztert. Ha $\mathbf{x}_i$ -től a legkisebb távolságra lévő középpont, akkor $\mathbf{x}_i$ a k -adik klaszter, $\mathbf{c}_k$ eleme lesz.
3. Határozzuk meg az így kialakított klaszterek új középpontjait. Egy klaszter új középponja legyen a klaszterekbe tartozó mintapontok átlaga:
$$\mathbf{c}_k = \frac{1}{P_k} \sum_{\mathbf{x}_i \in \mathcal{X}^{(k)}} \mathbf{x}_i \quad (5.32)$$
 ahol P_k az $\mathcal{X}^{(k)}$ klaszterbe tartozó tanítópontok száma.
4. Fejezzük be az eljárást, ha a mintapontok klaszterbe sorolása már nem változik, illetve adott számú iteráció után.

1.1.3. A középpontok ellenőrzött tanítása

A középpontok meghatározására további lehetőség az ellenőrzött tanítás alkalmazása. Ebben az esetben a bázisfüggvények paramétereit a kimeneti réteg súlyaihoz hasonló, tanítható paraméterekként kezeljük és például gradiens eljárással tanítjuk. Mivel a bázisfüggvények paramétereit a rejtett réteg leképezését befolyásolják, a kimeneti hibát a rejtett, bázisfüggvényes rétegre vissza kell terjeszteni, vagyis itt a hibavisszaterjesztés algoritmus egy RBF hálóra kidolgozott változatáról van szó.

Ha a pillanatnyi gradiens alapú eljárást alkalmazzuk, meg kell határozni a kimeneti négyzetes hiba adott középpontra vonatkozó parciális deriváltját. Feltételezve, hogy az alábbi általános (vagyis nem skalár σ szélességparaméterrel rendelkező) Gauss bázisfüggvényt alkalmazzuk:

$$g(\|\mathbf{x} - \mathbf{c}_i\|) = \exp[-(\mathbf{x} - \mathbf{c}_i)^T \Sigma_i^{-1} (\mathbf{x} - \mathbf{c}_i)], \quad (5.33)$$

valamint, hogy egykimenetű hálózatunk van, a láncszabályt alkalmazva a derivált a következőre adódik:

$$\begin{aligned} \frac{\partial \varepsilon^2(k)}{\partial \mathbf{c}_i(k)} &= 2 \varepsilon(k) \frac{\partial \varepsilon(k)}{\partial y(k)} \frac{\partial y(k)}{\partial g_i(k)} \frac{\partial g_i(k)}{\partial \mathbf{c}_i(k)} \\ &= -2 \varepsilon(k) w_i g'(\|\mathbf{x}(k) - \mathbf{c}_i(k)\|) \Sigma_i^{-1} (\mathbf{x}(k) - \mathbf{c}_i(k)) \end{aligned} \quad (5.34)$$

Itt $g'(\cdot)$ az i -edik bázisfüggvény $\mathbf{c}_i(k)$ szerinti deriváltja az $\mathbf{x}(k)$ bemenetre adott értéknél, w_i pedig az a súly, amivel az i -edik rejtett neuron kimenete szerepel a háló kimentében. A középpont módosító összefüggés ennek megfelelően:

$$\mathbf{c}_i(k+1) = \mathbf{c}_i(k) + 2\mu \varepsilon(k) w_i g'(\|\mathbf{x}(k) - \mathbf{c}_i(k)\|) \Sigma_i^{-1} (\mathbf{x}(k) - \mathbf{c}_i(k)), \quad (5.35)$$

ahol μ a szokásos tanulási tényező. A tanulási tényező értékének megválasztása gyakorlatilag csak tapasztalati úton lehetséges és ez az érték általában más is, mint a kimeneti réteg súlyainak tanításánál alkalmazott LMS eljárás tanulási tényezője. Míg az LMS eljárásnál μ konvergencia-tartományát meg tudjuk határozni, itt ez hasonló módon nem lehetséges. Ennek oka, hogy a kimeneti négyzetes hiba nem kvadrátikus függvénye $\mathbf{c}_i$ -nek, sőt ez a függvénykapcsolat nem is konvex. A nemkonvex hibafelület további következménye, hogy a gradiens alapú eljárás lokális minimumot eredményezhet.

A tanítással meghatározott középpontok természetesen nem fognak egybeesni a bemeneti pontokkal vagy azok részhalmazával. Sokkal inkább úgy igazodnak a tanítópontok elhelyezkedéséhez, mintha egy klaszterező eljárást alkalmaznánk. A középpontok tanítással történő meghatározása a középpontok számának meghatározását nem oldja meg, tehát hasonlóan a K -közép algoritmushoz, ezt a kérdést a tanítástól függetlenül kell megoldani.

1.2. A szélességparaméter meghatározása

Az RBF hálózatok – extrém értékektől eltekintve – nem érzékenyek a σ_i szélességparaméterek megválasztására. Ezt illusztrálja az 5.3 ábra. Az ábrán a szinusz függvény 1 periódusának approximációja látható, különböző szélességparaméter-értékek mellett, zajos mintapontok alapján. Az ábrán a bázisfüggvények száma és elhelyezése mindegyik esetben azonos. Az 5.3 (a) - (f) ábrákon látható eredmények rendre $\sigma=8, 7, 6, 5, 4$ és 3 értékkel készültek. Az (e) és (f) beállítások kivételével, ahol σ értéke már túl kicsi, a többi esetben az approximációs tulajdonságok jelentősen nem térnek el egymástól. Az ábra szaggatott vonallal a bázisfüggvényeket is mutatja.

Bár a szélességparaméter viszonylag széles határok között változhat anélkül, hogy az approximációs tulajdonságok lényegesen megváltoznának, a megfelelő σ tartomány, illetve σ érték meghatározására célszerű módszert adni. Egy egyszerű heurisztikus eljárás az R legközelebbi szomszéd módszer (*R-nearest neighbour heuristics*):

$$\sigma_k = \frac{1}{R} \sum_{j=1}^R \|\mathbf{c}_k - \mathbf{c}_j\|, \quad (5.36)$$

ahol az R különböző $\mathbf{c}_j$ középpont a $\mathbf{c}_k$ középpont R legközelebbi szomszédja.

RBF hálózatoknál gyakori, hogy minden rejtett rétegbeli elemnél azonos σ -t használunk, vagyis $\sigma_k = \sigma$, $k=1, \dots, K$ -ra. Tapasztalati eredmények azt mutatják, hogy a közös szélességparaméterre jó becslés, ha egy középpont és a hozzá legközelebbi középpont euklideszi távolságainak az átlagát vesszük:

$$\sigma = \frac{1}{K} \sum_{i=1}^K \|\mathbf{c}_i - \mathbf{c}_j\| \quad (5.37)$$

ahol a $\mathbf{c}_j$ középpont $\mathbf{c}_i$ legközelebbi szomszédja. Láthatóan (5.36) és (5.37) között csak az a különbség hogy az előbbi esetben R , az utóbbiban K távolság átlagát vesszük. Az R -heurisztikánál R értéke általában kicsi (2-3), míg K értékét az összes középpont száma adja meg.

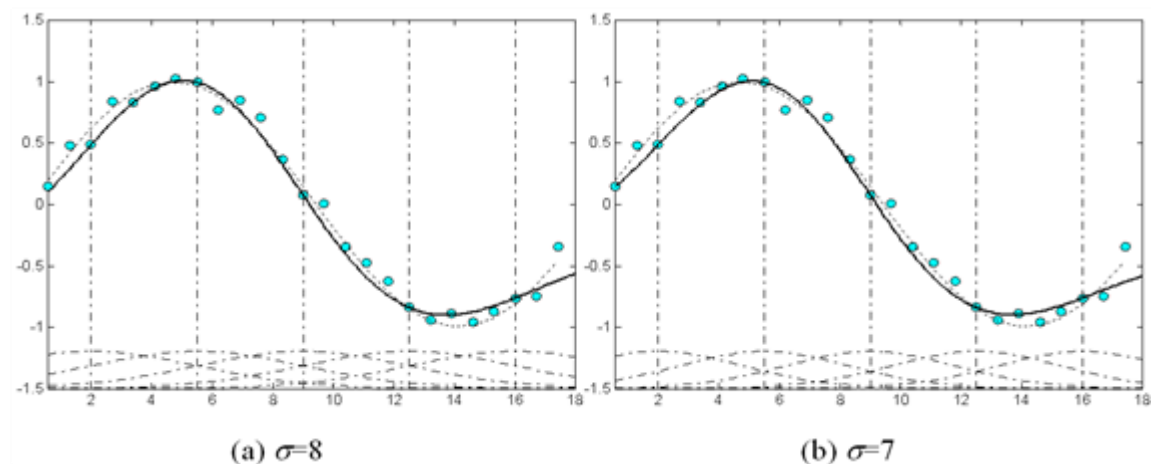
A σ_i értékek meghatározása ellenőrzött tanítással is lehetséges. Ekkor – hasonlóan a középpont-értékek tanításához – egy megfelelően származtatott hibavisszaterjesztéses tanítást használhatunk. Meg kell tehát határozni a kimeneti négyzetes pillanatnyi hiba szélességparaméterek szerinti gradiensét. Ha általános esetet tételezünk fel, minden rejtett neuronhoz más szélességparaméter tartozhat, sőt, mivel a bázisfüggvények többdimenziós térben értelmezendők az sem kötelező, hogy a bázisfüggvényekhez skalár σ_i értékek tartozzanak. Ekkor a (5.33) szerinti többdimenziós Gauss bázisfüggvénynél az összes dimenzió menti azonos σ szélességparaméter helyett egy Σ mátrixot tekintve Σ^{-1} tanító összefüggését kell meghatározni.

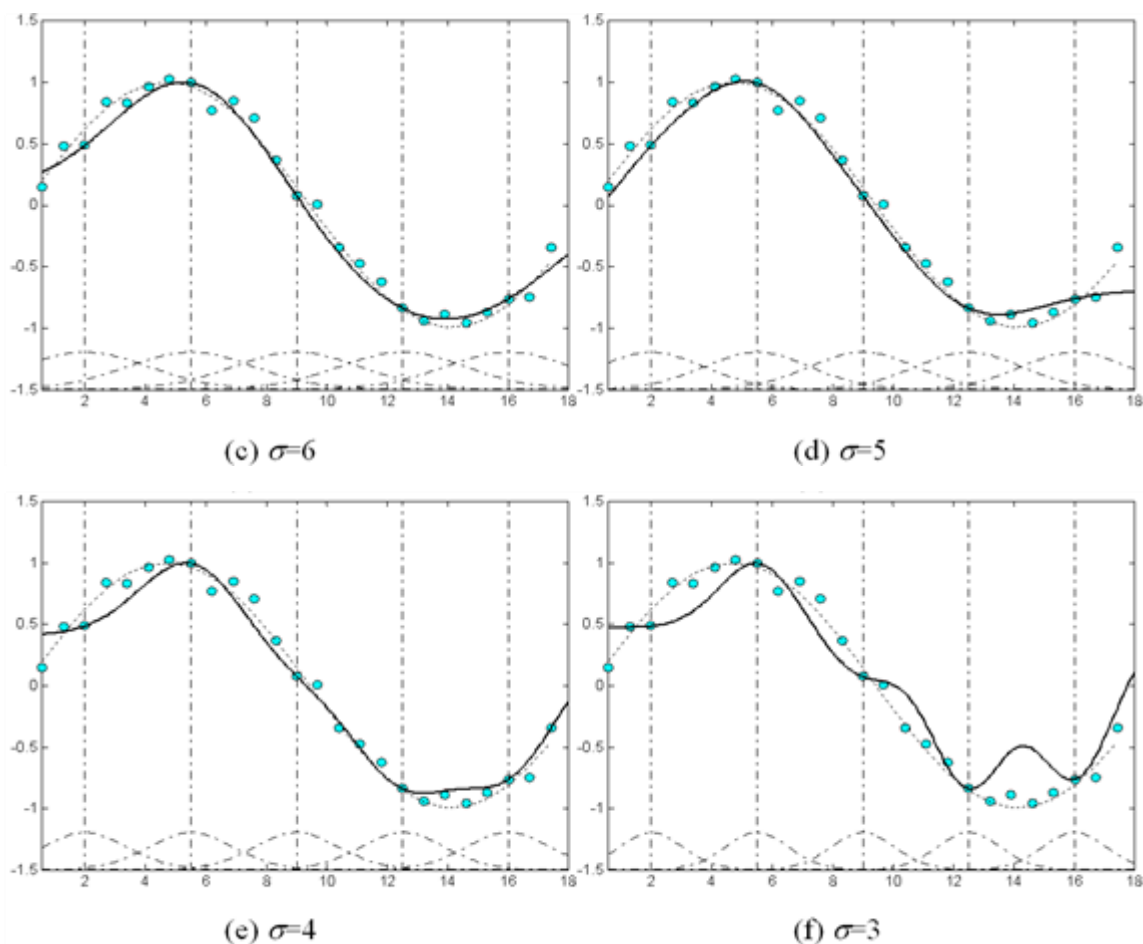
$$\frac{\partial \varepsilon^2(k)}{\partial \Sigma_i^{-1}} = -\varepsilon(k) w_i(k) g'(\|\mathbf{x}(k) - \mathbf{c}_i(k)\|) \mathbf{Q}_i(k) \quad (5.38)$$

ahol

$$\mathbf{Q}_i(k) = [\mathbf{x}(k) - \mathbf{c}_i(k)][\mathbf{x}(k) - \mathbf{c}_i(k)]^T \quad (5.39)$$

5.3. ábra - A σ szélességparaméter hatása az RBF approximációs képességére.





Ezzel a módosító összefüggés általános alakja:

$$\Sigma_i^{-1}(k+1) = \Sigma_i^{-1}(k) + \mu \varepsilon(k) w_i(k) g'(\|\mathbf{x}(k) - \mathbf{c}_i(k)\|) \mathbf{Q}_i(k) \quad (5.40)$$

A gradiens alapú tanítással meghatározott szélességparaméterekre érvényesek a középpontok ellenőrzött tanításánál említett általános megjegyzések. Tehát itt sem garantálja a tanító eljárás az optimális értékek elérését a lokális minimumok lehetősége miatt. A tanulási tényező megválasztására vonatkozó megállapítás is igaz, vagyis a szélességparaméter tanításánál alkalmazandó μ nagy valószínűséggel különbözni fog mind a kimeneti súlyok, mind a középpontok tanításánál alkalmazott tanulási tényezőktől.

1.3. RBF változatok

Az RBF hálózatoknak is több változata létezik. John Moody és Christian Darken javasolták a normalizált bázisfüggvények alkalmazását [Moo88]. Gauss függvény esetén ez az alábbi:

$$g_i(\mathbf{x}) = \frac{\exp\left[-\|\mathbf{x} - \mathbf{c}_i\|^2 / 2\sigma_i^2\right]}{\sum_k \exp\left[-\|\mathbf{x} - \mathbf{c}_k\|^2 / 2\sigma_k^2\right]} \quad (5.41)$$

A normalizálás itt azt jelenti, hogy $\sum_i g_i(\mathbf{x}) = 1$ bármely bemeneti vektornál ($\mathbf{x}$), vagyis az összes rejtett processzáló elem kimenetének (súlyozatlan) összege 1-et ad. A normalizált bázisfüggvények alkalmazását az approximációs tulajdonságok javulását mutató tapasztalatok indokolják [Wer93].

További változatok hozhatók létre attól függően, hogy a kimeneti rétegben alkalmazunk-e eltolás (*bias*) tagot vagy nem. Mind az 5.1, mind az 5.2 ábrán bemutatott hálózat tartalmaz eltolásértéket, amit w_0 biztosít. Ezek az ábrák általános bázisfüggvényes, illetve általános RBF struktúrát mutatnak. Eltolás alkalmazására azonban a bázisfüggvényes hálózatoknál – beleértve az RBF hálózatokat is – nincs mindig szükség. Az RBF hálózatoknak az a változata, amikor minden tanítópont egyben bázisfüggvény középpont is, nem igényli az eltolás

alkalmazását. Ezt az RBF háló változatot szokás regularizációs hálónak is nevezni, mert származtatása egy megfelelő regularizációs taggal kiegészített, négyzetes hibát minimalizáló eljárással is lehetséges [Pog90].

Az RBF hálózatok alkalmazásával kapcsolatban részben hasonló nehézségekkel kell szembenéznünk, mint az MLP hálózatok esetében. Itt is gondot jelent a hálózat méretének, vagyis a rejtett rétegbeli processzáló elemek számának (M) a meghatározása. Ismereteink szerint jelenleg erre nincs egzakt módszer, értékét tapasztalati megfontolások alapján választhatjuk meg. Ugyanakkor meg kell jegyezni, hogy az előbbieken bemutatott OLS eljárás valójában a háló komplexitási problémára adható egy lehetséges válasz.

Hasonlóan, a szélességparaméternél R értékének megállapítására is csak tapasztalati eljárások állnak rendelkezésünkre. Ezen módszerek lényegében különböző értékű M és R paraméterek mellett felépített hálózatok tanítását és tesztelését végzik, és a megtanított hálózatok képességeiből próbálnak a megfelelő hálózat-méretre következtetni, így tulajdonképpen a kereszt kiértékelés valamilyen változatának tekinthetők. A kereszt kiértékelési eljárásnak a tanítópontok számától függő valamelyik változata tehát alapvető fontosságú szerepet tölt be a hálók konstrukciójánál.

A rejtett neuronok számának meghatározásakor figyelembe kell venni az RBF (és általában a bázisfüggvényes hálózatok) egy specialitását. E hálózatok többsége – szemben az MLP-vel – lokális approximációt végez, ami azt jelenti, hogy egy bázisfüggvénynek a középpont egy véges környezetében van csak számottevő hatása. E véges környezet mérete függ a bázisfüggvénytől és a függvény σ szélességparaméterétől. Minél kisebb σ értéke, annál kisebb tartományba eső bemenet esetén lesz az adott bázisfüggvénynek (az adott rejtett rétegbeli neuronnak) jelentős hozzájárulása a kimenethez. Azt a tartományt, amelybe eső bemenetekre a bázisfüggvény jelentős értéket ad a bázisfüggvény érzékelési mezőjének (*receptive field*) szokás nevezni. Az RBF hálózatoknál az érzékelési mező a legtöbb esetben gyakorlatilag véges, még végtelen tartójú pl. Gauss bázisfüggvényeknél is.

A bázisfüggvények számát a szélességparaméter, de leginkább az approximálandó függvény Fourier transzformált tartománybeli tulajdonságai határozzák meg. Ha a tanítópontok egyben középpontok is, az RBF háló valójában a tanítópontok között végez interpolációt úgy, hogy a tanítópontokban a megtanulandó függvényt pontosan előállítja. A tanítópontokat ebben az esetben, mint mintavételi értékeket is értelmezhetjük, így az RBF hálózatos függvény-approximáció egy mintavételi értékekből történő (általában többdimenziós) jelviszsaállítás feladatként is felfogható. A megfelelő jelviszsaállítás – amennyiben a függvény Fourier tartománybeli viselkedését ismerjük – megszabja a mintavételezés sűrűségét, a mintavételi pontok számát, ami itt egyben a tanítópontok számát és a rejtett neuronok számát is jelenti. A gyakorlatban a mintavételezés Nyquist elmélete alapján történő rejtett réteg konstrukció általában nem lehetséges, hiszen az approximálandó függvény jellemzőit nem ismerjük. Ráadásul gyakorlati feladatoknál a rendelkezésünkre álló tanítópontok (a mintavételi pontok) a függvény értelmezési tartományának számunkra fontos részén belül szinte sohasem helyezkednek el egyenletesen, ezért a probléma a nemegyenletes (többdimenziós) mintavételezéssel, a nemegyenletesen elhelyezkedő mintavételi pontokból történő jelviszsaállítással [Kim90] hozható kapcsolatba.

Ha a tanítópontoknál kevesebb a rejtett rétegbeli neuronok száma, tehát amikor pl. klaszterezéssel határozzuk meg a középpont paramétereket, a helyzet még nehezebb, hiszen itt a középpontok mintavételi pontokként nem is értelmezhetők. A mintavételi értékekből való jelviszsaállítással való kapcsolat azonban arra felhívja a figyelmet, hogy a tanítópontok száma és elhelyezkedése, az approximálandó függvény Fourier tartománybeli viselkedése, a bázisfüggvények megválasztása és a rejtett neuronok száma nem függetlenek egymástól.

1.4. A kizáró VAGY (XOR) probléma megoldása RBF hálózattal

Láttuk, hogy az RBF háló interpretálható úgy is, mint egy olyan perceptron, ahol a bemenetek nem közvetlenül, hanem egy nemlineáris transzformációt követően kerülnek a perceptron bemeneteire. A nemlineáris transzformáció – a perceptron kapacitására vonatkozó Cover tétel alapján [Cov65] – általában dimenziónövelő transzformáció is egyben. Számos esetben azonban a nemlineáris transzformáció dimenziónövelés nélkül is biztosíthatja a lineáris szeparálhatóságot. Erre mutatunk példát a kizáró VAGY (XOR) kapcsolat RBF hálózattal történő megoldásával.

Konstruáljunk egy olyan RBF hálót, melynek két rejtett neuronja van, tehát két bázisfüggvényt alkalmazzunk (5.4 ábra).

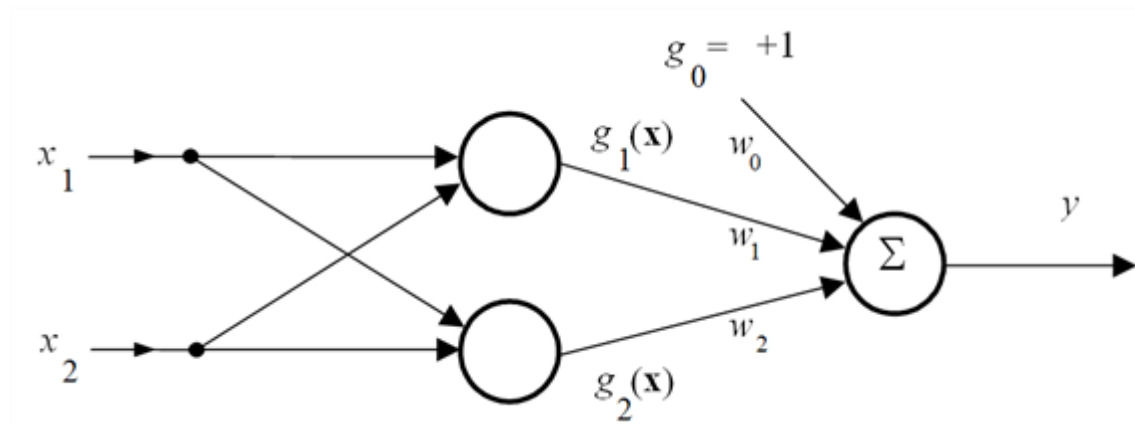
A Gauss bázisfüggvények legyenek az alábbiak

$$g_i(\mathbf{x}) = \exp\left[-\|\mathbf{x} - \mathbf{c}_i\|^2\right] \quad i = 1, 2 \quad (5.42)$$

ahol a $\mathbf{c}_1$ és $\mathbf{c}_2$ középpontvektorok

$$\mathbf{c}_1 = [1, 1]^T, \mathbf{c}_2 = [0, 0]^T. \quad (5.43)$$

5.4. ábra - A XOR problémát megoldó RBF háló



Az (5.42) összefüggés olyan Gauss bázisfüggvényt definiál, ahol $2\sigma^2=1$. Az ábrán látható RBF háló a XOR probléma négy mintapontja közül kettőt használ középpontnak. A hálónak három szabad paramétere van: a w_0 eltolásérték, valamint w_1 és w_2 . A súlyok analitikusan is meghatározhatók. Mivel négy tanítópontunk van, ehhez egy túlhatározott lineáris egyenletrendszer kell megoldanunk, ami a pszeudo-inverz meghatározásával lehetséges.

$$\mathbf{w}^* = \mathbf{G}^\dagger \mathbf{d} = (\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T \mathbf{d} \quad (5.44)$$

A rejtett réteg válaszaiból képezett $\mathbf{G}$ mátrix a fenti bázisfüggvények mellett a következőre adódik:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0,135 \\ 1 & 0,368 & 0,368 \\ 1 & 0,135 & 1 \\ 1 & 0,368 & 0,368 \end{bmatrix} \quad (5.45)$$

A kívánt válaszok vektora a XOR problémának megfelelően

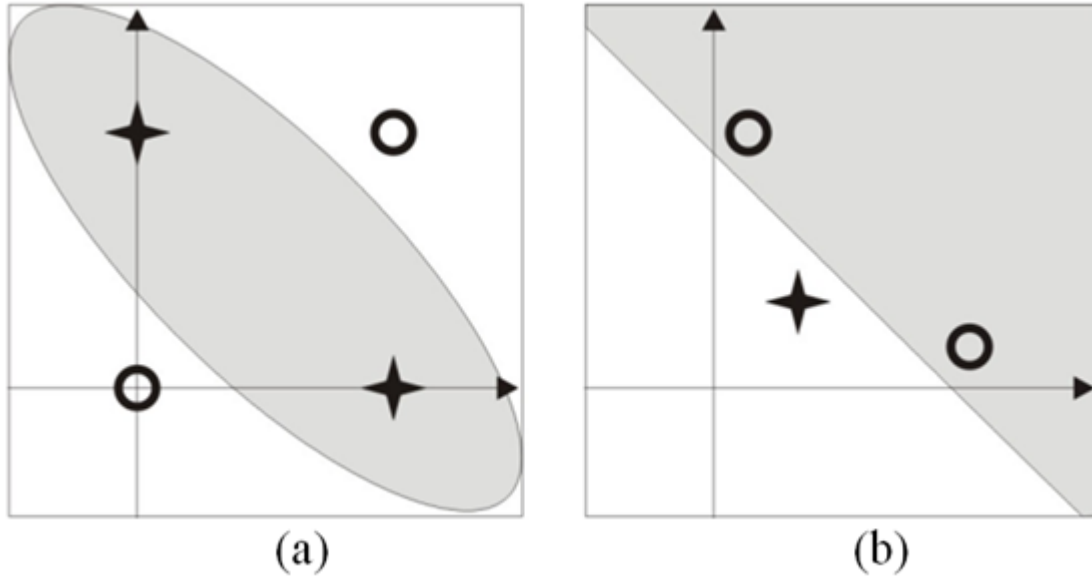
$$\mathbf{d} = [0 \ 1 \ 0 \ 1]^T \quad (5.46)$$

Elvégezve a pszeudo-inverz számítást a megoldásvektor a következőre adódik:

$$\mathbf{w} = [+2,840 \ -2,502 \ -2,502]^T \quad (5.47)$$

Az 5.5 (a) ábra a bemeneti térben, a (b) ábra a transzformált térben mutatja a mintapontokat. Látható, hogy a transzformált térben a pontok már lineárisan szeparálhatók. Ugyanakkor az is látható, hogy a szeparáló egyenes nem megy át az origón, ami azt jelenti, hogy szükség van eltolás tagra, amint ezt a megoldás mutatja is.

5.5. ábra - A mintapontok elhelyezkedése: (a) a bemeneti térben, (b) a transzformált térben (a kereszttel jelölt két mintapont a transzformált térben egybeesik)



2. A CMAC hálózat

A CMAC (*Cerebellar Model Articulation Controller*) hálózatot 1975-ben javasolta James Albus [Alb75]. A perceptrontól származtatott hálózat kétlépéses leképezést valósít meg, így sok szempontból rokonítható az RBF hálózatokhoz. A hálózat itt is a bemenet valamilyen nemlineáris transzformációját valósítja meg, majd a transzformált értékek egy lineáris, ellenőrzött tanítású réteg bemenetére kerülnek.

Az RBF és a CMAC hálózatok között az alapvető különbség a nemlineáris leképezésben van. A CMAC hálózat a bemenetet egy ún. asszociációs vektorra képezi le, mégpedig úgy, hogy véges számú, diszkrét bemeneti értéket tételez fel, és minden lehetséges bemeneti értékhez egy kölcsönösen egyértelmű asszociációs vektort rendel. Ez azt jelenti, hogy a CMAC hálózat kvantált bemenetekkel dolgozik. b -bit-es kvantálót feltételezve, egydimenziós bemenetnél $R = 2^b$, N -dimenziós bemenetnél $R^N = 2^{Nb}$ különböző bemeneti értékünk lehet. Az egyszerűség kedvéért először az egydimenziós bemenet esetét vizsgáljuk, majd ezután végezzük el a többdimenziós bemenetre való kiterjesztést.

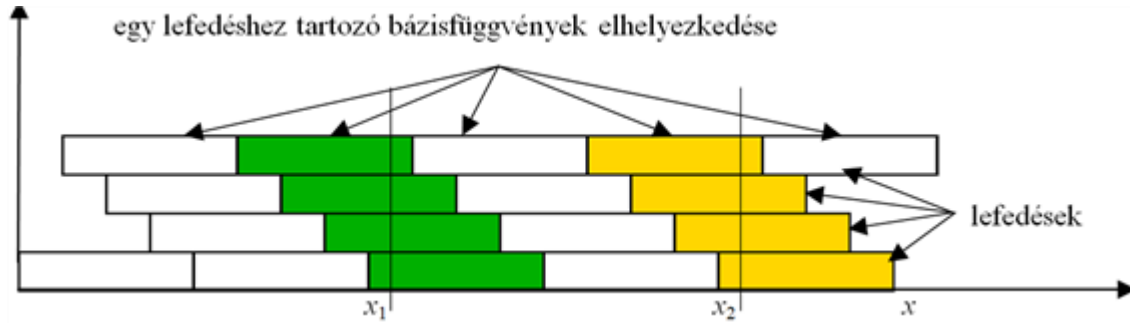
Az egyes bemeneti értékekhez rendelt asszociációs vektor egy olyan sokbit-es bináris vektor, amelyben minden diszkrét bemeneti értékhez rögzített számú, egymás mellett elhelyezkedő bit lesz aktív (1 értékű), míg a vektor összes többi bitje 0 értéket vesz fel. Az aktív bitek száma – amelyet a továbbiakban C -vel jelölünk – a hálózat fontos paramétere, befolyásolja a hálózat általánosító-képességét. Megjegyezzük, hogy a bináris asszociációs vektor miatt az Albus által javasolt CMAC hálózatot szokás bináris CMAC-nak is nevezni. Később javasoltak ún. magasabbrendű CMAC változatokat is, melyeknél az asszociációs vektor elemei nemcsak bináris értékeket vehetnek fel. Ezekkel a változatokkal a későbbiekben foglalkozunk.

Az asszociációs vektort úgy is elképzelhetjük, mint egy sokelemű neuronréteget, melynek neuronjai közül minden különböző diszkrét bemenetre pontosan C kerül gerjesztett állapotba, vagyis ad 1 értékű kimenetet. A gerjesztett neuronok célszerűen a rétegben egymás melletti helyeken találhatók és olyan módon kerülnek kiválasztásra, hogy az egymáshoz közeli bemeneti értékekre válaszul részben azonos neuronok aktivizálódnak. Az egyes neuronoknak itt is van érzékelési mezőjük (*receptive field*). Adott bemenetnél mindazon neuronok aktivizálódnak, melyek érzékelési mezői tartalmazzák az adott bemenetet. Minél közelebbi értékek kerülnek a hálózatra, annál nagyobb lesz az átfedés az aktív neuronok, vagyis az asszociációs vektor aktív bitjei között, hiszen a közeli bemenetek nagyrészt azonos érzékelési mezőkbe esnek. Szomszédos, tehát egymástól egy kvantumnyira lévő bemenetekhez a C aktív bitből $C-1$ lesz közös, míg a legalább C kvantum távolságban lévő bemenetekre adott asszociációs vektor válaszokban már nem találunk közös aktív biteket.

Az érzékelési mezőkkel történő értelmezés alapján az egyes neuronokat tekinthetjük úgy is, mintha minden neuronhoz egy bázisfüggvény tartozna. Ebben az értelmezésben a CMAC háló is egy bázisfüggvényes háló, ahol bináris és valóban véges tartójú bázisfüggvényekkel dolgozunk. Egydimenziós bemenet esetén a bázisfüggvények tartója C kvantumnyi intervallum, míg többdimenziós bemenetnél a tartó egy C kvantumnyi oldalhosszúsággal rendelkező N -dimenziós hiperkocka.

A bázisfüggvényeknek úgy kell elhelyezkedniük, hogy bármely megengedett diszkrét bemenet C bázisfüggvény tartójába essen. Így bármely bemenet hatására C rejtett neuron aktivizálódik. Ezt úgy valósíthatjuk meg, ha minden bemeneti kvantum (elemi cella) a bázisfüggvényekkel C -szeresen lesz lefedve. Egy lefedés azt jelenti, hogy a bázisfüggvények tartói „hézagmentesen” és átlapolódásmentesen a teljes bemeneti tartományt lefedik. Ezt mutatja egydimenziós esetre $C=4$ mellett az 5.6 ábra.

5.6. ábra - A bemeneti tartomány lefedése bázisfüggvényekkel egydimenziós esetben $C=4$ mellett



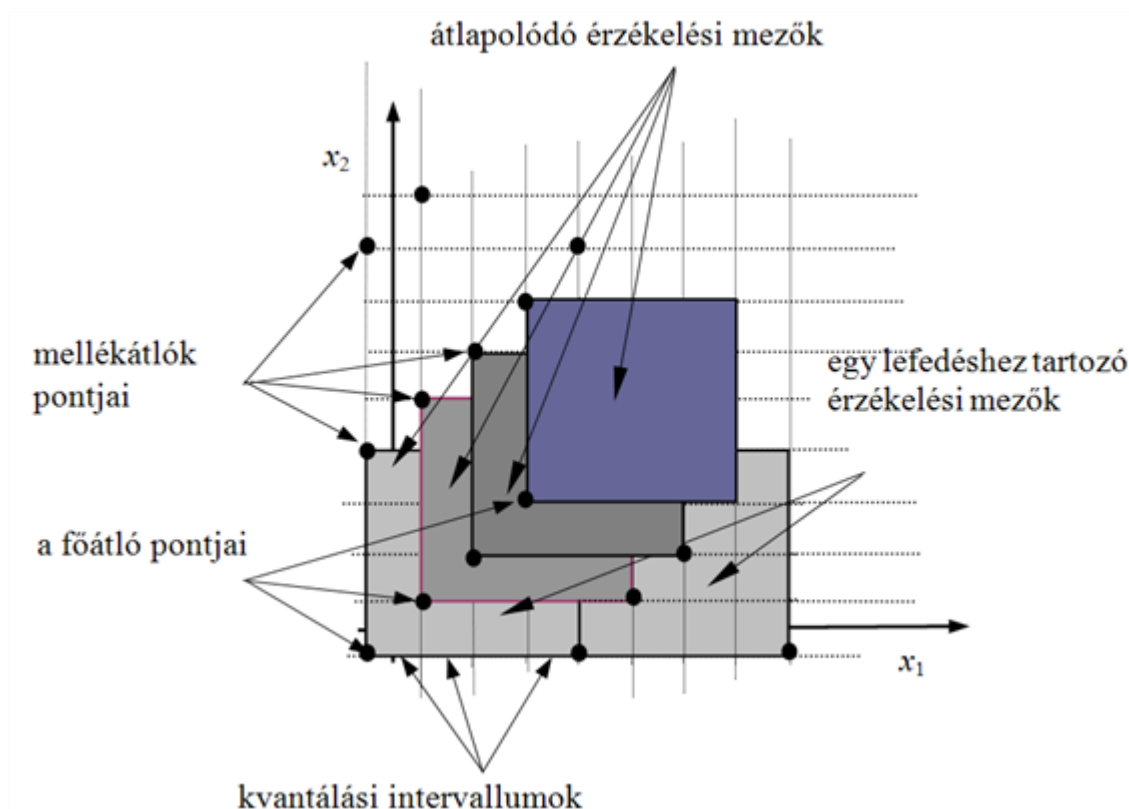
Az ábrán látható, hogy az egyes lefedésekhez tartozó bázisfüggvények tartói egy kvantumnyi távolsággal vannak eltolva egymáshoz képest. Így biztosított, hogy bármely diszkrét bemenet mindig C bázisfüggvény tartójába essen és a szomszédos bemenetek által aktivált bázisfüggvények közül $C-1$ közös. Az ábrán az is látható, hogy adott bemenetekhez mely bázisfüggvények lesznek kiválasztva: x_1 a sötétebben (zöld), x_2 pedig a világosabban (sárga) árnyékolt bázisfüggvények érzékelési mezőibe esik bele.

Ha többdimenziós esetben is hasonlóan fednének le bázisfüggvényekkel a bemeneti teret, akkor a minden dimenzió mentén egy kvantumnyi eltolás azt eredményezné, hogy N -dimenziós esetben C^N lefedést lehetne kialakítani. Ezt a lefedést teljes lefedésnek (*full overlay*) nevezzük, és ez azt jelentené, hogy a bázisfüggvények száma a dimenzióval exponenciálisan növekedne (a dimenzió átka probléma). Ha megelégszünk azzal, hogy dimenziótól függetlenül bármely bemenethez C aktív bit tartozzon az asszociációs vektorban, vagyis egy bemenet mindig csak C bázisfüggvény tartójába essen, jóval kevesebb bázisfüggvényre van szükségünk. Az 5.7 ábra kétdimenziós bemeneti tartományra mutatja a bázisfüggvények elhelyezésének egy lehetséges módját, amely a C -szeres lefedést biztosítja $C=4$ mellett.

Látható, hogy egy lefedés a bemeneti tér teljes „kicsompózását” jelenti. A bázisfüggvények úgy helyezkednek el, hogy minden lehetséges diszkrét bemenet egy lefedésen belül egy és csakis egy bázisfüggvény érzékelési mezőjébe esik. Az asszociációs vektorban a C aktív bit úgy biztosítható, ha pontosan C lefedést alkalmazunk.

Az 5.8 ábrán egy bemeneti mintapontot (a + jellel megjelölt x pont) és a kvantált bemenet miatt a pontot is magába foglaló elemi cellát (árnyékolt terület) lefedő C érzékelési mező elhelyezkedése látható ($C=4$). Minden egyes érzékelési mező egy bázisfüggvényhez tartozik és minden bázisfüggvény egy bitet jelent az asszociációs vektorban. Az adott x bemenet mellett az ábrán bejelölt érzékelési mezőknek megfelelő asszociációs bitek lesznek aktívak.

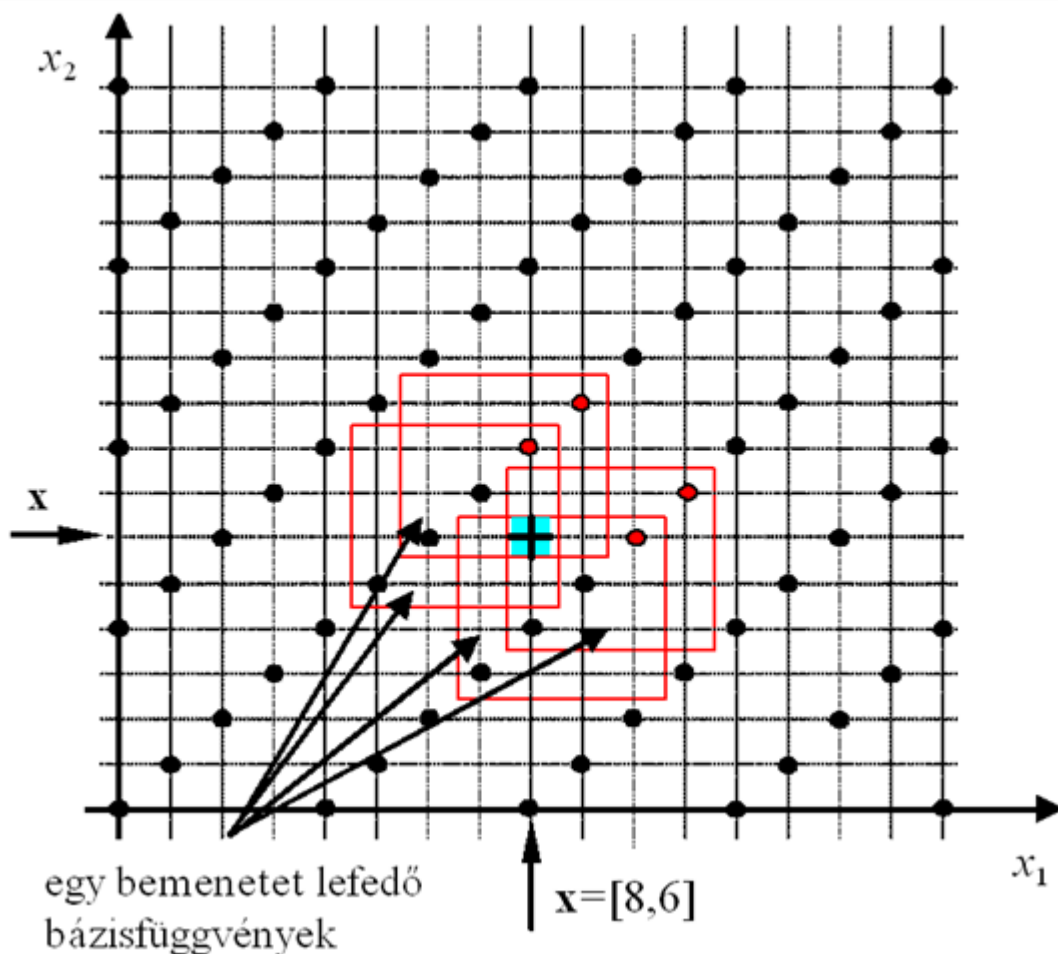
5.7. ábra - A bemeneti tér lefedése bázisfüggvényekkel kétdimenziós CMAC-nál $C=4$ mellett.



Az asszociációs vektor a hálózat kimeneti rétegének bemenetül szolgál. Minthogy bináris vektorral van dolgunk, ez azt jelenti, hogy az asszociációs vektorban lévő aktív bitek C súlyértéket választanak ki a kimeneti réteg súlyai közül, majd a kimeneti összegző ezen súlyok összegeként állítja elő a hálózat bemenetre adott választát. Vegyük észre, hogy ha $C=1$, akkor minden diszkrét bemenethez csak 1 aktív bit tartozik az asszociációs vektorban, és a hálózat kimenete ezen bit által kiválasztott súly értéke lesz. Ha tehát az asszociációs vektor pontosan annyi bites, ahány különböző diszkrét bemeneti értékünk lehet, akkor egy egyszerű táblázattal állunk szemben. Ha a táblázat értékei – az egyes súlyok – a diszkrét bemenetekre adandó kívánt válaszok, akkor a hálózat minden diszkrét bemeneti értékre nyilvánvalóan a pontos kívánt választ fogja adni.

A CMAC hálózat alapvető architektúráját és leképezéseit mutatja az 5.9 ábra. A gyakorlatban C értékét egynél nagyobbra, de az asszociációs vektor hosszánál jóval kisebbre választjuk. A bemenetről az asszociációs vektorra való leképezés tulajdonképpen egy bináris kód hozzárendelése a bemenethez, és mivel az M -bites vektorban csupán $C \ll M$ aktív bit lesz, a bináris kód ritka kód lesz.

5.8. ábra - Egy bemeneti mintapont tér lefedése bázisfüggvényekkel



Az aktív bitek határozzák meg, hogy a kimeneti lineáris összegző rétegben mely súlyok összegeként áll elő a kimeneti érték. Az aktív asszociációs bitek elrendezése biztosítja a hálózat általánosító képességet. Két bemeneti vektor, $\mathbf{x}_1$ és $\mathbf{x}_2$ hasonlóságát a hozzájuk tartozó $\mathbf{a}_1$ és $\mathbf{a}_2$ asszociációs vektorok metszete mutatja. Ezt az esetet illusztrálja az 5.9 ábra, ahol $\mathbf{a}_1$ és $\mathbf{a}_2$ aktív bitei közül kettő közös. Ugyanakkor az ábrán az is látható, hogy egymástól távol lévő bemeneti vektorok ($\mathbf{x}_3$ és $\mathbf{x}_1$ vagy $\mathbf{x}_2$) esetében nincsenek közös aktív asszociációs bitek ($\mathbf{a}_3$ és $\mathbf{a}_1$ vagy $\mathbf{a}_2$). A jobb áttekinthetőség kedvéért az ábrán csak az aktív biteket és az általuk kiválasztott súlyokat jelöltük. Természetesen nem szükséges – és többdimenziós bemenet esetén nem is lehetséges –, hogy minden bemeneti értékhez az asszociációs vektor szomszédos C bite legyen aktív. A bemenetről az asszociációs vektorokra való leképezésnél mindössze azt kell biztosítani, hogy minél közelebb van egymáshoz két bemeneti vektor, annál nagyobb átfedés legyen az asszociációs vektorok között, vagyis az asszociációs vektorok Hamming távolsága annál kisebb legyen.

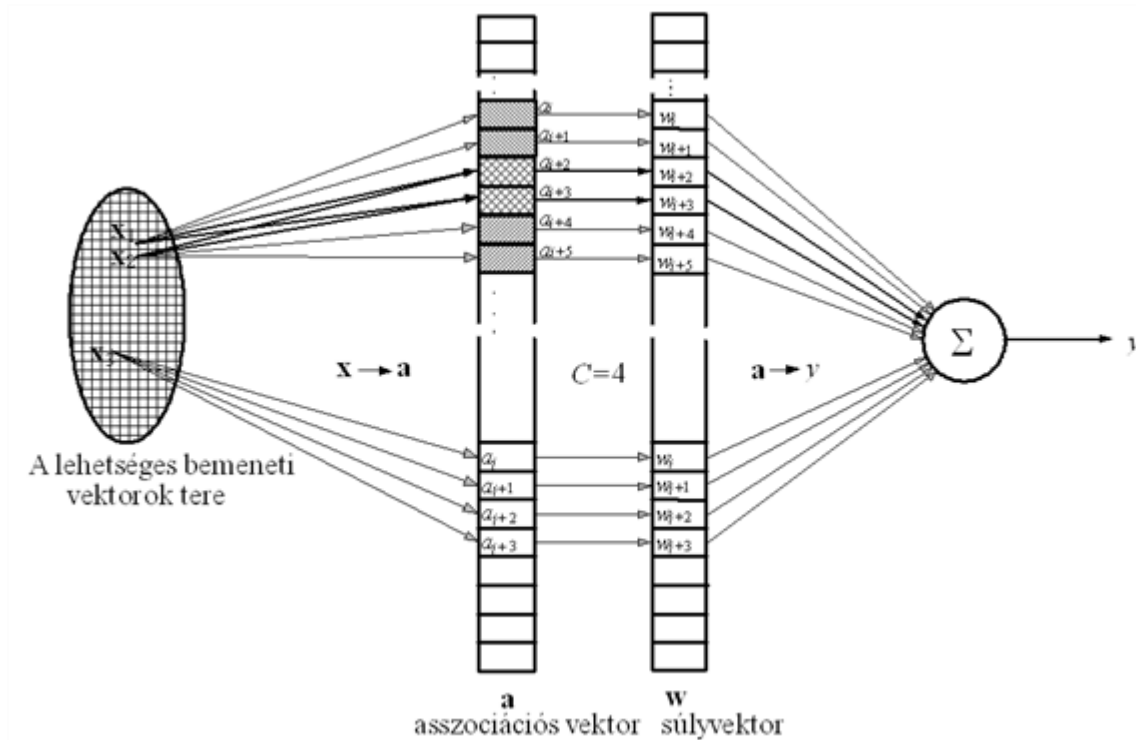
A kimeneti érték – hasonlóan bármely más bázisfüggvényes hálózathoz – az asszociációs vektor (a bázisfüggvényes rejtett réteg válasza, $\mathbf{a}(\mathbf{x})$ és a súlyvektor skalár szorzataként határozható meg:

$$y(\mathbf{x}) = \mathbf{a}(\mathbf{x})^T \mathbf{w}. \quad (5.48)$$

A CMAC asszociációs vektora bináris vektor, ezért a skalár szorzást valójában nem is kell elvégezni, elegendő az aktív bitek indexeit ismerni, mivel ezek fogják kijelölni a súlyokat tároló memória azon rekeszeit, amelyekben az adott bemenethez tartozó kimenetet meghatározó súlyok találhatók. A kimenet számítása ezért ténylegesen csak összeadást igényel, ami egyértelmű implementációs előnyt jelent minden eddigi hálóval szemben:

$$y(\mathbf{x}) = \sum_{i: a_i(\mathbf{x})=1} w_i \quad (5.49)$$

5.9. ábra - A CMAC hálózat leképezései



2.1. A háló komplexitása

Az 5.9 ábrán bemutatott elvi felépítésből látható, hogy bár a CMAC háló szintén bázisfüggvényes háló, jelentős mértékben különbözik az előbbieken tárgyalt RBF hálótól. A különbségnek több oka van. Egyrészt a CMAC háló kvantált bemenetekkel dolgozik, ezért itt csak véges számú különböző bemenet fordulhat elő. Másrészt az alkalmazott bázisfüggvények nem rendelkeznek szabad paraméterekkel. A háló első rétegének egyetlen megválasztható paraméterét, C -t előre rögzíteni kell, és ebből a háló teljes felépítése következik. Mivel a bázisfüggvények rögzített elhelyezésűek, a CMAC hálónál valóban csak a kimeneti lineáris réteg súlyait kell meghatározni, ami mind analitikus összefüggés alapján, mind iteratív tanulással (pl. LMS algoritmussal) lehetséges.

A háló komplexitását is C értéke és a bemenet dimenziója határozza meg. A háló első rétege a kvantált bemenetekből állítja elő az asszociációs vektor aktív biteinek megfelelő memóriacímeket. Ezek a címek adják meg, hogy az aktív bitekhez tartozó bázisfüggvények súlyai egy súlymemóriában hol találhatók. Mind az első réteg, mind a második, lineáris réteg komplexitását a bázisfüggvények száma, az asszociációs vektor hossza határozza meg. Az első réteg a diszkrét bemenetek és az asszociációs vektorok között kölcsönösen egyértelmű leképezést valósít meg, tehát minden lehetséges diszkrét bemenethez egy egyedi asszociációs vektor tartozik. Ebből adódik, hogy C aktív bit mellett egydimenziós esetben az asszociációs vektor hossza

$$M = R + C - 1 \quad (5.50)$$

lesz, ahol R a lehetséges diszkrét bemenetek száma. N -dimenziós esetben, teljes (C^N -szeres) lefedésnél ez

$$M = \prod_{i=1}^N (R_i + C - 1) \quad (5.51)$$

lenne, ami nagyon nagy érték is lehet. Az összefüggésekből látható, hogy az asszociációs vektor közelítőleg annyi bites, mint ahány lehetséges diszkrét bemeneti érték van. Pl. egy 10-dimenziós feladatnál, ahol minden bemeneti komponens 10 bitre kvantálunk, a lehetséges diszkrét bemeneti vektorok és így az asszociációs bitek száma már megvalósíthatatlanul nagy: 2^{100} . Eddigi formájában a CMAC hálózat súlyait tároló memória tehát csak meglehetősen kis bemeneti dimenzió és durva kvantálás mellett lenne realizálható méretű. Ha a teljes lefedés helyett az előbbieken már említett C -szeres lefedést alkalmazzuk, akkor az asszociációs vektor hossza közelítőleg már „csak”

$$M_C = \left\lceil \frac{1}{C^{N-1}} \prod_{i=1}^N (R_i + C - 1) \right\rceil \quad (5.52)$$

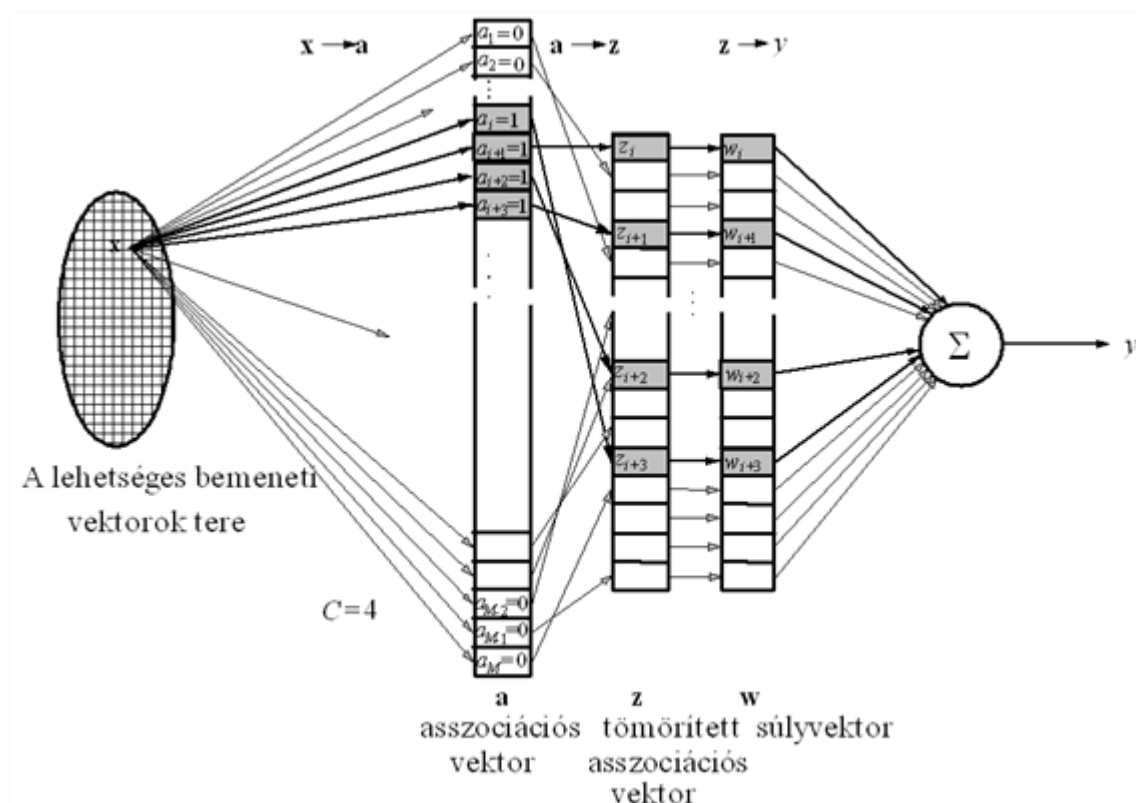
lesz (itt $\lceil \cdot \rceil$ benne lévő értéknél nagyobb legkisebb egészt jelöli). Bár M_C nagyságrendekkel kisebb is lehet, mint M , ha a dimenziószám, N nagy, a címtartomány még mindig akkora, hogy a súlyok tárolásához szükséges memória még mindig irreálisan nagy. Pl. az előzőekben említett 10-dimenziós bemeneti vektorok esetében, ha a bemeneti komponenseket 10 bitre kvantáljuk és $C=128$, a címtartomány mérete, M_C közelítőleg még mindig 2^{37} , ami 128 Gbyte.

A realizálhatóság érdekében a súlyok számát valamilyen módon tovább kell csökkenteni. Ez viszonylag egyszerűen akkor tehető meg, ha van valamilyen a priori ismeretünk a bemeneti mintapontok eloszlásáról. Ha a bemeneti mintatérben vannak sűrűsödési helyek, ahol nagy számban találhatók tanítópontok, akkor elsősorban ezen tartományokhoz kell aktív asszociációs biteket és így súlyokat rendelni. A bemeneti mintatér többi részéhez, ahol kis valószínűséggel találunk tanítópontot, egyáltalán ne rendeljünk súlyokat. A priori ismeret hiányában ez a megoldás azonban nem alkalmazható. Iktassunk be ezért egy újabb, tömörítő leképezést az asszociációs vektor és a súlyvektor közé. S minthogy a bemeneti mintapontok eloszlásáról nincs a priori ismeretünk, a legjobb amit tehetünk egy véletlen(szerű) leképezés alkalmazása.

Rendeljünk az a asszociációs vektorhoz egy z ún. tömörített asszociációs vektort úgy, hogy a minden egyes bitjéhez rendeljük hozzá z egy bitjét. A véletlen hozzárendelés miatt az egy bemeneti mintapontra válaszként kapott a asszociációs vektor aktív bitjei között fennálló kapcsolat (pl. szomszédosság) z aktív bitjeire már nem lesz igaz. A súlyokat ezek után nem a , hanem z bitjei választják ki, tehát a kimeneti érték előállításához azon súlyok összegét kell képezni, amelyekre z aktív bitjei mutatnak. A tömörítést is alkalmazó CMAC hálózat leképezéseit az 5.10 ábra mutatja.

Ha z bitjeinek száma sokkal kisebb, mint a -é, akkor jelentősen csökkenteni tudjuk a súlyok számát. Természetesen e tömörítő leképezés csak úgy valósítható meg, ha megengedjük, hogy a különböző cellái véletlenszerűen ugyanarra a z cellára legyenek leképezve, tehát valójában több-egyértelmű leképezést alkalmazunk. Az eljárás a nem kölcsönösen egyértelmű leképezés ellenére a gyakorlati feladatok megoldásánál jól működik. Ugyanis a hálózat bemenetére a lehetséges diszkrét bemeneti mintáknak rendszerint csak egy kis töredéke kerül, így annak valószínűsége, hogy különböző bemeneti mintákhoz azonos z asszociációs vektorok tartozzanak csaknem nulla. Annak valószínűsége azonban, hogy egymástól távol lévő bemeneti minták z asszociációs vektoraiban közös aktív bitek is legyenek már nagyobb. Ilyen esettel elsősorban akkor találkozhatunk, ha a tömörítés mértéke igen nagy, vagyis z sokkal kisebb méretű, mint a . Ezt a jelenséget "ütközés"-nek hívjuk. Az ütközés következtében az egymástól távol lévő tanító pontok részben azonos súlyokat választanak ki, ami a tanításnál interferenciát okoz. Ha az ütközések valószínűsége kicsi, a hatás elhanyagolható, illetve a tanító lépések számának növelésével az ütközésből adódó interferencia csökkenthető.

5.10. ábra - Tömörítést is alkalmazó CMAC hálózat leképezései

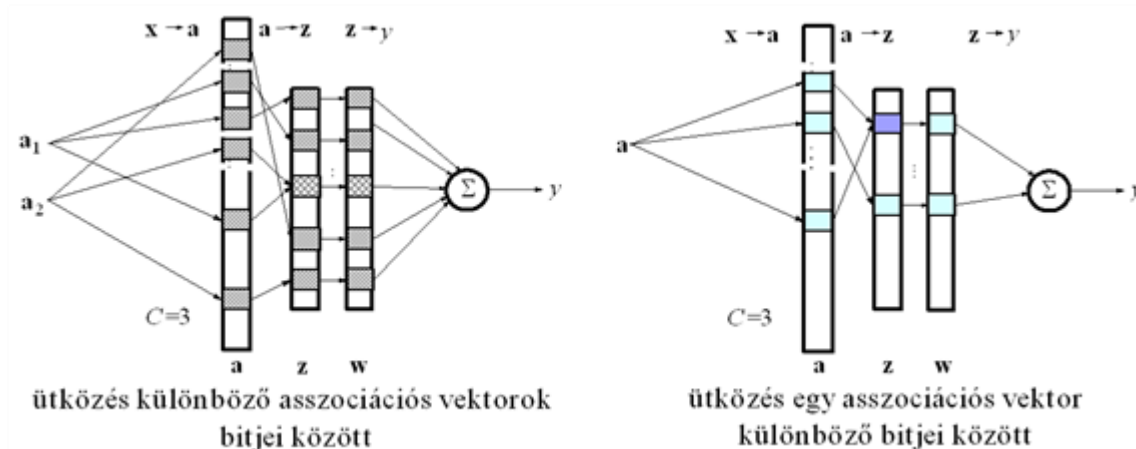


Az ütközésnek egy másik fajtája is előfordulhat a tömörítés következtében. Egy bemeneti vektorhoz C aktív bit tartozik az asszociációs vektorban, így előfordulhat, hogy a C bit közül kettőt vagy többet ugyanarra a z -beli bitre képezünk le, vagyis a tömörített vektorban az aktív bitek száma $\leq C$.

A kétfajta ütközést illusztrálja az 5.11. ábra. Az ütközések gyakorisága a tömörítő leképezés megvalósításától, továbbá az aktív bitek számának a tömörített asszociációs vektor hosszához viszonyított arányától is függ.

Megmutatható, hogy ha az aktív bitek száma kisebb, mint kb. 1%-a a tömörített asszociációs vektor hosszának, akkor csak kis valószínűséggel következik be ütközés. Annak valószínűsége pedig, hogy két eltérő asszociációs vektorban több mint két bitnél is ütközés következzen be, illetve, hogy egy asszociációs vektor kettőnél több bitje ugyanarra a tömörített vektor-bitre legyen leképezve gyakorlatilag nulla. Ez arra utal, hogy megfelelő tömörítő leképezés alkalmazása nem rontja jelentősen a hálózat képességeit [Ell91].

5.11. ábra - A tömörítő leképezés ütközései



A tömörítő leképezés azt jelenti, hogy a kiinduló (virtuális) címtartományt egy kisebb (fizikai) címtartományra képezzük le. A két címtartomány közötti leképezés *hash kódolással* történhet. A hashelés (a szó azt jelenti, hogy összekeverni, összefoglalni valamit) véletlenszerű leképezése különösen a tömörítés miatt fontos, hiszen ez

biztosítja, hogy az $\mathbf{a}$ asszociációs vektor bitjei a kisebb, tömörített asszociációs vektoron belül egyenletesen szétszórva helyezkedjenek el, csökkentve a véletlen ütközés valószínűségét.

A hash kódolás gyakorlati megvalósítására többféle eljárás alkalmazható [Knu88]. A CMAC hálózatoknál jól használható pl. a polinomosztás. Ha a virtuális címtartomány minden egyes címét, mint egy bináris polinomot képzeljük el, akkor ezen polinom osztása során keletkezett maradék-polinom a tömörített, fizikai címtartomány megfelelő címét reprezentálja. Az osztó polinom természetesen befolyásolja a leképezés tulajdonságait, ezért tetszőlegesen nem választható meg. Ennek részleteire azonban itt nem térhetünk ki, csupán a megfelelő irodalomra, pl. [Pet72] utalunk.

A módszert az alábbi példa illusztrálja.

5.1 Példa

Tételezzük fel, hogy egy 32 megaszó méretű virtuális memóriáról kell a hash kódolás során 64 kiloszavas fizikai memóriára elvégezni a tömörítést. A virtuális memória megcímezéséhez 25 bit szükséges, a fizikai címek megadásához 16 bit elegendő. A polinomosztással történő hashelés tehát azt jelenti, hogy egy 25 bites bináris címet reprezentáló 24-ed fokú polinomot kell elosztanunk egy olyan polinommal, hogy a maradék 16 bites legyen. A virtuális címnek megfelelő polinomot $V(x)$ -szel, a fizikai címnek megfelelő polinomot $P(x)$ -szel és az osztó polinomot $G(x)$ -szel jelölve:

$$P(x) = V(x) \bmod G(x).$$

A 16 bites fizikai címet 15-ödfokú polinommal reprezentálhatjuk, vagyis $G(x)$ fokszáma 16 kell legyen. Legyen az osztó polinom a következő:

$$G(x) = x^{16} + x^{15} + x^{14} + x^{13} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Ezzel pl. a 28.792.695 virtuális cím, amelynek 24-ed fokú polinom reprezentációja:

$$V(x) = x^{24} + x^{23} + x^{21} + x^{20} + x^{18} + x^{17} + x^{16} + x^{14} + x^{12} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^2 + x + 1$$

$G(x)$ -szel történő osztása után a maradékpolinom:

$$P(x) = x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^7 + x^6 + x^5 + x^4 + 1,$$

vagyis a fizikai cím: 28.401.

2.2. A CMAC hálózat fontosabb tulajdonságai

Az eddigiekből látható, hogy a CMAC hálózat az egyéb előrecsatolt hálózatokkal szemben számos előnyös, és mint látni fogjuk számos hátrányos tulajdonsággal is rendelkezik. Ebben az alfejezetben ezeket a tulajdonságokat elemezzük. Előbb megvizsgáljuk a háló approximációs- és általánosító-képességét, majd a háló tanításával kapcsolatos kérdésekkel foglalkozunk.

2.2.1. A háló modellező képessége

A modellező képesség, a hálózat által megvalósítható leképezés tulajdonságainak tárgyalásánál alapvetően két kérdésre keresünk választ: (*) milyen feltételek mellett tudja a CMAC háló *pontosan* megtanulni a tanító pontokat, továbbá, (**) milyen leképezést valósít meg a háló a tanító pontok között, milyen a háló általánosító-képessége.

Fenti kérdésekre adható válaszok bizonyos mértékig eltérnek az eddig tárgyalt (MLP, RBF) hálózatoknál kapott válaszoktól. Az eltérés a CMAC néhány lényeges specialitásából adódik. Az approximációs képesség szempontjából a CMAC hálózat legfontosabb sajátossága, hogy kvantált bemenetekkel dolgozik, vagyis adott kvantumon belül tetszőleges bemeneti érték mellett a hálózat kimenete azonos lesz. A hálózat ebből adódóan szakaszosan konstans kimenetet szolgáltat, vagyis a megvalósított leképezés eleve nem folytonos. A bemeneti kvantálás felbontását elvben tetszőlegesen finomra is választhatjuk, de ennek gyakorlati korlátai vannak. Egyrészt a bemeneti analóg jel kvantálására szolgáló A/D átalakítók felbontása véges, másrészt minél nagyobb felbontást alkalmazunk, annál nagyobb a hálózat súlyainak száma (ld. az (5.50), ill. az (5.52) összefüggéseket), ami különösen többdimenziós bemenetnél realizálhatatlanná teszi a hálózatot. Itt jegyezzük meg, hogy a

bemenet kvantálása nem szükségképpen kell, hogy egyenletes legyen, így azon tartományok, elemi cellák nagysága, amelyekben belül a háló leképezése konstans, a bemenő jeltől függően változhat is [Eld97]. Ugyancsak lehetséges, hogy a kvantálás finomsága dimenzióként eltérő legyen (ld. pl. [Gon98]), vagy több, különböző felbontással dolgozó CMAC háló kombinálásával egy moduláris háló architektúrát alakítunk ki [Moo89].

A továbbiakban a szakaszonként konstans leképezést, mint tény elfogadva azt vizsgáljuk, hogy a lehetséges kvantált diszkrét bemeneti értékekhez milyen kívánt válaszok előállítására alkalmas a hálózat, vagyis létezik-e olyan súlykészlet, amellyel a tanítópontokban előírt, tetszőleges bemeneti-kimeneti kapcsolatot pontosan előállítja a háló. A tanítópontokban adott válaszok mellett fontos kérdés, hogy a tanítópontok közötti pontokban milyen a háló válasza. Az előbbi kérdést, vagyis a háló tanítópontokban adott válaszainak a minősítését a háló approximációs képességének nevezhetjük, míg az utóbbit az általánosító-képességének. Triviális, de megjegyezzük, hogy általánosító-képességgel csak akkor rendelkezik a háló, ha $C > 1$.

2.2.1.1. A háló approximációs- és általánosító-képessége

A háló approximációs képességének vizsgálatánál meg kell különböztetnünk az egydimenziós és a többdimenziós esetet. Egydimenziós esetben tömörítő leképezésre nincs szükség, hiszen itt nincs komplexitásp probléma, ezért a háló leképezését döntően a bemenetről az asszociációs vektorra történő leképezés határozza meg. A háló kimenetét megadó (5.48) összefüggés alapján az összes tanítópontra adott válasz

$$\mathbf{y} = \mathbf{A}\mathbf{w} \quad (5.53)$$

formában adható meg. Az $\mathbf{A}$ asszociációs mátrix a tanítópontokhoz tartozó asszociációs vektorokból, mint sorvektorokból álló $P \times M$ -es mátrix, ahol P a tanítópontok száma, M pedig az asszociációs vektor hossza, amit egydimenziós esetben az (5.50) összefüggés ad meg. Amennyiben a $\mathbf{d}$ vektorral megadott kívánt válaszokat szeretnénk megtanítani, a megoldás súlyvektort az

$$\mathbf{A}\mathbf{w} = \mathbf{d} \quad (5.54)$$

egyenletet kielégítő (vagy azt átlagos négyzetes értelemben közelítő) $\mathbf{w}^*$ adja meg. Az asszociációs mátrix semmiképpen nem lehet négyzetes, mivel $M > P$ még akkor is, ha a bemeneti tér minden lehetséges pontja egyben tanítópont is. Az (5.54) összefüggés tehát egy alulhatározott lineáris egyenletrendszer, melynek végtelen sok különböző megoldása lehet. Tetszőlegesen rögzítve (pl. 0-ra választva) $M-P$ súly értékét, ha $\mathbf{A}$ rangja P , a lineáris egyenletrendszer egyértelmű megoldásra vezet, amit a pszeudo-inverz segítségével határozhatunk meg.

$$\mathbf{w}^* = \mathbf{A}^+ \mathbf{d} = \mathbf{A}^T (\mathbf{A}\mathbf{A}^T)^{-1} \mathbf{d} \quad (5.55)$$

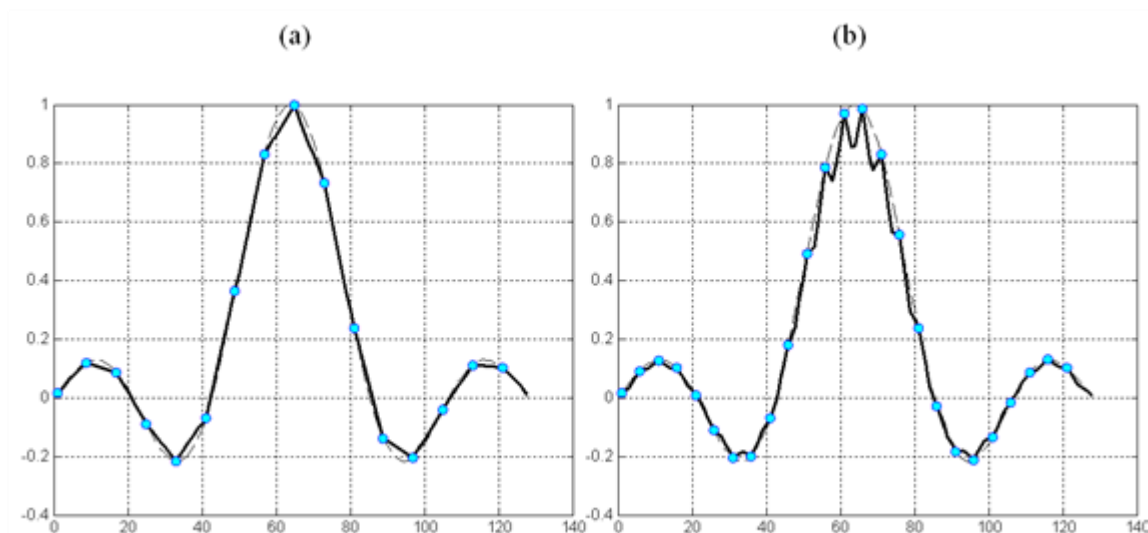
Ez a megoldás azt biztosítja, hogy a tanítópontokban a háló válasza pontosan megegyezzen a kívánt válaszokkal. A háló összes lehetséges bemenetre adott válasza ennek megfelelően az

$$\mathbf{y} = \mathbf{T}\mathbf{w}^* = \mathbf{T}\mathbf{A}^T (\mathbf{A}\mathbf{A}^T)^{-1} \mathbf{d} = \mathbf{T}\mathbf{A}^T \mathbf{B} \mathbf{d} \quad (5.56)$$

formában adható meg, ahol $\mathbf{T}$ az összes lehetséges bemenethez tartozó asszociációs vektorokból képezett $R \times M$ -es mátrix.

A tanítópontoktól eltérő bemenetekhez tartozó válaszok részletes elemzése a $\mathbf{T}\mathbf{A}$ és $\mathbf{B}$ mátrixok vizsgálata alapján lehetséges. Ezek a mátrixok szabják meg, hogy a háló milyen interpolációs képességgel rendelkezik. A mátrixoknak, és különösen a $\mathbf{B}$ mátrixnak az elemzése általános esetben komoly nehézségekbe ütközik. Speciális esetben azonban, amikor a tanítópontok egyenletesen, egymástól pontosan t kvantumnyi távolságra helyezkednek el, a mátrixok analitikusan is felírhatók, és így a háló válasza tetszőleges bemenethez analitikusan is meghatározható. A vizsgálat arra az eredményre vezet, hogy amennyiben a C/t arány egész, a háló a tanítópontok között lineárisan interpolál. Erre mutat példát az 5.12 (a) ábra. Természetesen a kvantálás hatását itt nem tekintjük, a lineáris interpoláció az egyes kvantumokhoz tartozó értékekre vonatkozik.

5.12. ábra - A CMAC válasza $C=8$ és (a) $t=8$, (b) $t=5$ mellett



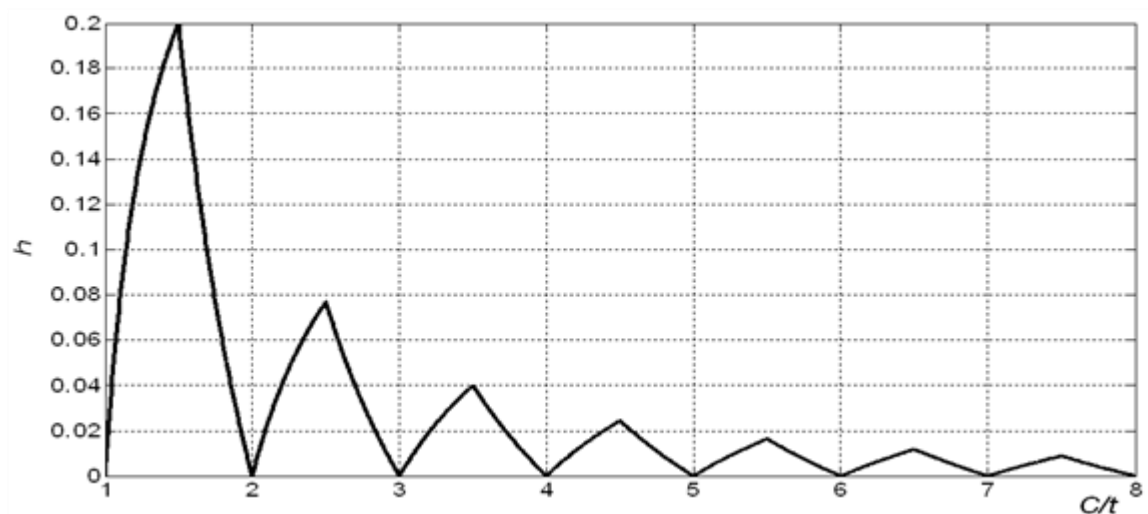
Ennél érdekesebb azonban, hogy egyéb esetekben, tehát ha a C/t arány nem egész szám, akkor a lineáris interpolációhoz képest jelentős eltérés lehet a válaszokban. Ez nyilvánvaló azokban az esetekben, amikor $C < t$, ugyanis a véges tartójú bázisfüggvények miatt ilyenkor található olyan súly, amelyet egyáltalán nem módosítunk a tanítás során. Ha $C < t/2$, az is előfordulhat, hogy egyes bemenetek csak olyan súlyokat választanak ki, melyek egyike sem módosul a tanításnál. Az ilyen bemenetekenél a háló választását a súlyok kezdeti értékei határozzák meg, tehát a válasznak semmi köze a tanítópontok által szolgáltatott információhoz. Az általánosítási hiba ezekre az esetekre analitikusan is könnyen származtatható. A nagy hiba miatt azonban a $C < t$ tartománynak nincs gyakorlati jelentősége. A háló általánosítási képességének vizsgálatából azonban az is kiderül, hogy még akkor is jelentős hiba léphet fel, amikor $C > t$. Ilyen esetet mutat az 5.12 (b) ábra, ahol a $\text{sinc}(\cdot)$ függvény CMAC-val történő approximációja látható $C=8$ és $t=5$ mellett.

Az általánosítási hiba, pontosabban a válaszok lineáris interpolációtól való eltérésének a lineáris interpolációs válaszhoz viszonyított relatív hibája $C > t$ mellett is felírható analitikusan. Ezt az értéket, amit relatív általánosítási hibának nevezünk, a következő összefüggés adja meg [Hor07]:

$$h = \frac{\min[(\text{mod}(C, t))(t - \text{mod}(C, t))]}{\left(2 \left\lfloor \frac{C}{t} \right\rfloor + 1\right) C - \left\lfloor \frac{C}{t} \right\rfloor \left(\left\lfloor \frac{C}{t} \right\rfloor + 1\right) t} \quad (5.57)$$

A fenti összefüggésben $\lfloor \cdot \rfloor$ az egész rész függvény. A relatív általánosítási hiba alakulását mutatja az 5.13 ábra. E szerint a hiba akár 20% is lehet.

5.13. ábra - Az egydimenziós CMAC relatív általánosítási hibája C/t függvényében



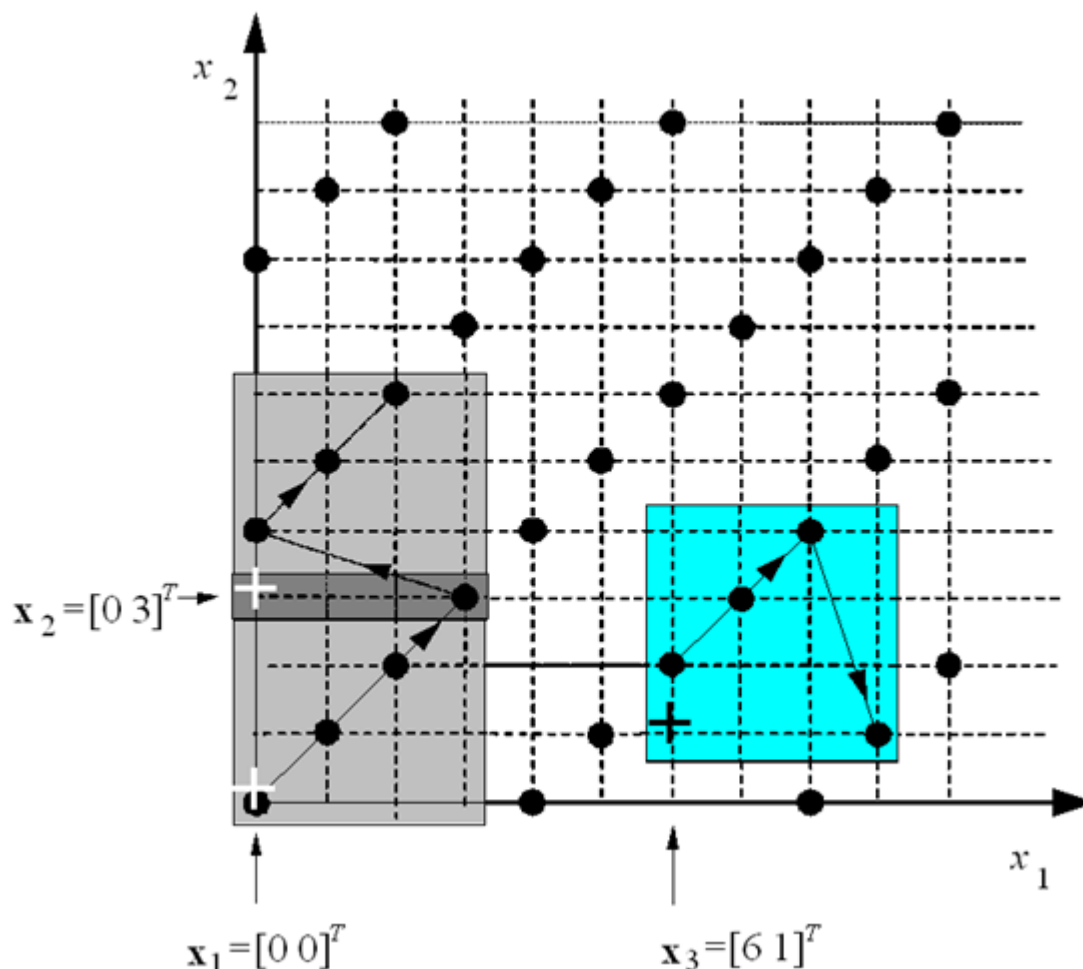
Ez a vizsgálat a tanítópontok egyenletes elhelyezkedése mellett érvényes. A gyakorlatban ritkán találkozunk ilyen esettel, a tanítópontok általában véletlen pozíciókban találhatók a bemeneti térben. A vizsgálat általánosítása tetszőleges pontelrendezésre azt mutatja, hogy kedvezőtlen esetben a relatív hiba még nagyobb is lehet, akár a 33%-ot is elérheti.

Többdimenziós esetben a helyzet bonyolultabb. Az egydimenziós esetnek megfelelő többdimenziós CMAC-t akkor kapnánk, ha a teljes, C^N -szeres lefedést alkalmaznánk. Láttuk, hogy a komplexitás exponenciális növekedése miatt ez legfeljebb 2-3 dimenzióig lehetséges. A komplexitás csökkentését szolgálta a teljes lefedés helyett a bemenetek dimenziójától függetlenül alkalmazott C -szeres lefedés és a tömörítő leképezés. A tömörítő leképezés hatása megfelelően megválasztott tömörítő eljárás mellett közelítőleg elhanyagolható, a pontos hatás elemzése viszont meghaladja a jelen kereteket, ezért ezzel a kérdéssel nem foglalkozunk. (Meg kell azonban jegyeznünk, hogy a tömörítés mindenképpen gyengíti a háló képességeit, ezért ahol csak lehet, célszerű az alkalmazását elkerülni [Wan96], [Zho97]. A későbbiekben, a kernel gépekkel foglalkozó 6. fejezetben be fogjuk mutatni a CMAC olyan értelmezését, mely segítségével a tömörítő hash kódolás elkerülhető, anélkül, hogy a háló komplexitását extrém módon megnövelnénk.)

A háló approximációs képességét többdimenziós esetben is a bemeneti réteg leképezése határozza meg. Itt is biztosítani kell, hogy az egymáshoz közeli bemeneti vektorok részben ugyanazokat a súlyokat válasszák ki, tehát, hogy a megfelelő memóriacímek részben azonosak legyenek. E követelményt teljesítő leképezés egy lehetséges megvalósításának elvét kétdimenziós bemenet esetére az 5.14 ábra mutatja, feltételezve, hogy a hálózat általánosító paramétere $C=4$.

Az ábrán nem a bázisfüggvények elhelyezkedését látjuk, hanem azt, hogy adott bemenet mellett mely bázisfüggvényekhez rendelt súlyok kiválasztása történik meg. A fekete pontok jelölik, hogy a bemeneti értékekhez hogyan rendelhetjük hozzá az asszociációs vektor aktív bitjeit. Minden egyes megjelölt fekete pont (rácspont) egy bitet képvisel az asszociációs vektorban (egy bázisfüggvényt képvisel), tehát ezekhez a pontokhoz kell a címeket hozzárendelnünk. Látható, hogy a kétdimenziós diszkrét mintatérben az egymástól C értékkel eltolt átlók rácspontjait használhatjuk fel. Az eljárás könnyen általánosítható N dimenzióra, ahol a megfelelő N -dimenziós C élhosszúságú hiperkockák testátlóinak rácspontjai képviselik az asszociációs vektor bitjeit. Az árnyékolt területek – melyek kétdimenziós bemeneteknél $C \times C$ -s négyzetek – különböző bemeneti vektorokhoz tartozó összerendelt rácspontokat és így az összerendelt címeket jelölik.

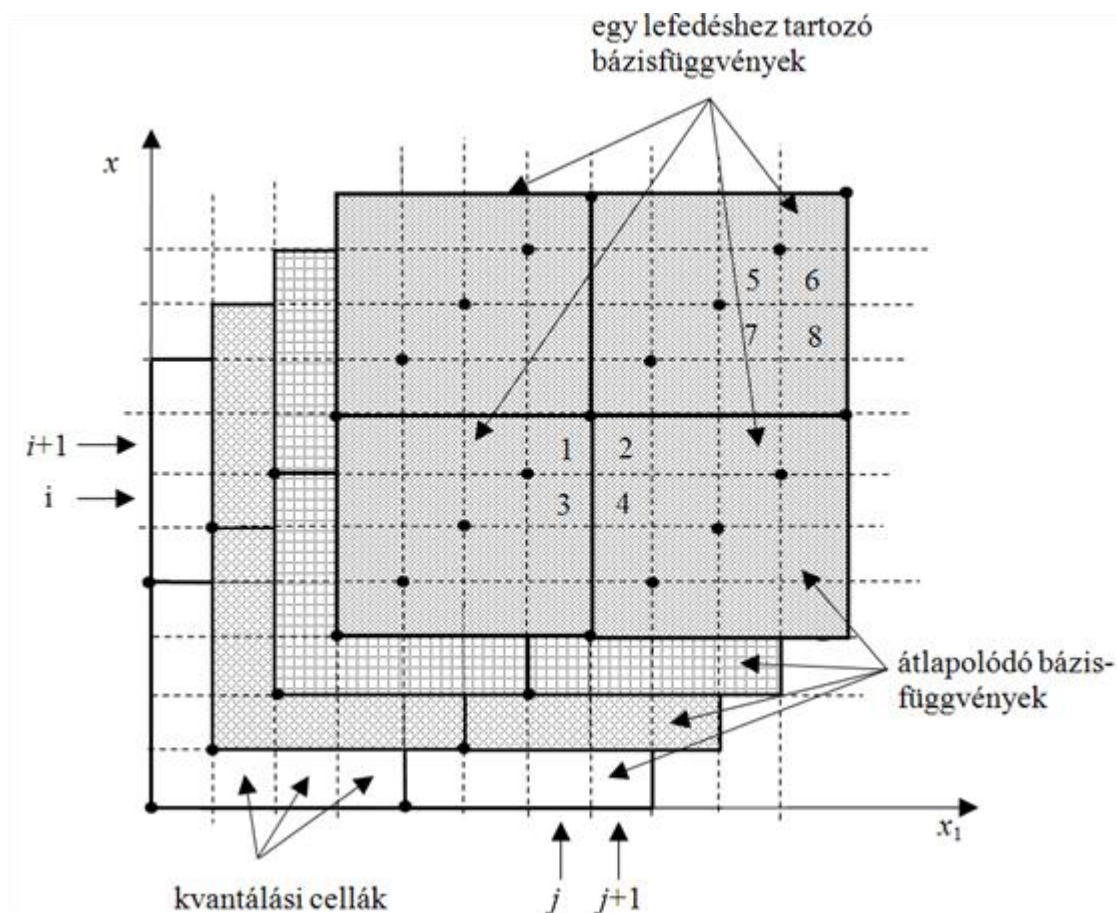
5.14. ábra - A címhozzárendelés elve kétdimenziós esetben



Az egymáshoz közeli bemeneteknél ($\mathbf{x}_1$ és $\mathbf{x}_2$) most is lesznek közös címek, míg a kellő távolságban lévőkénél ($\mathbf{x}_1$ és $\mathbf{x}_3$, illetve $\mathbf{x}_2$ és $\mathbf{x}_3$) ilyeneket nem találunk. Ugyanakkor az is látható, hogy a szükséges címek tartománya csak $C=1$ esetben egyezik meg a lehetséges diszkrét bemeneti vektorok számával, vagyis ilyenkor $\Pi_{i=1}^n R_i$ memóriacímre van szükség, ha R_i az i -edik bemeneti komponens lehetséges értékeinek száma. $C>1$ esetben a címtartomány mérete csökken, minthogy ekkor az összes rácspont helyett csak a főátlóbeli és az ezzel párhuzamos, tőle kC értékkel ($k=1,2,\dots$) eltolt mellékátlóbeli rácspontokhoz rendelünk asszociációs biteket.

Többdimenziós esetben a C -szeres lefedés azt eredményezi, hogy a tanítópontok számától függően az $\mathbf{A}\mathbf{w} = \mathbf{d}$ összefüggés jelenthet mind alulhatározott, mind túlhatározott egyenletrendszert. $\mathbf{A}$ -nak most is P sora van, hiszen P tanítópont áll rendelkezésünkre, az oszlopok számát (az asszociációs vektor hosszát) pedig az (5.52) összefüggés adja meg (M_C). Mivel $M_C < R$ és $P \leq R$, M_C és P viszonya bármi lehet. Az analitikus megoldást, a súlyvektor LS becslését az (5.55) összefüggés definiálja. Alulhatározott esetben most is végtelen sok megoldásunk lehet.

Iteratív tanításnál bármely, az $\mathbf{A}$ mátrix rangjának megfelelő számú tanítópontot tartalmazó megoldás, melyek túlhatározott esetben ($P > M_C$) különbözhetnek egymástól. Az egyes megoldások a megoldástérben egy tartományt jelölnek ki – melyet minimális konvergencia-tartománynak nevezhetünk. Ezen belül bármelyik érték kiadódhat attól függően, hogy mi a tanítópontok elővételi sorrendje és mekkora a tanulási tényező értéke. A tanulási eljárás közben, ha a tanítópontokat ciklikusan vesszük elő, a megoldás ebben a tartományban ciklikusan vándorol, megoldásként határciklust kaphatunk, míg véletlenszerű elővételnél a súlyvektor is véletlenszerű értékeket vesz fel, de a konvergencia-tartományból most sem lép ki. A pontonkénti tanításnál tehát ebben az esetben a súlymódosítás sohasem áll le. Batch tanításnál, ami megegyezik az analitikus LS megoldással, eredményként a minimális konvergencia-tartomány egy pontja adódik [Par92]. Túlhatározott esetben az analitikus LS megoldás és az iteratív LMS alapú megoldás nem feltétlenül azonos. Csak az biztos, hogy mindkét megoldás a minimális konvergencia tartományon belül van.

5.15. ábra - A szomszédos cellák lefedése kétdimenziós bináris CMAC-nál $C=4$ mellett

Többdimenziós esetben a bináris CMAC korlátozott approximációs képességgel rendelkezik. Ennek bemutatásához az 5.15 ábrán látható illusztrációt használhatjuk. Az ábra segítségével azt láthatjuk be könnyen, hogy a szomszédos cellák válaszai nem minden esetben lehetnek tetszőlegesek.

Vizsgáljuk meg a négy számozott elemi cella lefedését. Az 1. ($x_1=j$, $x_2=i+1$) és a 2. cellánál ($x_1=j+1$, $x_2=i+1$) a négy lefedő bázisfüggvény közül három azonos, csupán a negyedikben van eltérés. Ez azt jelenti, hogy a két bemeneti tartományhoz tartozó válaszok különbsége ennek a lefedésbeli különbségnek a következménye. A 3. ($x_1=j$, $x_2=i$) és a 4. ($x_1=j+1$, $x_2=i$) cellák lefedése között ugyanazt a különbséget találjuk, mint az előző két cellánál. Ebből adódóan a számozott cellákhoz tartozó válaszok nem lehetnek tetszőlegesek. A közöttük lévő kapcsolatot az ún. konzisztencia-egyenlet fejezi ki [Bro93]:

$$y(1) - y(2) = y(3) - y(4), \quad (5.58)$$

ahol $y(i)$ az i -edik cella válasza. A konzisztencia-egyenlet szerint a négy szomszédos elemi cella mindegyikéhez nem rendelhetünk tetszőleges kívánt választ; a négy válasz közül három rögzítése megszabja, hogy milyen értéket tud a hálózat előállítani a 4. cellánál. Hasonló kapcsolat található az 5-8 számozású cellák válaszai között is és minden további, hasonló pozíciójú szomszédos cellanégyes között. Ez azt jelenti, hogy az elemi cellák mindegyikénél nem szabható meg a kívánt leképezés, ha azt szeretnénk, hogy a háló pontosan állítsa elő a kívánt válaszokat. A konzisztencia-egyenleteket végigterjesztve a teljes bemeneti tartományon látható, hogy a bemeneti celláknak csak egy, a C értékétől függő töredékén szabható meg a kívánt válasz szabadon. Ezt illusztrálja $C=4$ mellett az 5.16 ábra, ahol, ha az árnyékolt cellánál megszabjuk a kívánt válaszokat, a többi cella által előállítható válaszok kiadódnak.

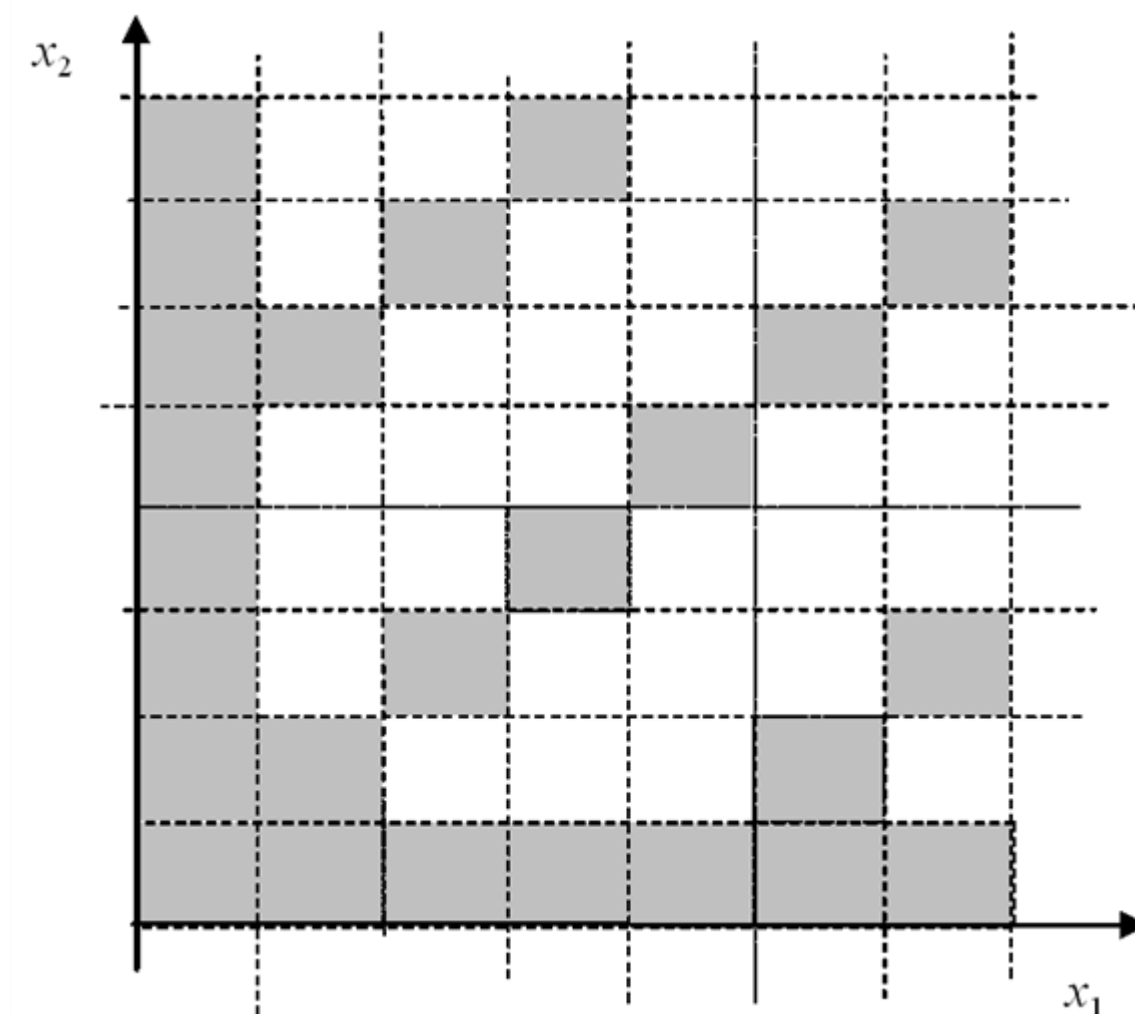
Fentiek alapján látható, hogy a bináris CMAC egydimenziós bemenet mellett – a kvantálásból adódó hatásoktól eltekintve – a tanítópontokat hibátlanul képes megtanulni. Többdimenziós bemeneteknél azonban ez már nem igaz. Minél nagyobb C -t alkalmazunk, annál kevesebb diszkrét bemeneti értékhez rendelhetünk szabadon kívánt választ, annál több megkötést teljesítő leképezést képes csak a hálózat megvalósítani. A konzisztencia-

egyenleteket tetszőleges C érték mellett csak az additív függvényosztály tagjai elégítik ki, vagyis csak olyan N -dimenziós leképezések valósíthatók meg pontosan, amelyek az alábbi formába írhatók:

$$y(\mathbf{x}) = \sum_{i=1}^n y_i(x_i) \quad (5.59)$$

Bár az $y_i(x_i)$ függvények tetszőleges egyváltozós függvények lehetnek, az additív függvényosztályra való korlátozás modellező képesség tekintetében hátrány az univerzális approximátor MLP és RBF hálókhoz képest.

5.16. ábra - A szabadon felvehető (árnyékolt cellák) és a kiadódó (nem árnyékolt cellák) cellánkénti leképezések egy lehetséges elrendezése $C=4$ mellett



A háló által megvalósítható leképezésnek a konzisztencia-egyenletekben megfogalmazott korlátozása a bázisfüggvények számának növelésével gyengíthető, sőt teljes lefedésnél ez a korlátozás meg is szűnik. A bázisfüggvények számának növelése azonban komplexitási okokból erősen korlátozott. A klasszikus Albus-féle CMAC a bázisfüggvényeket az eddigiekben bemutatott módon a bemeneti tér fő- és mellék-átlóbeli rácsponthoz rögzíti, míg a teljes lefedés az összes rácsponthoz rendel bázisfüggvényt. A két határeset között bármilyen közbelső megoldás alkalmazható, tehát az Albus-féle megoldáshoz további bázisfüggvények adhatók. A hozzáadott bázisfüggvények számát elsősorban a komplexitás növekedése korlátozza. A többlet bázisfüggvények elhelyezésére pl. [Lan92]-ban találunk javaslatokat.

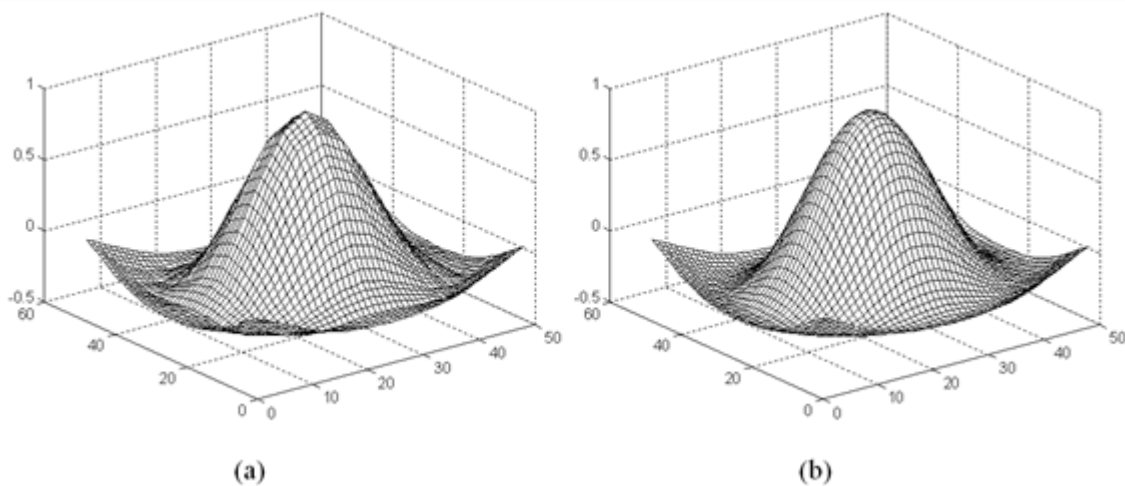
Az approximációs képességet anélkül is javíthatjuk, hogy a bázisfüggvények számát növelnénk. Ha a bázisfüggvények számát nem növeljük, a háló „szabadságfoka” nem nő, így a szabadon felvehető értékű mintapontok száma sem fog nőni. A szabadon felvehető és a kiadódó pontok elhelyezkedését azonban módosíthatjuk, elkerülve azt, hogy a kiadódó válaszok az 5.16 ábrán látható módon a bemeneti térben sávokban

jelentkezzenek. Ez részben a tanításnál a tanítópontok elővételi sorrendjének megváltoztatásával, részben a bázisfüggvények elhelyezésének módosításával érhető el.

A véletlenszerű tanítópont elővételi sorrend az iteratív tanulásnál biztosíthatja, hogy a pontosan megtanulható pontok ne az ábrán látható szabályos elrendezés szerint helyezkedjenek el. A bázisfüggvények elhelyezésére [Bro94]-ben találunk javaslatot. Az eredeti Albus CMAC által alkalmazott lefedések a főátló pontjaihoz, mint referenciapontokhoz vannak rögzítve (ld. az 5.7 és az 5.15 ábrákat). Ha ezeket a referenciapontokat a bemeneti térben véletlenszerűen helyezzük el, a hiba „sávósodása” elkerülhető. Meg kell jegyezzük, hogy ez utóbbi két módszer az approximációs hibát nem szünteti meg, csak a bemeneti téren belül egyenletesebben osztja szét.

Látható, hogy approximációs képesség tekintetében lényeges különbség van az egydimenziós és a többdimenziós CMAC hálók között. Az általánosítási képességet tekintve, viszont többdimenziós esetben is hasonló viselkedést tapasztalunk, mint egydimenziós esetben. Amennyiben a tanítópontok egyenletes t osztástávolságú rácson helyezkednek el és $C/t=l$ egész szám, a tanítópontok közötti lineáris interpoláció adja meg a háló választ. Erre mutat egy példát az 5.17 ábra. Az (a) ábrán a kétdimenziós sinc függvény egy szegmensének CMAC-vel történő approximációját látjuk $C=t=4$ beállítás mellett. Az eredeti függvény a (b) ábrán látható (az ábrán a kvantálás hatásától most is eltekintünk).

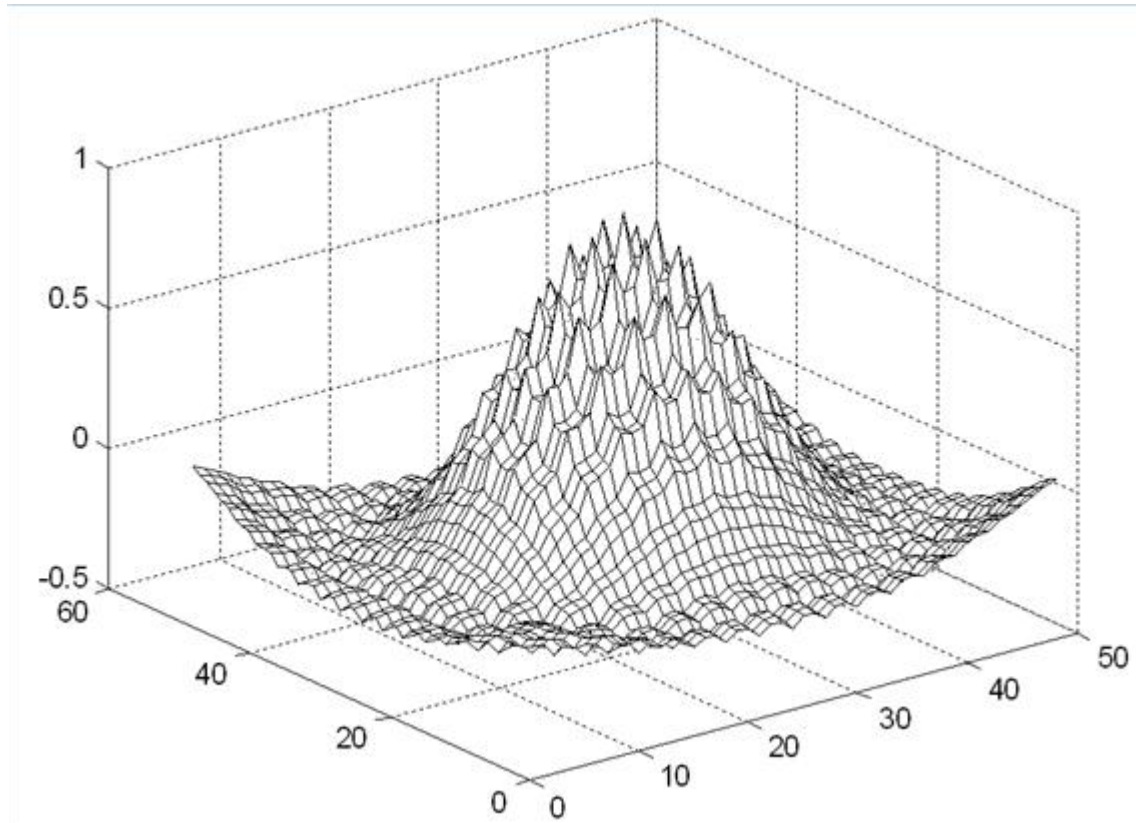
5.17. ábra - A 2D CMAC háló válasza $C/t=1$, $C=4$ mellett (a) és az eredeti függvény (b)



Ha $C/t \neq l$, az egydimenziós esethez hasonlóan igen jelentős általánosítási hiba léphet fel. Az előző feladat megoldása $C=4$ és $t=3$ mellett látható az 5.18 ábrán.

A nagy általánosítási hiba oka – ha C/t arány nem egész szám – az, hogy a súlyok nagyon különböző értéket vesznek fel még akkor is, ha a kívánt válaszok minden tanítópontban azonosak. A különböző értékek azért alakulnak ki, mert az egyes súlyoknál a módosítások száma eltérő. A súlymódosítások száma minden súlynál csak akkor lesz azonos, ha C/t egész szám. Ezt egydimenziós esetben könnyű belátni. Tekintsük azt az esetet, amikor $C/t=1$. Mivel a C tartójú bázisfüggvények miatt minden tanítópont C szomszédos bázisfüggvényt aktivizál és tanításnál az ezekhez tartozó súlyokat módosítjuk, nem lesz olyan súly, melyet több tanítópont is ki fog választani. Minden lépésben más és más C súly módosítására kerül sor úgy, hogy a tanítópontok egyszeri felhasználása során minden súlyt egyszer és csakis egyszer módosítunk. Mivel az egy tanítóponthoz tartozó C súly módosítása azonos mértékű (ld. az LMS algoritmus megfelelő összefüggését), ha a kezdeti értékek azonosak voltak, az egy tanítópont által kiválasztott C súly azonos értékű lesz egy teljes tanítási ciklus, epoch után is. Hasonló súlyeloszlást kapunk, ha $C/t=l$, ahol $l \geq 2$ egész. A különbség mindössze annyi, hogy most minden súlyt pontosan l -szer fogunk módosítani. Ezzel szemben, ha a C/t arány nem egész, a módosítások száma egyes súlycsoportoknál eltérő lesz, ami eltérő súlyértékeket is eredményez.

5.18. ábra - A CMAC háló válasza $C=4$ és $t=3$ mellett



A fenti hatás mérséklése lehetőséget ad a nagy általánosítási hiba csökkentésére. Olyan módon kell módosítani a tanulási eljárást, hogy a tanítópontokban fellépő hiba redukciója mellett azt is biztosítsuk, hogy az egy tanítópontokhoz tartozó súlyok minél inkább egyformák legyenek. Az eljárást a következő részben mutatjuk be.

2.3. A CMAC háló tanítása

Mivel a CMAC csak a kimeneti lineáris rétegben tartalmaz tanítható súlyokat, ezek meghatározása az előzőek szerint vagy analitikusan, vagy iteratív eljárással, leggyakrabban az LMS algoritmus alkalmazásával történhet. A hálózat speciális kialakítása miatt azonban még az LMS algoritmus is egyszerűsödik.

A hálózatban a kimeneti réteget kivéve csak bináris értékek találhatók, a bemeneti nemlineáris leképezést megvalósító rész tisztán digitális hálózat: a kvantált bemenetek bináris reprezentációjából az asszociációs vektor ritka bináris reprezentációját állítja elő. Mivel a kimeneti réteg bináris bemenetekkel dolgozik, tényleges szorzásra a hálózatban nincs is szükség. A súlymódosítás az LMS algoritmus szerint

$$w_i(k+1) = w_i(k) + \mu \varepsilon(k) \quad \forall i: a_i(k) = 1, \quad (5.60)$$

ahol $a_i(k)$ a k -adik bemenethez tartozó asszociációs vektor i -edik bitje. Tehát az asszociációs vektor aktív bitjei által kiválasztott C súlyt azonos mértékben kell módosítani. A módosítás mértéke pedig a hiba, ε valahányad, célszerűen kettő negatív egész hatványad része. Így tehát nemcsak a háló kimeneti értékének kiszámításához – a visszahívási fázishoz –, hanem a tanításhoz sem szükséges szorzás. Ez a tulajdonság a hálózatot különösen alkalmassá teszi a digitális VLSI megvalósításra, lehetővé téve így a nagysebességű parallel implementációt [Mil91].

További előny a CMAC hálózat ún. inkrementális tanulási képessége. Ennek oka, hogy hasonlóan az RBF hálózathoz a tanítópontok hatása csak korlátozott bemeneti tartományra terjed ki, a hálózat lokális általánosító-képességgel rendelkezik. Míg azonban az RBF hálónál valószínűleg nem lokális tartójú bázisfüggvényeket használunk, itt ténylegesen véges (C kvantum oldalhosszúságú hiperkocka) a bázisfüggvények tartója. Egy tanítóponttól nagyobb távolságra a leképezés kialakításában e pontnak nincs hatása, vagyis az egymástól távolabbi bemeneti tartományokhoz tartozó válaszok egymástól függetlenül alakíthatók ki. Egy tanító lépésben a kimeneti réteg súlyainak csak a töredéke – maximum C súly – módosul, vagyis az egymástól távol lévő tanítópontok a hálózat súlyainak eltérő, egymástól független csoportjaira hatnak, szemben a globális

általánosító-képességgel rendelkező többrétegű perceptronnal, ahol minden tanító lépésben az összes súly módosul.

A hálózat lényeges jellemzője a viszonylag gyors tanulás. A gyorsaság azt jelenti, hogy hasonló méretű feladat MLP és hibavisszaterjesztéses eljárás feltételezésével csak nagyságrendekkel több tanító lépés alatt oldható meg, mint CMAC hálózattal. A nagyobb sebesség részben a lokális tanulás következménye – ezért általában viszonylag kevés iterációs lépésre van szükség –, részben abból adódik, hogy egyszerű műveletekre van csak szükség. A hálózat leggyorsabban akkor tanítható, ha módunk van a tanítópontok olyan megválasztására, hogy az összes tanítópont egyszeri felhasználásával minden súlyt módosítsunk, de mindegyiket csak egyszer. Az előbbieken láttuk, hogy ez egydimenziós bemenetnél akkor érhető el, ha a tanítópontok pontosan C kvantumnyi távolságban vannak egymástól, így az egyes tanítópontok mindig másik C elemű súlykészletet módosítanak. Ebben az esetben nemcsak az érhető el, hogy a súlyokat egy tanítási ciklusban ki tudjuk alakítani, hanem az is, hogy a hálózat válasza minden tanítópontban megegyezik a kívánt válasszal, a tanítópontok közötti közbenső bemeneti értékekre adott válaszok pedig a szomszédos válaszok közötti lineáris interpoláció eredményeként adódnak, ahogy ezt az általánosítóképeség vizsgálatánál láttuk. A tanítópontok egyszeri felhasználása – egyetlen epoch – akkor ad hibátlan eredményt, ha megfelelő tanulási tényezőt alkalmazunk. Ez a fenti tanítópont elrendezésnél $\mu=1/C$ mellett lehetséges.

A tanulási szabály módosítása, súlykiegyenlítő regularizáció

Az általánosítási hiba okainak elemzése azt mutatta, hogy a hiba csökkentése lehetséges, ha a tanítás során törekszünk arra, hogy az egyszerre kiválasztott súlyok minél inkább azonosak legyenek. Ezt úgy érhetjük el, hogy a kritériumfüggvényt kiegészítjük egy olyan taggal, mely ezt a járulékos feltételt kényszeríti rá a súlyokra. A megoldás egy regularizációs eljárás, ahol a kritériumfüggvény a szokásos négyzetes tag mellett egy regularizációs tagot is tartalmaz:

$$C(\mathbf{w}) = \frac{1}{2} \left(d(k) - y(k) \right)^2 + \frac{\lambda}{2} \sum_{i=1}^C \left(\frac{d(k)}{C} - w_i(k) \right)^2 \quad (5.61)$$

Látható, hogy a regularizációs tag azt fogalmazza meg, hogy az egy bemenethez tartozó súlyok értékei minél inkább a d/C értéket vegyék fel, ahol d az adott bemenethez tartozó kívánt válasz. Azonos súlyértékeket általános esetben nyilván nem kaphatunk, hiszen ekkor a tanítópontokban kapnánk nagy hibát. A két tag egyensúlyát a λ regularizációs együttható megfelelő megválasztásával biztosíthatjuk.

A fenti kritériumfüggvénynek megfelelő iteratív, gradiens alapú tanító eljárás a következő:

$$w_i(k+1) = w_i(k) + \mu \varepsilon(k) + \lambda \left(\frac{d(k)}{C} - w_i(k) \right) \quad (5.62)$$

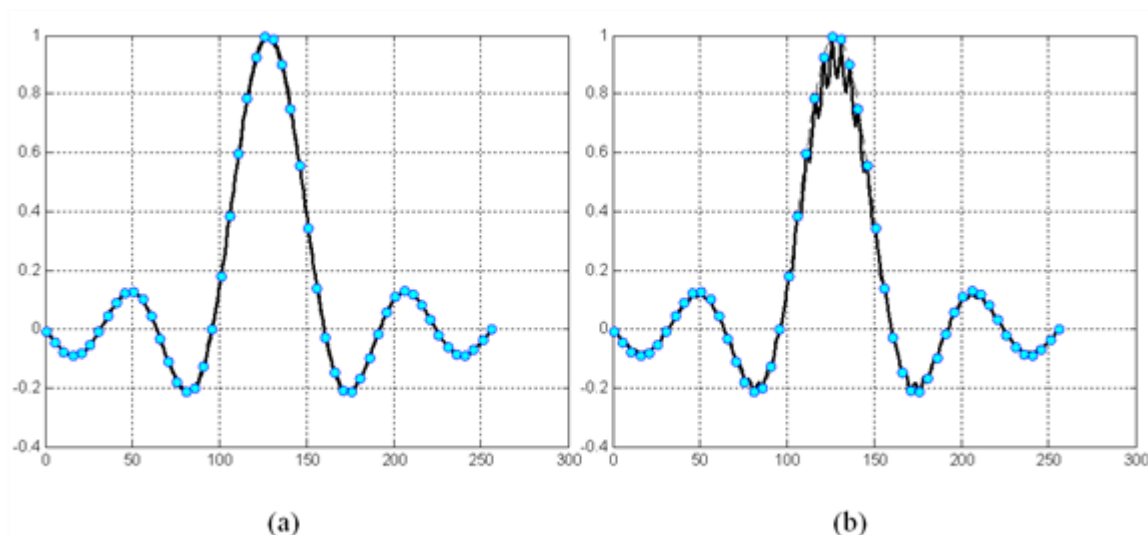
A kapott tanulási szabályt súlykiegyenlítő, vagy súlysimító regularizációs szabálynak nevezzük, ahol a két tag arányát μ és λ megválasztásával állíthatjuk be. A regularizációs eljárás hatása az 5.19 és az 5.20 ábrán látható.

Az előbbi egy egydimenziós, az utóbbi egy kétdimenziós sinc függvény esetén mutatja a háló választát $C=4$, $t=3$ értékekre. Az 5.19 (a) ábrán a regularizált változat válasza látható; az összehasonlításához a (b) ábrán az eredeti Albus változat választát is megadtuk. Látható, hogy regularizált esetben a válasz nemcsak sokkal simább, hanem az approximáció átlagos négyzetes hibája is jelentősen csökkent.

Az 5.20 ábra kétdimenziós mintapéldája is azt mutatja, hogy az 5.18 ábrán látható jellegzetes CMAC válasz helyett sima választ kapunk.

A súlykiegyenlítő regularizációnak az általánosítási hibát csökkentő hatását mutatja az 5.21 ábra, ahol a C/t arány függvényében az approximáció átlagos négyzetes hibáját adjuk meg az eredeti és a regularizált tanítás mellett az egydimenziós sinc függvény tanításánál. A hiba – ahogy vártuk – a kétféle tanító eljárás mellett a C/t arány egész értékeinél azonos, köztük viszont a regularizált esetben sokkal, egyes tartományokban akár több nagyságrenddel is kisebb. Sőt az is látszik, hogy regularizált esetben még akkor is jobb megoldást kaphatunk, mint regularizáció nélkül, ha közben a tanítópontok számát csökkentjük (C/t arányt növeljük). Az ábra konkrét esetre adja meg az approximációs hiba értékét, az eredmény azonban jellegre általánosan is érvényes.

5.19. ábra - A súlykiegyenlítő regularizáció hatása egydimenziós esetben: a háló válasza (a) regularizált esetben, (b) regularizáció nélkül

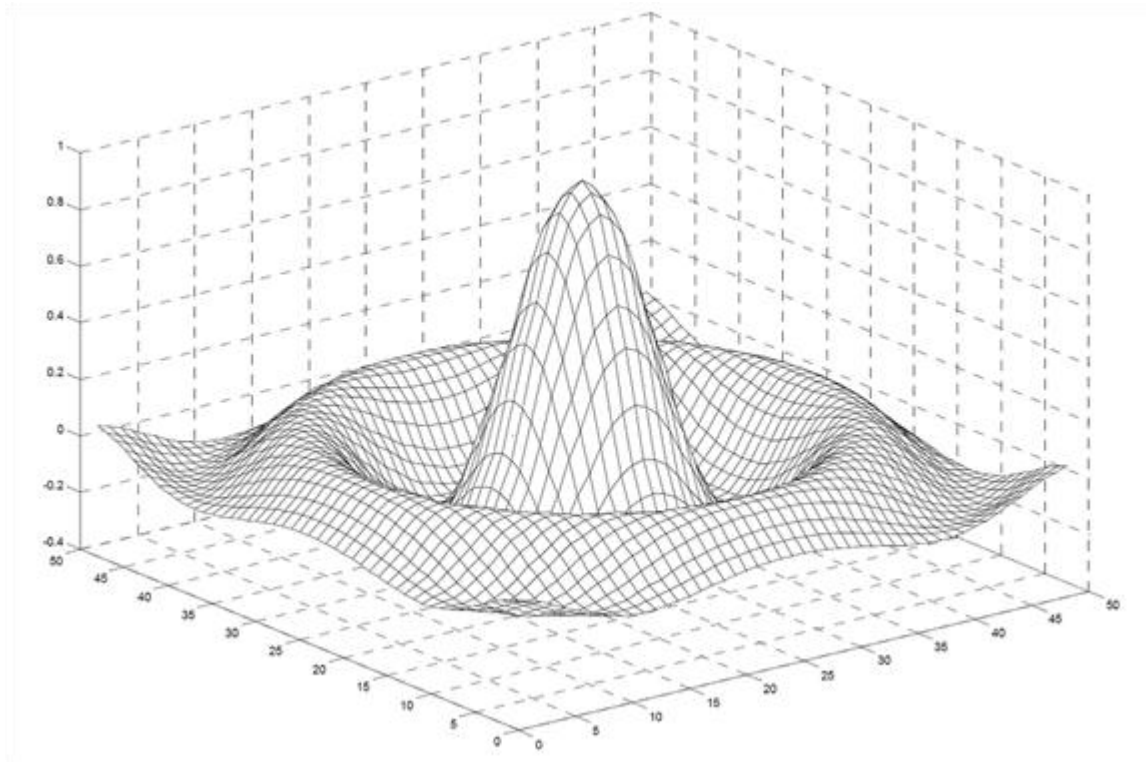


A regularizáció hatását zajos tanítópontok mellett mutatja az 5.22 (a) ábra. Összehasonlításként az 5.22 (b) ábrán a klasszikus Albus CMAC válasza látható azonos tanítópont-készlet és háló paraméterek mellett. Amint az várható az Albus CMAC-nál a tanítópontokban a válasz hibátlan, ami zajos esetben kimondottan káros. A tanítópontokra való tökéletes illeszkedés – ahogy ezt az előzőekben már bemutattuk – egydimenziós esetben az eredeti CMAC alaptulajdonsága a szabad paraméterek nagy száma miatt. A szabad paraméterek számának korlátozása, vagy valamilyen simasági mellékfeltétel figyelembevétele a háló eredő approximációs képességeit javítja.

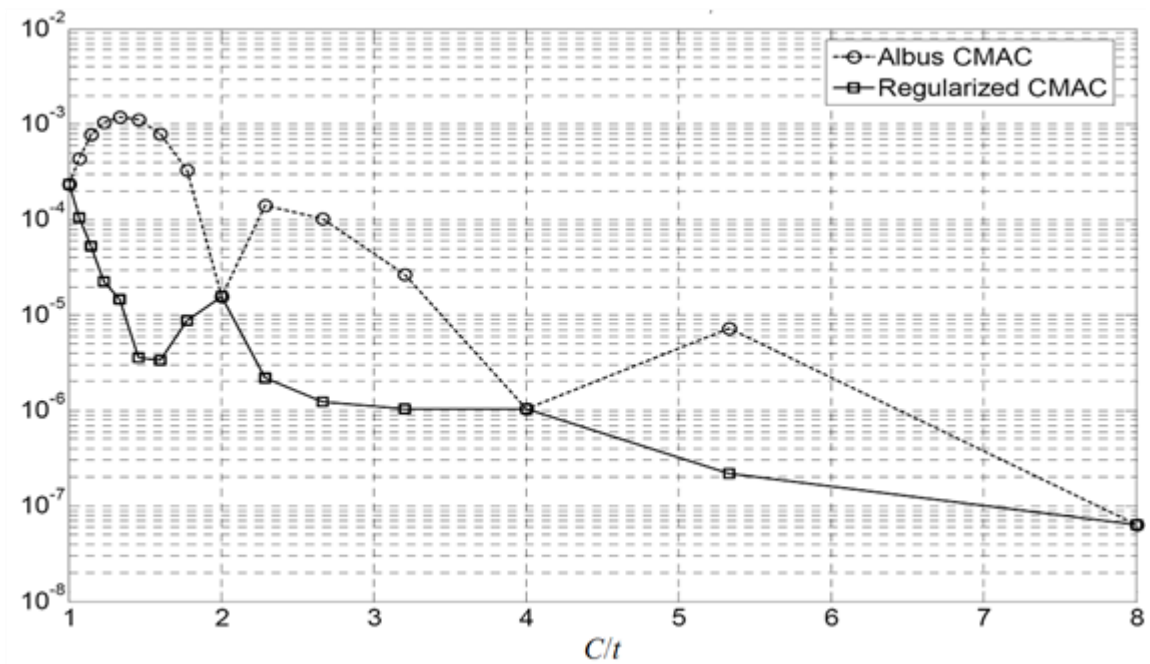
2.4. CMAC változatok

Az elmúlt több mint 30 évben a CMAC hálózatoknak igen sok változata alakult ki. Ezek a változatok olyan módosítások eredményeként születtek, melyek egyrészt a háló képességeinek javítását, másrészt a többdimenziós eset extrém nagy komplexitásának csökkentését célozzák. A következőkben előbb az ún. magasabbrendű CMAC hálózatokkal foglalkozunk, majd néhány olyan strukturális változtatást mutatunk be röviden, melyek a súlyok számának drasztikus csökkentését úgy próbálják elérni, hogy közben a háló teljesítőképessége ne romoljon.

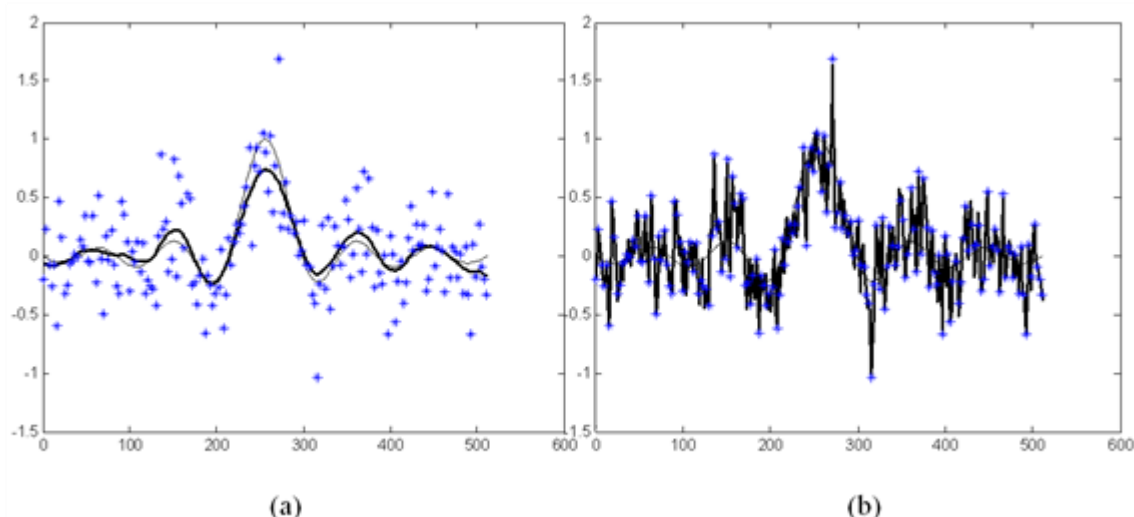
5.20. ábra - A súlykiegyenlítő regularizáció hatása kétdimenziós esetben



5.21. ábra - A CMAC átlagos négyzetes hibája C/t függvényében



5.22. ábra - Zajos tanítópontok mellett a CMAC válasza. (a) regularizációval, (b) regularizáció nélkül



2.4.1. Magasabbrendű CMAC hálózatok

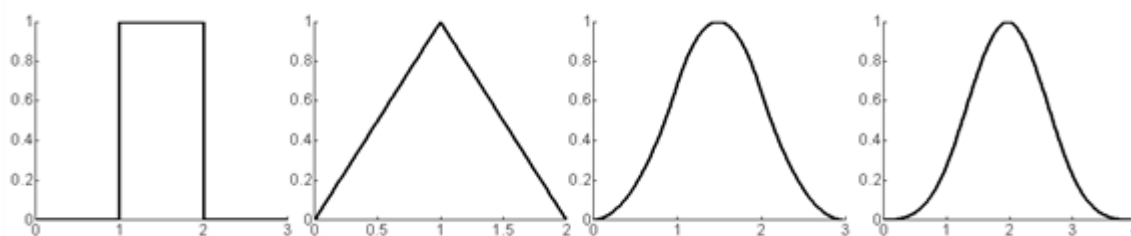
Az eredeti Albus-féle CMAC hálózat bináris bázisfüggvényeket használ, vagyis ezen bázisfüggvények értéke bármely olyan bemenetre, amely az érzékelési mezőbe esik 1, míg az összes többi bemenetre 0. A bináris bázisfüggvény következménye az egyszerű realizálhatóság – a szorzó nélküli struktúra –, de részben ennek a következménye a korlátozott modellező képesség is. Magasabbrendű CMAC hálózathoz a négyzetes függvény helyett lineáris, kvadrátikus, stb. bázisfüggvényeket használunk. Az egyváltozós lineáris bázisfüggvény a következő:

$$g(x) = \begin{cases} 1 - 2 \left| \frac{x-t}{C} \right| & \text{ha } |x-t| \leq \frac{C}{2} \\ 0 & \text{egyébként} \end{cases}, \quad (5.63)$$

ha t a C szélességű bázisfüggvény középpont paramétere. Ezek a bázisfüggvények a B-spline függvénycsalád elemeinek is tekinthetők. A bináris bázisfüggvény az elsőrendű, a lineáris a másod-, a négyzetes a harmad-, stb., és a $(k-1)$ -ed fokú polinommal létrehozott a k -ad rendű B-spline függvény. Az 5.23 ábra $k=1, 2, 3, 4$ esetre mutatja az első néhány B-spline függvényt. A B-spline függvények előnye a könnyű előállíthatóság és az, hogy egy k -ad rendű B-spline $(k-1)$ -szer deriválható.

A k -ad rendű B-spline bázisfüggvény az elsőrendű B-spline függvény önmagával vett k -szoros konvolúciója útján állítható elő. Ha a bemeneti tartományt kvantáljuk, a kvantált intervallumokhoz hozzárendelt B-spline függvényértékek a következő rekurzív összefüggések alapján is számíthatók:

5.23. ábra - Egyváltozós k -ad rendű B-spline bázisfüggvények $k=1-4$ esetekre



$$B_k'(x) = \frac{x - \lambda_{j,k}}{\lambda_{j+1} - \lambda_{j,k}} B_{k-1}'(x) + \frac{\lambda_j - x}{\lambda_j - \lambda_{j,k} + 1} B_{k-1}'(x)$$

$$B_k'(x) = \begin{cases} 1 & \text{ha } x \in I_j \\ 0 & \text{egyébként} \end{cases} \quad (5.64)$$

ahol $I_j = (\lambda_{j-1}, \lambda_j]$ a j -edik bemeneti intervallum.

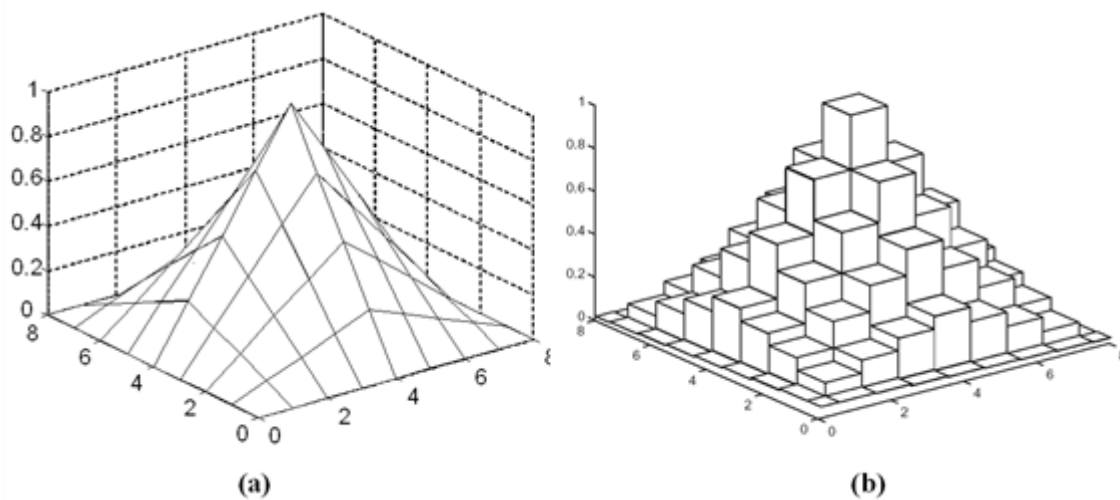
N -változós bázisfüggvény az egyváltozós függvények szorzataként nyerhető:

$$g(\mathbf{x}) = \prod_{i=1}^N g_i(x_i) \quad (5.65)$$

A lineáris bázisfüggvényt (másodrendű B-spline-nak megfelelő) kétdimenziós esetre az 5.24 (a) ábra mutatja.

Bár a magasabbrendű bázisfüggvények alkalmazása a modellező képességet javítja, a kvantált bemenet miatt a háló által megvalósított leképezés szakaszosan konstans tulajdonsága megmarad, ugyanis egy bemeneti cellán (kvantumon) belül, bármely bemeneti értékhez egyetlen asszociációs vektor és ennek megfelelően konstans válasz tartozik. A modellező képesség azon hiányossága, hogy a háló a kvantált bemenetek miatt a folytonos bemenet függvényében szakaszosan konstans leképezést valósít meg sok esetben nem jelent hátrányt, hiszen a bemeneti mintáink úgyis digitalizált formában állnak rendelkezésre (pl. digitális szabályozási feladatok esetén).

5.24. ábra - Kétdimenziós elsőrendű bázisfüggvény $C=8$ és $t=(4,4)$ paraméterekkel, (a) folytonos változat, (b) kvantált változat

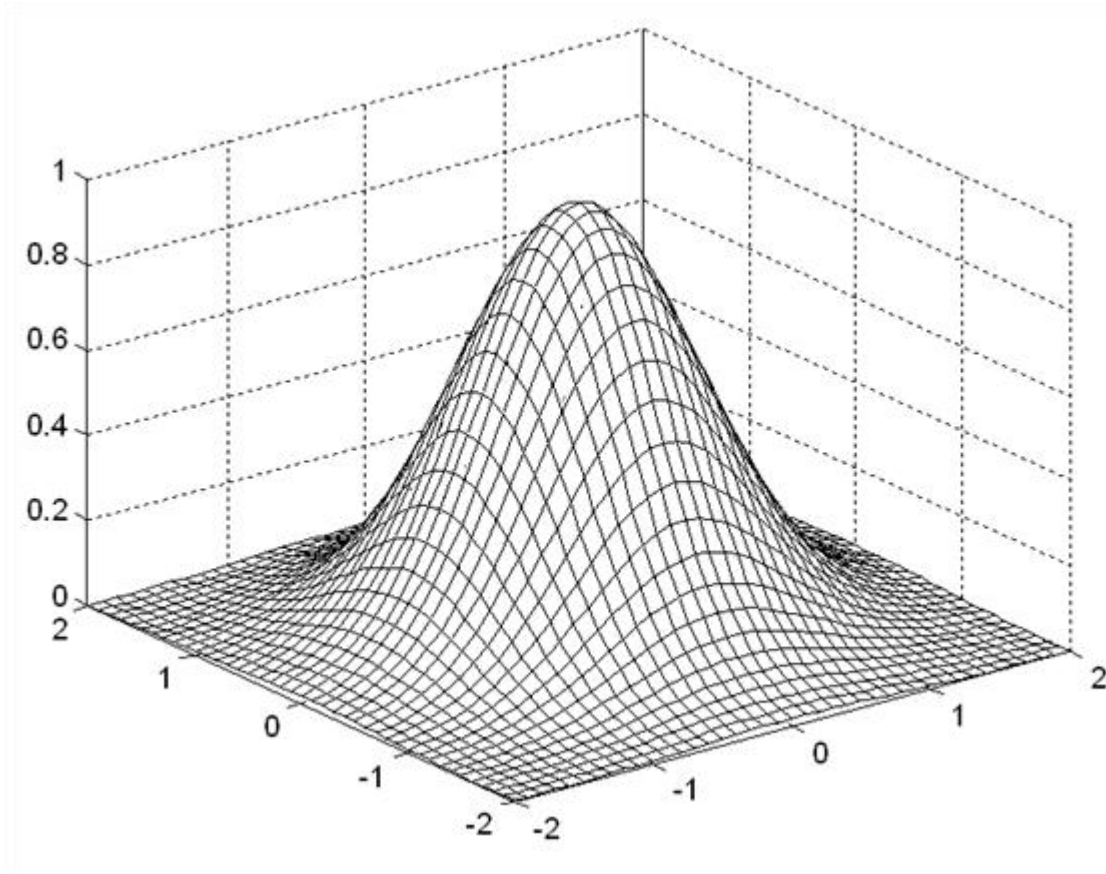


Ekkor a ténylegesen alkalmazott bázisfüggvény az 5.24 (a) ábrán látható helyett a (b) ábrán bemutatott lépcsős függvény lesz. Ennek ellenére már az elsőrendű bázisfüggvény alkalmazása is biztosítja, hogy a CMAC háló ne csak az additív függvényosztály elemeinek megvalósítására legyen alkalmas.

Amennyiben mégis analóg bemenetekkel dolgozunk, a kvantálás hatásának kiküszöbölésére több lehetőségünk is van. A magasabbrendű bázisfüggvények alkalmazása lehetővé tenné a folytonos leképezés megvalósítását akkor, ha egy bemeneti cellához nem konstans függvényértéket, vagyis nem lépcsős függvényt rendelnénk. Ez azt jelenti, hogy pl. kétdimenziós bemenet és lineáris bázisfüggvény mellett valóban az 5.24 (a) ábrán látható bázisfüggvénnyel dolgoznánk. Az így kapott B-spline [Lan92] bázisfüggvényekkel a folytonos leképezés megvalósítható, sőt nemcsak a függvényértékek, hanem a függvény bizonyos deriváltjai is előállíthatók: k -ad rendű B-spline bázisfüggvények alkalmazásával a $(k-1)$ -edik deriváltig bezárólag a függvény differenciálhányadosai is approximálhatók. E megoldás fő nehézsége a bázisfüggvényértékek meghatározásának nagyobb számításigénye, valamint az, hogy a bináris CMAC hálók egyik legfontosabb előnyét, nevezetesen, hogy sem a tanítási, sem a visszahívási fázisban nem igényelnek szorzást elveszítjük. A lépcsős bázisfüggvények mellett jóval kisebb a számításigény és jóval hatékonyabb hardver struktúra alakítható ki, mint a folytonos függvények alkalmazásakor [Hor98], [Ker95]. Alkalmazástól függ, hogy a nagyobb számításigényű magasabbrendű, esetleg folytonos változat, vagy az egyszerűbb, de korlátozott képességekkel rendelkező eredeti változat mellett döntünk.

Folytonos leképezés lépcsős bázisfüggvények mellett a kvantálásnál szokásos egyenletes eloszlású additív zaj (*dither*) [Lip92] alkalmazásával valósítható meg. Ennek ára, hogy visszahívási fázisban egy bemenetet nem egyszer, hanem többször, különböző additív zaj értékek mellett kell a hálózatra adni és a kimenetek átlaga adja a tényleges választ.

5.25. ábra - Kompakt tartójú Gauss bázisfüggvény CMAC hálózhoz, $C=4$.



A magasabbrendű változatok között megtalálható a Gauss függvényes megoldás is, amely egy olyan k -ad rendű B-spline bázisfüggvényes hálónak felel meg, ahol k elég nagy. Ugyanis k növelésével a B-spline függvények tartanak a Gauss függvényhez; $k > 6$ esetén a közelítés már igen jó. Gauss bázisfüggvényekkel a CMAC az RBF-hez hasonló, azzal az eltéréssel, hogy megtartjuk a CMAC-ra jellemző felépítést, vagyis a bázisfüggvények elhelyezési rendszerét és azt, hogy a bázisfüggvények véges tartójúak. Gauss-CMAC hálónál a tartó most is egy C kvantumnyi oldalhosszúságú hiperkocka (5.25 ábra).

2.4.2. Kredit hozzárendelés

A háló képességeit a tanuló eljárás módosításával is javíthatjuk. A kredit-hozzárendeléses tanításnál egy tanító lépésben a kiválasztott C súly módosítása nem azonos mértékű. Az egyes súlyokhoz hozzárendelt kredit attól függ, hogy az adott súlyt előzőleg hányszor módosítottuk. Minél többször módosítottunk egy súlyt, annál kisebb mértékben fogjuk a további tanítási ciklusokban módosítani. A CA-CMAC tanítási összefüggése:

$$w_i(k+1) = w_i(k) + \mu \frac{(f_i+1)^{-1}}{\sum_{i=1}^M (f_i+1)^{-1}} \varepsilon(k) \quad (5.66)$$

ahol f_i az i -edik súly eddigi módosításainak száma, az $(f_i+1)^{-1} / \sum_{i=1}^M (f_i+1)^{-1}$ összefüggés pedig az i -edik bázisfüggvényhez (a w_i súlyhoz) rendelt kredit érték.

A CA-CMAC eljárás [Su03] hatása részben hasonló a súlykiegyenlítő regularizáció hatásához. Egyrészt simább választ eredményez, másrészt a tanítást gyorsíthatja is.

2.4.3. Komplexitás csökkentés többdimenziós esetben

A többdimenziós CMAC esetenként extrém komplexitású lehet, annak ellenére, hogy a válaszok számításához csak a súlyok tárolására és egyszerű műveletek elvégzésére van szükség. A szükséges memória mérete azonban megvalósíthatatlanul nagy is lehet (a dimenzió átká probléma), ezért ennek redukálása az alkalmazhatóság előfeltétele. Az Albus által javasolt hash kódolás ugyan jelentősen csökkenti a súlymemória méretét, de az

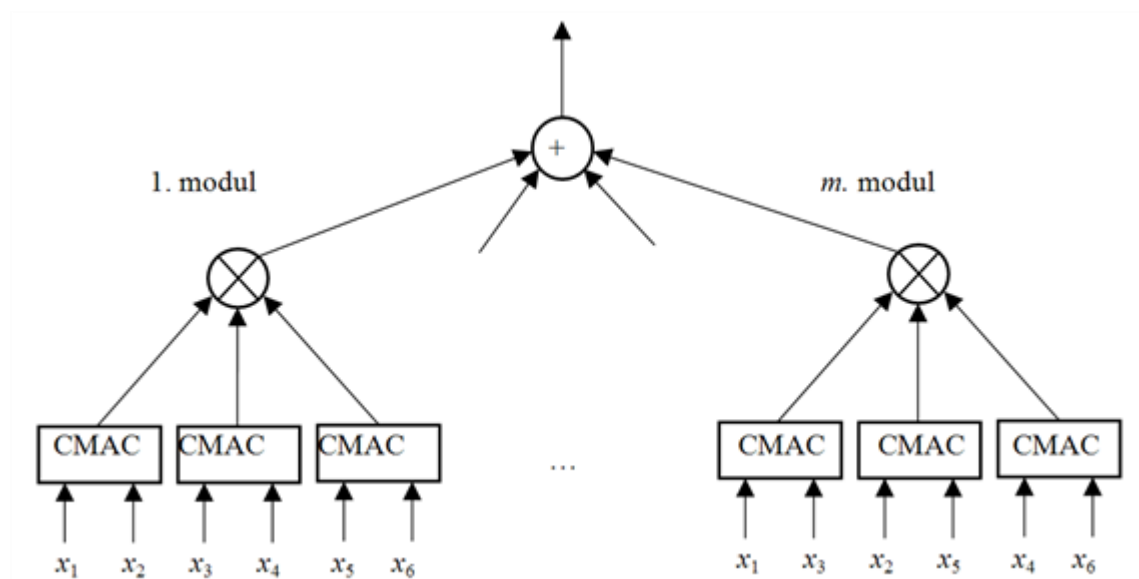
említett ütközések a tanítás konvergencia tulajdonságait rontják, annak ellenére, hogy az ütközések valószínűsége igen kis értékre szorítható [Wan96], [Zho97].

Hash kódolás helyett a súlymemória méretének csökkentésére több lehetőségünk van. Egyrészt egy N -változós CMAC dekomponálható kisebb dimenziós CMAC hálózatokra, melyek eredményeit valamilyen módon összesíteni kell, másrészt a CMAC-nál is alkalmazható az ún. kernel trükk, mely a problémát a bázisfüggvények teréből egy ún. kernel térbe transzformálja. A kernel térben a probléma komplexitása dimenziótól függetlenül a tanítópontok száma által korlátozott. Az így született kernel-CMAC-t a kernel gépeknél, a 6. fejezetben mutatjuk be. A kisebb-dimenziós hálózatokra történő dekomponálás fő eredményeit itt foglaljuk össze.

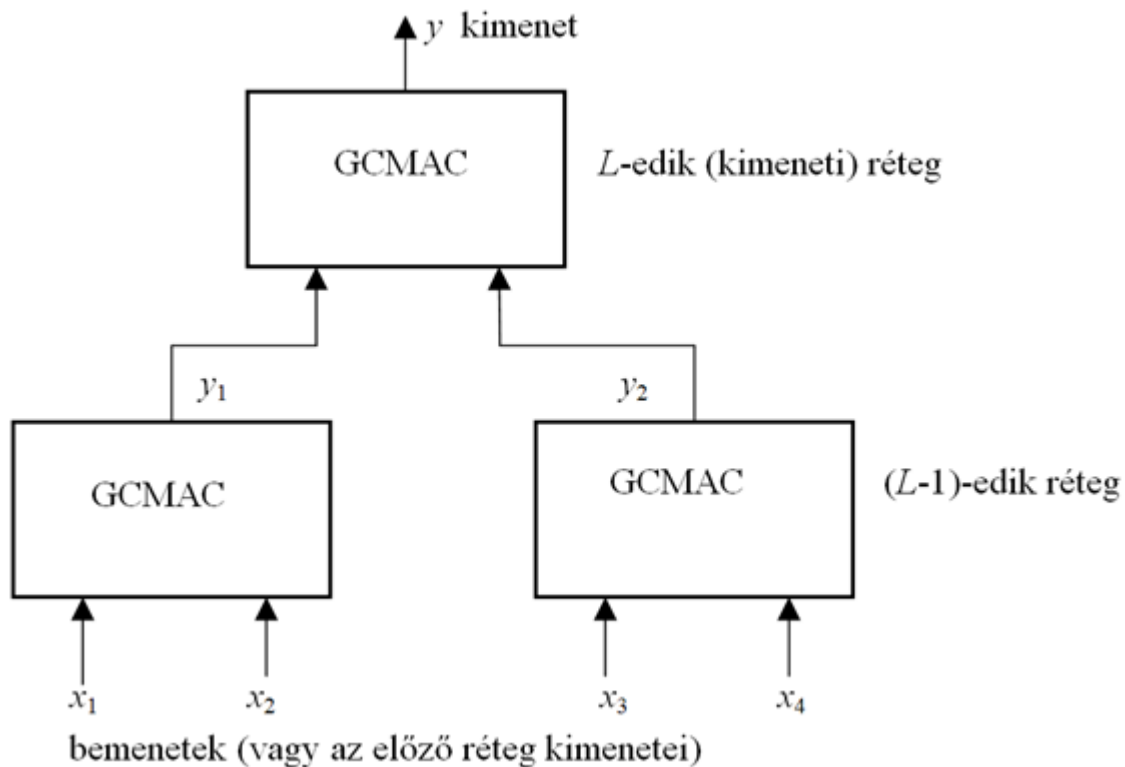
Egy N -változós CMAC komplexitását igen hatékonyan redukálhatjuk, ha dekomponáljuk egy- vagy kétváltozós hálókra, majd ezek eredményeit egyesítjük, hogy az eredő háló az eredeti N -változós feladat megoldását adja. A dekomponálásra több lehetőségünk is van. Ezek egyike az ún. szorzatok összege (*sum-of-product*, SOP) struktúra, amely egy olyan moduláris architektúra, ahol egy- vagy kétváltozós hálók kimeneteinek szorzata képezi egy modul kimenetét.

A teljes hálót több ilyen modul összegeként kapjuk meg. A modulok számát a tanulási eljárás részeként határozhatjuk meg fokozatos bővítés útján. Ha adott számú modul összegeként kapott eredő válasz hibája még nem elegendően kicsi, további modult alakítunk ki, melyet szintén integrálunk az eddigi hálózatba. Ha egyváltozós elemi CMAC hálókból építkezünk, minden modulban mind az N bemeneti változóhoz rendelünk egy egydimenziós hálót. Ha az elemi hálók kétdimenziósak, minden modul más-más kétdimenziós CMAC hálók szorzatát képezi [Lin96], [Li04]. A szorzatok összege struktúra kétdimenziós elemi hálókbal az 5.26 ábrán látható.

5.26. ábra - A szorzatok összege (*sum-of-product*, SOP) struktúra



5.27. ábra - A HCMAC felépítése kétdimenziós Gauss CMAC hálókbal



A HCMAC háló ugyancsak kétváltozós elemi hálókat alkalmaz, és egy hierarchikus bináris fa struktúrát épít fel [Lee03]. Itt az elemi kétváltozós hálók kimeneteit is kétváltozós CMAC hálók dolgozzák fel tovább, tehát egy többszintű hálóarchitektúra jön létre. N -változós bemenetek mellett $\log_2 N$ hierarchia szintre van szükség. Az egyes kétváltozós CMAC hálók Gauss bázisfüggvényekkel dolgoznak, ahol a bemeneti teret folytonosnak tekintjük. A Gauss bázisfüggvények elrendezése ettől függetlenül követi az Albus-féle bázisfüggvény elrendezést, ahogy ezt az 5.15 ábra mutatja. A hierarchikus architektúra miatt minden egyes rétegben találunk tanítandó paramétereket, a kétdimenziós CMAC súlyait. Ezek tanítása a hibavisszaterjesztéses algoritmus alkalmasan módosított változatával lehetséges. A háló architektúráját az 5.27 ábra adja meg. Az ábrán GCMAC a Gauss bázisfüggvényekkel dolgozó hálókat jelöli.

A dekompozíció egy más formáját alkalmazza az MS-CMAC háló [Hun99], [Hun01]. Az N -változós hálót itt úgy bontjuk fel egy egyváltozós hálóból építkező hierarchikus architektúrára, hogy az egyes hálókat egymástól függetlenül lehessen tanítani. A háló az egydimenziós elemi CMAC hálók esetében megtartja az eredeti Albus-féle felépítést, úgy, hogy egy hierarchiaszinten az N változóból csak egynek az értéke változhat, míg a maradék $N-1$ értéke rögzített. Ez azt jelenti, hogy az első hierarchia szinten, a bementi rétegnél csak egy változó, pl. x_1 a valódi bemeneti változó, és annyi egydimenziós hálót kell felépíteni, ahány rögzített értéket a tanítópontokban felvehet a maradék $N-1$ változó: $x_2, \dots, x_N$. Láthatóan ez a megoldás kihasználja, hogy a bemenetek kvantáltak, de a kvantálás nem lehet nagyon finom, különben az egydimenziós hálók száma nagyon nagy lehet. Pl. egy 3-bemenetű hálónál, ha a tanítópontoknál az egyes bemeneti komponensek csupán 10-10 diszkrét értéket vehetnek fel, az első rétegben 100, a másodikban 10, a harmadikban pedig 1 egydimenziós CMAC hálóra van szükség.

Bár összességében az így megvalósított N -dimenziós CMAC jóval kevesebb súlyt igényel, mint az eredeti verzió, nagy bemeneti változószám és finomabb felbontás mellett ez is elfogadhatatlanul nagy memória-komplexitású megoldásra vezet. Ráadásul további korlátozásoknak kell eleget tenni: a háló csak akkor tanítható, ha a tanítópontok egy szabályos rács pontjain helyezkednek el. A háló válasza bármely diszkrét érték mellett meghatározható, de ha a tanítópontok közötti bemenetekre adott válaszokat akarjuk megkapni, a bemeneti réteg kivételével a többi réteg elemi CMAC hálójánál visszahíváskor be kell iktatni egy tanítási fázist is.

3. Az MLP és a bázisfüggvényes hálózatok összehasonlítása

Az MLP és a bázisfüggvényes hálók alapvetően hasonló feladatok megoldására alkalmasak, ezért célszerű, ha a főbb jellemzők alapján egy rövid összehasonlítást teszünk. Mindkét hálósztályba tartozó hálók előrecsatolt,

többrétegű architektúrával rendelkeznek és mindkét hálócsoport a bemenet(ek) és a kimenet(ek) között nemlineáris memória nélküli leképezést valósít meg. A hasonlóság mellett azonban lényeges különbségek is vannak, melyek figyelembevétele fontos lehet a gyakorlati alkalmazásoknál.

A két hálóosztály összehasonlításánál a következő főbb szempontokat célszerű figyelembe venni. Ezek:

- a háló felépítése,
- a hálók approximációs képessége,
- a hálók általánosító-képessége,
- a megoldás egyértelműsége,
- a háló tanításának jellege,
- a tanulás és a működés sebessége,
- a háló számítási- és memória-komplexitása.

A hálók felépítését tekintve a legfontosabb különbség, hogy az MLP paramétereiben nemlineáris, míg a bázisfüggvényes hálók alapvetően paramétereiben lineáris leképezést valósítanak meg. Ennek oka, hogy a bázisfüggvényes hálók kimeneti rétege lineáris és többnyire csak ez a kimeneti réteg tartalmaz tanítható paramétereket.

Az approximációs- és általánosító-képesség tekintetében a fő különbség nem az MLP és a bázisfüggvényes hálók között van, hanem a bázisfüggvényes hálók egyes típusai között. Láttuk, hogy mind az MLP, mind egyes bázisfüggvényes hálók (pl. az RBF) univerzális approximátorok. Ehhez az MLP aktivációs függvényeinek, illetve a bázisfüggvényeknek meghatározott tulajdonságokkal kell rendelkezniük. Az univerzális approximátor képesség azt jelenti, hogy nemcsak a tanítópontokban, hanem a teljes értelmezési tartomány minden pontjában biztosítható a tetszőlegesen kis hibájú közelítés. Az univerzális approximátor képesség tehát az általánosító-képességre is vonatkozik.

A bináris CMAC ezzel szemben – bár felépítése az RBF-hez hasonló – nem tekinthető univerzális approximátornak. Az előzőekben láttuk, hogy a gyakorlatban is megvalósítható CMAC komplexitás mellett az approximációs- és az általánosító-képesség külön vizsgálendő, sőt külön vizsgálandók az egy- és a többdimenziós hálózatok tulajdonságai is. Általánosságban elmondható, hogy mind a leképezés jellege, mind az általánosítási hiba tekintetében a bináris CMAC hálók elmaradnak az MLP és RBF hálóktól. Ugyanakkor azt is láttuk, hogy a CMAC hálózatoknak léteznek olyan változatai, melyekkel gyakorlatilag hasonló minőségű approximáció biztosítható. Így elmondható, hogy a felmerülő nehézségek ellenére a CMAC háló akár az MLP reális alternatívája is lehet.

Az általánosító-képesség tekintetében azonban az MLP és a bázisfüggvényes hálózatok többsége között lényeges különbség van. Míg az MLP globális általánosító-képességgel rendelkezik, a bázisfüggvényes hálózatokra a lokális általánosítás a jellemző. Ez egyértelműen igaz a CMAC hálókra, de praktikusán igaz az RBF hálókra is. Pontosabban mindazon hálókra igaz, ahol a bázisfüggvények véges vagy gyakorlatilag véges tartójúak. A bázisfüggvényes hálónál a véges tartó következménye, hogy az approximáció jellege eltérő még az MLP és az RBF hálóknál is. Általában az MLP válaszai simábbak, mint az RBF válaszai, és az MLP kevésbé hajlamos a túltanulásra, ahogy ezt az MLP-re vonatkozó 4.7 és az RBF approximációját mutató 5.3 ábrák illusztrálják. Azt is mondhatjuk, hogy az MLP implicit regularizációval rendelkezik. A lehetséges végtelen sok megoldás közül inkább a sima megoldásokat részesíti előnyben. A túltanulást illetően a CMAC nemcsak, hogy hajlamos a túltanulásra, hanem a nagyszámú szabad paraméter miatt – különösen egydimenziós esetben – a tanítópontokat akár hibátlanul is megtanulja, még akkor is, ha azok nagymértékben zajosak (ld. 5.22 (b) ábra). A megfelelő általánosító-képesség biztosításához a CMAC hálónál a regularizációnak nagyon fontos szerepe van.

A véges tartó következménye a lokális tanulás és az ún. inkrementális tanulás is. A lokális általánosító-képesség azt jelenti, hogy egy tanítópontnak csak egy szűk környezetre van hatása. Az ehhez kapcsolódóan jelentkező inkrementális tanulási képesség pedig azt, hogy az egymástól távol lévő tanítópontokat a háló egymástól közel függetlenül tanulja meg. Így a megtanult ismeret általában nem degradálódik, ha újabb tanítópontok formájában újabb ismeretet kívánunk a hálónak megtanítani, feltéve, hogy az újabb tanítópontok kellően távol vannak a már megtanított pontoktól. Az MLP nem rendelkezik az inkrementális tanulás

képességével. Minden tanítópont a háló összes súlyát módosítja, ezért az újabb ismeret leronthatja a már megszerzett ismeretet, hacsak a régebbi tanítópontokat nem használjuk fel ismételten az újabb pontok tanításánál is.

A megoldás egyértelmősége tekintetében is lényeges különbségeket találunk az MLP és a bázisfüggvényes hálók között. Az MLP paraméterekben nemlineáris leképezése miatt a hibafelület nem kvadratikusságú. Ez még akkor is igaz, ha az MLP-nél is lineáris kimeneti réteget alkalmazunk. A felületen létezhetnek lokális minimumok, sőt a globális minimum sem egyértelmű: a súlyok megfelelő permutációjával különböző, de egyenértékű megoldásokhoz juthatunk. A lokális minimumok, mint láttuk a háló tanításánál (a gradiens alapú tanuló eljárásoknál) okozhatnak komoly nehézséget. A bázisfüggvényes hálók, amennyiben csak a kimeneti lineáris réteget tanítjuk, négyzetes hibafelülettel rendelkeznek, ami biztosítja, hogy nincsenek lokális minimumok. A paraméterekben lineáris struktúra biztosítja azt is, hogy a megoldás akár analitikusan is meghatározható.

A tanítás jellegét vizsgálva az látható, hogy az MLP-nél analitikus megoldást nem kapunk, a tanítópontok egyenkénti vagy kötegelte felhasználásával juthatunk el a megoldáshoz. A bázisfüggvényes hálónál ezzel szemben mind analitikus, mind iteratív megoldás nyerhető.

A tanulás sebességét nézve a CMAC háló lehet a leggyorsabb. Ennek oka a bemeneti nemlineáris réteg által megvalósított egyszerű leképezés és a véges bázisfüggvény tartó, amiből adódik, hogy egy tanító lépésben a súlyok töredékét módosítjuk csak. Visszahíváskor szintén csak a súlyok töredékét kell meghatároznunk, továbbá a bináris Albus-féle hálóban sem szorzásra, sem nemlineáris függvény (aktivációs függvény, nemlineáris bázisfüggvény) számítására nincs szükség.

A legkevésbé konkrét megállapítások a hálók komplexitásában tehetők. Bár jelenleg erre vonatkozó elméleti eredmény legjobb tudomásunk szerint nincs, adott feladat megoldásánál nagy valószínűséggel az MLP igényli a legkevesebb szabad paramétert. Ez annak ellenére igaz, hogy nem tudjuk meghatározni a rejtett neuronok (szükséges és elégséges) minimális számát, ha pl. adott pontosságú függvényapproximáció a feladat. Ugyanakkor az RBF hálónál a bázisfüggvények számának felső korlátja meghatározható, hiszen ez a tanítópontok számával megegyezik. A CMAC esetében a bázisfüggvények száma a háló felépítésből és a C paraméter értékéből egyértelműen kiadódik, és ahogy láttuk, ez extrém nagy is lehet. A CMAC háló, tehát annak ellenére, hogy nagyon egyszerű építőelemekből áll, többdimenziós esetben a nagy memóriaigény miatt speciális megoldásokat igényel.

Feladatok

5.1 Konstruáljon RBF hálót, amely megoldja a két egymásba tekeredő spirál szétválasztását. Vizsgálja meg, hogy hogyan befolyásolja a háló eredményét a σ szélességparaméter értéke. A kettős spirál mintakészletet az alábbi MATLAB kód felhasználásával generálja:

```
function [xy, dr] = TwoSpiral()

xy_A=[];

d_A=[];

xy_B=[];

d_B=[];

for i=0:96

angle = i * pi /16;

radius = 6.5 * (104 - i)/104;

x = radius * sin(angle);

y = radius * cos(angle);

xy_A= [xy_A; x, y];

d_A = [d_A; 1];

xy_B = [xy_B; -x, -y];
```

```
d_B = [d_B; -1];
end;
xy= [xy_A; xy_B];
dr = [d_A; d_B];
save two_spiral_1 xy dr;
plot(xy_A(:,1),xy_A(:,2), '+');
hold on;
plot(xy_B(:,1),xy_B(:,2), 'o');
hold off;
```

5.2 Oldja meg a paritásproblémát (N -bites bemeneti minták páros és páratlan paritású csoportokra való szétválasztása) RBF hálózattal. Vizsgálja meg, hogy a bázisfüggvények száma és elhelyezése, valamint a σ szélességparaméter hogyan befolyásolják a megoldást.

5.3 Egy RBF hálózaton a rejtett rétegbeli neuronok paramétereinek, a bázisfüggvények középpont- és szélesség-paramétereinek meghatározását is ellenőrzött tanítással, gradiens módszerrel kívánjuk végezni. Határozza meg a tanítási összefüggéseket, ha Gauss bázisfüggvényeket használ.

5.4 A

$g(r) = (c^2 + r^2)^{1/2}$ $c > 0, r \geq 0$ multikvadratikus függvény

és a

$g(r) = (c^2 + r^2)^{-2}$ $c > 0, r \geq 0$ inverz multikvadratikus függvény

két lehetséges radiális bázisfüggvény választás. Az inverz multikvadratikus függvényt alkalmazó RBF háló nemlineáris leképezések lokális approximációját végzi. Ezzel szemben a multikvadratikus függvény az RBF lokális approximációs működésére ellenpélda. Igazolja ezt a két állítást.

5.5 A XOR feladat nemlineárisan szeparálható feladat. Mutassa meg, hogy megfelelően megválasztott középpontokkal az RBF rejtett rétegének kimenetén a bemeneti mintapontok olyan transzformált értékeit nyerjük, melyek már lineárisan szeparálhatók lesznek. Mutassa meg, hogy 2 rejtett elem elegendő a lineáris szeparálhatóság biztosításához, ha eltolás-értéket is használunk. Konstruáljon olyan RBF hálót a feladat megoldására, ahol nincs eltolás tag. Hány bázisfüggvénnyel tudja megoldani a feladatot?

5.6 Mutassa meg, hogy az additív függvényosztály kielégíti a bináris CMAC konzisztencia egyenleteit függetlenül attól, hogy az általánosító paraméter C értéke mekkora.

5.7 Határozza meg, hogy mi az a függvényosztály, melyet az elsőrendű, a másodrendű, stb. CMAC hálózatok képesek modellezni.

5.8 Határozza meg, hogy milyen választ ad a CMAC háló a C távolságra lévő tanítópontok között négyszögletes, lineáris illetve magasabbrendű bázisfüggvények alkalmazásánál.

5.9 Mutassa meg, hogy nulla kezdeti értékből kiindulva tömörítés nélküli esetben a CMAC tanítása konvergens.

5.10 Képes-e megtanulni egy bináris (elsőrendű, másodrendű) CMAC az $y = \sin(x_1) \sin(x_2)$ kétváltozós függvényt? Mi a helyzet az RBF illetve az MLP hálózatokkal? Konstruáljon egy-egy megoldást a három különböző hálózattal és hasonlítsa össze az eredményt az approximáció minősége, a háló mérete és a tanítás sebessége szempontjából.

5.11 Hogyan tudná a CMAC hálózaton a kvantálásból adódó szakaszonként konstans leképezést folytonos leképezéssé módosítani?

5.12 Oldja meg a négydimenziós sinc problémát a szorzatok összege (SOP) architektúrával, ha a szorzat modulokban kétdimenziós CMAC hálókat használ. Mekkora lesz a kapott háló memória komplexitása? Hasonlítsa össze ezt a komplexitást egy ugyanilyen felbontású bináris CMAC komplexitásával.

5.13 Hasonlítsa össze a súlysimító regularizáció és a kredit hozzárendelés hatását a CMAC háló válaszára.

6. fejezet - Kernel módszerek

Ebben a fejezetben olyan tanuló rendszerekkel foglalkozunk, amelyek a válaszokat ún. kernel-függvények (vagy magfüggvények) súlyozott összegeként állítják elő. A megközelítés bizonyos rokonságot mutat a bázisfüggvényes hálózatokkal, hiszen ott is függvények súlyozott összegeként kapjuk a megoldást, azonban a származtatás a kétféle megközelítésnél jelentősen eltér. A bázisfüggvényes hálónál a bázisfüggvények által megvalósított nemlineáris leképezés a mintapontokat a bemeneti térből egy ún. jellemzőtérbe képezi le, és a megoldást a jellemzőtérbeli reprezentáció felhasználásával nyerjük. A jellemzőtérre való leképezés célja, hogy egy eredetileg csak nemlineáris eszközzel megoldható problémát lineáris eszközzel megoldhatóvá transzformáljunk: egy nemlineárisan szeparálható problémát lineárisan szeparálhatóvá tegyünk, vagy egy nemlineáris regressziós feladatot lineáris regressziós feladattá transzformáljunk.

A bázisfüggvényes hálózatok és ezen belül is elsősorban az RBF hálók konstrukciójánál az egyik leginkább megoldatlan kérdés a jellemzőtér dimenziójának, vagyis a bázisfüggvények számának a meghatározása. Annyit tudunk csak biztosan, hogy ha a jellemzőtér dimenziója „kellően nagy”, akkor pl. a lineáris szeparálás lehetősége garantált. Azonban az általában nem definiált, hogy mit tekinthetünk „kellően nagy”-nak. Ha a jellemzőtér dimenzióját „túl nagy”-ra választjuk, akkor egyrészt feleslegesen növeljük a tanuló rendszer komplexitását, másrészt a szabad paraméterek száma túl nagy lesz, aminek túltanulás és a teljesítőképesség romlása is lehet a következménye. Olyan megoldásra volna szükségünk, ahol a jellemzőtér dimenziója „automatikusan kiadódik” nemcsak azt biztosítva, hogy lineáris megoldást kapunk, hanem azt is, hogy a megoldás hibája, a 2. fejezetben definiált valódi kockázat, a lehető legkisebb lesz – függetlenül attól, hogy hány dimenziós a jellemzőtér. Az ún. kernel gépek ezt a célt próbálják elérni.

A kernel gépeknél is alkalmazzuk a jellemzőtérbe való transzformációt, a feladatot azonban nem a jellemzőtérbeli reprezentáció felhasználásával oldjuk meg, hanem erről áttérünk egy ún. kernel reprezentációra és a kernel térben kapjuk a megoldást. A jellemzőtérbeli és a kernel térbeli reprezentáció közötti kapcsolat egy újabb transzformációként is értelmezhető. Míg a jellemzőtérben valójában bázisfüggvényes megoldással dolgozunk, addig a kernel térbeli reprezentáció a jellemzőtérbeli reprezentációból belső szorzattal nyerhető.

A kernel térbeli reprezentáció azzal az előnnyel jár, hogy itt a szabad paraméterek száma független a jellemzőtérbeli reprezentáció szabad paramétereinek számától. Ezt az előnyt ki is tudjuk használni, ha a belső szorzatot egy megfelelően megválasztott kernel függvénnyel helyettesítjük, vagyis ahelyett, hogy adott bázisfüggvényeket definiálnánk és ezek skalár szorzatát felhasználva állítanánk elő a kernel reprezentációt, közvetlenül a kernel függvényt választjuk meg. A kernel függvény megválasztása implicit módon meghatározza a jellemzőtérbeli leképezés bázisfüggvényeit és így a jellemzőtérbeli reprezentációt is, anélkül, hogy a bázisfüggvényeket közvetlenül kellene definiálni és használni.

A fejezet célja, hogy bemutassa a kernel módszerek alap gondolatát és a legfontosabb kernel gépek konstrukciójának a lépéseit. A bemutatásnál a legegyszerűbb, lineáris feladatból kiindulva jutunk el a nemlineáris problémák kernel géppel történő megoldásáig. A fejezet azt is bemutatja, hogy kernel reprezentációs megoldásra több, különböző megközelítés vezethet. Ezek közül talán a legfontosabb az elsősorban Vladimir Vapnik nevéhez köthető szupport vektor gép (*support vector machine*, SVM). A szupport vektor gépek azon túl, hogy a kernel térben szolgáltatják a megoldást, további fontos tulajdonságokkal is rendelkeznek. Az SVM-nél – az eddig bemutatott hálóktól eltérően – a megoldás komplexitása és teljesítőképessége kézen tartható. A fejezet ugyanakkor azt is bemutatja, hogy kernel gépek származtathatók a klasszikus négyzetes hiba-minimalizáló eljárással is, tehát bizonyos értelemben az eddig tárgyalt neuronhálóknak is létezhetnek kernel változatai. Ehhez kapcsolódóan a fejezet tárgyalja az LS-SVM hálózatokat és az ún. ridge regressziós (*ridge regression*) megoldást is.

A kernel gépek elméleti háttere szorosan kapcsolódik a belső szorzat terek (*inner product spaces*, *dot product spaces*) és a Hilbert terek (*Hilbert spaces*), témakörökhöz. A fejezet ezzel a kapcsolattal csak utalás szinten foglalkozik, a témakör iránt mélyebben érdeklődők a bőséges irodalomban, pl. [Sch02], [Vap98], [Sző72] tájékozódhatnak.

A fejezet felépítése a következő. Először bemutatjuk az irodalomban gyakran kernel trükknek nevezett módszert, ami lehetőséget ad a kernel függvények közvetlen felírására, alkalmazására. Ezt követően bemutatjuk a legismertebb és legelterjedtebb kernel módszert alkalmazó eljárást, a szupport vektor gépeket (*Support Vector Machines* – SVM), valamint ennek néhány változatát. A szupport vektor gépek jelentős számításigényű eljárások. A számítási igények csökkentésére ezért az SVM megközelítésétől néhány ponton eltérő kernel megoldások is születtek. Ezek között a legfontosabbak az LS-SVM és az ezzel matematikailag azonos, de más

indítékból származó ridge regresszió. A fejezetben ezeknek a megoldásoknak a származtatását is összefoglaljuk, és arra is kitérünk, hogy mi az ára az egyszerűbb számításoknak. Az LS-SVM a szupport vektor gépek egyik fontos tulajdonságát, nevezetesen, hogy ritka (*sparse*) megoldást elveszíti. Elsősorban e hiányosság kiküszöbölését célozza az LS²-SVM, valamint a ridge regresszió egy redukált változata, melyeket szintén összefoglalunk. A fejezet végén egy konkrét hálózat kapcsán megmutatjuk, hogy a bázisfüggvényes hálózatok kernel gép formában is megfogalmazhatók, biztosítva így a kernel megközelítés előnyeit.

1. Egy egyszerű kernel gép

A kernel reprezentációt a legegyszerűbben talán egy lineáris regressziós feladat kapcsán mutathatjuk be. Adott egy lineáris be-kimeneti leképezést megvalósító eszköz, amit a továbbiakban lineáris gépnek nevezünk:

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b, \quad (6.1)$$

Itt az eddigieknek megfelelően $\mathbf{w}$ a lineáris gép súlyvektora, b pedig az eltolásérték. Amennyiben rendelkezésünkre áll egy $\{\mathbf{x}_i, d_i\}_{i=1}^P$ tanítópont készlet, olyan súlyvektort keresünk, ami mellett a

$$C(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^P (d_i - \mathbf{w}^T \mathbf{x}_i - b)^2 \quad (6.2)$$

kritériumfüggvény minimumot vesz fel. A minimális négyzetes hibát biztosító súlyvektor – ahogy ezt az előző fejezetekben már láttuk – a kritériumfüggvény deriváltja alapján határozható meg. A b eltolásértéket a súlyvektor nulladik komponenseként kezelve és bevezetve a kibővített súlyvektort $\hat{\mathbf{w}} = [b, \mathbf{w}^T]^T$, valamint a kibővített bemeneti vektort $\hat{\mathbf{x}} = [1, \mathbf{x}^T]^T$ is, a lineáris gép válasza $\hat{\mathbf{x}}$ bemenet mellett

$$y(\hat{\mathbf{x}}) = \hat{\mathbf{w}}^T \hat{\mathbf{x}} = \sum_{i=0}^n \hat{w}_i \hat{x}_i \quad (6.3)$$

formába írható, ahol N a bemeneti $\mathbf{x}$ vektorok dimenziója. A tanítópontokban a lineáris gép válaszai és a kívánt válaszok közötti eltérésekből képezhetünk egy hibavektort

$$\mathbf{e}(\hat{\mathbf{w}}) = \mathbf{d} - \hat{\mathbf{X}}\hat{\mathbf{w}}, \quad (6.4)$$

melynek segítségével az eredő négyzetes hiba a

$$C(\hat{\mathbf{w}}) = \frac{1}{2} \mathbf{e}^T(\hat{\mathbf{w}}) \mathbf{e}(\hat{\mathbf{w}}) = \frac{1}{2} (\mathbf{d} - \hat{\mathbf{X}}\hat{\mathbf{w}})^T (\mathbf{d} - \hat{\mathbf{X}}\hat{\mathbf{w}}) \quad (6.5)$$

összefüggéssel adható meg. Itt $\mathbf{d}$ a tanítópontokban a kívánt válaszok vektora, $\hat{\mathbf{X}}$ pedig a tanítópontok kibővített bemeneti vektoraiból képezett mátrix:

$$\hat{\mathbf{X}} = \begin{bmatrix} \hat{\mathbf{x}}_1^T \\ \hat{\mathbf{x}}_2^T \\ \vdots \\ \hat{\mathbf{x}}_P^T \end{bmatrix}. \quad (6.6)$$

Elvégezve a gradiens számítást és a gradienst nullává téve

$$\frac{\partial C(\hat{\mathbf{w}})}{\partial \hat{\mathbf{w}}} = -\hat{\mathbf{X}}^T \mathbf{d} + \hat{\mathbf{X}}^T \hat{\mathbf{X}} \hat{\mathbf{w}} = \mathbf{0} \quad (6.7)$$

a súlyvektor legkisebb négyzetes hibájú (LS) becslését most is a pszeudo-inverz segítségével kapjuk meg:

$$\hat{\mathbf{w}}^* = \hat{\mathbf{X}}^T (\hat{\mathbf{X}} \hat{\mathbf{X}}^T)^{-1} \mathbf{d}. \quad (6.8)$$

Ha a kapott súlyvektort behelyettesítjük a (1.3) összefüggésbe, akkor adott $\hat{\mathbf{x}}$ -re a lineáris gép válasza a következő lesz¹:

$$y(\hat{\mathbf{x}}) = \hat{\mathbf{x}}^T \hat{\mathbf{w}}^* = \hat{\mathbf{x}}^T \hat{\mathbf{X}}^T (\hat{\mathbf{X}} \hat{\mathbf{X}}^T)^{-1} \mathbf{d}. \quad (6.9)$$

Vezessük be az

$$\boldsymbol{\alpha} = (\hat{\mathbf{X}} \hat{\mathbf{X}}^T)^{-1} \mathbf{d} \quad (6.10)$$

jelölést. Ekkor a kimenet az alábbi formában is felírható:

$$y(\hat{\mathbf{x}}) = \hat{\mathbf{x}}^T \hat{\mathbf{X}}^T \boldsymbol{\alpha} = \sum_{i=1}^P \alpha_i (\hat{\mathbf{x}}^T \hat{\mathbf{x}}_i) = \sum_{i=1}^P \alpha_i K_i(\hat{\mathbf{x}}) \quad (6.11)$$

ahol α_i az $\boldsymbol{\alpha}$ vektor i -edik komponense és $K_i(\hat{\mathbf{x}}) = \hat{\mathbf{x}}^T \hat{\mathbf{x}}_i$. A (6.11) összefüggés származtatásánál figyelembe vettük a tanítópontokhoz tartozó bemeneti vektorok (6.6) összefüggését. Így $\hat{\mathbf{x}}^T \hat{\mathbf{X}}^T$ egy olyan vektor, melynek elemei az $\hat{\mathbf{x}}$ bemenet és az $\hat{\mathbf{x}}_i$ tanítópont-bemenetek skalár szorzataiként állnak elő:

$$\hat{\mathbf{x}}^T \hat{\mathbf{X}}^T = [\hat{\mathbf{x}}^T \hat{\mathbf{x}}_1, \hat{\mathbf{x}}^T \hat{\mathbf{x}}_2, \dots, \hat{\mathbf{x}}^T \hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}^T \hat{\mathbf{x}}_P] \quad (6.12)$$

A (6.11) összefüggés érdekessége, hogy a lineáris gép egy $\hat{\mathbf{x}}$ bemenetre adott válasza a skalár szorzattal definiált $K_i(\hat{\mathbf{x}}) = \hat{\mathbf{x}}^T \hat{\mathbf{x}}_i$ függvények súlyozott összegeként határozható meg. Egy lineáris gépnél ezeket a függvényeket nevezzük kernel függvényeknek vagy magfüggvényeknek.

A (6.11) összefüggés a lineáris regressziós leképezés alternatív felírását jelenti. Míg az eredeti (1.3) összefüggés a bemenetek súlyozott összegeként adja meg a választ, ahol a súlyokat a $\hat{\mathbf{w}}$ súlyvektor képviseli, addig a kerneles reprezentációban a kimenet a $K_i(\hat{\mathbf{x}}) = \hat{\mathbf{x}}^T \hat{\mathbf{x}}_i$ függvények súlyozott összegeként áll elő, ahol a súlyokat az α_i együtthatók jelentik.

A fenti (6.3) és (6.11) összefüggések ugyanannak a feladatnak két eltérő reprezentációját képviselik. Nyilvánvaló, hogy a két reprezentáció ekvivalens. A (6.3) összefüggés a megoldást a „bemeneti térben”, közvetlenül $\hat{\mathbf{x}}$ függvényeként adja meg, míg a (6.11) összefüggés ugyanezt a „kernel térben”, a $K_i(\hat{\mathbf{x}}) = \hat{\mathbf{x}}^T \hat{\mathbf{x}}_i$ kernel értékek függvényeként teszi meg.

A most vizsgált lineáris leképezésnél a kernel reprezentáció különösebb előnnyel nem jár, mindössze egy alternatív megadási formát jelent. A különbség a kétféle reprezentáció között csupán annyi, hogy a két szummás kifejezésben a tagok száma eltérő. A bemeneti térben történő összegzés $N+1$ tagból, míg a kernel térbeli P tagból áll. A kernel reprezentáció tehát akkor előnyös, ha a mintapontok száma, P jóval kisebb, mint a bemeneti adatok dimenziója, N .

Más a helyzet akkor, ha nemlineáris leképezést akarunk megvalósítani. A nemlineáris feladatok egy lehetséges megoldása a bázisfüggvényes hálók alkalmazása, vagyis, ha a kimenetet az (5.1) összefüggéssel megadott

$$y(\mathbf{x}) = \sum_{i=0}^M w_i \varphi_i(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) \quad (6.13)$$

alakban állítjuk elő, ahol $\mathbf{w}^0$ képviseli az eltolás tagot (bias), amennyiben $\varphi_0(\mathbf{x}) \equiv 1$.

A bázisfüggvényes leképezés a bemeneti térből az ún. jellemzőtérbe transzformál. A jellemzőtérbeli megoldás alternatívájaként viszont most is alkalmazható a kernel térre való áttérés. A lineáris esetre vonatkozó fenti gondolatmenetet követve a (6.13)-mal megadott leképezés kerneles változata úgy nyerhető, hogy a (6.11) összefüggésben minden $\hat{\mathbf{x}}$ helyére $\boldsymbol{\varphi}(\mathbf{x})$ kerül. (A továbbiakban – hacsak ez nem okoz értelmezési zavart – külön nem jelezzük, hogy kibővített bemeneti vektorról van szó.) Ezzel:

¹ A válasz (6.9) szerinti meghatározásához az $(\hat{\mathbf{X}} \hat{\mathbf{X}}^T)^{-1}$ inverznek léteznie kell. Ha $(\hat{\mathbf{X}} \hat{\mathbf{X}}^T)$ szinguláris vagy közel szinguláris, a regularizáció alkalmazható, amikor $(\hat{\mathbf{X}} \hat{\mathbf{X}}^T)^{-1}$ helyett $(\hat{\mathbf{X}} \hat{\mathbf{X}}^T + \lambda \mathbf{I})^{-1}$ szerepel. (ld. még 2.1 szakasz illetve Függelék.)

$$y(\mathbf{x}) = \boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\Phi}^T \boldsymbol{\alpha} = \sum_{i=1}^P \alpha_i (\boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\varphi}(\mathbf{x}_i)) = \sum_{i=1}^P \alpha_i K_i(\boldsymbol{\varphi}(\mathbf{x})), \quad (6.14)$$

ami azt jelenti, hogy a kimenetet most is a skalár szorzattal definiált kernel értékek súlyozott összegeként állítjuk elő. Itt a

$$\boldsymbol{\varphi}(\mathbf{x})^T \boldsymbol{\varphi}(\mathbf{x}_i) = K_i(\boldsymbol{\varphi}(\mathbf{x})) = K(\mathbf{x}, \mathbf{x}_i). \quad (6.15)$$

függvény a kernel függvény, amit tehát a bázisfüggvények skalár szorzatával nyerhetünk, míg a

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\varphi}(\mathbf{x}_1)^T \\ \boldsymbol{\varphi}(\mathbf{x}_2)^T \\ \vdots \\ \boldsymbol{\varphi}(\mathbf{x}_P)^T \end{bmatrix} \quad (6.16)$$

mátrix a tanítópontok jellemzőtérbeli reprezentációiból felépülő mátrix.

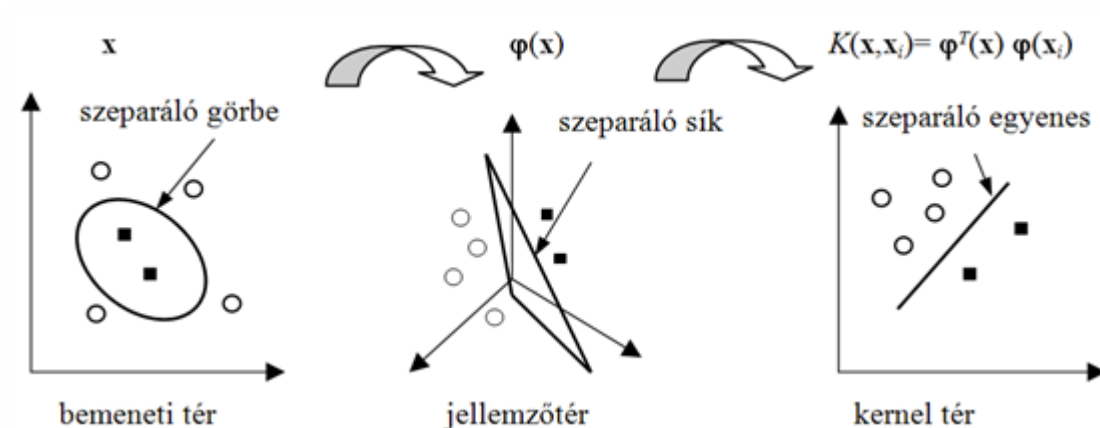
A (6.15) összefüggéssel megadott kernel reprezentáció itt is ekvivalens az eredeti reprezentációval ((6.13) összefüggés), és alkalmazása önmagában különösebb előnnyel itt sem jár az összegzésben szereplő tagok esetleges kisebb számán túl. A kernel reprezentáció előnye akkor látszanak, ha a kernel függvényeket nem a bázisfüggvények skalár szorzataként határozzuk meg, hanem közvetlenül felvesszük, tehát ha nem a bázisfüggvényekből, hanem a kernel függvényből indulunk ki. Kernel függvényként azonban bármilyen függvény nem használható, hiszen a kernel függvény még akkor is bázisfüggvények skalár szorzatával származtatható függvény kell legyen, ha a származtatásnál nem ezt az utat választjuk. Érvényes kernel függvénynek bizonyos feltételeket teljesítenie kell. A kernel függvény megválasztása implicit módon meghatározza a jellemzőtérre való leképezést biztosító bázisfüggvényeket is. Ez viszont már magában rejt a kernel megközelítés egy nagyon lényeges előnyét.

Amint az a (6.14) összefüggésből látszik a kerneles reprezentáció a tanítópontoknak megfelelő számú (P) kernel függvény-érték súlyozott összegeként áll elő, függetlenül attól, hogy az implicit módon definiált jellemzőtér dimenziója (M) mekkora. A kernel függvény megválasztásától függően a jellemzőtér dimenziója nagyon nagy, akár végtelen is lehet, ami a (6.13) szerinti kimenet előállítását nagyon megnehezítené, sőt akár lehetetlenné is tenné, miközben a kernel reprezentáció komplexitása a tanítópontok száma által mindenképpen korlátozott. Minthogy a kernel térbeli megoldás ekvivalens a jellemzőtérbeli megoldással, a kernel módszerekkel azt tudjuk elérni, hogy a megoldás komplexitását akkor is korlátozni tudjuk, ha egyébként a megfelelő jellemzőtérbeli megoldás extrém módon komplex lenne. A kernel függvények bevezetésének ezt a hatását kernel trükknek (*kernel trick*) nevezzük.

Nemlineáris esetben a kernel reprezentációt – legalábbis koncepcionálisan – két lépésben érjük el. Először a bemeneti térről megfelelő bázisfüggvények alkalmazásával nemlineáris dimenzió növelő transzformációt hajtunk végre. Ennek eredménye a jellemzőtérbeli reprezentáció. A feladatot azonban nem itt, hanem egy újabb transzformáció után a kernel térben oldjuk meg. A jellemzőtér és a kernel tér között a skalár szorzat teremti meg a kapcsolatot, és mindkét származtatott reprezentációra igaz, hogy az eredetileg nemlineáris probléma a származtatott reprezentációkban már lineárisan megoldható. A reprezentációk kapcsolatát mutatja a 6.1 ábra.

A koncepcionális származtatás mellett azonban fontos hangsúlyozni, hogy a kernel reprezentáció alkalmazásánál nem a 6.1 ábrán bemutatott utat követjük, a kernel térbe nem a jellemzőtérre keresztül jutunk el, hanem épp ellenkezőleg a kernel térből indulunk ki, és ez automatikusan definiálja a jellemzőteret. A fordított út előnye – ami a kernel trükk következménye –, hogy nem kell a jellemzőteret definiálnunk, ami egyébként sok esetben komoly nehézséget jelentene, dolgoznunk sem kell a praktikus nehezen kezelhető jellemzőtérben, hanem közvetlenül a kernel teret definiáljuk és a megoldást is itt nyerjük. Mindehhez az egyik legfontosabb követelmény a megfelelő kernel függvény megválasztása.

6.1. ábra - A nemlineáris leképezések az eredeti probléma tértől a kernel térig.



2. Kernel függvények

Az előzőekből nyilvánvaló, hogy kernel függvényként csak olyan függvény használható, amely belső szorzat segítségével is származtatható. A származtatás módjából következik, hogy a kernel függvények bizonyos tulajdonságokkal kell rendelkezzenek.

Egy kernel függvénynek mindig két argumentuma van és ezekre nézve a függvény szimmetrikus kell legyen:

$$K(\mathbf{x}, \mathbf{x}_j) = \varphi(\mathbf{x}_j)^T \varphi(\mathbf{x}) = \varphi(\mathbf{x})^T \varphi(\mathbf{x}_j) = K(\mathbf{x}_j, \mathbf{x}). \quad (6.17)$$

A kernel függvények – minthogy az adatok közötti hasonlóságot mérik – általában kielégítik a következő követelményeket is:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &\geq 0 \\ K(\mathbf{x}_i, \mathbf{x}_j) &= K(\|\mathbf{x}_i - \mathbf{x}_j\|) \\ K(\mathbf{x}, \mathbf{x}) &= \max_{i,j} K(\mathbf{x}_i, \mathbf{x}_j) \quad i, j = 1, \dots, P. \\ \lim_{t \rightarrow \infty} K(t) &= 0, \quad \text{ha } t = \|\mathbf{x}_i - \mathbf{x}_j\| \end{aligned} \quad (6.18)$$

A fentiek közül az első a nemnegativitást, a második a radiálisan szimmetrikus tulajdonságot jelenti. A harmadik feltételnek eleget tevő függvény maximumértéket vesz fel, ha mindkét argumentuma azonos, míg az utolsó azt fogalmazza meg, hogy a függvény a két argumentum távolságának monoton csökkenő függvénye.

A kernel függvényekkel támasztott követelmények precízebben is megfogalmazhatók: $K(\mathbf{x}_i, \mathbf{x}_j)$ lehet bármely olyan szimmetrikus függvény, amely kielégíti a Mercer tétel feltételeit [Vap98]:

$$\iint K(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0 \quad \forall f \neq 0 \text{ függvényre, ahol } \int f^2(\mathbf{x}) d\mathbf{x} < \infty, \quad (6.19)$$

ugyanis a Mercer tételt kielégítő függvények előállíthatók valamilyen jellemzőtérbeli $\varphi_j(\mathbf{x})$ függvények skalár szorzataként:

$$K(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \varphi_j(\mathbf{x}) \varphi_j(\mathbf{z}) \quad (6.20)$$

ahol $\lambda_j > 0$. Néhány gyakran alkalmazott kernel függvényt az alábbi táblázatban foglalunk össze.

6.1. táblázat - A legelterjedtebben használt magfüggvények (kernel függvények).

Lineáris	$K(\mathbf{x}, \mathbf{x}_i) = \mathbf{x}_i^T \mathbf{x}$
Polynomiális	$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}_i^T \mathbf{x} + 1)^d$

(d fokszerű)	
Gauss (RBF)	$K(\mathbf{x}, \mathbf{x}_i) = \exp\{-\ \mathbf{x} - \mathbf{x}_i\ ^2 / \sigma^2\}$, ahol σ konstans.
Tangens hiperbolikus (MLP)	$K(\mathbf{x}, \mathbf{x}_i) = \tanh(k \mathbf{x}_i^T \mathbf{x} + \theta)$, k és θ megfelelően választott konstansok, mert nem minden kombináció eredményez magfüggvényt.

A táblázat azt is jelzi, hogy egyes kernel függvények alkalmazásával olyan kernel térbeli hálóstruktúrát kapunk, amely bizonyos klasszikus neuronháló struktúrákhoz hasonló. Így pl. a Gauss kernel függvények az RBF hálózattal rokon struktúrát, a tangens hiperbolikus kernel függvény pedig olyan MLP hálózattal rokon struktúrát eredményez, melynek a kimeneti rétege lineáris.

A fenti, gyakran alkalmazott függvények mellett számos további függvényt használnak kernel függvényként. Megválasztásuk az egyes kernel gépeknél alapvetően fontos és sokszor igen nehéz kérdés. A választás nehézsége akkor jelentkezik, ha adott feladathoz „optimális” kernelt szeretnénk választani. Általában a gyakorlati feladatok megoldásánál nem is törekszünk „optimális” kernel megválasztására, hanem az előbb felsoroltak, vagy egyéb, szintén bevált függvények (pl. B-spline) közül választunk.

A kernel függvények megválasztásának illetve konstrukciójának részleteivel itt nem foglalkozunk, csak az igen széleskörű irodalomra utalunk ld. pl. [Sch02], stb. Annnyit azonban megjegyzünk, hogy megfelelő kernelek konstrukciójánál fontos szerepet játszhat, hogy különböző műveletekkel érvényes kernelekből további kernelek képezhetők. Ha pl. mind K_1 , mind K_2 érvényes kernel, továbbá, ha a tetszőleges pozitív konstans, akkor a következő műveletekkel további érvényes kernelek konstruálhatók:

$$\begin{aligned} K(\mathbf{x}, \mathbf{z}) &= K_1(\mathbf{x}, \mathbf{z}) + K_2(\mathbf{x}, \mathbf{z}) \\ K(\mathbf{x}, \mathbf{z}) &= \alpha K_1(\mathbf{x}, \mathbf{z}) \\ K(\mathbf{x}, \mathbf{z}) &= K_1(\mathbf{x}, \mathbf{z}) K_2(\mathbf{x}, \mathbf{z}) \end{aligned} \quad (6.21)$$

Fentiekből az is következik, hogy kernelek lineáris kombinációja szintén kernel, ha a lineáris kombináció együtthatói nemnegatív számok.

A kernel függvények előre történő megválasztása mellett az utóbbi időben számos olyan eredmény született, mely adatfüggő vagy adaptív kernelekkel oldja meg a feladatokat. Erre vonatkozó eredményeket mutat be többek között pl. [Cri99], [Ama99], [Vin00], [Xio05] és [Mer06].

A későbbiekben látni fogjuk, hogy a kernel gépek konstrukciójánál fontos szerepet tölt be a kernel függvény $(\mathbf{x}_i, \mathbf{x}_j)$ $i, j=1, \dots, P$ tanító minta-páron értelmezett értékeiből képezett $\mathbf{K}_{ij}=K(\mathbf{x}_i, \mathbf{x}_j)$ kernel mátrix, amely tartalmazza a P tanítópont összes lehetséges mintapárjánál a kernel függvény értékét.

$$\mathbf{K} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \dots & K(\mathbf{x}_1, \mathbf{x}_i) & \dots & K(\mathbf{x}_1, \mathbf{x}_P) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_i, \mathbf{x}_1) & \dots & K(\mathbf{x}_i, \mathbf{x}_i) & \dots & K(\mathbf{x}_i, \mathbf{x}_P) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_P, \mathbf{x}_1) & \dots & K(\mathbf{x}_P, \mathbf{x}_i) & \dots & K(\mathbf{x}_P, \mathbf{x}_P) \end{bmatrix} \quad (6.22)$$

A kernel mátrix fontos jellemzője, hogy egy $P \times P$ -es szimmetrikus mátrix. Az ilyen mátrixot szokás Gram mátrixnak is nevezni.

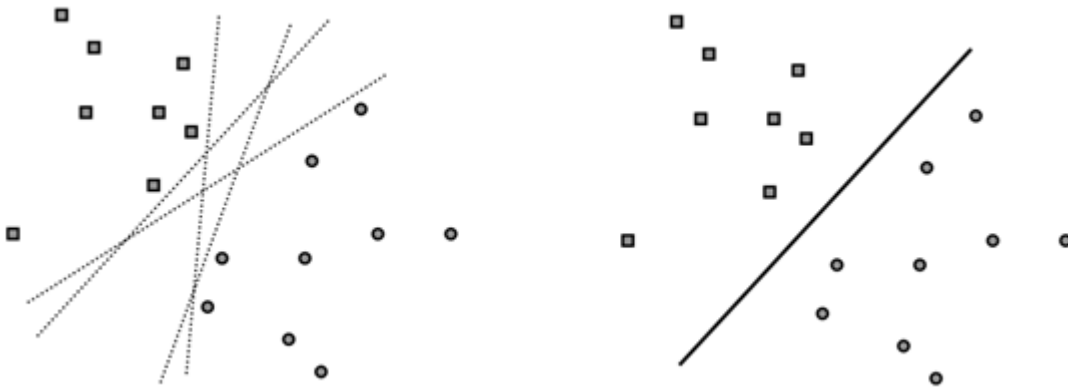
3. Szupport Vektor Gépek

Az előzőekben bemutatott egyszerű kernel gép arra adott példát, hogy az 5. fejezetben bemutatott bázisfüggvényes hálók kernel gépként is értelmezhetők. Ennél az értelmezésnél a probléma megfogalmazása nem változott az eredeti bázisfüggvényes megfogalmazáshoz képest: mindkét esetben az átlagos négyzetes hiba minimumát biztosító megoldásra törekedtünk.

Kernel gépek azonban más megközelítésből is konstruálhatók. A Bernhard Boser, Isabel Guyon és Vladimir Vapnik által javasolt szupport vektor gép [Bos92] talán a legfontosabb, több új szempontot is hozó kernel gépnek tekinthető. A szupport vektor gépek olyan kernel gépek, melyek a statisztikus tanulásmélet eredményeit is hasznosítják. Alapváltozatuk lineáris szeparálásra képes, amely azonban kiterjeszthető nemlineáris szeparálásra és nemlineáris regressziós feladatokra is.

Amint azt korábban láttuk, a lineáris szeparálási feladat megoldható egy egyszerű perceptronnal. A Rosenblatt perceptron azonban egy lineárisan szeparálható feladat egy lehetséges megoldását adja, miközben tudjuk, hogy lineárisan szeparálható mintapontok esetében általában végtelen sok különböző megoldásunk lehet. Ezek a megoldások egyenértékűek abból a szempontból, hogy mindegyikük egy olyan lineáris elválasztó felületet határoz meg, amely a tanítópontokat hibátlanul szétválasztja. A szeparálás minősége azonban az egyes megoldásokban mégis jelentősen különbözhet. Ezt illusztrálja a 6.2 ábra.

6.2. ábra - Egy lineárisan szeparálható feladat különböző megoldásai



Bár az ábra bal oldali részén látható összes egyenes hibátlanul szétválasztja a tanítópontokat, az a sejtésünk, hogy a jobb oldalon látható folytonos vonallal rajzolt egyenes kitüntetett a lehetséges megoldások között. A mintapontokból történő tanulásnál ugyanis nemcsak a tanítópontok megtanulására törekszünk, hanem megfelelő általánosítóképesség elérésére is. A mintapontok valamilyen – számunkra általában ismeretlen – eloszlású mintahalmazokból származnak, és a feladat nemcsak a véges számú ismert válaszú tanítópont helyes osztályozása, hanem az egész osztályozási feladat minél jobb megoldása is. Feltételezve, hogy a tanítópontok kellően reprezentálják a megoldandó problémát, azt várjuk, hogy az az elválasztás eredményez jobb általánosítóképességet, amely a két osztályba tartozó tanítópontok között, a tanítópontoktól a lehető legnagyobb távolságra helyezkedik el. A lineáris kétosztályos osztályozási feladat megoldását adó szupport vektor gép ezt az „optimális” elválasztó felületet határozza meg.

A következőkben előbb ezt a legegyszerűbb esetet és a megfelelő megoldást mutatjuk be, majd a szupport vektor gépeket kiterjesztjük lineárisan nem megoldható osztályozási, végül nemlineáris regressziós feladatok megoldására is.

3.1. Lineárisan szeparálható feladat lineáris megoldása

Fentiek értelmében optimális lineáris szeparálásnak azt a megoldást tekintjük, amikor az elválasztó egyenes (sík, hipersík) a két osztályba tartozó tanítópontok között a pontoktól a lehető legnagyobb távolságra helyezkedik el. A pontok között középre elhelyezett szeparáló felületet a tanítópontoktól egy margó (*margin*), azaz egy biztonsági sáv választja el, ezért az így megoldható feladatokat maximális tartalékot vagy maximális margót biztosító lineárisan szeparálható osztályozási feladatoknak is nevezzük. A következőkben ennek a feladatnak a megoldását mutatjuk be.

Legyenek adottak az $\{\mathbf{x}_i, d_i\}_{i=1}^P$ tanítópontok, ahol az $\mathbf{x}_i \in \mathbb{R}^n$ N -dimenziós bemeneti pontok két osztály valamelyikéből származhatnak, vagyis a kívánt válaszok a két lehetséges érték egyikét veszik fel: $d_i \in \{-1, 1\}$. Egy olyan $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$ szeparáló függvényt keresünk, amely a tanítópontokat hiba nélkül osztályozza:

$$\begin{aligned} \mathbf{w}^T \mathbf{x}_i + b &\geq a > 0 \quad \text{ha } d_i = +1 \\ \mathbf{w}^T \mathbf{x}_i + b &\leq -a < 0 \quad \text{ha } d_i = -1 \end{aligned} \quad i=1,2,\dots,P \quad (6.23)$$

továbbá, ahol a hipersíkhöz legközelebb álló tanítópontoknak a síktól vett távolsága maximális. A (6.23) egyenlőtlenségben a valamilyen kis pozitív konstans, d_i pedig az $\mathbf{x}_i$ -hez tartozó kívánt válasz.

A hipersík paramétereinek megfelelő skálázásával biztosítható, hogy

$$\begin{aligned} \mathbf{w}^T \mathbf{x}_i + b &\geq +1 \quad \text{ha } d_i = +1 \\ \mathbf{w}^T \mathbf{x}_i + b &\leq -1 \quad \text{ha } d_i = -1 \end{aligned} \quad (6.24)$$

legyen, vagyis a síkhoz legközelebb eső pontokban az elválasztó függvény abszolút értéke 1 legyen. Így a feladat az alábbi tömörebb formában is felírható:

$$d_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad i=1,2,\dots,P \quad (6.25)$$

Jelölje az optimális szeparáló lineáris felület paramétereit $\mathbf{w}^*$ és b^* , és határozzuk meg egy $\mathbf{x}$ pontnak ettől a hipersíktól való távolságát. A 6.3 ábrát követve jelölje $\mathbf{x}_p$ az $\mathbf{x}$ pont merőleges vetületét a szeparáló felületre. Ekkor a pont felírható a vetület és a szeparáló felület normálvektora segítségével:

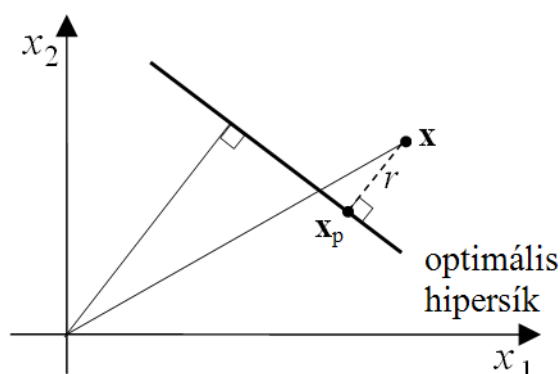
$$\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}^*}{\|\mathbf{w}^*\|}. \quad (6.26)$$

Mivel $g(\mathbf{x}_p) = 0$, $g(\mathbf{x}) = \mathbf{w}^{*T} \mathbf{x} + b^* = r \|\mathbf{w}^*\|$, vagyis az $\mathbf{x}$ pontnak az optimális szeparáló hipersíktól való távolságára

$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}^*\|} \quad (6.27)$$

adódik, ahol $\|\mathbf{w}^*\|$ a $\mathbf{w}^*$ vektor euklideszi normáját jelöli. Az optimális hipersík a síkhoz legközelebbi tanítópontok távolságának maximumát biztosítja.

6.3. ábra - A margó geometriai értelmezése.



Az egységnyi távolságúra skálázott feladatban a határoló felülethez legközelebb eső pontokra $g(\mathbf{x}_i^+) = 1$ vagy $g(\mathbf{x}_i^-) = -1$. Ekkor a szeparáló hipersíkhöz legközelebb eső, különböző osztályba tartozó bemeneti vektorok közötti, a síkra merőlegesen mért távolság

$$\rho = \frac{g(\mathbf{x}_i^+)}{\|\mathbf{w}^*\|} - \frac{g(\mathbf{x}_i^-)}{\|\mathbf{w}^*\|} = \frac{1}{\|\mathbf{w}^*\|} - \frac{-1}{\|\mathbf{w}^*\|} = \frac{2}{\|\mathbf{w}^*\|}. \quad (6.28)$$

Az optimális hipersík által biztosított margó tehát:

$$r = \frac{\rho}{2} = \frac{1}{\|\mathbf{w}^*\|}. \quad (6.29)$$

Az egyenlet szerint az osztályozási tartalék akkor lesz maximális, ha $\|\mathbf{w}^*\|$ minimális értékű.

A megoldandó feladat ezek után a következőképpen fogalmazható meg: adott egy lineárisan szeparálható mintapont készlet

$$S = \{(\mathbf{x}_1, d_1), \dots, (\mathbf{x}_n, d_n)\}, \quad (6.30)$$

és keressük azt a minimális normájú $\mathbf{w}^*$ vektort és azt a b^* skalár értéket, melyekkel a tanítópontok mindegyikét helyesen osztályozzuk, vagyis keressük a

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} (\mathbf{w}^T \mathbf{w}) \quad (6.31)$$

megoldását, azzal a feltétellel, hogy

$$d_i(\mathbf{w}^{*T} \mathbf{x}_i + b^*) \geq 1 \quad i = 1, 2, \dots, P. \quad (6.32)$$

A feladatot tehát feltételes szélsőérték-keresési problémaként tudjuk megfogalmazni, ahol a feltételek egyenlőtlenségek formájában vannak megadva. A feltételes szélsőérték-keresési feladat megoldását egy Lagrange kritérium (ld. Függelék) megoldásával kereshetjük:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^P \alpha_i [d_i(\mathbf{w}^T \mathbf{x}_i + b) - 1], \quad (6.33)$$

ahol az $\alpha_i \geq 0$ együtthatók a Lagrange multiplikátorok. Az optimalizálási feladatnak ezt a felírását elsődleges alaknak (*primal problem*) nevezzük.

A optimalizálási feladat megoldásához a Karush-Kuhn-Tucker (KKT) elmélet (ld. Függelék) szerint a fenti Lagrange kritériumot kell minimalizálni $\mathbf{w}$ és b szerint és maximalizálni α_i szerint, vagyis a Lagrange kritérium által definiált kritériumfelület nyeregpontját (*saddle point*) kell meghatározni. A megoldás két lépésben érhető el:

- először $\mathbf{w}$ és b szerinti szélsőértéket keresünk a megfelelő deriváltak számításával,
- majd ezek eredményét behelyettesítve a Lagrange kritériumba kapjuk az ún. másodlagos feladatot (*dual problem*), melynek megoldásai az α_i Lagrange multiplikátorok.

Az $L(\mathbf{w}, b, \alpha)$ Lagrange kritérium $\mathbf{w}$ és b szerinti parciális deriválásából az optimumra az alábbi két feltételt kapjuk:

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_{i=1}^P \alpha_i d_i \mathbf{x}_i, \quad (6.34)$$

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = 0 \rightarrow \sum_{i=1}^P \alpha_i d_i = 0 \quad \alpha_i \geq 0 \quad (6.35)$$

Látható, hogy ha a Lagrange multiplikátorokat ismerjük, a feladatot megoldottuk, hiszen $\mathbf{w}$ csak az α_i Lagrange multiplikátoroktól és a tanítópont-értékektől függ.

A feltételes optimalizálási feladat megoldásához felírható annak duális alakja, melyben már csak az α_i Lagrange multiplikátorok az ismeretlenek. Mivel az

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^P \alpha_i d_i \mathbf{w}^T \mathbf{x}_i - b \sum_{i=1}^P \alpha_i d_i + \sum_{i=1}^P \alpha_i \quad (6.36)$$

kifejtésében a harmadik tag (1.35) alapján nulla, és mivel (6.34) alapján

$$\mathbf{w}^T \mathbf{w} = \sum_{i=1}^P \alpha_i d_i \mathbf{w}^T \mathbf{x}_i = \sum_{i=1}^P \sum_{j=1}^P \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j \quad (6.37)$$

az $L(\mathbf{w}, b, \alpha) = Q(\alpha)$ duális optimalizálási feladat az alábbi:

$$Q(\alpha) = \sum_{i=1}^P \alpha_i - \frac{1}{2} \sum_{i=1}^P \sum_{j=1}^P \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j \quad (6.38)$$

a

$$\sum_{i=1}^P \alpha_i d_i = 0 \quad (6.39)$$

és a

$$\alpha_i \geq 0, \quad i = 1, \dots, P \quad (6.40)$$

feltételekkel.

A duális probléma egy olyan, a keresett α_i értékekben másodfokú szélsőérték-keresési feladat, mely mellett a megoldásnak egy egyenlőség, (6.39) és egy egyenlőtlenség, (6.40) típusú mellékfeltételt is ki kell elégítenie. Az ilyen típusú optimalizálási feladatok kvadrátikus programozással (*quadratic programming, QP*) oldhatók meg. Az optimumhoz tartozó Lagrange multiplikátorok – az α_i^* értékek – ismeretében, felhasználva az optimális súlyvektorra vonatkozó (6.34) összefüggést, a lineáris kétosztályos osztályozó szupport vektor gép válasza felírható, mint

$$y(\mathbf{x}) = \text{sgn} \left[\sum_{i=1}^P \alpha_i^* d_i \mathbf{x}_i^T \mathbf{x} + b^* \right]. \quad (6.41)$$

Vegyük észre, hogy a válasz összefüggésében a bemeneti mintavektorok nem közvetlenül, hanem egy skálár szorzat részeként szerepelnek, tehát a (1.11) összefüggéshez hasonlóan itt is kernel megoldást kaptunk. A megoldás érdekessége, hogy az α_i^* -k nagyrésze általában 0, így mind a súlyvektor kifejezésében, mind a szupport vektor gép válaszában a tanítópontoknak csak egy része – a P tanítópontból csak P_s – vesz részt.

Azokat a tanítópontokat, amelyek résztvesznek a megoldás kialakításában, amelyekhez tartozó Lagrange multiplikátorok értéke nem nulla, szupport vektoroknak (*support vectors*) nevezzük.² A szupport vektor gépek tehát olyan kernel gépek, ahol a kernel tér tényleges dimenziója nem a tanítópontok számával (P), hanem a szupport vektorok számával (P_s) egyezik meg. Ez jelentős egyszerűsítést jelenthet a válasz számításában, különösen, ha $P_s \ll P$. A szupport vektor gépeknek ezt a tulajdonságát ritkasági (*sparseness*) tulajdonságnak nevezzük.

Azt is észrevehetjük, hogy azon tanítópontokra, ahol $\alpha_i^* > 0$, teljesül a

$$d_i (\mathbf{w}^{*T} \mathbf{x}_i + b^*) = 1 \quad (6.42)$$

összefüggés, vagyis ezek a tanítópontok – tehát a szupport vektorok – lesznek az elválasztó hipersíkhöz legközelebb lévő mintapontok. Itt $\mathbf{w}^*$ már az optimális Lagrange együtthatókkal meghatározható súlyvektor:

$$\mathbf{w}^* = \sum_{i=1}^P \alpha_i^* d_i \mathbf{x}_i, \quad (6.43)$$

A szupport vektoroknál (6.42) alapján a (6.24) és a (6.25) egyenlőtlenségek egyenlőségként állnak fenn. Az optimális eltolásérték (*bias*) (b^*) a $d_i (\mathbf{w}^{*T} \mathbf{x}_i + b^*) = 1$ egyenletből számítható, ahol $\mathbf{x}_i$ egy szupport vektor.

3.1.1. A maximális margójú lineáris szeparálás általánosítóképessége

² Úgy is mondhatjuk, hogy ezek a mintapontok „tartják” a megoldást, ezért ezek a mintapontok a tartó vektorok vagy támasztó vektorok (*support vectors*). A tartó vektor elnevezés azonban a magyar nyelvű szakirodalomban egyelőre nem terjedt el, ezért a könyvben a szupport vektor elnevezést használjuk.

Az előzőekben láttuk, hogy egy tanuló rendszer létrehozásánál a konstrukciós lépések meghatározásán túl fontos kérdés, hogy a létrehozott rendszert minősíteni is tudjuk. A 3. fejezetben részletesebben tárgyaltuk a minősítés lehetőségeit és bemutattuk, hogy a minősítéshez valójában az ún. valódi kockázat meghatározására lenne szükség. Azt is láttuk, hogy a valódi kockázatot – a mintapont-eloszlás ismeretének hiányában – nem tudjuk meghatározni. A statisztikus tanuláselmélet azonban megadja azokat a feltételeket, melyek fennállta esetén a valódi kockázatnak legalább egy felső korlátja megadható.

A (2.31) összefüggés osztályozási feladatra adott felső korlátot. A felső korlát két tagból állt: egyrészt a tanítópontokban meghatározható eredő hibából, a tapasztalati kockázatból, és egy ún. konfidencia tagból, ami a megoldás komplexitásától függ. A statisztikus tanuláselmélet a komplexitás mértékeként a VC-dimenziót használja. Ha tehát egy megoldás VC-dimenziójára tudunk mondani valamit, a valódi kockázat egy felső korlátja megadható. A felső korlát (2.31) összefüggése alapján azt is láttuk, hogy olyan megoldásra van szükségünk, amely egyidejűleg mindkét tagot minimalizálja, tehát a tapasztalati hibát úgy próbálja minél kisebbre leszorítani, hogy közben a VC-dimenzió értéke is minél kisebb legyen. A megoldás elve a strukturális kockázat minimalizálás (SRM) elv volt. A strukturális kockázat minimalizálás elve szerint azt a legkevésbé komplex megoldást keressük, melynél a két tag összege minimumot ad.

A lineáris osztályozó szupport vektor gépnél, azon túl, hogy a tanítópontok hibátlan osztályozását kívánjuk elérni, még azt is célnak tekintjük, hogy a szeparálás maximális biztonsággal, maximális margó mellett történjen meg. A maximális margójú megoldás a lineáris szeparálások között kitüntetett. A maximális margó biztosítása mellett ugyanis a megoldás komplexitásának mértékére, a VC-dimenzióra is adható egy felső korlát, és ennek segítségével a lineáris szupport vektor gép általánosítóképességéről is tudunk állítani valamit.

A statisztikus tanuláselmélet eredményei szerint [Vap95], [Vap98] a

$$\mathbf{w}^* \cdot \mathbf{x}_i + b^* = 0 \quad (6.44)$$

által megadott optimális szeparáló hipersík VC-dimenziójára az alábbi felső korlát adható:

$$h \leq \min \left(\left\lceil \frac{R^2}{r^2} \right\rceil, N \right) + 1, \quad (6.45)$$

ahol N a bemeneti tér dimenziója, r a szeparációs tartalék, a margó, R pedig az összes bemeneti vektort tartalmazó legkisebb átmérőjű gömb sugara. Az összefüggésben a $\lceil \cdot \rceil$ művelet azt a legkisebb egészet jelöli, ami nagyobb vagy egyenlő, mint a művelet argumentuma.

Bár egy függvényosztály VC-dimenzióját általános esetben nehéz meghatározni, N -dimenziós szeparáló hipersík esetén könnyen megmutatható, hogy a VC-dimenzió értéke $h=N+1$. Ha azonban nemcsak a mintapontok lineáris szeparálását végezzük, hanem *maximális margójú* lineáris szeparálást biztosítunk, már a (6.45) összefüggéssel megadott felső korlát érvényes. Ez az összefüggés azt mondja, hogy maximális margójú szeparáló hipersík mellett – ha az r margó értéke elegendően nagy – a VC-dimenzió értéke lehet $(N+1)$ -nél kisebb is. Ennek különösen nemlineáris esetben van jelentősége, ahogy ezt a későbbiekben látni fogjuk. Az a tény, hogy egy felső korlát adható a VC-dimenzióra azt jelenti, hogy a megoldás általánosítóképessége a bemeneti tér dimenziójától függetlenül kontrol alatt tartható a szeparálás tartalékának megfelelő megválasztásával.

A tartalék (6.29) összefüggése szerint a margó fordítottan arányos a súlyvektor normájával. A (6.33) Lagrange kritérium olyan optimalizálási feladatot fogalmaz meg, ahol a tanítópontok megfelelő osztályozásán túl a súlyvektor normanegyzetének minimumát is biztosítani szeretnénk. A szupport vektor gépeknél az osztályozási hiba minimuma mellett a maximális tartalék, vagyis a minimális VC-dimenzió elérésére törekszünk, összhangban a strukturális kockázat minimalizálási elvvel. A szupport vektor gépekre ezért a statisztikus tanuláselméletnek a 2. fejezetben összefoglalt eredményei alkalmazhatók. Ez lényeges különbség, ha a szupport vektor gépeket az előzőekben bemutatott ún. klasszikus hálókkal, de elsősorban a többretegű perceptronnal hasonlítjuk össze.

Az MLP hálózatoknál a komplexitás a rejtett réteg méretétől függ. A háló struktúrájának rögzítése a rejtett réteg méretét is meghatározza. A tanítás során ezen már nem változtatunk. Ezzel szemben a szupport vektor gépeknél nem a szabad paraméterek száma határozza meg a komplexitás mértékét, hanem a VC-dimenzió, melynek felső korlátja a maximális margó értékével befolyásolható. A kétféle megközelítés közötti alapvető különbség tehát, hogy míg az MLP-nél csak a tapasztalati hiba, addig a szupport vektor gépeknél mind a tapasztalati hiba, mind az általánosítóképességet befolyásoló komplexitásmérték, a VC-dimenzió minimalizálása megtörténik.

3.2. A lineárisan nem szeparálható feladat lineáris megoldása

Az előzőekben tárgyalt maximális margójú lineáris szeparálásra képes megoldás az alapelvek bemutatására alkalmas. A gyakorlatban azonban ritkán találkozunk olyan feladatokkal, melyek úgy szeparálhatók lineárisan, hogy még osztályozási tartalék is biztosítható. Általában vagy csak olyan lineáris szeparálás lehetséges, ahol a két osztály között csak nagyon kis biztonsági sáv alakítható ki, vagy lineáris szeparálás hibamentesen nem is lehetséges. A kis biztonsági sávot növelhetnénk, ha nem ragaszkodnánk ahhoz, hogy minden tanítópont a sávon kívül helyezkedjen el. Ha a pontok többsége a sávon kívül lesz, de néhány mintapontonál megengedjük, hogy a sávon belülre kerüljenek (sőt esetleg azt is megengedjük, hogy egyes mintapontok az elválasztó hipersík „rossz” oldalára kerüljenek), akkor ezen az áron a sáv mérete esetleg jelentősen növelhető. A következőkben olyan lineáris megoldást mutatunk be, ahol megfelelő kompromisszum alapján dől el a biztonsági sáv mérete és a hibás osztályozás valószínűsége.

Ha megengedjük, hogy a biztonsági sávban is legyenek tanítópontok, miközben továbbra is cél a lehető legnagyobb margó biztosítása, ún. lágy vagy szoft margójú megoldásról beszélünk. Azoknál a pontoknál, amelyek a biztonsági sávban belül helyezkednek el a maximális margójú osztályozást biztosító

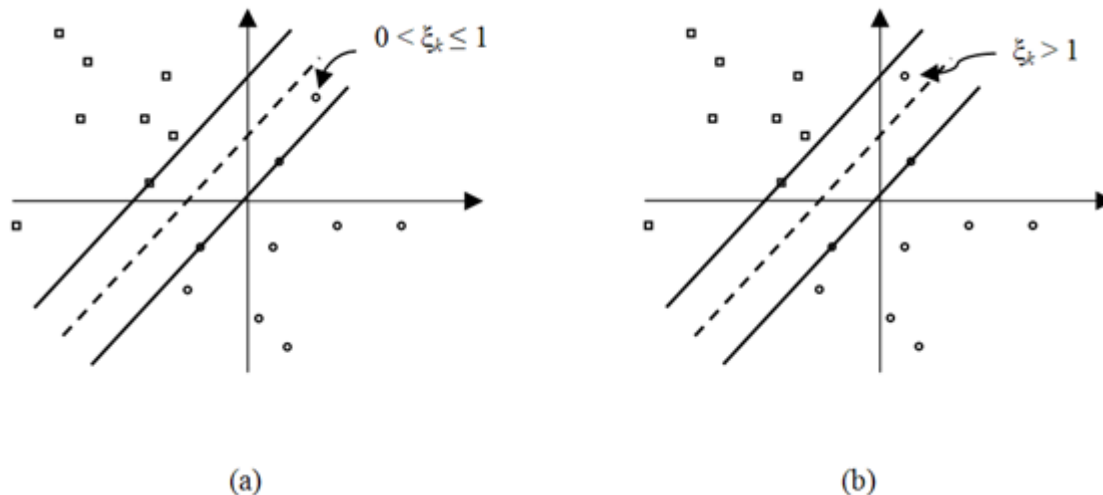
$$d_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad (6.46)$$

egyenlőtlenség nem áll fenn. Az ilyen mintapontokra vonatkozó, az előző egyenlőtlenségnek megfelelő formális kapcsolat ún. gyengítő (*slack*) ξ_i változók bevezetésével lehetséges. A gyengítő változók bevezetése lehetővé teszi, hogy a (6.46) összefüggés az egyes tanítópontoknál különböző mértékben gyengítve érvényesüljön. Ennek megfelelően az összes pontra most a következő egyenlőtlenség írható fel:

$$d_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i=1,2,\dots,P \quad (6.47)$$

Azon tanítópontoknál, ahol $\xi_i = 0$, visszakapjuk az alapfeladatot. Ha $0 < \xi_i \leq 1$, az adott tanítópont a hipersík megfelelő oldalán, de a biztonsági sávban vagy a sávhatáron helyezkedik el, ha pedig $\xi_i > 1$ az adott tanítópont a sáv ellenkező oldalán (a hibás oldalon) van. Ezt illusztrálja a 6.4. ábra.

6.4. ábra - A gyengítő változók használata: (a) a mintapont osztályozása helyes, de a mintapont a biztonsági sávba esik, (b) a mintapont osztályozása hibás.



Az optimális hipersíkot úgy kell meghatározni, hogy a hibás osztályozások száma minimális legyen, miközben továbbra is törekszünk a lehető legnagyobb margó elérésére. A minimalizálandó kifejezés – amit a továbbiakban $J(\mathbf{w})$ -vel jelölünk – ennek megfelelően két tagból áll:

$$J(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^P \xi_i, \quad (6.48)$$

ahol C a két tag közötti kompromisszumot beállító együttható. Ha $C=0$, visszakapjuk az előző, gyengítő változó nélküli esetet. Ha C értéke kicsi, a minimalizálandó összefüggésben a második tag súlya kicsi, vagyis nem

nagyon büntetjük, ha egy tanítópont a margón belülre, netán a rossz oldalra kerül. Ez azt jelenti, hogy a biztonsági sáv szélesebb lehet, de több pont kerül a sávon belülre vagy a rossz oldalra. Ha C értékét növeljük, ezeket az eseteket jobban büntetjük, ami keskenyebb biztonsági sávot eredményez, hiszen ekkor nyilván kevesebb pont eshet a sávon belülre.

A megoldás most is a megfelelő Lagrange kritérium felírását igényli, hiszen itt is feltételes szélsőérték-keresési feladattal állunk szemben. Egyfelől (6.48) minimumát kívánjuk elérni, de közben a (6.47) által a tanítópontokra megfogalmazott egyenlőtlenségeket is teljesíteni kívánjuk. Ennek megfelelően a következő Lagrange kritériumot tudjuk felírni.

$$L(\mathbf{w}, \xi, \alpha, \gamma, b) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^P \xi_i - \sum_{i=1}^P \alpha_i \{d_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i\} - \sum_{i=1}^P \gamma_i \xi_i \quad (6.49)$$

A kritérium a (6.47) által a tanítópontokra megfogalmazott gyengített feltételek figyelembevételén kívül egy új taggal is bővült. Az utolsó tag – amely szintén egy Lagrange multiplikátoros tag – azt fejezi ki, hogy törekszünk arra, hogy a ξ_i értékek lehetőleg nullák, de legalábbis minél kisebbek legyenek. Itt γ_i -k a megfelelő Lagrange multiplikátorok. A módosított Lagrange kritériumot $\mathbf{w}$, b és ξ_i -k szerint minimalizálni, míg α_i -k és γ_i -k szerint maximalizálni kell. A feladatot most is két lépésben oldhatjuk meg. Először a Lagrange kritérium $\mathbf{w}$, b és ξ szerinti deriváltját írjuk fel.

$$\frac{\partial L(\mathbf{w}, \xi, \alpha, \gamma, b)}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_{i=1}^P \alpha_i d_i \mathbf{x}_i, \quad (6.50)$$

$$\frac{\partial L(\mathbf{w}, \xi, \alpha, \gamma, b)}{\partial b} = 0 \rightarrow \sum_{i=1}^P \alpha_i d_i = 0 \quad (6.51)$$

$$\frac{\partial L(\mathbf{w}, \xi, \alpha, \gamma, b)}{\partial \xi_i} = 0 \rightarrow \gamma_i + \alpha_i = C. \quad (6.52)$$

Ezt követően a gradiensek értékének nullává tételével kapott összefüggéseket behelyettesítve a Lagrange kritériumba kapjuk a (6.38) összefüggéssel megegyező duális feladatot:

$$Q(\alpha) = \sum_{i=1}^P \alpha_i - \frac{1}{2} \sum_{i=1}^P \sum_{j=1}^P \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j, \quad (6.53)$$

ahol

$$\sum_{i=1}^P d_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, P, \quad (6.54)$$

és melyből az α_i -k most is kvadratikus programozással határozhatók meg. Figyeljük meg, hogy a gyengítő változók bevezetése az optimalizálási feladat duális alakját csak a Lagrange multiplikátorokra vonatkozó feltételekben módosítja. Az osztályozó eredménye is megegyezik a lineárisan szeparálható feladatra megadottakkal (6.41). Ennek megfelelően itt is lesznek nulla értékű α_i -k, melyekhez tartozó tanítópontok nem vesznek részt a kimenet előállításában. Ha pedig $\alpha_i \neq 0$, a megfelelő tanítópontok szupport vektorok lesznek, melyekre a

$$d_i(\mathbf{w}^{*T} \mathbf{x}_i + b^*) - 1 + \xi_i = 0 \quad (6.55)$$

egyenlőség áll fenn. Ez azt jelenti, hogy most a szupport vektorok sem feltétlenül a margó határán helyezkednek el. A margóhoz viszonyított helyzetüket a ξ_i értékek befolyásolják.

A (6.48) minimalizálandó kritériumban szereplő C konstans hiperparaméternek is szokás nevezni. Szerepe – ahogy az előbb említettük – az, hogy a súlyvektor hosszának, illetve a tanító mintákra számított osztályozási hibának a viszonyát beállítsa. C meghatározása nem könnyű feladat, általában kereszt kiértékeléssel történhet.

3.2.1. A szoft margójú lineáris szeparálás általánosító képessége

A gyengítő változó bevezetése azt eredményezte, hogy az összes pont már nem esik a biztonsági sávon kívülre. Ennek a megoldás általánosítóképességére is lesz hatása. A VC-dimenzió (6.45) összefüggése a szoft margójú megoldásra már nem lesz érvényes, így nem lesz érvényes a valódi kockázat felső korlátját megadó (2.31) összefüggés sem. Azonban a lágy margójú megoldás általánosítási hibájára is származtattak felső korlátokat, melyek figyelembe veszik a gyengítő változók értékeit is. Ezek bemutatásával itt nem foglalkozunk. A származtatás és az eredmények az irodalomban [Cri00], [Sha02] megtalálhatók.

3.3. Nemlineáris szeparálás

A valós osztályozási feladatok túlnyomó része nem szeparálható lineárisan. A lineárisan nem szeparálható feladatoknál – ahogy ezt már többször láttuk – egy megfelelő (és általában dimenziónövelő) $\mathbf{x} \rightarrow \Phi(\mathbf{x})$ nemlineáris transzformációval lineárisan szeparálhatóvá alakítjuk a feladatot. Az SVM előnyeinek kihasználására az így kapott jellemzőtérben az előzőeknek megfelelő optimális (maximális margójú) megoldást keressük.

A nemlineárisan szeparálható feladat SVM-el történő megoldásában tehát két lépés szerepel:

- nemlineáris transzformáció a bemeneti térből a jellemzőtérbe,
- a jellemző térben az optimális lineáris szeparáló hipersík meghatározása.

Ez utóbbi lépést azonban itt sem a jellemzőtérben, hanem az ebből származtatott kernel térben oldjuk meg.

A nemlineáris megoldás a lineáris eset megoldásából egyszerűen az $\mathbf{x} \rightarrow \Phi(\mathbf{x})$ helyettesítéssel nyerhető. Az optimális hipersíkot a következő alakban keressük:

$$\mathbf{w}^T \Phi(\mathbf{x}) + b = 0 \quad (6.56)$$

Ennek alapján a korábbi lineárisan szeparálható esetre kapott eredményünknek megfelelően az optimális súlyvektor:

$$\mathbf{w}^* = \sum_{i=1}^P \alpha_i^* \Phi(\mathbf{x}_i) \quad , \quad (6.57)$$

ahol $\Phi(\mathbf{x}_i)$ az $\mathbf{x}_i$ mintához tartozó jellemző (feature) vektor. Az optimális eltolásérték (bias) (b^*) a $\mathbf{w}^{*T} \Phi(\mathbf{x}_s) + b^* = +1$ vagy a $\mathbf{w}^{*T} \Phi(\mathbf{x}_s) + b^* = -1$ egyenletből számítható, ahol $\mathbf{x}_s^+$ ill. $\mathbf{x}_s^-$ egy-egy megfelelő osztályba tartozó szupport vektor. A jellemzőtérben megkonstruált szeparáló felületet tehát a következő egyenlet adja meg:

$$\sum_{i=1}^P \alpha_i^* \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}) + b^* = 0 \quad . \quad (6.58)$$

Látható, hogy a megoldáshoz most is csak az α_i Lagrange multiplikátorok meghatározása szükséges, amit a (6.38) összefüggésnek megfelelő duális feladat kvadratikusan történő megoldásával nyerhetünk. A különbség mindössze annyi, hogy a duális feladatnál $\mathbf{x}$ helyére mindenhol $\Phi(\mathbf{x})$ kell kerülni. Ha a jellemzőtérben maximális margójú megoldás kapható, akkor a Lagrange multiplikátorokra az $0 \leq \alpha_i$ feltétel, ha csak a gyengített változat, akkor az $0 \leq \alpha_i \leq C$ feltétel teljesül.

A $\Phi(\mathbf{x}_i)^T \Phi(\mathbf{x})$ szorzatot a kernel trükknek megfelelően egy magfüggvényként írjuk fel, azaz

$$K(\mathbf{x}_i, \mathbf{x}) = \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}) \quad . \quad (6.59)$$

A nemlineáris osztályozó tehát:

$$y(\mathbf{x}) = \text{sign} \left[\sum_{i=1}^P \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^* \right] \quad (6.60)$$

alakban adja meg egy $\mathbf{x}$ bemenetre a választ. Az összegzés az összes $\tan \alpha_i = 0$ -ra vonatkozik, valójában azonban csak a szupport vektorokra kell elvégezni, hiszen a többi esetben $\alpha_i = 0$. A szupport vektor gépek tehát nemlineáris esetben is ritka (*sparse*) megoldást eredményeznek.

A szupport vektor gépes megoldás előnye a kerneles megoldások előnyéhez hasonlóan itt is azáltal jelentkezik, hogy a feladatot nem a jellemzőtérben oldjuk meg, hanem a kernel térben, azáltal, hogy a $\Phi(\mathbf{x})$ nemlineáris transzformációt nem közvetlenül, hanem csak a kernel függvényen keresztül, közvetve definiáljuk.

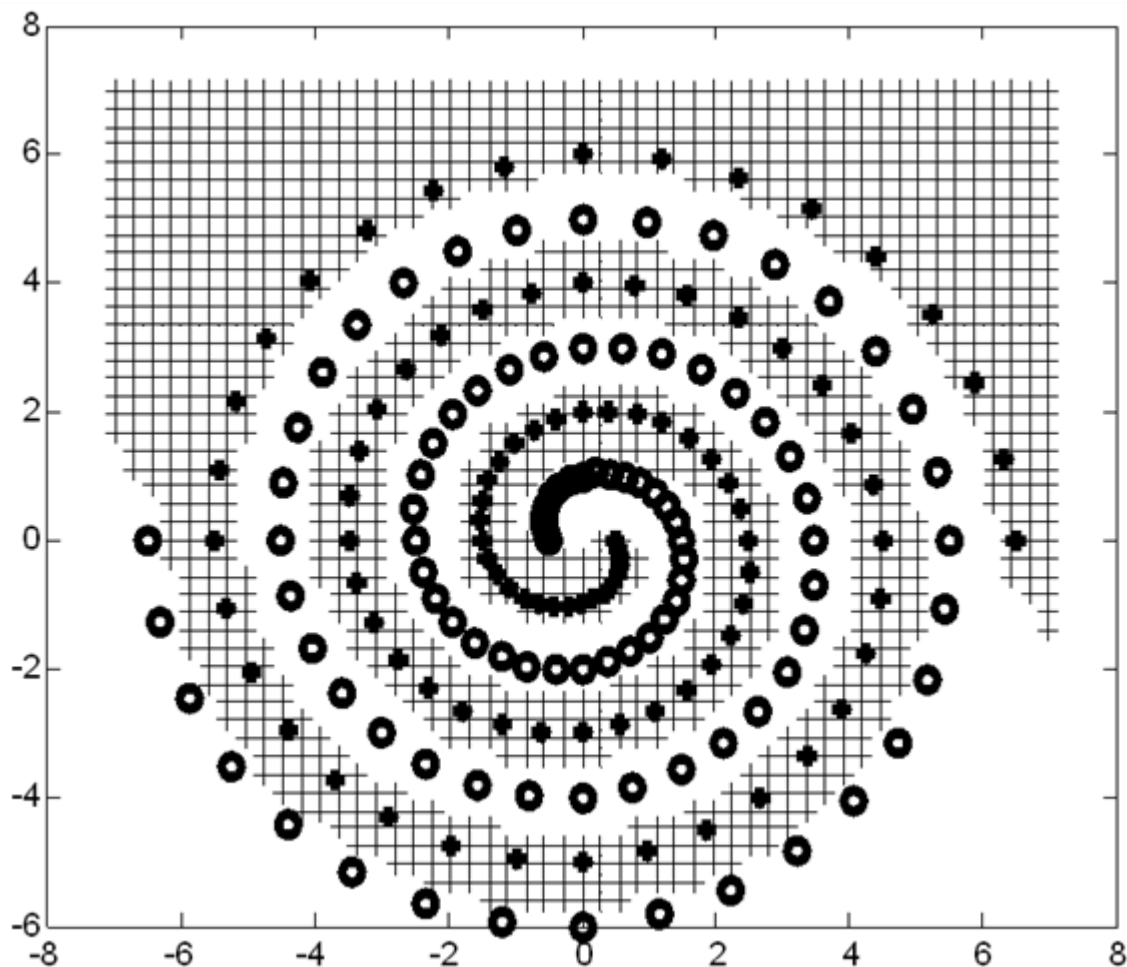
A 6.5 ábrán a nemlineáris SVM osztályozás illusztrálására gyakran használt kettős spirál feladat egy megoldását mutatjuk be. A szeparálandó két osztály két egymásba tekeredő spirál pontjaival van megadva, melyek tanítópontokként szerepelnek. A két osztályba tartozó tanítópontokat (+), illetve (o) jelek jelölik az ábrán. A feladat tehát olyan elválasztó görbe meghatározása, mely a két spirál pontjait szeparálja. A feladatot, mint benchmark problémát gyakran használják osztályozó rendszerek, köztük neuronhálók teljesítőképességének összehasonlítására. Ezt az indokolja, hogy bár a feladat nagyon egyszerűnek tűnik, megoldása nem könnyű, pl. egy MLP számára különösen nehéz.

6.1 példa

Példaként oldjuk meg a XOR problémát szupport vektor géppel. Minthogy a feladat lineárisan nem szeparálható, csak a nemlineáris változat használható. Első lépésként meg kell választani a kernel függvényt. A feladat viszonylagos egyszerűsége miatt az eddig említett kernel függvények bármelyike használható. Jelen esetben használjunk polinomiális kernelt: $K(\mathbf{x}, \mathbf{x}) = (\mathbf{x}^T \mathbf{x} + 1)^2$. A polinomiális kernel előnye, hogy ebben az egyszerű esetben még a jellemzőtérre való leképezést biztosító $\Phi(\mathbf{x})$ függvények is meghatározhatók. A jellemzővektorra a következő adódik:

$$\Phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T \quad (6.61)$$

6.5. ábra - A kettős spirál probléma megoldása SVM-el. Az SVM a világos és a négyzethálós tartományokat különböztette meg, mint a két osztályba tartozó pontokat.



Látható, hogy a jellemzőtér egy 6-dimenziós tér (5 dimezió + az eltolás). Az osztályozási feladat megoldásához fel kell írunk és meg kell oldanunk a duális egyenletet.

$$Q(\alpha) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2} \sum_{i=1}^4 \sum_{j=1}^4 \alpha_i \alpha_j d_i d_j K(\mathbf{x}_i^T \mathbf{x}_j), \quad (6.62)$$

a következő feltételek mellett:

$$\sum_{i=1}^4 d_i \alpha_i = \alpha_1 - \alpha_2 + \alpha_3 - \alpha_4 = 0, \quad \text{és} \quad 0 \leq \alpha_i, \quad i = 1, 2, \dots, 4 \quad (6.63)$$

A kvadratikus egyenlet felírásánál feltételeztük, hogy a bemenetek és a kimenetek nem bináris, hanem bipoláris megadásúak, vagyis a XOR probléma leképezése a következő:

Tanítópont index i	$[\mathbf{x}_1, \mathbf{x}_2]$	d
1	[1, 1]	1
2	[1, -1]	-1
3	[-1, -1]	1
4	[-1, 1]	-1

A kvadratikus programozási feladat megoldásához fel kell írni a kernel mátrixot. Mivel a XOR problémánál összesen 4 pontunk van, melyek mind tanítópontok, a kernel mátrixra a következő adódik:

$$\mathbf{K} = \begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix} \quad (6.64)$$

A kvadratikus programozás eredményeképpen azt kapjuk, hogy mind a négy α azonos értékű lesz, és mind különbözik nullától: $\alpha_1^* = \alpha_2^* = \alpha_3^* = \alpha_4^* = 1/8$. Ez azt jelenti, hogy az összes pont szupport vektor, ami a feladat ismeretében nem meglepő. A szupport vektoroknál a háló válasza alapján az eltolásérték is meghatározható, ami $b^*=0$ -ra adódik. A háló válasza a kernel térben tehát:

$$y(\mathbf{x}) = \text{sign} \left[0,125 \sum_{i=1}^4 d_i (\mathbf{x}_i^T \mathbf{x} + 1)^2 \right] \quad (6.65)$$

A szupport vektor gépeknél a jellemzőtérbeli reprezentáció és megoldás általában nem határozható meg, és nincs is rá szükség. Ebben az egyszerű példában azonban ismerjük a jellemzőtérre való leképezést biztosító $\varphi(\mathbf{x})$ függvényeket, és a Lagrange multiplikátorok ismeretében a $\mathbf{w}^*$ jellemzőtérbeli súlyvektor is meghatározható:

$$\mathbf{w}^* = \sum_{i=1}^4 \alpha_i^* d_i \mathbf{x}_i = [0 \quad 0 \quad 0 \quad 1/\sqrt{2} \quad 0 \quad 0]^T \quad (6.66)$$

Ennek ismeretében a megoldás a jellemzőtérben is megkapható. A döntési függvény:

$$g(\mathbf{x}) = x_1 x_2. \quad (6.67)$$

Arra az érdekes eredményre jutottunk, hogy bár a választott kernel által meghatározott nemlineáris transzformáció hatdimenziós jellemzőteret definiál, a döntés egyetlen dimenzió alapján meghozható, melyet a két bementi komponens szorzata határoz meg. Ez egyben arra is példa, hogy előfordulhat olyan eset, amikor a „dimenziónövelő” nemlineáris transzformáció valójában kisebb dimenziójú térbe képez le, mert a lineáris szeparálás ott is lehetséges. Ez természetesen általánosan nem igaz, de feladattól függően, ha megfelelő nemlineáris transzformációt választunk, előfordulhat. A nemlineáris SVM a jellemzőtérben biztosítja a maximális margót, ami itt $\sqrt{2}$ -re adódik.

3.4. Szupport vektor gépek regressziós feladatra

A szupport vektor gépek az osztályozási feladatokon túl regressziós feladatokra is alkalmazhatók. A regressziós vagy függvényapproximációs feladatok esetén azonban nem definiálható az osztályozásnál értelmezett margó, azaz az osztályozási tartalék fogalma. Ennek megfelelően szükség van a regressziós feladat egy olyan értelmezésére, felírására, ami lehetővé teszi az SVM kiegészítő feltételének felhasználását.

Hasonlóan az osztályozónál bemutatottakhoz itt is lehetőség lenne először a lineáris esetet, majd annak nemlineáris kiterjesztését tárgyalni, de a tömörebb tárgyalás kedvéért itt közvetlenül a nemlineáris (azaz a jellemzőtérben lineáris) regressziót mutatjuk be.

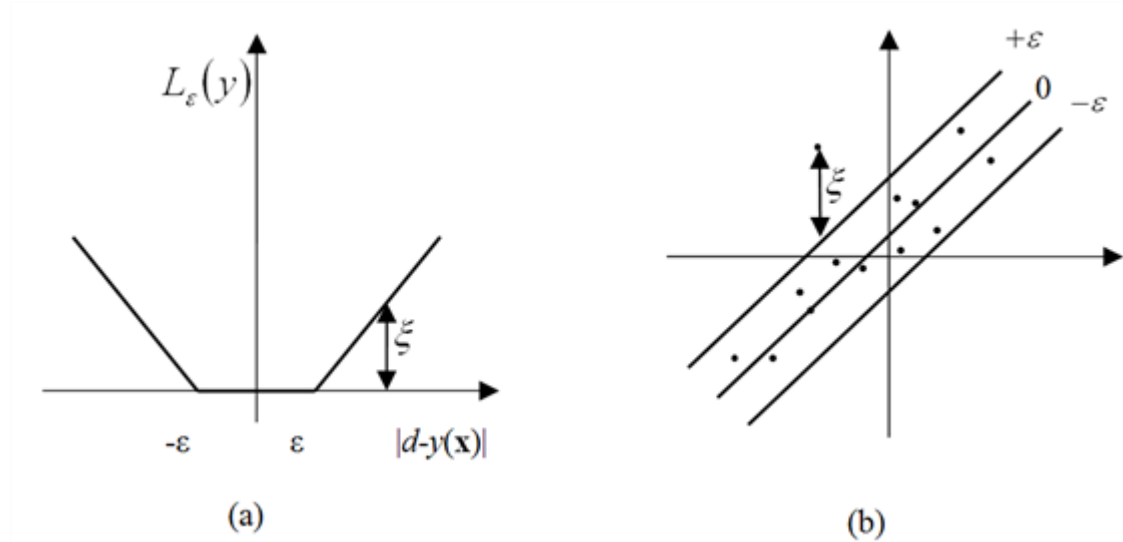
Az osztályozós és a regressziós feladatok között a kapcsolat viszonylag könnyen megteremthető, ha a függvényapproximációs feladatoknál az átlagos négyzetes hiba helyett egy ún. ε -érzékenységi sávval rendelkező abszolútérték hibafüggvényt (ε -insensitive loss function) alkalmazunk az eltérés mérésére.

$$L_\varepsilon(y) = \begin{cases} 0 & \text{ha } |d - y(\mathbf{x})| < \varepsilon \\ |d - y(\mathbf{x})| - \varepsilon & \text{egyébként} \end{cases} \quad (6.68)$$

Az ε érzékenységi veszteségfüggvény felhasználásával tulajdonképpen az osztályozási esethez nagyon hasonló probléma áll elő. Azokra a mintákra, melyek az érzékenységi sávon belül esnek, a hiba értéke 0, míg a kívül eső pontok hibáját büntetjük. Ez megfeleltethető egy olyan osztályozási feladatnak, ahol a helyes osztályozás hibája 0, míg biztonsági sávba eső vagy a rosszul osztályozott pontokat pedig gyengítő ($\frac{\varepsilon}{2}$) változók bevezetésével büntetjük. Ennek megfelelően az alábbiakban bemutatott regressziós megoldást a lineárisan nem szeparálható osztályozási feladatra kapott eredményekkel célszerű összevetni. Hibafüggvényként az abszolútérték-függvény használatát az is indokolja, hogy ekkor a megoldás az esetlegesen jelen lévő kilógó adatokra (outlier) kevésbé lesz érzékeny, mint a gyakran alkalmazott négyzetes hibafüggvényénél. Ennek

ellenére szokás ε -érzékletlenségi sávú négyzetes hibafüggvényt is alkalmazni. Jelen tárgyalásnál azonban megmaradunk a Vapnik féle alapg megoldásnál, ahol az abszolútérték-függvény szerepel.

6.6. ábra - Az ε -érzékletlenségi sávval rendelkező abszolútérték hibafüggvény és alkalmazása.



Írjuk fel a háló kimenetét. Ez most is nemlineáris bázisfüggvények lineáris kombinációjaként áll elő, azaz a $\varphi(\mathbf{x})$ -ek által reprezentált (M -dimenziós) jellemzőtérben lineáris megoldást keresünk:

$$y(\mathbf{x}) = \sum_{j=1}^M w_j \varphi_j(\mathbf{x}) + b = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) + b, \quad \mathbf{w} = [w_1, w_2, \dots, w_M]^T,$$

$$\boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \dots, \varphi_M(\mathbf{x})]^T \quad (6.69)$$

A kielégítendő feltételek (a gyengítő változók bevezetésével) a következők:

$$\begin{aligned} d_i - (\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b) &\leq \varepsilon + \xi_i, \\ (\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b) - d_i &\leq \varepsilon + \xi'_i, \\ \xi_i &\geq 0, \\ \xi'_i &\geq 0, \end{aligned} \quad i = 1, 2, \dots, P \quad (6.70)$$

A minimalizálandó költségfüggvény:

$$J(\mathbf{w}, \xi, \xi') = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \left(\sum_{i=1}^P (\xi_i + \xi'_i) \right) \quad (6.71)$$

a (6.70) egyenlőtlenségek által megfogalmazott mellékfeltételekkel ($i = 1, \dots, P$):

A megfelelő Lagrange kritérium az $\alpha_i, \alpha'_i, \gamma_i, \gamma'_i \geq 0$ Lagrange multiplikátorok felhasználásával:

$$\begin{aligned} L(\mathbf{w}, b, \xi, \xi', \boldsymbol{\alpha}, \boldsymbol{\alpha}', \boldsymbol{\gamma}, \boldsymbol{\gamma}') &= C \sum_{i=1}^P (\xi_i + \xi'_i) + \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ &- \sum_{i=1}^P \alpha_i [\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b - y_i + \varepsilon + \xi_i] \\ &- \sum_{i=1}^P \alpha'_i [y_i - \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) - b + \varepsilon + \xi'_i] - \sum_{i=1}^P (\gamma_i \xi_i + \gamma'_i \xi'_i) \end{aligned} \quad (6.72)$$

A Lagrange függvényt kell $\mathbf{w}$, b és a ξ és ξ' gyengítő vektorváltozók szerint minimalizálni, $\boldsymbol{\alpha}$, $\boldsymbol{\alpha}'$, valamint $\boldsymbol{\gamma}$ és $\boldsymbol{\gamma}'$ szerint maximalizálni. Elvégezve a deriválást, és a deriváltak értékét nullává téve, a következő összefüggéseket kapjuk:

$$\begin{aligned} \mathbf{w} &= \sum_{i=1}^P (\alpha_i - \alpha_i') \varphi(\mathbf{x}_i) \\ \gamma_i &= C - \alpha_i \\ \gamma_i' &= C - \alpha_i' \end{aligned} \quad (6.73)$$

Behelyettesítve ezeket a Lagrange függvénybe jutunk a duális feladatra:

$$\begin{aligned} Q(\boldsymbol{\alpha}, \boldsymbol{\alpha}') &= \sum_{i=1}^P d_i (\alpha_i - \alpha_i') - \varepsilon \sum_{i=1}^P (\alpha_i + \alpha_i') \\ &- \frac{1}{2} \sum_{i=1}^P \sum_{j=1}^P (\alpha_i - \alpha_i') (\alpha_j - \alpha_j') K(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \quad (6.74)$$

a

$$\sum_{i=1}^P (\alpha_i - \alpha_i') = 0, \quad 0 \leq \alpha_i \leq C, \quad 0 \leq \alpha_i' \leq C, \quad i=1, 2, \dots, P \quad (6.75)$$

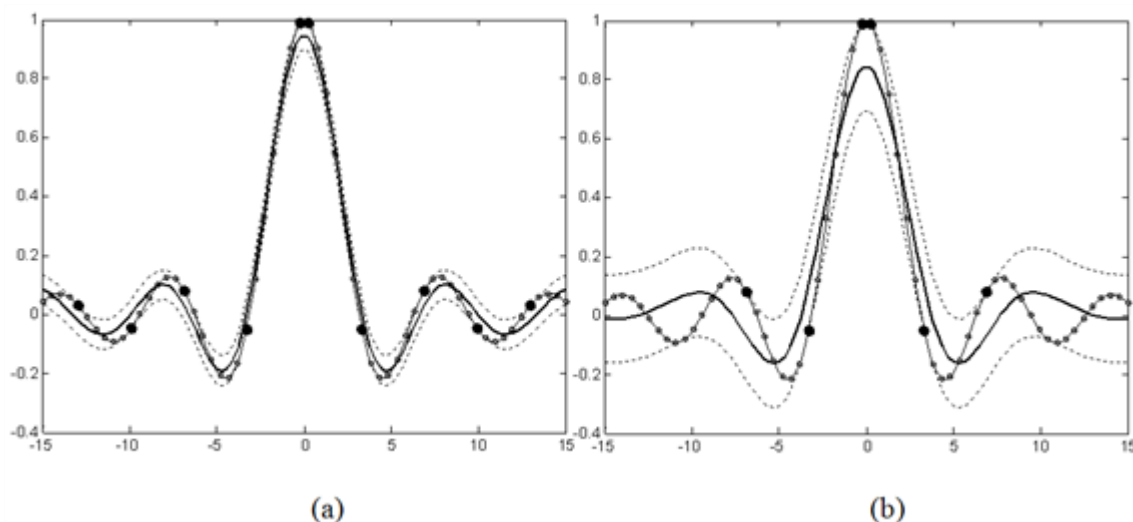
megkötésekkel. Az eredmény a kvadratikus programozással meghatározható α_i Lagrange multiplikátorok, a kiszámított b és a $K(\mathbf{x}, \mathbf{x}_i)$ kernel függvény alapján:

$$y(\mathbf{x}) = \sum_{i=1}^P (\alpha_i - \alpha_i') K(\mathbf{x}, \mathbf{x}_i) + b \quad (6.76)$$

Csakúgy, mint az osztályozási esetben, az SVM modell most is ritka, ugyanis csak azok a tanítópontok vesznek részt a kimenet előállításában, melyekre $(\alpha_i - \alpha_i') \neq 0$. Fontos megjegyezni, hogy most is a kernel függvényből indulunk ki, így a jellemzőtér bázisfüggvényeit a kernel függvény implicit módon definiálja. Ez a jellemzőtér azonban nagyon nagy dimenziós, akár végtelen dimenziós is lehet, ezért míg lineáris esetben a probléma az eredeti térben is megoldható, nemlineáris esetben a jellemzőtérben már nem minden esetben lehetséges a megoldás.

Figyeljük meg, hogy a regressziós megoldásnál az SVM kialakításában az osztályozós esethez képest eggyel bővül a hiperparaméterek száma, hiszen az ε érzéketlenségi paraméter megadására is szükség van. Az ε hiperparaméternek az approximációra gyakorolt hatása zajmentes tanítópontok esetén szemléletesen is értelmezhető (6.7 ábra).

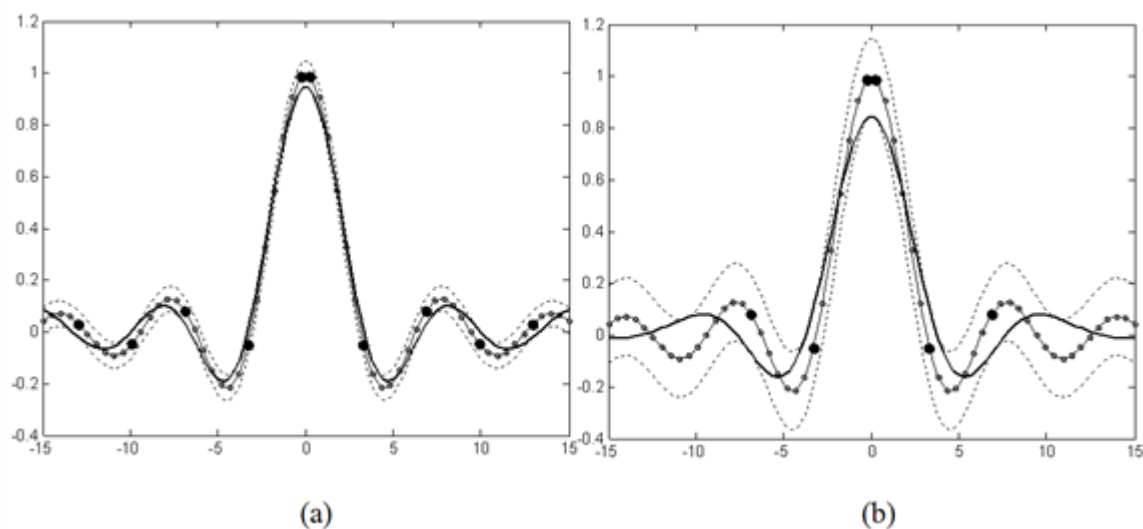
6.7. ábra - Az ε -érzéketlenségi sáv szerepe a függvényapproximációban (a) $\varepsilon=0,05$, (b) $\varepsilon=0,15$. A vastagabb folytonos vonal a háló választát, a szaggatott vonalak az ε szélességű sáv széleit mutatják. A vékonyabb folytonos vonal az approximálandó függvényt a rajta lévő tanítópontokkal (kisebb pontok) mutatja. A nagyobb fekete pontok a szupport vektorok



$\pm \varepsilon$ ábrán látható, hogy ha a megoldás köré egy ε szélességű sávot húzunk úgy, hogy az SVM válaszfüggvényét ε -nal eltoljuk, akkor minden tanítópont ezen a sávon belül helyezkedik el. A sávhatárra eső pontok lesznek a szupport vektorok (az ábrán a nagyobb fekete pontok). Az ábra azt is mutatja, hogy kisebb szélességű sáv több, a nagyobb szélességű pedig kevesebb szupport vektort eredményez. Ezt úgy biztosítja az SVM, hogy minél nagyobb ε értéke, annál simább megoldást kapunk. Ezt a hatást illusztrálja a 6.8 ábra is. Ezen az ábrán is szerepel egy ε szélességű sáv, ami azonban most az approximálandó függvényt veszi közre, és annak ε -nal történő eltolásával kapható. Az SVM zajmentes esetben mindig olyan megoldást ad, amely ebbe az ε szélességű sávba befér, miközben a válasz a lehető legsimább. Látható, hogy a szélesebb sáv simább, a keskenyebb kevésbé sima megoldást eredményez.

Fentiek alapján ε hatása zajmentes esetben jól követhető, és ennek alapján a megfelelő ε érték megválasztható, hiszen minél kisebb ez az érzéketlenségi sáv, annál pontosabban próbálja a modell a tanítópontokra illeszteni a megoldást. A valós feladatokban azonban szinte mindig zajos bemenetekkel van dolgunk.

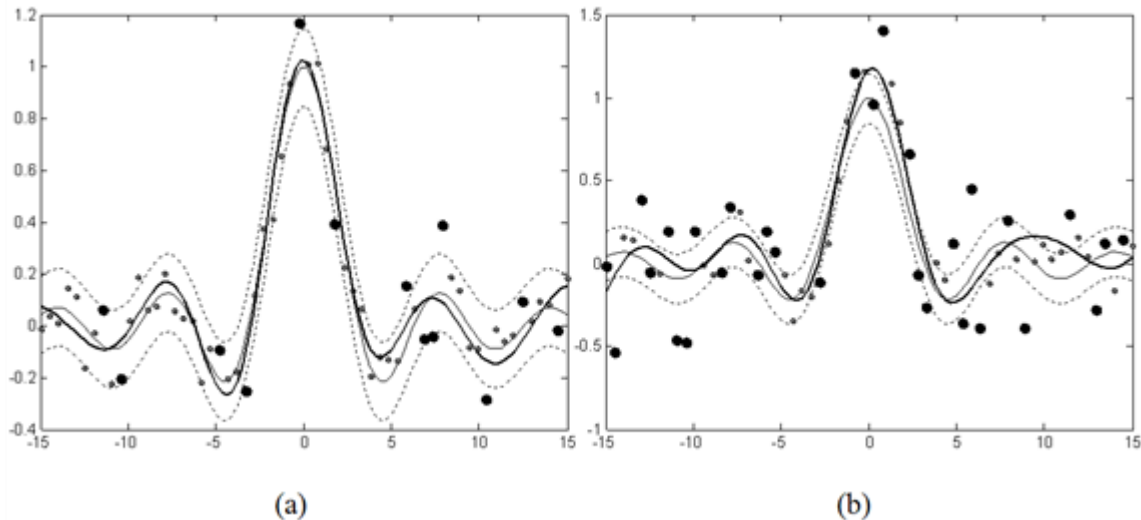
6.8. ábra - Az ε -érzéketlenségi sáv hatása a válasz simaságára függvényapproximációban (a) $\varepsilon=0,05$, (b) $\varepsilon=0,15$. A vastagabb folytonos vonal a hálózati választ, a szaggatott vonalak az ε szélességű sáv széleit mutatják. A vékonyabb folytonos vonal az approximálandó függvényt a rajta lévő tanítópontokkal mutatja. A nagyobb fekete pontok a szupport vektorok.



Ebben az esetben a kis ε arra készítené a modellt, hogy minden pontot jól (kis hibával) közelítsen, ami túltanuláshoz vezethet, holott a cél itt éppen egy simább függvény illesztése lenne. Nagyobb zaj esetén tehát célszerű nagyobb ε -t, szélesebb érzéketlenségi sávot használni. Zajos tanítópontok mellett ε hatására a 6.9 ábra mutat példát.

Az SVM minél simább válasza annak a következménye, hogy a súlyvektor normáját minimalizáljuk. A költségfüggvényben szereplő $\mathbf{w}^T \mathbf{w}$ tag osztályozós esetben a maximális margóért, regressziós esetben pedig a sima és minél laposabb megoldásért felelős.

6.9. ábra - A sinc(x) regressziója zajos adatok alapján. (Gauss kernel; $\sigma=\pi$; $C=10$, $\varepsilon=0,15$. A Gauss zaj szórása (a) $\sigma_n=0,01$, (b) $\sigma_n=0,04$.



A megoldás hibája, valamint komplexitása közötti kompromisszumot regressziós esetben is a C paraméter állítja be, azaz az ε és a C hiperparaméterek értékét összehangoltan kell megválasztani. Általánosságban is elmondható, hogy a szupport vektor gépek hiperparamétereinek (az ε , a C , valamint a kernel függvény paramétereinek – Gauss kernel esetén a σ szélességparaméternek) a megválasztása igen összetett feladat, mert az egyes paraméterek nem függetlenek egymástól, így a lehetséges beállítások terében kell egy jó, esetleg optimális kombinációt megtalálni. A kernel paraméterek hatása – ha Gauss kernelt választunk – hasonló a Gauss bázisfüggvényes RBF hálók szélességparaméterének hatásához: a nagy σ simább, míg a kis σ változékonyabb (erősen lokalizált) közelítést eredményez, ami hatással van például a megfelelő C értékre is. C értékét az előbbi esetben kisebb, míg az utóbbi esetben várhatóan nagyobbra kell megválasztani. A hiperparaméterek megfelelő megválasztásával számos irodalom pl. [Sch02], [Cha02], [Kwo03] foglalkozik.

3.4.1. Az SVM működése

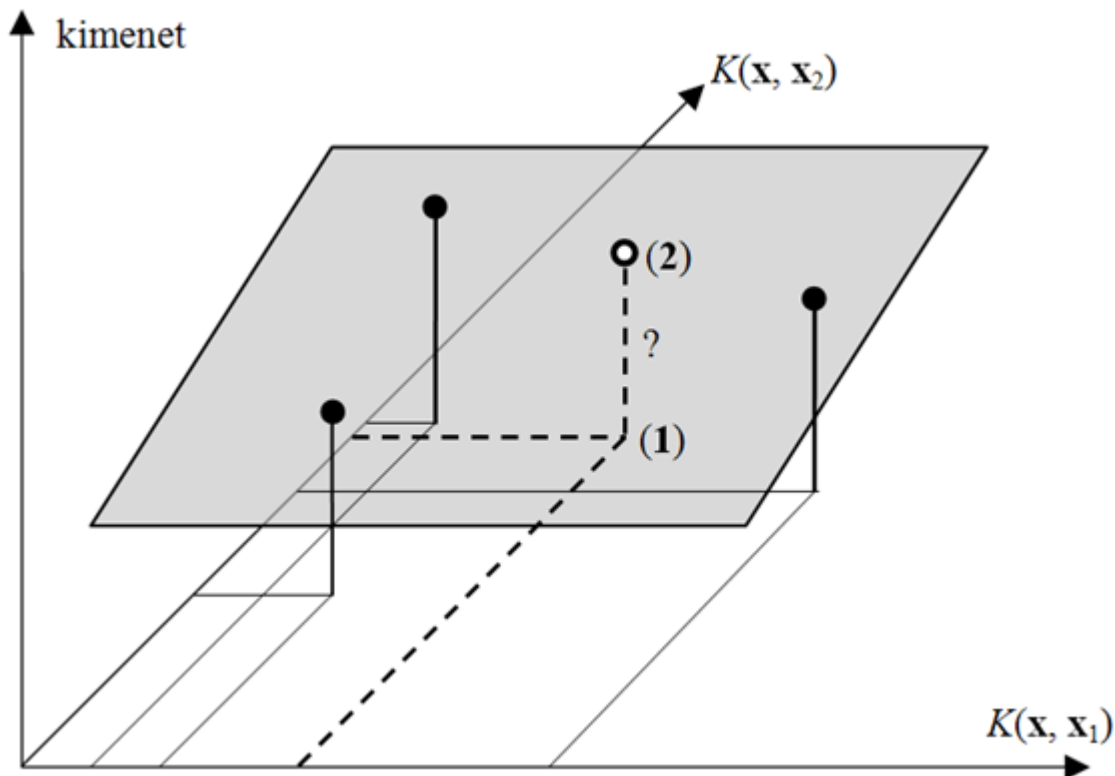
Láttuk, hogy a nemlineáris problémák lineáris kezelésének kulcsa, hogy az eredeti feladatot egy olyan többdimenziós térbe transzformáljuk, ahol az már lineárisan megoldható. A kernel gépek mindezt a kernel térben tudják megoldani. A szupport vektor gépek ezt kiegészítik azzal, hogy a lineáris megoldás kiszámításához olyan módszert adnak, ami alapján a lehetséges megoldások közül a legjobb számítható ki.

A működés mechanizmusának jobb megértéséhez a kernel tér alábbi jellemzőit hangsúlyozzuk:

- A kernel tér a jellemzőtér, illetve az ehhez tartozó leképezés konkrét ismerete nélkül is elérhető.
- A kernel tér véges dimenziójú.
- A kernel függvény és a kernel középpontok (szupport vektorok) ismeretében minden bemeneti vektor leképezhető a kernel térbe.

A probléma a kernel térben már lineáris, így itt az SVM-nek megfelelő optimális hipersíkot kell meghatározni. Ez osztályozás esetén megfelel a kernel térbe leképezett két pontthalmazt (a jellemző térben) maximális margóval elválasztó hipersíknak, regresszió esetén pedig a kernel térbeli pontokra illesztett hipersíknak. A tanítási lépés során tehát a hipersíkot leíró egyenlet szabad paramétereit, az α_i Lagrange multiplikátorokat és b -t kell meghatározni.

6.10. ábra - A kernel térbeli lineáris megoldás illusztrációja két szupport vektor és egydimenziós kimenet esetén. A fekete pontok a tanító mintákat illusztrálják, míg a fehér pont egy kiszámítandó kimenetet illusztrál.



Amennyiben a szupport vektorok száma M , a kernel tér $(M+1)$ dimenziós, ahol M koordinátát a $K(\mathbf{x}_i, \mathbf{x}_j)$ kernel leképezéssel számítunk ki, míg a további egy dimenzió a kimenetnek felel meg. A keresett hipersík a leképezett tanítópontok és a hozzájuk tartozó kimenet közötti összerendelést definiálja. A működés szemléltetéséhez tekintsük a regressziós esetet. A megtanított SVM működését a 6.10 ábra szemlélteti.

Az SVM modell eredményét leíró

$$y(\mathbf{x}) = \sum_{i=1}^P (\alpha_i - \alpha_i') K(\mathbf{x}, \mathbf{x}_i) + b \quad (6.77)$$

egyenlet alapján a működés a következő. A vizsgált $\mathbf{x}$ bemeneti vektort leképezzük a kernel térbe. Ez a szupport vektoroknak megfelelő számú $K(\mathbf{x}, \mathbf{x}_i)$ kernel érték kiszámítását jelenti. Az eredmény a hipersík által definiált pont a leképezéssel kapott pozícióban.

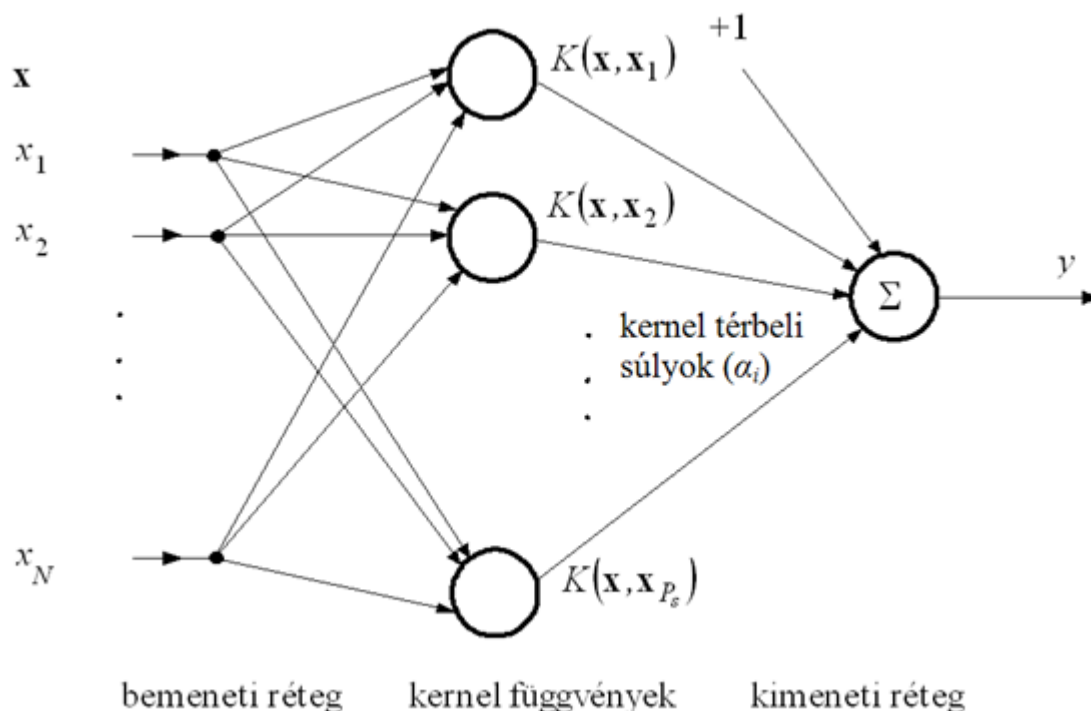
A regressziós feladat megoldása során az SVM által konstruált optimális hipersík – amennyire lehetséges – jó választ ad a tanító pontokra, azaz illeszkedik rájuk. A modell akkor működik megfelelően, ha nemcsak a tanítópontok esetében, hanem később a teszteléshez használt pontok esetén is a hipersíkra esik a kimenet, vagy másképpen a hipersík értéke jól közelíti a valós kimenet értéket.

3.5. Az SVM neurális értelmezése

Bár a gyakorlatban a szupport vektor gépeket ritkán alkalmazzák neurális hálózat alakjában, ez az értelmezés hasznos, mert egyszerűbb tárgyalásmódot tesz lehetővé a pusztán matematikai megközelítésnél.

A szupport vektor gépek tanítása, és használata is megfelelő számítások sorozata, ahol a válasz számításának összefüggése pontosan megfeleltethető egy egy-rejtett-rétegű neurális hálózathoz. A rejtett réteg tipikusan nemlineáris neuronokat tartalmaz. A 6.11 ábra egy szupport vektor gépnek megfeleltethető neurális hálózatot mutat be.

6.11. ábra - A szupport vektor gépnek megfeleltethető neurális hálózat



A bemenet egy N -dimenziós vektor. A nemlineáris kernel függvényeket a rejtett réteg neuronjai tartalmazzák. Ezen neuronok száma megegyezik a szupport vektorok számával (P_s). A hálózat válasza (y) a rejtett réteg neuronjai kimenetének súlyozott összege. A súlyokat a tanítás során kiszámított Lagrange multiplikátor értékek (α_i) határozzák meg. Ennek megfelelően, minél kisebb a hálózat, annál kevesebb számításra van szükség a válasz megadásához, így a cél a lehető legkisebb hálózat elérése.

Láthatóan a kernel gép neurális interpretációja egy struktúrájában a bázisfüggvényes hálózatokkal gyakorlatilag tökéletesen megegyező háló. Ez azonban csak formai hasonlóság, a kétféle megközelítés közötti lényeges különbséget nem szabad szem elől téveszteni.

3.6. Az SVM hatékonyabb megvalósításai

A szupport vektor gépek gyakorlati felhasználásának egyik legnagyobb akadálya az SVM tanításához szükséges kvadratikusan programozás nagy algoritmikus komplexitása, valamint a kernel mátrix nagy mérete. Az alábbiakban olyan módszereket mutatunk be röviden, melyek a számítás gyorsítását, valamint a szükséges memória méretének csökkentését célozzák. Ezt a feladatot kétféleképpen oldhatjuk meg. Ezek:

- az eredeti felírás megoldásához szükséges számítások (a kvadratikusan programozás) felgyorsítása, valamint ezek kisebb tárterületen történő megoldása,
- az eredeti felírás módosítása úgy, hogy könnyebben megoldható legyen, s így gyorsabb algoritmussal hasonló tulajdonságú megoldásra vezessen. Itt olyan módosításról beszélünk, ami az eredeti SVM optimumot adja, tehát a feladatot nem, csak annak matematikai felírását változtatja. (A későbbiekben olyan SVM változatok bemutatására is sor kerül, ahol azonos elméleti háttér mellett már egy másik probléma (optimum) kerül kiszámításra.)

3.6.1. A QP hatékonyabb megoldása

A tanítási sebesség növeléséhez az SVM tanításban használt kvadratikusan programozási lépést kell gyorsabban (és kevesebb memória felhasználásával) végrehajtani. Erre számos hatékony QP megoldó áll rendelkezésre. Ezek iteratív eljárások, amelyek az eredeti optimalizálási feladatot kisebb részfeladatok sorozataként oldják meg. A részfeladatok kiválasztásának alapját az optimalizálási feladat KKT (Karush-Kuhn-Tucker) feltételei jelentik. Azok a minták, amelyek teljesítik az optimumra vonatkozó KKT feltételeket, nem vesznek részt az optimalizálásban, míg a feltételeket megszegő pontok igen. A KKT feltételek osztályozás esetére az alábbiak $\forall i$ -re:

$$\begin{aligned}\alpha_i = 0 &\Rightarrow d_i p(\mathbf{x}_i) \geq 1 \\ 0 < \alpha_i < C &\Rightarrow d_i p(\mathbf{x}_i) = 1 \\ \alpha_i = C &\Rightarrow d_i p(\mathbf{x}_i) \leq 1\end{aligned}\quad (6.78)$$

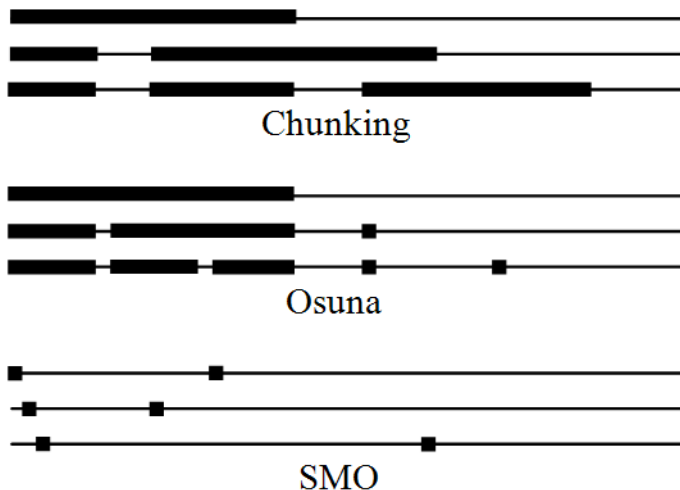
Egy mintapont akkor és csak akkor lehet optimum pontja (6.53)-nak, ha teljesülnek a KKT feltételek és a kernel mátrix pozitív semidefinit.

A szupport vektor gépekhez használt három legfontosabb, azonos alapelvek szerint működő, egymást továbbfejlesztő gyorsítási algoritmus [Sch99]:

- **Szeletelés** - A szeletelési (*chunking*) eljárás [Vap79] alapja, hogy a nulla α_i -khöz tartozó minták nem befolyásolják a megoldást. Először a minták egy részhalmazán oldjuk meg a QP-t, majd az adott alproblémából mindig a szupport vektorokat tartjuk meg. Ezekhez azt az M tanítómintát veszünk hozzá, amelyek a leginkább megszegik a KKT-feltételeket, majd újra megoldjuk a kvadratis optimalizálást. Ha M -nél kevesebb minta szegi meg a feltételeket, akkor mindet beillesztjük a következő alfeladatba. Az iteráció végére az összes nem nulla multiplikátort meghatározzuk, így az eredeti feladatot is megoldottuk. A kezelt alprobléma nagysága az iterációk során általában nő. A chunking nem jól használható, ha nagyon sok a tanítópont vagy sok a nem nulla α_i , hiszen ebben az esetben még mindig nagy a memóriaigény.
- **Osuna algoritmus** - Az előzőhöz hasonló módszer Edgar Osuna dekompozíciós módszere [Osu99], amely mindig egy állandó méretű részhalmazon dolgozik, miközben a többi változót rögzített értéken tartja. A kvadratis programozást így mindig egy rögzített méretű munkahalmazon oldja meg, és kisebb optimalizálási feladatok sorozatán keresztül jut el a teljes megoldásig.
- **SMO** - A Sequential Minimal Optimization (SMO) olyan dekompozíciós eljárás [Pla99], amely során iterációnként mindössze két α_i változó optimalizálása történik meg. Előnye, hogy teljesen kikerüli a QP számítást, mert a két változó analitikusan is optimalizálható. Két változó analitikus optimalizálása után a következő két α_i kiválasztására kerül sor, amely heurisztikusan történik.

A 6.12. ábra az említett három alapeljárás szeletelési stratégiáját mutatja. Mindhárom módszernél három lépést mutat az ábra, ahol a legfelső a kezdőlépés. A vízszintes vonalak minden esetben a tanító készletet reprezentálják, a megvastagított fekete szakaszok pedig az adott iterációs lépésben optimalizált Lagrange multiplikátorokat jelzik.

6.12. ábra - A legelterjedtebben használt QP megoldók „szeletelési” (dekomponálási) stratégiái.



3.6.2. Az SVM módosítása

Az SVM módosított verziói közül a hatékonyság és a memóriaigény szempontjából Olvi Mangasarian és munkatársainak a munkái a legfontosabbak.

3.6.2.1. SSVM

Az SSVM (Smooth Support Vector Machine) [Lee01a] az eredeti feladatot egy olyan optimalizálási feladatra fogalmazza át, ami egyszerű gradiens módszerekkel megoldható. Az így kapott kvadratikusan hibafelület minimuma (itt már nincsenek feltételek, mint az eredeti QP-ben) megfelel az SVM által megoldott kvadratikusan programozási feladat minimumának.

3.6.2.2. RSVM

Az RSVM (Reduced Support Vector Machine) [Lee01b] valójában az SSVM egy kiterjesztése, ahol az összes mintapontnak csak egy részhalmazából kerülhetnek ki szupport vektorok. Az eljárás ezzel valójában tovább redukálja a megoldandó feladat méretét, hiszen az eredeti kvadratikusan kernel mátrix helyett egy kisebb mátrixot kell csak előállítani. Ennek a kulcsa, hogy ellentétben a kvadratikusan programozással, ami csak kvadratikusan mátrixra futtatható, a gradiens módszerekkel már egy ilyen egyenletrendszer is megoldható.

4. SVM változatok

A szupport vektor gépek alkalmazásának legjelentősebb akadálya a nagy memória- és számítási komplexitás. Mint láttuk, ennek oka, hogy az SVM tanítása során egy nagy számításigényű kvadratikusan programozási feladatot kell megoldani. Az előzőekben röviden összefoglalt néhány hatékony eljárás vagy a kvadratikusan programozás gyors megvalósításával, vagy az SVM optimalizálási feladatának módosításával ér célzt. Ezek a megoldások azonban mind az eredeti feladat megoldását keresik.

A probléma egy másik lehetséges megközelítése, ha az optimalizálási feladatot módosítjuk, az SVM elméleti hátterének, valamint előnyeinek lehetőség szerinti megtartása mellett. Ekkor azonban már nem az eredeti SVM feladat megoldására törekszünk. A biztonsági sáv maximalizálása, a duális felírás és a kernel trükk alkalmazása jelentik a módszer azon magját, amelyet meghagyva a számítási feladat átfogalmazásával az eredeti módszerhez hasonló kernel gépeket kaphatunk. Az SVM változatokkal szembeni fő elvárás, hogy egyszerűsítsék a szükséges számításokat, mindemellett megtartsák az eredeti eljárás jó tulajdonságait, mind a jó általánosítóképességet, mind a kapott megoldás ritkaságát (*sparseness*).

Az egyik legelterjedtebb ilyen módosítás a négyzetes hibára optimalizált szupport vektor gép, az LS-SVM (Least Squares Support Vector Machine) [Suy00], [Suy02]. Ez a megoldás az SVM-től eltérően a klasszikus neuronhálónál szokásos négyzetes hibafüggvénnyel dolgozik, továbbá a kielégítendő feltételeket egyenlőségek formájában fogalmazza meg. Ennek köszönhetően a megoldás egy lineáris egyenletrendszer eredményeként áll elő, amelynek megoldása lényegesen egyszerűbb, mint a kvadratikusan programozási feladaté.

Az SVM változatok közül először a legelterjedtebb LS-SVM-et mutatjuk be, ami megfelel a statisztikából ismert ridge regresszióknak, amelyet később szintén bemutatunk röviden. A legkisebb négyzetes hiba alkalmazásából adódóan az eredeti LS-SVM és a ridge regresszió elveszíti a ritkasági tulajdonságot. Megfelelő módosításokkal, kiterjesztésekkel azonban itt is elérhető, hogy ritka megoldást kapjunk. A ritka megoldásra vezető egyik lehetséges kiterjesztés az LS^2 -SVM. Ennek tárgyalását a redukált rangú kernel ridge regresszió (*reduced rank kernel ridge regression*, *RRKRR*) bemutatása követi, ami más megközelítést alkalmazva, de ugyancsak ritka megoldást biztosít.

4.1. Az LS-SVM

A Johan Suykens által kidolgozott LS-SVM [Suy02] a hagyományos szupport vektor gépek olyan módosítása, ami az idő- és erőforrás-igényes kvadratikusan programozással szemben, egy lineáris egyenletrendszer megoldására vezet.

Mivel az LS-SVM a hagyományos SVM módosítása, a kiinduló ötletek, illetve összefüggések hasonlóak, ezért a továbbiakban elsősorban az alapváltozattól való eltérések bemutatására törekszünk. A fő különbség a két módszer között az ε -érzékenységi sávval rendelkező költségfüggvény négyzetes függvényre való lecserélése, valamint, hogy az SVM mellékfeltételeit megfogalmazó egyenlőtlenségeket egyenlőségekre cseréljük.

Az LS-SVM tárgyalásánál csak a legáltalánosabb nemlineáris változatot származtatjuk.

4.1.1. Az LS-SVM osztályozó

Az osztályozási feladat $d_i \in \{0,1\}$ egészokott: adott egy $\{\mathbf{x}_i, d_i\}_{i=1}^P$ tanítópont-halmaz, ahol $\mathbf{x}_i \in \mathbb{R}^n$ egy N -dimenziós bemeneti vektor, illetve az ehhez tartozó kívánt kimenet. A cél egy olyan modell létrehozása, ami jól reprezentálja a tanítópontok által leírt kapcsolatot.

Az osztályozó esetében az optimalizálási feladat az alábbi:

$$\min_{\mathbf{w}, b, \mathbf{e}} J(\mathbf{w}, b, \mathbf{e}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \frac{1}{2} \sum_{i=1}^P e_i^2 \quad (6.79)$$

a

$$d_i [\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b] = 1 - e_i, \quad i = 1, \dots, P \quad (6.80)$$

mellékfeltételekkel. A fenti képletben a négyzetes hibát megadó e_i értékek a négyzetes költségfüggvényből adódó hibaértékek. Szerepük az SVM-ben bevezetett e_i gyengítő változóknak felel meg. Látható, hogy a feltételekben az SVM osztályozóhoz képest (lásd (6.24) összefüggés) az egyenlőtlenség helyett egyenlőség szerepel, valamint hogy a célfüggvényben a hiba négyzete (e_i^2) jelenik meg.

Az osztályozó most is a szokásos bázisfüggvényes alakban írható fel:

$$y(\mathbf{x}) = \text{sign}[\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) + b] \quad (6.81)$$

ahol a $\boldsymbol{\varphi}(\cdot): \mathbb{R}^n \rightarrow \mathbb{R}^M$ leképezés az SVM-ben is használt nemlineáris leképezés egy magasabb, M -dimenziós jellemzőtérbe.

A fenti egyenletekből az alábbi Lagrange kritérium írható fel:

$$L(\mathbf{w}, b, \mathbf{e}, \boldsymbol{\alpha}) = J(\mathbf{w}, b, \mathbf{e}) - \sum_{i=1}^P \alpha_i \{ d_i [\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b] - 1 + e_i \}, \quad (6.82)$$

ahol az α_i értékek a Lagrange multiplikátorok, amelyek az egyenlőségi feltételek miatt itt pozitív és negatív értékeket is felvehetnek.

Az optimumra vonatkozó feltételek az alábbiak.

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} = 0 & \rightarrow \mathbf{w} = \sum_{i=1}^P \alpha_i d_i \boldsymbol{\varphi}(\mathbf{x}_i) \\ \frac{\partial L}{\partial b} = 0 & \rightarrow \sum_{i=1}^P \alpha_i d_i = 0 \\ \frac{\partial L}{\partial e_i} = 0 & \rightarrow \alpha_i = C e_i \quad i = 1, \dots, P \\ \frac{\partial L}{\partial \alpha_i} = 0 & \rightarrow d_i [\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b] - 1 + e_i = 0 \quad i = 1, \dots, P \end{aligned} \quad (6.83)$$

A fenti egyenletekből egy lineáris egyenletrendszer írható fel, mely az alábbi kompakt formában is megadható:

$$\begin{bmatrix} 0 & \mathbf{d}^T \\ \mathbf{d} & \boldsymbol{\Omega} + C^{-1} \mathbf{I} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}, \quad (6.84)$$

ahol:

$$\begin{aligned} \mathbf{d}^T &= [d_1, d_2, \dots, d_P] \\ \mathbf{1}^T &= [1, \dots, 1] \\ \boldsymbol{\alpha}^T &= [\alpha_1, \alpha_2, \dots, \alpha_P] \\ \Omega_{ij} &= d_i d_j \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j) = d_i d_j K(\mathbf{x}_i, \mathbf{x}_j) \quad i, j = 1, \dots, P \end{aligned} \quad (6.85)$$

és $\mathbf{I}$ egy megfelelő mére $\frac{\partial L}{\partial \alpha_i} = 0$ égmátrix. Látható, hogy az egyenletrendszer első sora a $\sum_{i=1}^P \alpha_i d_i = 0$ feltételt írja le, míg a további sorok ϵ_i feltételből adódó egyenleteket írják le, behelyettesítve a (6.83)-ban megadott derivált számításokból ϵ_i -re és ϵ_i -re származtatott összefüggéseket.

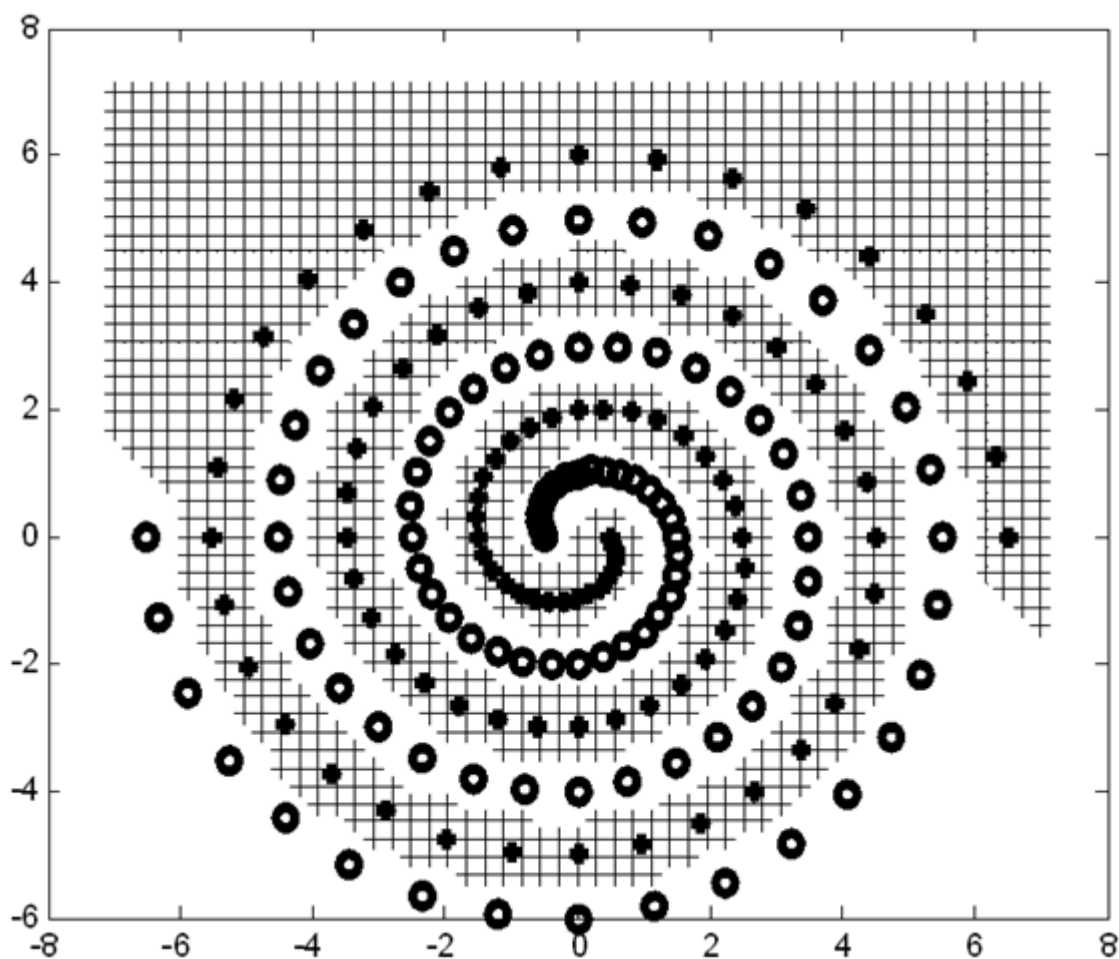
Az osztályozó válasza – hasonlóan a hagyományos SVM-hez – a következő alakban írható fel:

$$y(\mathbf{x}) = \sum_{i=1}^P \alpha_i d_i K(\mathbf{x}, \mathbf{x}_i) + b. \quad (6.86)$$

Az α_i súlyok és a b eltolásérték (*bias*) a fenti egyenletrendszerből számíthatók. Az eredmény jellemzője, hogy a megoldásban az összes tanítópont szerepel, tehát mind szupport vektor, azaz az egyenlőség használata miatt nincsenek nulla súlytényezők. Ezzel elveszítjük az SVM ritkasági tulajdonságát, nevezetesen, hogy a bemeneti vektoroknak csak egy kis része lesz szupport vektor. Az α_i súlyok nagyság szerinti sorba rendezéséből látszik, hogy melyek a fontosabb, illetve melyek a kevésbé fontos bemeneti vektorok. Ezen alapul a később bemutatásra kerülő komplexitás redukciós metszési (*pruning*) eljárás, ami az SVM ritkasági tulajdonságát adja vissza.

A 6.13 ábrán a szupport vektor gépek összehasonlításánál gyakran alkalmazott kettős spirál probléma LS-SVM-mel történő megoldását mutatjuk. Jól látható, hogy ennél az erősen nemlineáris feladatnál az LS-SVM – hasonlóan az SVM-hez – nagyon jól teljesít.

6.13. ábra - A kettős spirál probléma megoldása LS-SVM-el.



4.1.2. Az LS-SVM regresszió

Az LS-SVM – hasonlóan az eddig látott hálózatokhoz az osztályozás feladatán túl $\{ \mathbf{x}_i, d_i \}_{i=1}^P$ esetre is alkalmazható. Ekkor $\mathbf{x}_i \in \mathbb{R}^n$ igyeknek megfelelően egy függvény $d_i \in \mathbb{R}$ közelítése a cél adott tanítópont-halmaz alapján, ahol egy N -dimenziós bemeneti vektor és a kívánt kimenet.

A regresszió esetén az optimalizációs feladat az alábbi:

$$\min_{\mathbf{w}, b, \mathbf{e}} J(\mathbf{w}, b, \mathbf{e}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \frac{1}{2} \sum_{i=1}^P e_i^2 \quad (6.87)$$

a

$$d_i = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i, \text{ ahol } i = 1, \dots, P \quad (6.88)$$

feltételekkel.

A regresszió válasza a következő alakban írható fel:

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) + b, \quad (6.89)$$

ahol a $\boldsymbol{\varphi}(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^M$ nemlineáris transzformáció most is a jellemzőtérbe való leképezést biztosítja. Az osztályozás esethez hasonlóan itt is felírhatunk egy Lagrange függvényt:

$$L(\mathbf{w}, b, \mathbf{e}; \boldsymbol{\alpha}) = J(\mathbf{w}, b, \mathbf{e}) - \sum_{i=1}^P \alpha_i \{ \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i - d_i \}, \quad (6.90)$$

ahol az α_i értékek a Lagrange multiplikátorok. Az optimumra vonatkozó feltételek az alábbiak.

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} = 0 & \rightarrow \mathbf{w} = \sum_{i=1}^P \alpha_i \boldsymbol{\varphi}(\mathbf{x}_i) \\ \frac{\partial L}{\partial b} = 0 & \rightarrow \sum_{i=1}^P \alpha_i = 0 \\ \frac{\partial L}{\partial e_i} = 0 & \rightarrow \alpha_i = C e_i \quad i = 1, \dots, P \\ \frac{\partial L}{\partial \alpha_i} = 0 & \rightarrow \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i - d_i = 0 \quad i = 1, \dots, P \end{aligned} \quad (6.91)$$

A $\mathbf{w}$ és az e kifejezése után a következő lineáris egyenletrendszer írható fel:

$$\begin{bmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & \boldsymbol{\Omega} + C^{-1} \mathbf{I} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix} \quad (6.92)$$

ahol az $\boldsymbol{\Omega}$ kivételével az egyenletrendszerben szereplő többi jelölés megegyezik a (6.85)-ben megadottakkal. Az $\boldsymbol{\Omega}$ mátrix (i, j) -edik eleme:

$$\Omega_{i,j} = \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j), \quad i, j = 1, \dots, P. \quad (6.93)$$

Az eredmény – hasonlóan a hagyományos SVM-hez – a következő alakban írható fel:

$$y(\mathbf{x}) = \sum_{i=1}^P \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b, \quad (6.94)$$

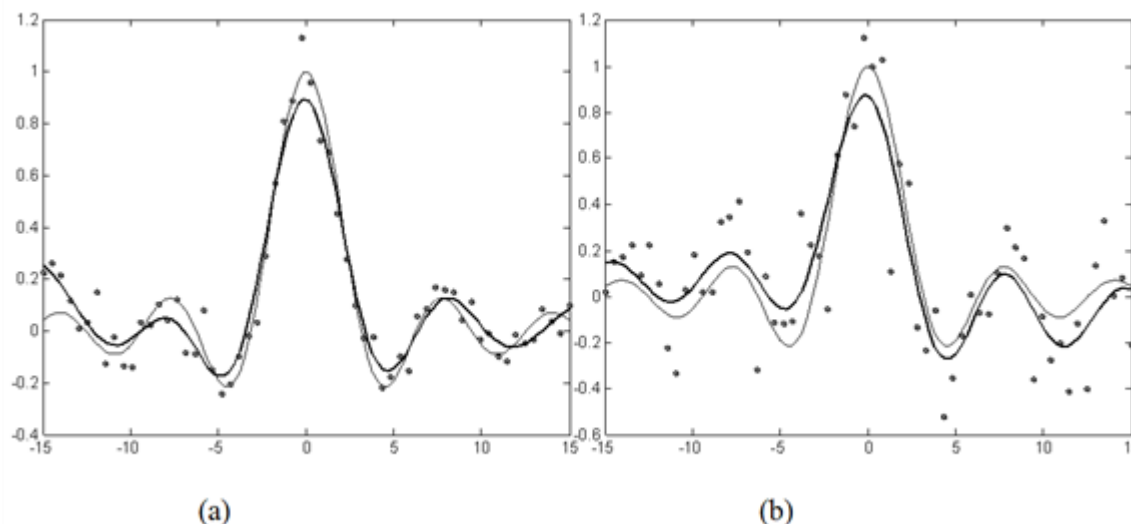
ahol az α_i -k és b a fenti lineáris egyenletrendszer megoldásai.

Az egyenletrendszer felírása a parciális deriváltakból az osztályozás esetén bemutatottak szerint történik. A regressziós esetben azonban az eredmény kiszámítása ((6.94) összefüggés) és az egyenletrendszer sorai között egyértelműen látszik a kapcsolat, nevezetesen, hogy az egyenletrendszer sorai a regularizációt ($C^{-1} \mathbf{I}$) leszámítva pontosan megfelelnek az eredmény kiszámítására szolgáló összefüggésnek. Ez nem meglepő, hiszen a tanítópontok által reprezentált feltételek gyakorlatilag azt fogalmazzák meg, hogy az adott bemenetre milyen kimenetet várunk a modelltől. Ez a működés jól követhető a 6.10 ábrán. A tanítás során bevezetett regularizáció

pedig megadja, hogy mennyire (mekkora hibával) kell illeszkedni a tanítópontokban. Az erősebb regularizáló hatás pontatlanabb illeszkedést követel és simább (egyszerűbb) megoldást, simább válaszfüggvényt eredményez.

A 6.14 ábrán egy egyszerű példát mutatunk be LS-SVM használatával megoldott regressziós feladatra. Látható, hogy még jelentősebb mértékben zajos tanítópontok mellett is jó megoldást kapunk.

6.14. ábra - A zajos sinc(x) LS-SVM modellje (Gauss kernel; $\sigma=\pi$; $C=10$; A Gauss zaj szórása (a) $\sigma_n=0.01$ (b) $\sigma_n=0.04$). Az összes tanítópont szupport vektor.



4.1.3. Ritka LS-SVM

A Least-Squares megoldás egyik hátránya, hogy a megoldás nem ritka (*sparse*), hiszen mint láttuk, minden tanító vektor szupport vektor is egyben. Ha a végeredményt neurális hálózatként értelmezzük, akkor ebben P (azaz a felhasznált tanítópontok számával megegyező számú) nemlineáris neuron található, így az eredmény komplexitása (a háló mérete) nagyobb, mint a hagyományos SVM-mel nyert hálózatoké, melyek valóban szelektálnak a bementek közül. Ez abból is látszik, hogy a megoldásban felhasználjuk az $\alpha_i = C e_i$ egyenletet, ami mutatja, hogy a i -edik tanítópontban kapott hiba arányos a tanítóponthoz, mint szupport vektorhoz tartozó α_i súllyal. Figyeljük meg, hogy a négyzetes hibafüggvény alkalmazása miatt az ∞ sáv hiányában gyakorlatilag nincs 0 hibaérték, így nincs $\alpha_i = 0$ érték sem. Ahhoz, hogy a hagyományos SVM ritkasági tulajdonságát visszanyerjük további lépésekre van szükség.

Intuitíven állíthatjuk, hogy a kisebb α_i -k kevésbé járulnak hozzá a megoldáshoz, azaz a kialakított modellhez. A következő „pruning” (metszési) eljárás egy ezen alapuló iteratív módszer, mellyel az LS-SVM alkalmazásánál is elérhető egy egyszerűbb eredmény, azaz kisebb hálózat. A megoldás menete az alábbi:

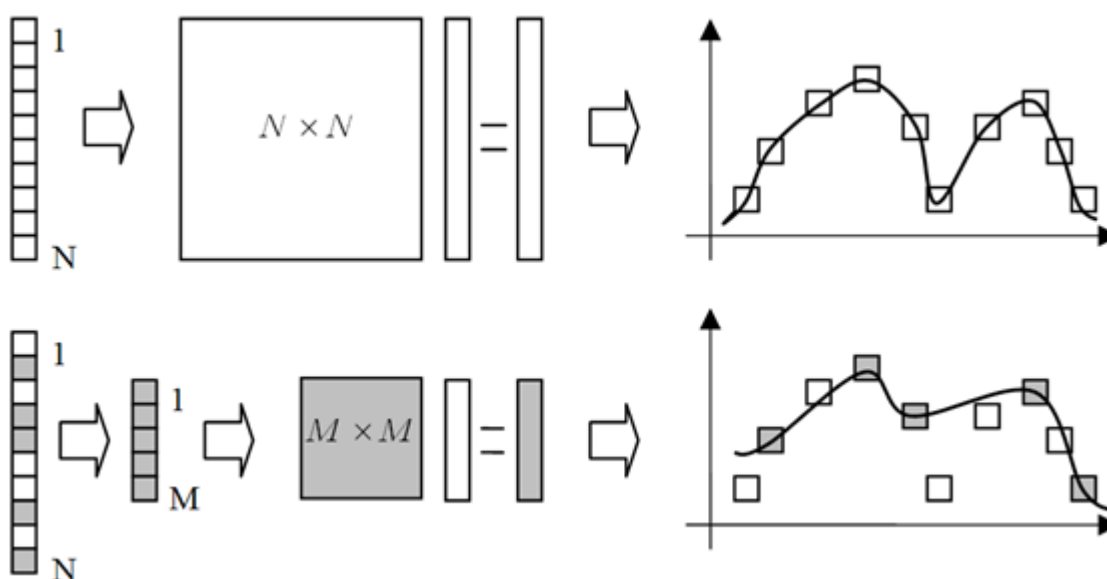
- Tanítsuk a hálózatot az összes rendelkezésre álló (P) tanítóponttal.
- Távolítsuk el egy kisebb részét (pl. 5%-át) a pontoknak úgy, hogy azokat hagyjuk el, melyekhez a legkisebb $|\alpha_i|$ -k tartoznak.
- Tanítsuk újra az LS-SVM-et a kisebb tanítókészlettel.
- Folytassuk a 2. lépéstől, amíg a válasz minősége nem romlik. Ha a teljesítmény romlik, akkor a hiperparaméterek (C és Gauss kernel esetén a σ) hangolásával esetleg korrigálhatjuk, illetve még tovább csökkenthetjük a megoldás méretét.

A negyedik pontban említett hiperparaméter hangolásának oka jól látható például egy Gauss kerneles hálózat esetén. Ha a metszési (*pruning*) eljárás eredményeképpen csökken a kernelek száma, akkor az egymástól távolabb eső középpontok miatt gyakran növelni kell a Gauss függvények szélességét (σ).

Fontos megjegyezni, hogy a metszési eljárás során, a minták elhagyásával gyakorlatilag a tanítókészlet méretét redukáljuk, ami – főként jelentősebb redukció esetén – információvesztéssel és így a háló minőségének romlásával jár(hat). Ez egyértelmű különbség a Vapnik-féle SVM-hez képest, hiszen ott a ritka megoldást úgy nyertük, hogy közben a Lagrange multiplikátorok meghatározásában az összes tanítópont részt vett. Ennek a kedvezőtlen hatásnak egyfajta mérséklését, vagy kiküszöbölését oldja meg a későbbiekben bemutatásra kerülő LS^2 -SVM. A mintapontok redukciójának hatását illusztrálja a 1.15 ábra. Az ábrán az üres négyzetek képviselik az összes tanítópontot, az árnyékolt négyzetek pedig a redukált mintakészlet pontjait.

A 6.16 ábraszorozat az eljárás valós példán történő futtatásának eredményét mutatja. Az (a) ábrán az eredeti LS -SVM megoldása látható. A további ábrák az egyre nagyobb mértékű redukció eredményét mutatják: a (b), (c) és (d) ábrákon egyre több pontot hagytunk el az eredeti tanítókészletből, miközben az approximáció minősége jelentősen nem változott.

6.15. ábra - A metszési eljárás, azaz a tanító minták elhagyásának hatása a feladat méretére valamint az approximáció eredményére.



4.1.4. Súlyozott LS -SVM

A négyzetes hibafüggvény a nem normál eloszlású zaj (például kilógó minták, „outlier”-ek) esetén nem optimális, ezért ilyen tanítómintáknál a modellt egy további módosítással hangolhatjuk. A módszer, hasonlóan a „pruning”-hoz az $\alpha_i = C e_i$ összefüggésen alapul. Ez az összefüggés azt mutatja, hogy az egyes pontokban kapott hiba, e_i arányos az eredményként kapott α_i súlyokkal. A módosított eljárás célja, hogy az egyes mintapontok szerepét a szerint mérlegelje, hogy ott mekkora a hiba. Ha e_i nagy, akkor az ehhez tartozó mintapont kevésbé megbízható, tehát a teljes eljárásban a szerepét célszerű csökkenteni, míg a pontosabb mintapontok szerepét, ahol e_i kisebb, érdemes növelni. Ezt a hatást egy újonnan bevezetett v_i súlykészlettel érjük el. Az optimalizálandó egyenlet (a regressziós esetre felírva) ekkor a következőképpen módosul:

$$\min_{\mathbf{w}, \mathbf{b}, \mathbf{e}} J(\mathbf{w}, \mathbf{b}, \mathbf{e}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \frac{1}{2} \sum_{i=1}^P v_i e_i^2. \quad (6.95)$$

A tanítópontokra vonatkozó feltételek nem változnak:

$$d_i = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i, \quad i = 1, \dots, P. \quad (6.96)$$

A lineáris egyenletrendszerben ez a következőket jelenti:

$$\begin{bmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & \boldsymbol{\Omega} + \mathbf{V}_c \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix} \quad (6.97)$$

ahol a $\mathbf{V}_c$ az alábbi diagonál mátrix:

$$\mathbf{V}_c = \text{diag}\left(\left[\frac{1}{Cv_1}, \dots, \frac{1}{Cv_P}\right]\right) \quad (6.98)$$

A v_i súlyokat az $e_i = \frac{\alpha_i}{C}$ értékek alapján választhatjuk meg, például a következők szerint:

$$v_i = \begin{cases} 1 & \text{ha } |e_i/s| \leq c_1 \\ \frac{c_2 - |e_i/s|}{c_2 - c_1} & \text{ha } c_1 \leq |e_i/s| \leq c_2 \\ 10^{-4} & \text{egyébként} \end{cases} \quad (6.99)$$

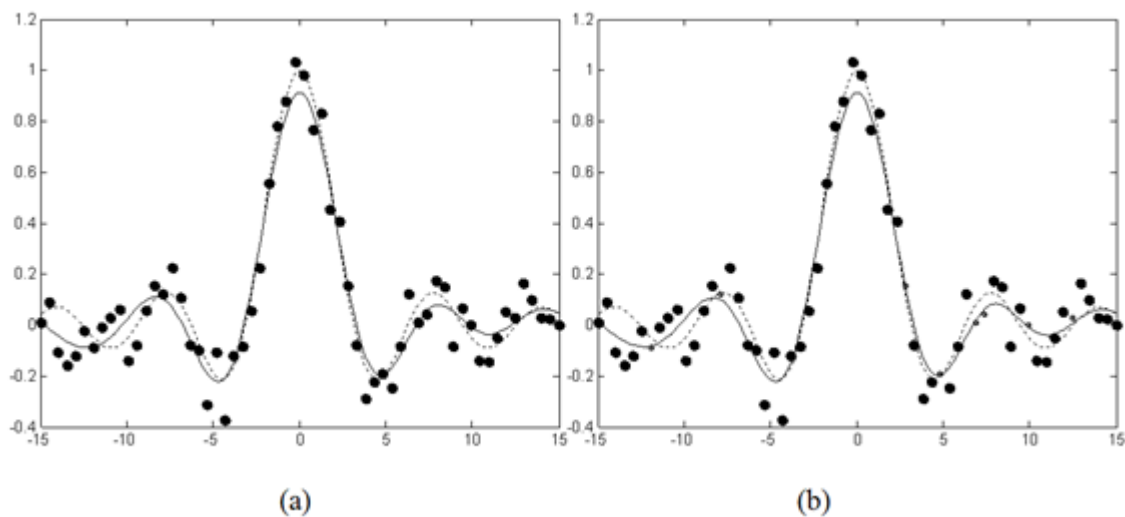
ahol c_1 , c_2 és s megválasztása a statisztikában ismert módszerek alapján történhet [Suy02].

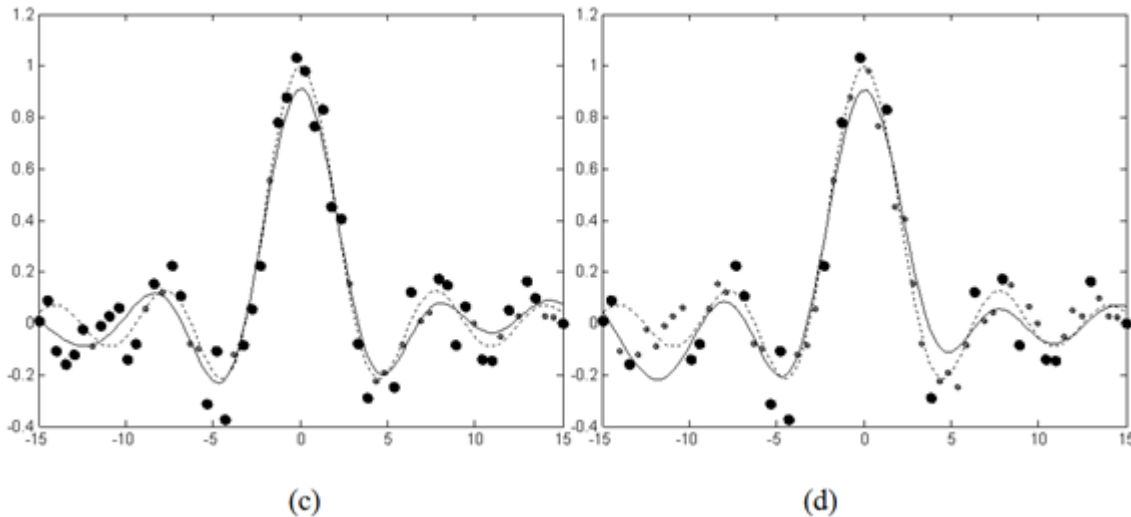
Az algoritmus menete a következő:

- Tanítsuk a hálózatot súlyozás nélkül az összes rendelkezésre álló (P) tanítóponttal és határozzuk meg az e_i értékeket.
- Határozzuk meg a v_i súlyokat az előzőek ((6.99) összefüggés) szerint.
- Számítsunk ki egy súlyozott LS-SVM modellt a v_i súlyok segítségével.

Ez az eljárás iteratívan ismételhető, de a gyakorlatban egy súlyozó lépés általában elegendő.

6.16. ábra - A metszési eljárás folyamatának négy állapota egy zajos sinc(x) megoldása során (RBF kenel; $\sigma = \pi$; $C = 10$; a zaj szórása: $\sigma_n = 0.01$). A mintapontok közül végül (d) 22-t tartunk meg (nagy fekete pontok).





4.2. Az LS-SVM hatékonyabb megoldása

Az LS-SVM tanításának komplexitása jóval kisebb, mint a hagyományos SVM tanításé, hiszen a QP helyett itt „csak” egy lineáris egyenletrendszert kell megoldani. Az LS-SVM használatának azonban vannak hátrányai is, mint pl. a ritkaság elvesztése. Mindemellett még az LS-SVM esetén is felmerül a hatékonyabb megoldás iránti igény, hisz a gyakorlati felhasználásokban még ez a módszer is túl számítás- és memóriaigényesnek bizonyulhat. Itt a fő probléma a kernel mátrix méretének csökkentése, ami így természetesen gyakran összekapcsolódik az LS-SVM egyéb hátrányainak a kiküszöbölésével.

4.2.1. A számítás gyorsítása

A tanítási sebesség növelésére, valamint a memóriaigények csökkentésére az LS-SVM lineáris egyenletrendszerét kell gyorsabban, kisebb memóriában megoldani. Erre Johan Suykens a Heistens-Stiefel konjugált gradiens módszert javasolja [Suy02]. Ez a lineáris egyenletrendszer egy iteratív megoldása, ami ez eredeti LS-SVM felírásnak megfelelő egyenletrendszer eredményére vezet.

Más lehetőség a ritka megoldás és a hatékonyság növelésének együttes alkalmazása, ami az LS-SVM módosítását jelenti. Ennek eléréséhez például a következőkben bemutatásra kerülő LS^2 -SVM-et vagy az LS-SVM-mel lényegében ekvivalens ridge regresszió egy redukált változatát, a csökkentett rangú kernel ridge regressziót alkalmazhatjuk.

4.3. LS^2 -SVM

Az LS^2 -SVM fő célja egy ritka LS-SVM megoldás elérése, csökkentve a modell komplexitását, azaz a rejtett rétegbeli nemlineáris neuronok számát. A módszer két fontos lépést tartalmaz:

- Az *első lépés* átalakítja az LS-SVM megoldást úgy, hogy az a tanítópontoknak csak egy részhalmazát használja „szupport vektornak”. Ennek következtében egy túlhatározott egyenletrendszert kapunk, melynek LS megoldása adja az eredményt.
- A *második lépés* a “szupport vektorok” (a kernel függvényeket meghatározó vektorok) automatikus meghatározására ad eljárást.

4.3.1. Túlhatározott egyenletrendszer

Az új megközelítés kiindulópontja az LS-SVM lineáris egyenletrendszere. Ha a tanító készlet P mintapontot tartalmaz, akkor az egyenletrendszer $(P+1)$ ismeretlen, az α -kat és a b -t, $(P+1)$ egyenletet és $(P+1)^2$ együtthatót tartalmaz. Az együtthatók a $K(\mathbf{x}_i, \mathbf{x}_j)$ $i, j = 1, \dots, P$ kernel függvény értékek. A mintapontok száma tehát meghatározza az egyenletrendszer méretét, ami egyúttal a megoldás komplexitását, a hálózat méretét is megszabja. Hogy ritka megoldást, azaz egy kisebb modellt kapjunk, az egyenletrendszert, illetve az együtthatómátrix méretét redukálni kell.

Vizsgáljuk meg közelebbről az LS-SVM regressziós problémát leíró egyenletrendszert és jelentését. Az első sor jelentése:

$$\sum_{k=1}^p \alpha_k = 0 \quad (6.100)$$

míg a j -edik sor a

$$b + \alpha_1 K(\mathbf{x}_j, \mathbf{x}_1) + \dots + \alpha_k [K(\mathbf{x}_j, \mathbf{x}_k) + C^{-1}] + \dots + \alpha_p K(\mathbf{x}_j, \mathbf{x}_p) = d_j \quad (6.101)$$

feltételt, megkötést tartalmazza.

Az egyenletrendszer redukálásánál sorokat, illetve oszlopokat hagyhatunk el.

- Ha a j -edik oszlopot hagyjuk el, akkor az ennek megfelelő $K(\mathbf{x}, \mathbf{x}_j)$ kernel szintén “eltűnik”, így az eredményként megkapott hálózat mérete csökken. Az első sor által támasztott feltétel azonban automatikusan alkalmazkodik, hisz a megmaradó α_k -k összege 0 marad.
- Ha a j -edik sort töröljük, akkor az $(\mathbf{x}_j, d_j)$ tanítópontnak megfelelő információ elvész, hiszen a j -edik megkötést elveszítjük.

A fentiek alapján az egyenletrendszert leíró mátrix legfontosabb része a $[\Omega + C^{-1}\mathbf{1}]$ részmátrix, ahol Ω az összes lehetséges tanítóvektor kombinációját tartalmazza ($\Omega_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$). A mátrix redukálása során abból sorokat, oszlopokat, illetve mindkettőt (sort a hozzá tartozó oszloppal) törölhetünk.

Minden oszlop egy neuronnak felel meg, míg a sorok bemenet-kimenet relációkat, azaz a megoldás által teljesítendő feltételeket fogalmazznak meg. A mátrix az alábbi két módon redukálható:

- Hagyományos teljes redukció

Az $(\mathbf{x}_j, d_j)$ tanítópontot teljesen elhagyjuk, azaz mind a hozzá tartozó sort, mind pedig az oszlopot töröljük. Az alábbi egyenlet a tanító minták teljes elhagyásának hatását mutatja. A törölt elemeket szürkével jelöltük.

$$\begin{bmatrix} 0 & \mathbf{1}^T \\ \Omega_{11} + \frac{1}{C} & \Omega_{12} & \dots & \Omega_{1p} \\ \Omega_{21} & \Omega_{22} + \frac{1}{C} & \dots & \Omega_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{1}^T & \Omega_{(p-1)1} & \Omega_{(p-1)2} & \dots & \Omega_{(p-1)p} \\ \Omega_{p1} & \Omega_{p2} & \dots & \Omega_{pp} + \frac{1}{C} \end{bmatrix} \begin{bmatrix} b \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} 0 \\ d_1 \\ d_2 \\ \vdots \\ d_p \end{bmatrix} \quad (6.102)$$

A hagyományos metszési (pruning) technika esetében pontosan ez történik, hisz a metszési algoritmus iteratívan kihagyja a tanítópontok egy részét. Az elhagyott tanítópontok által hordozott információ ezért teljesen elvész. A tanítópontok által hordozott információ megőrzése a részleges redukció mellett lehetséges.

- Részleges redukció

A $(\mathbf{x}_j, d_j)$ tanító mintát csak részben hagyjuk el, úgy, hogy töröljük a ponthoz tartozó oszlopot, de megtartjuk a hozzá tartozó sort. Így a tanítóminta által hordozott megkötés továbbra is érvényben marad, hisz az adott sor súlyozott összegének amennyire csak lehet egyezni kell a d_j kívánt kimenettel.

Ha kiválasztunk M ($M < P$) „szupport vektort” (oszlopot), de megtartjuk az összes megkötést (sort), az egyenletrendszer túlhatározottá válik. A részleges redukció hatását a mutatja a következő összefüggés, ahol a szürkével jelölt részeket távolítjuk el.

$$\left[\begin{array}{c|ccc} 0 & & \mathbf{1} & \\ \hline & \Omega_{11} + \frac{1}{C} & \Omega_{12} & \cdots & \Omega_{1P} \\ & \Omega_{21} & \Omega_{22} + \frac{1}{C} & \cdots & \Omega_{2P} \\ & \vdots & \vdots & \ddots & \vdots \\ \mathbf{1}^T & \Omega_{(P-1)1} & \Omega_{(P-1)2} & \cdots & \Omega_{(P-1)P} \\ & \Omega_{P1} & \Omega_{P2} & \cdots & \Omega_{PP} + \frac{1}{C} \end{array} \right] \begin{bmatrix} b \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_P \end{bmatrix} = \begin{bmatrix} 0 \\ d_1 \\ d_2 \\ \vdots \\ d_P \end{bmatrix} \quad (6.103)$$

A fentiek alapján kapott túlhatározott ($P \times M$ méretű) egyenletrendszert legkisebb négyzetes hiba értelemben oldhatjuk meg. Egyszerűsítsük a jelöléseinket úgy, hogy a fenti egyenletrendszer mátrix alakja legyen:

$$\mathbf{A}\mathbf{u} = \mathbf{v}, \quad (6.104)$$

melynek legkisebb négyzetes hibájú megoldása:

$$\mathbf{u} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{v}. \quad (6.105)$$

Az oszlopok törlése a sorok megtartása mellett biztosítja, hogy a kernelek (neuronok) száma csökkenjen, miközben az eljárás az összes ismert megkötést (tanítómintát) figyelembe veszi. Ez a kulcsa annak, hogy a modell komplexitása a pontosság megtartása mellett is csökkenthető legyen.

Látható, hogy a kernel mátrix redukciója során az oszlopok elhagyásával a regularizálás „aszimmetrikussá” válik, hiszen a továbbiakban nem minden sorban szerepel regularizáció. Ennek kiküszöbölésére megtehető, hogy a regularizációt már csak a kernel térbeli megoldás során használjuk, ami a kernel térbeli margó maximalizálásnak ($\|\alpha\|$ minimalizálásának) felel meg. Ez hasonló az SV megoldások nemlineáris kiterjesztéséhez, ahol a $\mathbf{w}$ súlyvektor minimalizálása valójában a jellemző térben történik. A kernel térben regularizált megoldás:

$$(\mathbf{A}^T \mathbf{A} + C^{-1} \mathbf{I}) \mathbf{u} = \mathbf{A}^T \mathbf{v}, \quad (6.106)$$

ahol

$$\mathbf{A} = \begin{bmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & \Omega \end{bmatrix}, \quad \Omega = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \cdots & K(\mathbf{x}_1, \mathbf{x}_M) \\ \vdots & \ddots & \vdots \\ K(\mathbf{x}_P, \mathbf{x}_1) & \cdots & K(\mathbf{x}_P, \mathbf{x}_M) \end{bmatrix}. \quad (6.107)$$

A módosított, részlegesen redukált egyenletrendszert legkisebb négyzetes hibájú (least-squares) értelemben oldjuk meg, ezért nevezzük ezt a módszert Least Squares LS-SVM-nek vagy röviden LS²-SVM-nek [Val04].

Az itt bemutatott részleges redukció hasonlít a hagyományos SVM kiterjesztéseként bevezetett redukált szupport vektor gép (Reduced Support Vector Machine – RSVM) alapelveihez [Lee01b]. Az RSVM esetén azonban, mivel az SVM eleve kisebb (ritka) modellt eredményez, a részleges redukció célja az algoritmikus komplexitás csökkentése, míg az LS-SVM esetében elsődlegesen a modell komplexitásának, a hálózat méretének csökkentése, azaz a ritka LS-SVM elérése a cél.

4.3.2. A kiválasztási eljárás

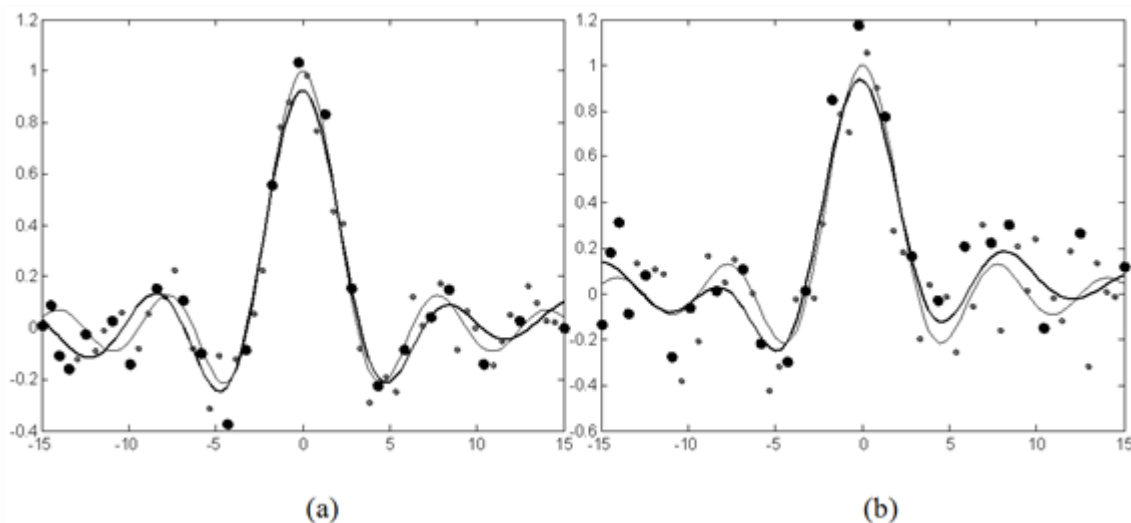
A fenti részleges redukció alkalmazása esetén szükség van valamilyen módszerre, ami meghatározza a szükséges szupport vektorokat. A kernel mátrix oszlopaiból kiválasztható egy lineárisan független részhalmoz, melynek elemeiből lineáris kombinációival a többi előállítható. Ez a kernel mátrix „bázisának” meghatározásával érhető el, hiszen az oszlopvektorok terének bázisa az a legkisebb vektorkészlet, amellyel a feladat megoldható. Ha a bázis meghatározásánál nem követeljük meg, hogy az oszlopvektorok által kifeszített tér és a kiválasztott bázisok által kifeszített tér azonos legyen, hanem bizonyos toleranciát is megengedünk, akkor a tolerancia mértékének meghatározásával egy szabad paramétert kaptunk. Ez a szabad paraméter lehetővé teszi, hogy kontroláljuk a bázisvektorok számát (M), hiszen a célunk az, hogy a P darab P -dimenziós oszlopvektorból $M < P$ bázisvektort határozzunk meg, ahol M minél kisebb.

A bázisvektorok száma, ami egyben a szupport vektorok száma is, valójában nem függ a mintapontok számától (P), csak a probléma nehézségétől, hiszen M a mátrix lineárisan független oszlopainak száma. A gyakorlatban ez azt jelenti, hogy ha egy probléma nehézsége M neuront igényel, akkor a tanításhoz felhasznált mintapontok számától függetlenül a modell mérete nem növekszik.

A szupport vektorok kiválasztása az A^T mátrix RRE alakra (*reduced row echelon form*, *RREF*) hozása során részleges pivotálással végrehajtott Gauss-Jordan eliminációval történik [Pre02], [Gol89b] (ld. Függelék). A tolerancia figyelembevétele a pivot elem (p) ellenőrzésével lehetséges. Eredetileg a pivotálás lényege, hogy az elimináció végrehajtása során szükséges osztásban a lehető legnagyobb elemet használjuk fel a numerikus stabilitás érdekében. A kiválasztott pivot elem nagysága azonban információt hordoz arról is, hogy mennyire fontos a hozzá tartozó oszlop a pontos megoldáshoz. Ha $p \leq \varepsilon'$ (ahol ε' a tolerancia paraméter), akkor az adott oszlop elhagyható, ellenkező esetben az oszlopnak megfelelő bemenet egy szupport vektor. A módszer azon oszlopvektorok listáját adja meg, melyek az ε' toleranciaérték értelmében lineárisan függetlenek.

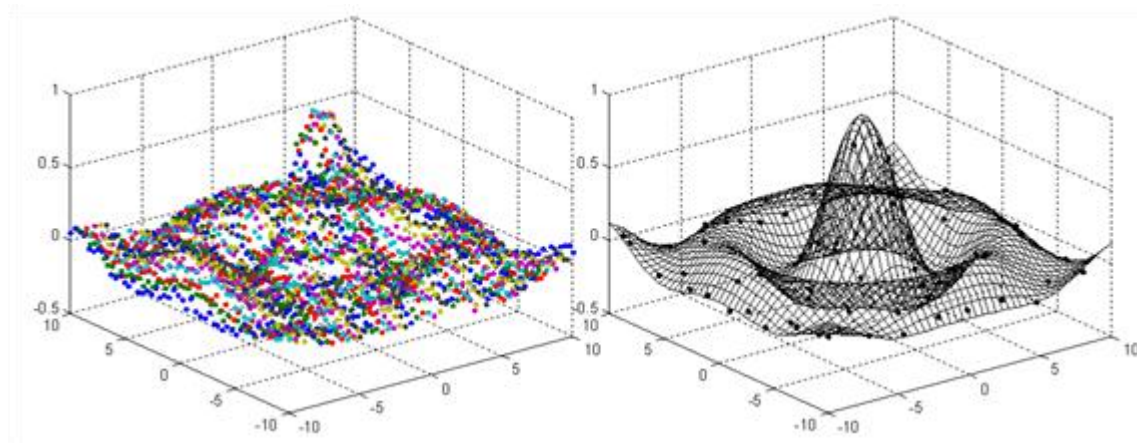
A megfelelő tolerancia helyes meghatározása hasonló a többi hiperparaméter (C , ε , valamint a kernel paraméterek) megválasztásához. Egy lehetséges megoldás a kereszt kiértékelés (cross-validation) alkalmazása. Minél nagyobb a tolerancia, annál kisebb a hálózat, és annál nagyobb az approximáció hibája. Az ε' helyes megválasztása egy olyan kompromisszum alapján lehetséges, ahol a pontos approximáció, illetve a modell komplexitása áll szemben egymással.

6.17. ábra - A zajos sinc(x) LS²-SVM modellje (RBF kernel; $\sigma=\pi$; $C=100$; $\varepsilon'=0,01$; A zaj szórása: (a) $\sigma_n = 0,01$ (b) $\sigma_n = 0,04$). Az RREF módszer eredményeképp a megoldás 23 szupport vektoron alapul.



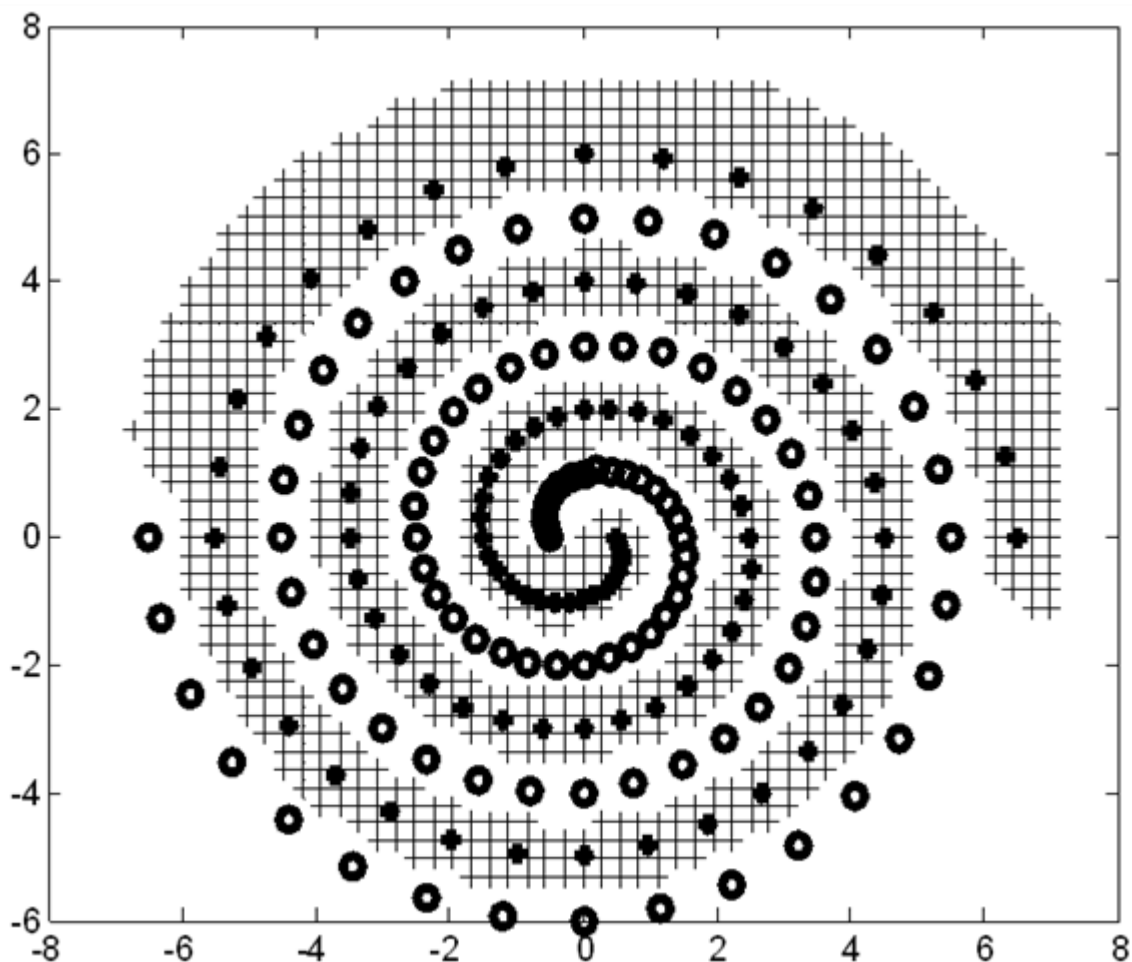
A 6.17 ábra zajos mintapontok mellett mutatja az LS^2 -SVM eredményét. Fontos hangsúlyozni, hogy nulla tolerancia esetén az LS^2 -SVM és az LS-SVM megoldás azonos, mivel ebben az esetben a kiválasztási módszer a mátrix minden oszlopát, azaz az összes tanítópontot megtartja (kivéve, ha a tanítópontok között volt redundancia).

6.18. ábra - Egy kétdimenziós sinc függvény approximációja LS^2 -SVM-el. A baloldali képen az összes (2500) tanítópont, a jobboldalin az LS^2 -SVM eljárással kapott 63 szupport vektor (fekete pontok) és az approximáció eredménye látható. (Gauss kernel, $\sigma=\pi$, $C=1000$, $\varepsilon'=0,15$.)



Az alábbiakban az LS^2 -SVM alkalmazására mutatunk két mintapéldát. A 6.18 ábrán egy kétdimenziós sinc függvény LS^2 -SVM-mel történő approximációja látható. A bal oldali ábra a kiinduló tanítópontokat mutatja, a jobboldali pedig a megoldást, ahol külön megjelennek az eljárás által meghagyott tanítópontok (szupport vektorok) is. Láthatóan a tanítópontok elenyésző töredéke elég a megfelelően pontos approximáció eléréséhez. A 6.19 ábra a klasszikus kettős spirál probléma megoldását adja meg. A módszer hatékonyságnövelő hatása itt is jól követhető.

6.19. ábra - A kettős spirál probléma megoldása LS^2 -SVM alkalmazásával. A mintapontok száma 194, a szupport vektorok száma 119. (Gauss kernel, $\sigma=0,5$; $C=10$, $\varepsilon'=0,9$.)



4.3.3. Súlyozott LS²-SVM

Az LS-SVM-nél megmutattuk, hogy egy iteratív eljárással súlyozott megoldás is adható, ami zajos esetben, főként kilógó minták (outlierek) esetén jelentősen jobb megoldásra vezet.

Amennyiben ismert vagy becsülhető a tanító mintákat terhelő zaj mértéke, ezt az információt az LS²-SVM esetén is fel lehet használni. A mintapontok azonos kezelése helyett ebben az esetben a pontosabb mintákat nagyobb, a pontatlan, azaz zajos mintákat kisebb súllyal célszerű figyelembe venni. A hiba súlyozott számítása:

$$\varepsilon_r = \sum_{i=1}^P v_i e_i^2 = \sum_{i=1}^P v_i (d_i - y_i)^2 \quad (6.108)$$

A túlhatározott LS²-SVM egyenletrendszer súlyozott megoldása az előbbieken bevezetett átírást alkalmazva:

$$\mathbf{A}^T \mathbf{V} \mathbf{A} \mathbf{u} = \mathbf{A}^T \mathbf{V} \mathbf{v}, \quad (6.109)$$

ahol a $\mathbf{V}$ súlyozó mátrix egy diagonálmátrix, amely a főátlóban a v_i súlyokat tartalmazza. A súlyokat úgy kell beállítani, hogy jól tükrözzék az egyes tanítóminták hatását a lineáris illesztésre. A zaj mértéke, illetve a súlyozás meghatározható egy előzetes megoldás alapján (vagy *a priori* információ alapján), de a túlhatározott egyenletrendszer megoldására használhatunk olyan statisztikai megoldó módszereket is, amelyek csökkentik a nagyon hibás adatok hatását. Ilyen például a Least Absolute Residuals – LAR, a Bisquare weights vagy a Least Trimmed Squares - (LTS) módszer [Ciz04], [Hol77], [Hub81].

4.4. Ridge regresszió

A fejezet elején egy egyszerű kernel gép származtatásánál láttuk, hogy a négyzetes hiba minimalizálását célzó lineáris gép értelmezhető kernel gépként is. A megoldást a tanítópontok bemeneti vektoraiból képezett mátrix

pszeudo-inverze segítségével tudtuk meghatározni (ld. (6.8) összefüggés) A könnyebb követhetőség miatt ezt az összefüggést itt megismételjük.

$$\hat{\mathbf{w}}^* = \hat{\mathbf{X}}^T (\hat{\mathbf{X}} \hat{\mathbf{X}}^T)^{-1} \mathbf{d}. \quad (6.110)$$

Azt is megmutattuk, hogy az így származtatott súlyvektorral kapott háló kernel gépként is értelmezhető, ahol a választ az alábbi formában is felírhatjuk:

$$y(\hat{\mathbf{x}}) = \hat{\mathbf{x}}^T \hat{\mathbf{X}}^T \hat{\mathbf{w}} = \sum_{i=1}^P \alpha_i (\hat{\mathbf{x}}^T \hat{\mathbf{x}}_i) = \sum_{i=1}^P \alpha_i K_i(\hat{\mathbf{x}}). \quad (6.111)$$

Amennyiben az $\hat{\mathbf{X}} \hat{\mathbf{X}}^T$ mátrix rangja nem teljes, vagy bármilyen pl. numerikus instabilitás miatt nem invertálható, a következőképp módosított összefüggés alapján nyerhető a megoldás:

$$\hat{\mathbf{w}}^* = \hat{\mathbf{X}}^T (\hat{\mathbf{X}} \hat{\mathbf{X}}^T + \lambda \mathbf{I})^{-1} \mathbf{d}, \quad (6.112)$$

ahol $\mathbf{I}$ az $\hat{\mathbf{X}} \hat{\mathbf{X}}^T$ mátrixnak megfelelő méretű egységmátrix, λ pedig egy kis pozitív konstans. Ez az összefüggés valójában a rosszul kondicionált mátrixok invertálásánál alkalmazott regularizációs eljárás, ami a matematikában Tyihonov regularizáció néven is ismert. Erre az összefüggésre azonban több úton is eljuthatunk. A fenti megoldás a Tyihonov regularizációtól függetlenül az ún. ridge regresszió (*ridge regression*) [Hoe70] útján is származtatható.

A következőkben röviden bemutatjuk a ridge regresszió származtatását és azt, hogy a ridge regresszió és az LS-SVM probléma-megfogalmazása, és így a megoldás is lényegében azonos. Mivel ez a tárgyalás főként a neurális modellekhez, illetve a szupport vektor gépekhez kapcsolódik, a ridge regressziós eljárásra csak röviden térünk ki. A ridge regressziót többnyire eltolásérték (*bias*) nélkül alkalmazzák, a következőkben itt is ezt a változatot mutatjuk be.

4.4.1. Lineáris ridge regresszió

A korábban bemutatott osztályozási problémának megfelelően itt is az $\{\mathbf{x}_i, d_i\}_{i=1}^P$ tanító mintakészlet alapján keressük az $y(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ lineáris modellt. A ridge regresszió is a négyzetes hibát minimalizálja, de az LS-SVM-hez hasonlóan közben a súlyvektor hosszának minimumát is biztosítani szeretné, ami regressziós feladatoknál a minél laposabb megoldást biztosítja [Sch02]. Ennek megfelelően a $\mathbf{w} \in \mathbb{R}^N$ súlyvektort ridge regresszióval a következőképpen határozzuk meg. Jelölje $J(\mathbf{w})$ most is a minimalizálandó kritériumfüggvényt (az 1/2-es szorzó a deriválás utáni eredmény egyszerűsítése miatt szerepel):

$$J(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2 + \frac{1}{2} C \sum_{i=1}^P (d_i - \mathbf{x}_i^T \mathbf{w})^2 \quad (6.113)$$

A $\frac{\partial J}{\partial \mathbf{w}}$ parciális deriválás eredményéből az optimális $\mathbf{w}$ az alábbiak szerint számítható:

$$\sum_{i=1}^P (d_i - \mathbf{x}_i^T \mathbf{w}) \mathbf{x}_i = \frac{1}{C} \mathbf{w} \quad (6.114)$$

$$\mathbf{w} = \left(\sum_{i=1}^P \mathbf{x}_i \mathbf{x}_i^T + \frac{1}{C} \mathbf{I} \right)^{-1} \left(\sum_{i=1}^P d_i \mathbf{x}_i \right) \quad (6.115)$$

Mátrix alakban felírva (ahol $\mathbf{d} = [d_1 \ \dots \ d_P]^T$ a tanítópontok kívánt válaszaiból képezett vektor és $\mathbf{X}$ az $\mathbf{x}_i$ $i=1 \dots P$ tanítóminta-bemenetekből, mint sorvektorokból képezett mátrix):

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_P^T \end{bmatrix}, \quad (6.116)$$

a megoldás súlyvektor:

$$\mathbf{w} = \left(\mathbf{X}^T \mathbf{X} + \frac{1}{C} \mathbf{I} \right)^{-1} \mathbf{X}^T \mathbf{d} = \mathbf{X}^T \left(\mathbf{X} \mathbf{X}^T + \frac{1}{C} \mathbf{I} \right)^{-1} \mathbf{d} \quad (6.117)$$

Az

$$\boldsymbol{\alpha} = \left(\mathbf{X} \mathbf{X}^T + \frac{1}{C} \mathbf{I} \right)^{-1} \mathbf{d} \quad (6.118)$$

jelölés bevezetésével

$$\mathbf{w} = \mathbf{X}^T \boldsymbol{\alpha} \quad (6.119)$$

így a végeredmény:

$$y(\mathbf{x}) = \mathbf{x}^T \mathbf{X}^T \boldsymbol{\alpha} = \sum_{i=1}^P \alpha_i \mathbf{x}^T \mathbf{x}_i \quad (6.120)$$

Vegyük észre, hogy ez a felírás egy kis különbséggel megegyezik a 6.1 részben bevezetett egyszerű kernel gép összefüggéseivel. A kis különbség abból adódik, hogy itt a kiinduló kritériumfüggvény a súlyvektor hosszának négyzetét, mint minimalizálandó mennyiséget, is tartalmazza. Ennek következménye a $\mathbf{w}$ illetve az $\boldsymbol{\alpha}$ vektorok összefüggéseiben ((6.117), illetve (6.118) összefüggések) a $C^{-1} \mathbf{I}$ regularizációs tag megjelenése. A ridge regresszió tehát az egyszerű kernel gép regularizált változata. Azt is észrevehetjük, hogy ez az eredmény megegyezik az eltolás-érték nélküli lineáris LS-SVM-re kapott összefüggéssel.

Természetesen a ridge regresszió is kiterjeszthető nemlineáris esetre a szokásos $\boldsymbol{\varphi}(\mathbf{x})$ transzformáció, és a kernel trükk alkalmazásával.

4.4.2. Nemlineáris kernel ridge regresszió

A nemlineáris ridge regresszió esetén – csakúgy, mint az SVM vagy az LS-SVM kiterjesztésénél – a lineáris regressziós modellt egy magasabb dimenziós térben, a jellemző térben alkalmazzuk, ami a szokásos módon egy nemlineáris transzformáció eredménye.

A $\boldsymbol{\varphi}(\cdot)$ leképezés segítségével a lineáris esetben megismert célfüggvény a következőképpen módosul:

$$J(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2 + \frac{1}{2} C \sum_{i=1}^P e_i^2 \quad (6.121)$$

ahol:

$$e_i = d_i - \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i), \quad i = 1, \dots, P. \quad (6.122)$$

Az LS-SVM regressziónál bemutatott lépéseknek megfelelően az $L(\mathbf{w}, \mathbf{e}, \boldsymbol{\alpha})$ Lagrange-függvény írható fel az α_i ($i = 1, \dots, P$) Lagrange együtthatókkal. A deriválások után a kernel trükk alkalmazásával az alábbi megoldás nyerhető (mátrix alakban):

$$\boldsymbol{\alpha} = \left(\boldsymbol{\Omega} + \frac{1}{C} \mathbf{I} \right)^{-1} \mathbf{d} \quad (6.123)$$

ahol az $\boldsymbol{\Omega}_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$ a tanítómintákból képzett kernel mátrix. Látható, hogy a nemlineáris eset itt is a kernel térben kerül megoldásra. A megoldás ugyanakkor megfelel egy regularizált kernel térbeli megoldásnak. Ez annyit tesz, hogy a $\|\mathbf{w}\|^2$ jellemzőtérbeli minimalizálása az $\|\boldsymbol{\alpha}\|^2$ kernel térbeli minimalizálására vezet.

A kernel ridge regressziós modell:

$$y(\mathbf{x}) = \sum_{i=1}^P \alpha_i K(\mathbf{x}, \mathbf{x}_i) \quad (6.124)$$

ami a regularizációtól eltekintve ismét megegyezik az egyszerű nemlineáris kernel gép válaszával (ld. (6.14) összefüggés).

4.4.3. Csökkentett rangú kernel ridge regresszió

Az eredeti ridge regressziós megoldásokban a kernel mátrix négyzetes, ami azt jelenti, hogy a modell a tanító pontoknak megfelelő számú szupport vektort tartalmaz, azaz nem ritka. A csökkentett rangú kernel ridge regresszió (*Reduced Rank Kernel Ridge Regression - RRKRR*) [Caw02] eljárás a ritka megoldást a kernel mátrix redukciójával éri el. Az RRKRR eljárás tehát hasonló célt tűz ki, mint az LS^2 -SVM, a célhoz azonban más úton jut el.

Az eljárást eltolásérték (bias) alkalmazása mellett mutatjuk be, hogy az eredmény összevethető legyen az LS -SVM eredményével.

A ritka megoldáshoz az approximációban felhasznált kernel függvények – azaz a kernel középpontoknak megfelelő szupport vektorok – számát kell csökkenteni. Az eredeti ridge regresszió (és természetesen az LS -SVM is) olyan kernel mátrixot használ fel, ahol mind az oszlopok, mind a sorok számát a tanítópontok száma határozza meg, hiszen minden tanítópont egy kernel középpont – ezek felelnek meg az oszlopoknak – és ugyanakkor minden tanítópont a jellemzőtérben egy lineáris egyenletet is jelent – ezek felelnek meg a soroknak. Az RRKRR eljárás az oszlopok számát redukálja úgy, hogy a tanítópontokból kiválaszt egy részhalmazt – ezek fogják képezni a kernel középpontokat –, és megtartja az összes tanítópontot, tehát megtartja a sorok számát. Az RRKRR eljárás tehát az LS^2 -SVM-hez hasonlóan részleges redukciót alkalmaz.

Az oszlopok kiválogatásához az alábbiakból indulhatunk ki. Az LS -SVM (6.83) és (6.91) összefüggései a $\mathbf{w}$ súlyvektort, mint a $\Phi(\mathbf{x}_i)$ jellemzőtérbeli vektorok súlyozott összegét fejezik ki.:

$$\mathbf{w} = \sum_{i=1}^P \alpha_i \Phi(\mathbf{x}_i) \quad (6.125)$$

Ez azt jelenti, hogy a súlyvektorok terét a $\Phi(\mathbf{x}_i)$ vektorok meghatározzák; mint bázisvektorok kifeszítik a teret. Kérdés azonban, hogy az összes $\Phi(\mathbf{x}_i)$, $i=1, \dots, P$ jellemzőtérbeli vektorra szükség van-e a súlyvektor (adott pontosságú) reprezentációjához. Tételezzük fel, hogy ehhez a tanítópontok egy S részhalmaza elegendő. Ekkor a $\mathbf{w}$ súlyvektor felírható az S részhalmazba tartozó jellemzővektorok súlyozott összegeként:

$$\mathbf{w} = \sum_{i \in S} \beta_i \Phi(\mathbf{x}_i) \quad (6.126)$$

Ha ez a megadás pontos, akkor az S -be tartozó $\Phi(\mathbf{x}_i)$ vektorok is kifeszítik a súlyvektorok terét, tehát bázist alkotnak ebben a térben. Ha S elemeinek száma kisebb, mint P , eredményként ritka megoldást kapunk. Természetesen itt a β_i súlyok különbözni fognak az (6.125) szerinti megadás α_i súlyaitól.

Ha az S részhalmazba tartozó jellemzővektorok bázist alkotnak a jellemzőtérben, akkor bármely $\mathbf{x}$ bemenet jellemzőtérbeli képe is kifejezhető, mint az S -beli $\Phi(\mathbf{x}_i)$ vektorok lineáris kombinációja.

$$\Phi(\mathbf{x}) = \sum_{i \in S} \zeta_i \Phi(\mathbf{x}_i) \quad (6.127)$$

Ha a bázisvektorok $\Phi(\mathbf{x})$ -nek csak közelítő reprezentációját, $\hat{\Phi}_S(\mathbf{x})$ -et adják, akkor olyan bázisfüggvényeket kell kiválasztanunk, melyek mellett ez a közelítés a legkisebb hibájú. A hibát most mint a két vektor normalizált euklideszi távolságát definiálhatjuk:

$$\delta_j = \frac{\|\Phi(\mathbf{x}_j) - \hat{\Phi}_S(\mathbf{x}_j)\|^2}{\|\Phi(\mathbf{x}_j)\|^2} \quad (6.128)$$

Megmutatható [Bau01], hogy ez a hiba a kernel trükk felhasználásával a kernel mátrix segítségével is felírható:

$$\delta_j = 1 - \frac{\mathbf{K}_S^T \mathbf{K}_{SS}^{-1} \mathbf{K}_S}{k_{jj}} \quad (6.129)$$

ahol $\mathbf{K}_{ss}$ a $\mathbf{K}$ kernel mátrix almátrixa: $\mathbf{K}_{ss} = \{K_{ij}\}_{i,j \in S}$, $\mathbf{K}_{sj} = \{K_{ij}\}_{i \in S}$ pedig egy belső szorzatokból képezett oszlopvektor, míg $k_{jj} = \mathbf{K}_{jj}$, a kernel mátrix j -edik főátlóbeli eleme.

A jellemzőtér bázisának meghatározásához az így definiált hiba átlagát kell minimalizálni a tanítókészlet összes pontját tekintetbe véve. Vagyis maximalizálni kell a

$$J(S) = \frac{1}{P} \sum_{j=1}^P \frac{\mathbf{K}_{sj}^T \mathbf{K}_{ss}^{-1} \mathbf{K}_{sj}}{k_{jj}} \quad (6.130)$$

mennyiséget. Az eljárást az üres halmazzal, vagyis $S = \emptyset$ -val indítjuk, és mohó stratégiát alkalmazunk. Minden egyes lépésben azt a tanítóvektort adjuk S -hez, mely a (6.130) kifejezést maximálja. Az eljárás magától leáll, ha $\mathbf{K}_{ss}$ már nem invertálható, jelezve, hogy a megfelelő bázist megtaláltuk, illetve leállítható, ha a hiba adott korlát alá csökken vagy egy előre meghatározott számú bázisvektort már kiválasztottunk.

Mivel a súlyvektor (6.126) szerint kifejezhető S -be tartozó bázisvektorok lineáris kombinációjaként, a következő duális egyenletet kapjuk, melyet $\mathbf{b}$ és b szerint kell minimalizálnunk.

$$J(\mathbf{b}, b) = \frac{1}{2} \sum_{i,j \in S} \beta_i \beta_j K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{2} C \sum_{i=1}^P (d_i - \sum_{j \in S} \beta_j K(\mathbf{x}_i, \mathbf{x}_j) - b)^2 \quad (6.131)$$

A $\mathbf{b}$ és b szerinti parciális deriváltak alapján megkeresve a minimumhelyeket, valamint ezt C -vel leosztva az alábbi egyenletet kapjuk:

$$\sum_{i \in S} \beta_i \sum_{j=1}^P K(\mathbf{x}_i, \mathbf{x}_j) + b = \sum_{j=1}^P d_j \quad (6.132)$$

és $\forall r \in S - r \in$

$$\begin{aligned} \sum_{i \in S} \beta_i \left(\frac{1}{C} K(\mathbf{x}_i, \mathbf{x}_r) + \sum_{j=1}^P K(\mathbf{x}_j, \mathbf{x}_r) K(\mathbf{x}_j, \mathbf{x}_i) \right) + b \sum_{i=1}^P K(\mathbf{x}_i, \mathbf{x}_r) = \\ = \sum_{j=1}^P d_j K(\mathbf{x}_i, \mathbf{x}_r) \end{aligned} \quad (6.133)$$

A fenti egyenletek az alábbi $(|S|+1) \times (|S|+1)$ méretű lineáris egyenletrendszerre vezetnek ($|S|$ az S halmaz elemeinek a száma):

$$\begin{bmatrix} \Omega & \Phi \\ \Phi^T & P \end{bmatrix} \begin{bmatrix} \mathbf{b} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \sum_{i=1}^P d_i \end{bmatrix}, \quad (6.134)$$

ahol

$$\Omega_{r,i} = \frac{1}{C} K(\mathbf{x}_i, \mathbf{x}_r) + \sum_{j=1}^P K(\mathbf{x}_j, \mathbf{x}_r) K(\mathbf{x}_j, \mathbf{x}_i), \quad i, r \in S$$

$$\Phi_r = \sum_{i=1}^P K(\mathbf{x}_i, \mathbf{x}_r), \quad \forall i \in S$$

$\mathbf{c}$ pedig egy olyan $|S|$ -dimenziós vektor, melynek r -edik eleme:

$$c_r = \sum_{j=1}^P d_j K(\mathbf{x}_i, \mathbf{x}_r) \quad \forall i \in S$$

Az egyenletrendszer megoldása egy csökkentett rangú (bázisú) ritka megoldásra vezet, mivel az csak $|S|$ számú kernelt és az eltolást tartalmazza.

5. Kernel CMAC: egy LS-SVM gép véges tartójú kernel függvényekkel

Az eddigiekben láttuk, hogy a bázisfüggvényes hálózatok értelmezhetők kernel gépként is, amennyiben a jellemzőtérből a belső szorzat felhasználásával áttérünk a kernel térre. A kernel interpretáció származhat újfajta megközelítésből (szupport vektor gépek), melynek nincs a klasszikus neuronhálók között ekvivalense, de olyan úton is származtatható, hogy az eredmény valójában a klasszikus bázisfüggvényes megoldással ekvivalens (LS-SVM, ridge regresszió és változataik). Az előbbi esetben a komplexitásra vonatkozó előnyökön túl további előnyök is jellemzik a kernel gépeket: a szupport vektor gépeknél az általánosító-képességre is megfogalmazhatók állítások. Az utóbbi esetben ilyen következménye nincs a kernel megközelítésnek. A komplexitás jelentős redukciója, a limitált komplexitás biztosítása azonban önmagában is jelentős előny. Az LS-SVM vagy a ridge regressziós megközelítést tehát érdemes akkor is alkalmazni, ha ezzel csupán a kapott háló komplexitását tudjuk redukálni. Ennek különösen akkor van jelentősége, ha az eredeti változatot a nagy komplexitás miatt esetleg meg sem tudjuk valósítani, miközben a kernel változat könnyen implementálható. Ilyen esetre ad példát az előző fejezetben bemutatott CMAC háló, amely számos előnyös tulajdonsága ellenére, többdimenziós esetben komplexitás-problémák miatt csak korlátozottan alkalmazható.

A következőkben azt mutatjuk be, hogy ez a komplexitás-probléma a kernel reprezentáció segítségével hogyan oldható meg. A kernel-CMAC származtatásához induljunk ki a CMAC választ megadó (5.48) összefüggésből:

$$y(\mathbf{x}) = \mathbf{a}(\mathbf{x})^T \mathbf{w}. \quad (6.135)$$

A választ megkapjuk, ha az optimális súlyvektor (5.55) kifejezését ebbe behelyettesítjük. Ezzel egy adott $\mathbf{x}$ -re a CMAC válasza:

$$y(\mathbf{x}) = \mathbf{a}(\mathbf{x})^T \mathbf{w}^* = \mathbf{a}(\mathbf{x})^T \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} \mathbf{d} \quad (6.136)$$

Vegyük észre, hogy ez az összefüggés formailag azonos az egyszerű kernel gép válaszával ((1.9) összefüggés), ha az ottani $\hat{\mathbf{x}}$ bemenet helyére a CMAC kimeneti rétegének bemenetét (az asszociációs vektort) helyettesítjük, és hasonlóan az $\hat{\mathbf{X}}$ mátrix helyére az asszociációs mátrixot $\mathbf{A}$ -t írjuk. A CMAC bázisfüggvényes háló, tehát az asszociációs vektor a bázisfüggvényes hálók $\Phi(\mathbf{x})$ vektorának felel meg. Vagyis a CMAC válasza az (6.14) összefüggéshez hasonlóan kernel formában is megadható:

$$y(\mathbf{x}) = \mathbf{a}(\mathbf{x})^T \mathbf{A}^T \boldsymbol{\alpha} = \sum_{i=1}^P \alpha_i (\mathbf{a}(\mathbf{x})^T \mathbf{a}(\mathbf{x}_i)) = \sum_{i=1}^P \alpha_i K_i(\mathbf{a}(\mathbf{x})) \quad (6.137)$$

A kernel függvény a CMAC-nál az asszociációs vektorok skalár szorzataként származtatható:

$$K_i(\mathbf{a}(\mathbf{x})) = \mathbf{a}(\mathbf{x})^T \mathbf{a}(\mathbf{x}_i) = K(\mathbf{a}(\mathbf{x}), \mathbf{a}(\mathbf{x}_i)) = K(\mathbf{x}, \mathbf{x}_i). \quad (6.138)$$

Az $\boldsymbol{\alpha}$ vektor a kernel térbeli súlyvektor:

$$\boldsymbol{\alpha} = (\mathbf{A} \mathbf{A}^T)^{-1} \mathbf{d} \quad (6.139)$$

Előfordulhat, hogy az $\mathbf{A} \mathbf{A}^T$ mátrix nem invertálható, pl. numerikus problémák miatt. Ekkor a szokásos regularizáció alkalmazható, vagyis $\mathbf{A} \mathbf{A}^T$ helyett a regularizált $(\mathbf{A} \mathbf{A}^T + \epsilon^{-1} \mathbf{I})$ mátrixot invertáljuk. Vegyük észre, hogy ekkor valójában egy eltolásérték (*bias*) nélküli LS-SVM-et oldottunk meg, hiszen – mint láttuk – a mátrix inverziójánál bevezetett regularizáció hatása ekvivalens egy olyan LS-SVM megoldásával, ahol a kritériumfüggvénybe a súlyvektor hosszának négyzete, mint minimalizálandó mennyiség is szerepel.

A kernel reprezentáció előnye nyilvánvaló: amíg az eredeti jellemzőtérbeli reprezentációnál a w_i súlyok száma a tanítópontok számánál jóval nagyobb (ld. a (5.50)-(5.52) összefüggéseket), addig a kernel térbeli α_i súlyok száma a tanítópontok számával egyezik meg. A különbség különösen többdimenziós esetben jelentős. Az 5. fejezetben láttuk, hogy a súlymemória többdimenziós esetben még viszonylag kisméretű (néhány dimenziós) feladatnál is olyan óriási, hogy semmilyen eszközzel nem implementálható. A különféle módosítások (hash kódoló réteg bevezetése, a többdimenziós feladat egy- vagy kétdimenziós részfeladatokra való dekomponálása, stb.) célja az volt, hogy a hatalmas méretű memóriát megvalósítható méretűre redukáljuk. Láthatóan a kernel reprezentáció is egy eszköz az extrém komplexitás letörésére.

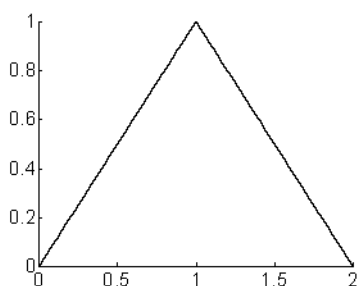
A kernel térben a CMAC választ kernel függvények súlyozott összegeként állítjuk elő. A kernel gépekre ugyanakkor az is jellemző, hogy $\phi(\mathbf{x})$ kernel függvényekből indulunk ki, és a jellemzőtérbeli leképezés nemlineáris transzformációját megvalósító leképezést csak implicit módon határozza meg a kernel függvény. A kernel CMAC-nál nem ez a helyzet. Itt valóban a CMAC négyzetes (bináris) bázisfüggvényeiből belső szorzat útján határozzuk meg a kernel függvényeket.

A bináris véges tartójú bázisfüggvények miatt kernel függvényként egydimenziós esetben a háromszögfüggvényt kapjuk. Többdimenziós esetben a kernel függvény attól függ, hogy teljes lefedést vagy csak C -szeres lefedést alkalmazunk. Az eredeti Albus CMAC esetében az extrém komplexitás miatt a teljes lefedés szóba sem kerülhet, ha a bemeneti dimenzió 3-4-nél nagyobb. A kernel reprezentációnál azonban a komplexitást nem a jellemzőtér, hanem a kernel tér dimenziója határozza meg. A kernel tér dimenziója semmiképpen sem lehet nagyobb, mint a mintapontok száma, bármekkora – akár végtelen – is a jellemzőtér dimenziója. Ez azt jelenti, hogy a kernel reprezentáció esetén nincs szükség a lefedések számának korlátozására, a teljes lefedés is alkalmazható. A kétféle megoldás mindössze annyiban tér el egymástól, hogy más kernel függvényt eredményez, de a kernel függvények száma azonos lesz.

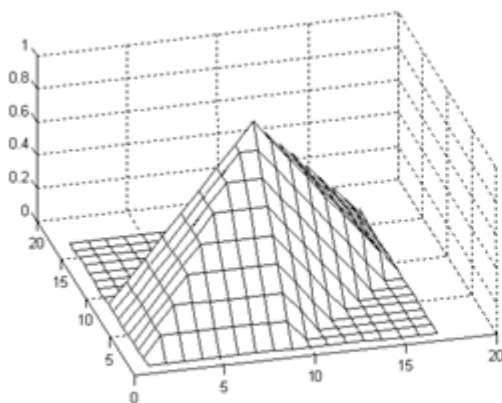
A teljes lefedés azzal az előnnyel is jár, hogy így elkerülhető a C -szeres lefedés következtében előálló approximációs képességbeli korlátozás. Láttuk, hogy többdimenziós esetben a C -szeres lefedésű bináris CMAC csak az additív függvényeket tudja hibátlanul megtanulni (ld. (5.58) konzisztencia-egyenlet). Teljes lefedés, illetve az annak megfelelő kernel változat mellett ez a korlátozás megszűnik.

A 6.20 ábrán a bináris CMAC kernel függvényét mutatja egydimenziós (a) és többdimenziós esetben. A többdimenziós CMAC-nál mind a C -szeres lefedésnek (b), mind a teljes lefedésnek (c) megfelelő kernel függvény látható. A teljes lefedéshez tartozó kernel függvényt kvantált változatban is megadjuk (d).

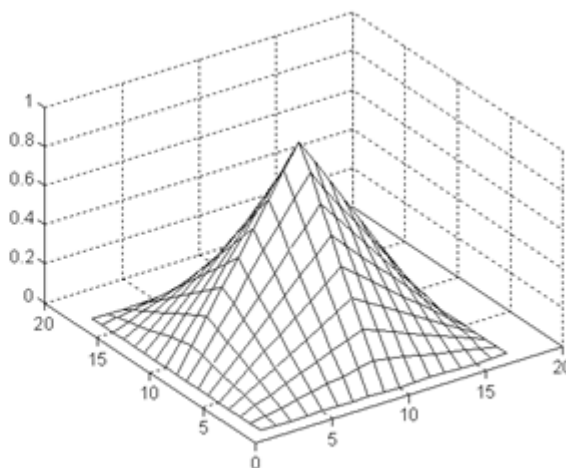
6.20. ábra - A CMAC hálózat kernel függvényei (a) egydimenziós eset, (b) kétdimenziós C -szeres lefedésű, (c) kétdimenziós teljes lefedésű, (d) kétdimenziós teljes lefedésű kvantált.



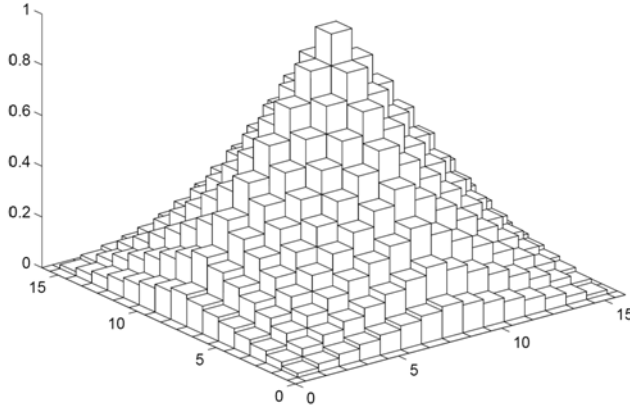
(a)



(b)



(c)



(d)

A kernel változatnak azonban van egy hátrányos következménye. Az ábrából látható, hogy a kernel függvények már nem bináris függvények, emiatt a kimenet előállításánál a szorzásokat ténylegesen el kell végezni, a szorzónélküli felépítés, mint előnyös tulajdonság elveszett. A véges tartójú bázisfüggvények következtében azonban a kernel függvények is véges tartójúak, ami azt jelenti, hogy adott bemenet mellett a kimenet előállításában a kernel függvényeknek csak egy töredéke vesz részt, így a szorzások száma is erősen korlátozott.

A válaszokat tekintve a kernel változat ekvivalens az eredeti változattal. Ez azt jelenti, hogy a kernel reprezentáció mellett is megjelenik a nagy általánosítási hiba mindazon esetekben, ha a C/t nem egész (t az egyes dimenziók mentén az egyenletesen elhelyezkedő mintapontok távolságát jelenti kvantumokban mérve). Ennek a hibának a jelentős mérséklését egy súlykiegyenlítő vagy súlysimító regularizáció alkalmazásával értük el (ld. 5. fejezet). A következőkben röviden azt foglaljuk össze, hogy ez a súlykiegyenlítő regularizáció a kernel CMAC esetében is alkalmazható.

5.1. Kernel CMAC súlykiegyenlítő regularizációval

A regularizált kernel CMAC megoldását a minimalizálandó kritériumfüggvény felírásából származtathatjuk [Hor07].

$$J(\mathbf{w}, \mathbf{e}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{j=1}^P e_j^2 + \frac{\lambda}{2} \sum_{j=1}^P \sum_{i=1}^C \left(\frac{d_j}{C} - w_{j,i} \right)^2 \quad (6.140)$$

ahol $\alpha_{j,i}$ a j -edik bemenethez rendelt asszociációs vektor i -edik bitje. (A továbbiakban a négyzetes hibtag együtthatóját γ -val jelöljük az eddigi C -vel szemben, hogy a CMAC C paraméterével való jelölésbeli ütközést elkerüljük.) Látható, hogy a kritériumfüggvény az LS-SVM kritériumfüggvényének felel meg, azzal a különbséggel, hogy egy regularizációs taggal bővült. A regularizációs tag együtthatója λ . Valójában az LS-SVM kritériumfüggvénye is értelmezhető regularizált kritériumfüggvényként, így itt most két egymástól független regularizációt alkalmazunk.

Az LS-SVM-nél szokásos utat követve felírhatjuk a Lagrange függvényt:

$$L(\mathbf{w}, \mathbf{e}, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{j=1}^P e_j^2 + \frac{\lambda}{2} \sum_{j=1}^P \sum_{i=1}^C \left(\frac{d_j}{C} - w_{j,i} \right)^2 - \sum_{j=1}^P \alpha_j (\mathbf{w}^T \mathbf{a}_j + e_j - d_j) \quad , \quad (6.141)$$

ami a következő alakra hozható:

$$L(\mathbf{w}, \mathbf{e}, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \sum_{j=1}^P e_j^2 - \sum_{j=1}^P \alpha_j (\mathbf{a}_j^T \text{diag}(\mathbf{a}_j) \mathbf{w} + e_j - d_j) + \frac{\lambda}{2} \sum_{j=1}^P \frac{d_j^2}{C} - \lambda \sum_{j=1}^P \frac{d_j}{C} \mathbf{a}_j^T \text{diag}(\mathbf{a}_j) \mathbf{w} + \frac{\lambda}{2} \sum_{j=1}^P \mathbf{w}^T \text{diag}(\mathbf{a}_j) \mathbf{w} \quad (6.142)$$

ahol $\text{diag}(\mathbf{a}_j)$ egy olyan diagonálmátrix, melynek főátlójában $\mathbf{a}_j$ található. A megfelelő deriválások és behelyettesítések elvégzése után itt is egy lineáris egyenletrendszert kapunk, melynek megoldása:

$$\alpha = \left(\mathbf{K}_D + \frac{1}{\gamma} \mathbf{I} \right)^{-1} \left(\mathbf{I} - \frac{\lambda}{C} \mathbf{K}_D \right) \mathbf{d} \quad (6.143)$$

Itt $\mathbf{D} = \sum_{j=1}^P \text{diag}(\mathbf{a}_j)$ és

$$\mathbf{K}_D = \mathbf{A} (\mathbf{I} + \lambda \mathbf{D})^{-1} \mathbf{A}^T \quad (6.144)$$

a regularizált esetben megfelelő kernel mátrix. Látható, hogy a regularizáció hatására a kernel mátrix megváltozott. A regularizáció tehát módosítja a kernel függvényt. A (6.138) által megadott kernel függvény helyett a

$$K_{D,j}(\mathbf{a}(\mathbf{x})) = \mathbf{a}(\mathbf{x})^T (\mathbf{I} + \lambda \mathbf{D})^{-1} \mathbf{a}(\mathbf{x}_j) = K_D(\mathbf{a}(\mathbf{x}), \mathbf{a}(\mathbf{x}_j)) = K_D(\mathbf{x}, \mathbf{x}_j) \quad (6.145)$$

kernel függvényt kapjuk. Mivel $\mathbf{D}$ a tanítópontok elhelyezésének függvénye, valójában adatfüggő kernel függvényt kaptunk. A kernel függvény azonban csak a tanítópontoknak a bemeneti térben való elhelyezkedésétől függ, a kívánt válaszoktól nem.

A háló válasza a regularizált esetben a következőre adódik:

$$y(\mathbf{x}) = \mathbf{a}^T(\mathbf{x}) (\mathbf{I} + \lambda \mathbf{D})^{-1} \mathbf{A}^T \left[\alpha + \frac{\lambda}{C} \mathbf{d} \right] \quad (6.146)$$

Ez az alábbi formában is felírható:

$$y(\mathbf{x}) = \sum_{j=1}^P \beta_j K_D(\mathbf{x}, \mathbf{x}_j) = \mathbf{K}_D^T(\mathbf{x}) \boldsymbol{\beta} \quad (6.147)$$

ahol a $\boldsymbol{\beta}$ együtthatóvektor a módosított kernel függvényekkel felépített kernel térbeli megoldás súlyvektorát jelöli:

$$\boldsymbol{\beta} = \alpha + \frac{\lambda}{C} \mathbf{d} \quad (6.148)$$

A regularizálás a kernel reprezentációs megoldás mellett is biztosítja a háló sima válaszát, tehát az általánosítóképesség jelentős javítását. A regularizált kernel CMAC tehát nemcsak a komplexitás jelentős redukcióját biztosítja, hanem az approximációs- és általánosítóképességet is javítja.

A regularizált kernel CMAC azon túl, hogy egy jobb tulajdonságú tanuló rendszert eredményez, arra is példa, hogy regularizációs formában az alapkritérium mellett további mellékfeltételek is belefogalmazhatók egy feladatba. Az eljárás hatékonysága attól függ, hogy az így definiált optimalizálási feladat milyen nehézségek árán oldható meg.

6. A kernel gépek összefoglaló értékelése

A kernel megközelítés az adatokból történő tanuló rendszerek konstrukciójánál két szempontból hozott újat.

Az első újdonság a megoldás komplexitásához köthető. A kernel trükk alkalmazása a komplexitást a tanítópontok száma által meghatározott mértékben mindenképpen korlátozza. A kernel reprezentáció önmagában előnyös mindazon esetekben, amikor a bemeneti tér vagy a bemenetből nemlineáris

transzformációval előállított jellemzőtér sokdimenziós. A kernel reprezentáció azt teszi lehetővé, hogy a feladat komplexitása a bemeneti tér, illetve a jellemzőtér dimenziójától független lesz. Ebből adódóan a jellemzőtér tetszőlegesen sokdimenziós, akár végtelen dimenziós is lehet. Ezt az előnyt a kernel gépek mindegyike biztosítja.

A második, az összes kernel gép közül az elsősorban Vladimir Vapnik nevéhez köthető szupport vektor gépekhez (SVM) kapcsolódik. A szupport vektor gépek azon túl, hogy a tanítópontokban értelmezett hiba minimalizálását célozzák, azt is célnak tekintik, hogy a megoldás „optimális komplexitású” legyen, ezáltal a modell minőségéről is tudjanak valamit állítani. A modell komplexitásának „mérésére” a szupport vektor gépek a VC-dimenziót alkalmazzák. Az SVM a VC-dimenzió korlátozásával a strukturális kockázat minimalizálás (SRM) elvét valósítja meg. Ennek következménye, hogy az SVM hibájára – a valódi kockázatra – felső korlátok fogalmazhatók meg. Az SRM elv érvényesítését a szupport vektor gépek a súlyvektor hosszának minimalizálása útján biztosítják. A felső korlátok és az SVM egyes változatai közötti kapcsolatot pedig az teremti meg, hogy a súlyvektor hosszának minimalizálása a margó maximalizálásán keresztül a VC-dimenziót is minimalizálja.

A súlyvektor hosszának minimalizálása a többi kernel gép származtatásának is része, vagy része lehet. Ez általánosan is „simító” regularizációs hatást eredményez. Az LS-SVM és a ridge regressziós megoldásoknál, valamint ezek változatainál a simító hatás ellenére jelenleg nincs olyan érvényes eredmény, amely az SVM-hez hasonló felső korlátokat tudna megállapítani. Ez annak ellenére van így, hogy a gyakorlati tapasztalatok szerint a kétféle kernel gép jellegében nagyon hasonló eredményekre vezet. Az SVM és az LS-SVM ilyen szintű kapcsolatának részletes elemzése még hátra van.

Az SVM és a többi kernel gép között abban a tekintetben is eltérés van, hogy a szupport vektor gépek ritka megoldást adnak, kiválogatják azokat a tanítópontokat, melyek a megoldáshoz szükségesek, amelyek a megoldást „tartják”. Hasonlóan ritka megoldás az LS-SVM-nél, illetve a ridge regresszióval is biztosítható (pl. LS^2 -SVM, RRRR), ezekben az esetekben azonban ez nem az alapeljárás része, hanem az alapeljárást követő redukció eredménye.

Bár a kernel gépek alapvetően egy számítási eljárásnak jelennek meg, neuronhálóként is értelmezhetők. Mint neuronháló, a statikus előrecsatolt hálók családjába tartoznak. Ez lehetővé teszi, hogy a klasszikus előrecsatolt statikus hálózatokkal – MLP, bázisfüggvényes hálózatok – is összevetésre kerüljenek. Az MLP-vel való összehasonlítás a legnehezebb, hiszen a kétféle megoldás strukturálisan is eltérő. A kernel gépek és elsősorban az SVM előnye, hogy a komplexitás-problémára megoldást adnak. Minőség tekintetében viszont nem lehet rangsort felállítani, még akkor sem, ha a minőségre vonatkozó korlátok a klasszikus hálókra nem vonatkoznak. A korlátok hiánya ugyanis nem azt mondja, hogy pl. egy MLP-vel nem tudunk hasonlóan jó, vagy netán jobb megoldást is elérni, mint egy SVM-mel, csupán azt, hogy az MLP-vel elérhető valódi kockázatról általános eredményt jelenleg nem tudunk megfogalmazni.

A bázisfüggvényes hálókkal való összehasonlításnál az SVM-et és a többi kernel gépet külön kell vizsgálni. Míg az LS-SVM és a ridge regresszió a különböző változatokkal együtt valójában a bázisfüggvényes hálózatok eltérő, de ekvivalens megvalósítása, az SVM a kiinduló probléma más megfogalmazása miatt elvileg is más eredményt ad. Erre vonatkozó kísérleti eredmények is ezt a különbséget támasztják alá. (ld. pl. az RBF és a Gauss kerneles SVM összehasonlítására vonatkozó eredményt [Sch96]).

A fejezet végén bemutatott kernel CMAC egy konkrét példa a kernel reprezentáció hasznára bázisfüggvényes hálózatoknál. Ennek specialitása a véges tartójú bázisfüggvény és az ebből következő véges tartójú kernel függvény. A regularizált kernel CMAC pedig arra egy példa, hogy regularizációs formában az alapkritérium mellett további mellékfeltételek is belefogalmazhatók egy feladatba. Az eljárás hatékonysága attól függ, hogy az így definiált optimalizálási feladat milyen nehézségek árán oldható meg.

A kernel gépekkel kapcsolatos első eredmények alapvetően a kilencvenes években születtek. A számos nyitott kérdés miatt azonban folyamatosan jelennek meg újabb és újabb eredmények, melyek többirányú kutatás eredményei. A fontosabb irányok: a kernel függvény megválasztása, adaptív és adatfüggő kernel konstrukció, az általánosítóképességre vonatkozó újabb és kevésbé pesszimista korlátok meghatározása, továbbá a ritka megoldások elérése és a számítási komplexitás további mérséklése. Fontos kérdés az is, hogy a kernel gépek alkalmazásával minél több tapasztalat gyűljön össze, melyek a hatékony gyakorlati felhasználást segíthetik. Néhány alkalmazási területről a következő fejezetben lesz szó.

Feladatok

6.1 Mutassa meg, hogy egy $\mathbf{x}$ pontnak a $\mathbf{w}^T \mathbf{x} + b = 0$ egyenlettel definiált lineáris szeparáló felülettől való távolsága

$$r = \frac{|\mathbf{w}^T \mathbf{x} + b|}{\|\mathbf{w}\|}$$

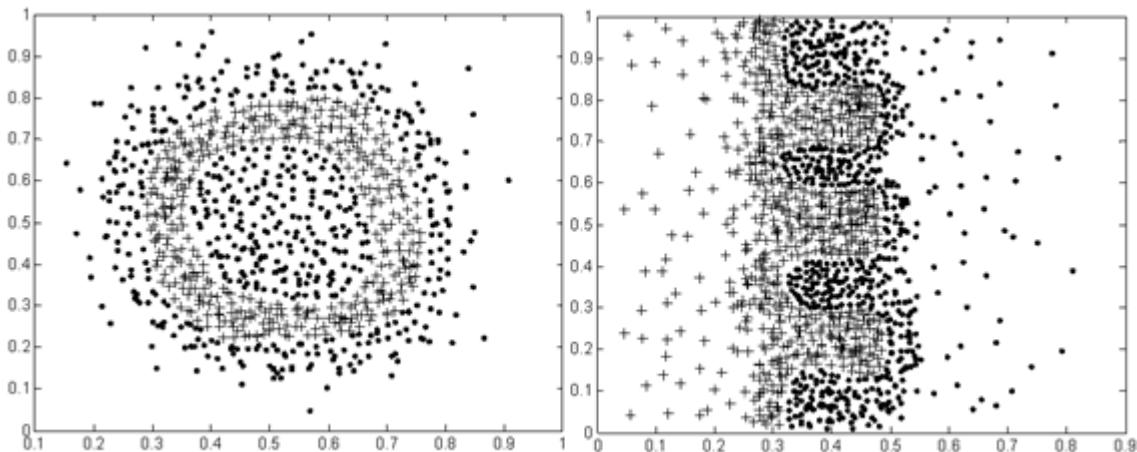
6.2 A XOR probléma megoldásában az $(\mathbf{x}^T \mathbf{x} + 1)^p$ polinomiális kernel függvény alkalmazható, ha $p=2$. Vizsgálja meg a polinomiális kernel alkalmazhatóságát más pozitív egész p értékekre.

6.3 Konstruáljon szupport vektor gépet az egydimenziós sinc függvény approximálására $(\mathbf{x}^T \mathbf{x} + 1)^p$ polinomiális kernel függvény mellett. Vizsgálja meg p szerepét az approximáció minőségét illetően.

6.4 Készítsen egy osztályozós mintakészletet (ld. 6.21 ábra), valamint egy paraméterezhető (kernel paraméterek – pl. RBF esetén σ , C , ϵ) tesztkörnyezetet, ami egy tanítókészlet alapján megtanít egy SVM osztályozót és meghatározza a hibás osztályozások számát (arányát) a tesztkészletre. (A feladatok során mindig legyen figyelemmel a megoldás hibájára, a számítási időre valamint a szupport vektorok számára.)

- Bontsa fel a mintapontokat tartalmazó fájlokat egy tanító és egy teszt készletre. A fájlban a pontok elrendezése (sorrendje) nem ismert, ezért célszerű figyelni arra, hogy mindkét osztályból és egyenletesen válasszunk elemeket. (Segítség: keverje össze az adatfájlt a válogatás előtt.)
- Határozza meg az SVM egy olyan paraméterezését, melyre az osztályozás elfogadható eredményt ad. (Segítség: használjon kicsi – max. 200 mintapontból álló – tanítókészletet és Gauss kernelt.)
- Vizsgálja meg, hogy hogyan függ az eredmény a kernel függvény megválasztásától. Tesztelje a hálózatot az egyes kernelek paramétereinek változtatásával is.
- Hogyan függ az eredmény a C paraméter megválasztásától?
- Vizsgálja meg, hogy hogyan függ az eredmény és a futási idő a tanítópontok számától. (Segítség: célszerű kisebb tanítókészletből kiindulni és addig növelni a pontok számát, amíg a számítás kivárható.)

6.21. ábra - Két lehetséges mintakészlet az osztályozós feladatokhoz.



6.5 Készítsen egy mintakészletet a sinc(x) függvény pontjaiból, valamint egy paraméterezhető tesztkörnyezetet (kernel paraméterek – pl. RBF esetén σ , C , ϵ), ami egy tanítókészlet alapján megtanít egy SVM regresszort és meghatározza a regresszió négyzetes hibáját.

- Bontsa fel a mintapontokat egy tanító és egy teszt készletre (Segítség: Vegye például az adatfájl minden n -edik elemét, vagy keverje össze a mintapontokat és ezt ossza ketté.)
- Határozza meg az SVM egy olyan paraméterezését, mely elfogadható eredményt ad. (Segítség: használjon kis – max. 200 mintapont – tanítókészletet és Gauss kernelt.)
- Vizsgálja meg, hogy hogyan függ az eredmény a kernel függvény megválasztásától. Tesztelje a hálózatot az egyes kernelek paramétereinek változtatásával is.

- d. Hogyan függ az eredmény a C paraméter megválasztásától?
 - e. Vizsgálja meg, hogy hogyan függ az eredmény és a futási idő a tanítópontok számától. (Segítség: célszerű kisebb tanítókészlettel indulni és addig növelni a pontok számát, amíg a számítás kivárható.)
- 6.6 Az előző regressziós feladatokat végezze el LS-SVM illetve LS^2 -SVM használatával is.
- 6.7 Származtassa a magasabbrendű CMAC kernel függvényét, ha a CMAC hálózat bázisfüggvénye k -adrendű B-spline.
- 6.8 Mutassa meg, hogy az RRR eljárásnál a bázisvektorok kiválasztásának alapját képező approximációs hiba (6.128) és (6.129) összefüggései ekvivalensek.

7. fejezet - Ellenőrzött tanítású statikus hálók alkalmazásai

Az eddigiekben bemutatott neuronhálók – bár felépítésükben, származtatásukban és tanításukban jelentősen különböznek egymástól – mind az ellenőrzött tanítású statikus hálózatok közé tartoznak. Ahogy láttuk ezeknek a hálózatoknak a többsége szinte tetszőleges folytonos nemlineáris leképezés tetszőleges pontosságú közelítésére képes. Ez az univerzális approximációs képesség és a hálózatok adaptivitása az alapja e hálótípusok gyakorlati alkalmazásainak.

A felhasználási területek nagyon szerteágazók, és jellegükre nézve két fő csoportba sorolhatók. Az egyik csoportba azok tartoznak, melyeknél a feladat rögzített, vagyis ha a tanítópontok alapján egyszer létrehoztuk a hálót, akkor ez a továbbiakban módosítás nélkül alkalmazható. A másik csoportba tartozó alkalmazási területekre, gyakorlati feladatokra az jellemző, hogy maga a feladat is változik, és ezt a változást a hálónak követnie kell. Itt tehát a megtanított háló működése közben a változó feltételekhez, körülményekhez való adaptálódás is követelmény.

Ebben a fejezetben néhány tipikusnak mondható feladat neuronhálós megoldását mutatjuk be röviden. Ezek a feladatok többségükben az első feladatosztályba tartoznak, ahol a feladatot egyszer kell a hálónak megtanítani, az alkalmazás folyamatos adaptálódást nem igényel. Az on-line adaptációs képesség elsősorban a dinamikus feladatoknál jelentkezik követelményként, melyekkel a következő fejezetben, az időfüggő hálózatok kapcsán fogunk foglalkozni.

A néhány feladat messze nem fedi le az alkalmazási lehetőségek teljes spektrumát. A teljes spektrum áttekintése nem is célja a fejezetnek, sőt még az sem, hogy a legfontosabb alkalmazási területek többségét érintse. A viszonylag egymástól távoli alkalmazási területek ugyanakkor azt kívánják alátámasztani, hogy a neuronhálók egyfajta univerzális feladatmegoldó eszközök. Ennek az sem mond ellent, hogy most is, és a későbbi alkalmazásokhoz kapcsolódó fejezetek során is külön hangsúlyozni szeretnénk, hogy az univerzális képességek ellenére a sikeres alkalmazáshoz az adott tématerület minél jobb megismerése, a feladat minél teljesebb megértése elengedhetetlen.

A neurális hálózatok népszerűségüket az univerzális képességükön túl annak is köszönhetik, hogy megértésük, használatuk viszonylag egyszerű, legalábbis az általuk megoldani célzott feladatokhoz képest. A használt algoritmusok nem bonyolultak, így kis programozói gyakorlattal bárki tud készíteni vagy használni egy hálózat szimulátor programot. Ezt azután csak megfelelő adatokkal kell feltölteni, és a beépített tanulás segítségével az adott feladatot a program – szerencsés esetben – elsajátítja. A tapasztalat azonban az, hogy így csak olyan problémák oldhatók meg kielégítő szinten, amelyek számos másfajta, hagyományos eljárás segítségével is megoldhatók volnának.

Egy alkalmazás sikeres elkészítéséhez, mind az adott feladat, mind az alkalmazandó eszköz, a neurális hálózat alapos ismeretere van szükség. Az alkalmazások többsége a többrétegű perceptront használja. Azt mondhatjuk, hogy az MLP messze a leggyakrabban alkalmazott neuronháló, bár az újabb eszközök, különösen a kernel gépek különböző változatai is egyre gyakrabban kerülnek alkalmazásra. A sok különböző alkalmazási terület közül néhány fontosabb:

- egy- és többdimenziós jelfeldolgozás (képfeldolgozás, beszédfeldolgozás, stb.) ideértve a különböző felismerési problémákat is: pl. nyomtatott, vagy kézzel írott karakterek, szövegek felismerése, beszéd és beszélő felismerés, stb.
- rendszer-identifikáció és vezérlés,
- robotika,
- orvosi és műszaki diagnosztika,
- pénzügyi és közgazdasági folyamatok modellezése, jellemzők becslése.

Az említett alkalmazási területekhez kapcsolódó feladatok egy része megoldható az eddig tárgyalt statikus nemlineáris leképezésre alkalmas, előrecsatolt hálózatokkal, más részük dinamikus feladat, így ezen alkalmazásokhoz a későbbiekben tárgyalt időfüggő hálózatok szükségesek.

Egy gyakorlati feladat megoldásánál először a probléma elemzése alapján azt kell eldönteni, hogy a feladat mely részeit próbáljuk neurális eszközzel megoldani, és milyen hálózattípus tűnik alkalmasnak a megoldásra. A fent leírtak alapján azokat a részfeladatokat célszerű neurális hálózattal megvalósítani, amelyekre nem tudunk gyors algoritmikus megoldást adni. A hálózat típusánál meghatározó szempont az is, hogy mennyi a rendelkezésre álló tanítási idő, milyen jellegű az adatbázis (hány dimenziós adatokból és hány adatból áll, stb.), továbbá milyen más a priori információ áll rendelkezésünkre a problémáról. A feladatról lévő előzetes ismereteinknek a hálózat struktúrájának és tanítási algoritmusának kiválasztásán kívül az adatok megfelelő kódolásában, előfeldolgozásában is fontos szerepük van. A most bemutatásra kerülő alkalmazások ezekre a kérdésekre is kitérnek, a kérdések általánosabb tárgyalásával azonban a 13. fejezetben foglalkozunk.

1. Felismerési feladatok (képosztályozás)

Az alakfelismerés (*pattern recognition*), és ezen belül is a képosztályozás a neurális hálózatok egyik tipikus alkalmazási területe. Nemcsak azért, mert számos sikeres megoldás született különböző problémákra (pl. arcfelismerés, irányítószám-felismerés, rendszámfelismerés), de azért is, mert a feladatok megoldásánál alkalmazott módszerek jól reprezentálják az alkalmazások készítésénél felmerülő problémákat: az adatrepresentáció megválasztását, a megfelelő hálózati struktúra és tanulási algoritmus kijelölését, a tanító mintapéldák kiválasztását. Ebben az alfejezetben két régebbi (1989-es ill. 1992-es), és egy újabb (2002-es) neurális hálózat alapú számjegyfelismerő rendszert mutatunk be, amelyek lényegesen eltérő megközelítéseket alkalmaznak. A megoldott feladat mindhárom esetben kézzel írt irányítószám számjegyeinek az azonosítása volt.

Az egyik legnehezebben eldönthető probléma az alkalmazások megvalósításánál a megfelelő adatrepresentáció megválasztása. Ha viszonylag alacsony szintű, „nyers” reprezentációt alkalmazunk, akkor a megoldási eljárásnak, a neurális hálózatnak nehéz feladatot kell megoldania, mivel az adatokból neki kell kiválasztani a lényeges információkat. Ha a mintafelismerést végző hálózat már egy előfeldolgozott, magas szintű adatrepresentációt felhasználva dolgozik, akkor a számára megoldandó feladat már viszonylag egyszerűbb lesz. Ennek a módszernek hátránya, hogy az előzetes adat-transzformálás gyakran időigényes és információvesztéssel jár. Tehát, míg az első esetben az adatokat gyorsan lehet a neurális hálózat bemenetére juttatni, viszont a hálózat működése lassú és a bonyolult megoldandó probléma miatt esetleg nem lesz kielégítő minőségű, addig az utóbb említett módszernél az adatok előfeldolgozása lassíthatja az eljárást és az itt elvesztett információ okozhat gyengébb eredményt a felismerés során. Azt, hogy melyik megközelítést érdemes használni, minden alkalmazásnál külön meg kell vizsgálni.

Akár alacsony szintű, akár magas szintű reprezentációt használunk, az a priori információk felhasználása fontos kérdés. Az első alkalmazási példánál ez az osztályozást végző hálózat struktúrájának kialakításánál, a második módszer esetében az előfeldolgozás során történik. A harmadik esetben a tanító készlet bővítésével hasznosították az előzetes ismereteket.

Ezek után nézzük meg a három karakterfelismerési alkalmazást, amelyek között mindkét bemutatott megközelítés előfordul: az első és a harmadik módszer alacsony szintű, míg a második módszer magas szintű reprezentációt használ.

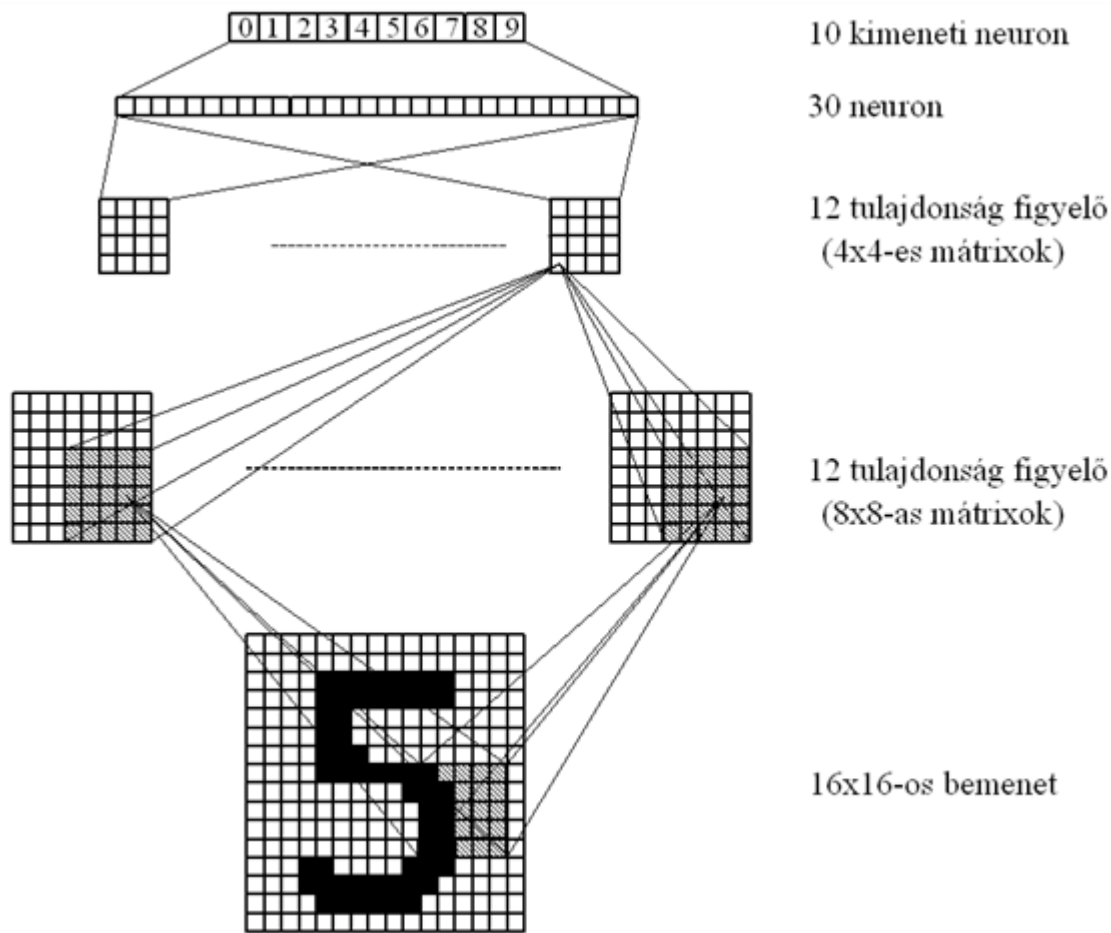
Az irányítószám felismerés feladatánál a megoldás első lépése a rögzített képen a számjegyek helyének meghatározása és a karakterek szegmentálása. Ezt, az önmagában is bonyolult feladatot mindhárom esetben egy külön rendszer végezte, amivel itt most nem foglalkozunk. A probléma megoldását attól a ponttól vizsgáljuk, ahol a bemenet az izolált számjegyek pixel képe.

1.1. Számjegyfelismerés speciális struktúrájú MLP-vel

Az első alkalmazási megoldásban [LeC89] a többrétegű perceptron struktúra egy módosított változatát használták fel. A hálózat bemenetére közvetlenül a kézzel írt számjegy 16×16-os felbontású fekete-fehér pixel képe került. A bemenetek 3 rejtett rétegen keresztül kerültek a kimeneti réteg 10 elemére, amelyek rendre a 0-9 számjegyeknek feleltek meg.

Az első két rejtett réteg a bemeneti kép különböző részeiben felismerhető tulajdonságok felismeréséért felelős, míg az utolsó rejtett réteg és a kimeneti réteg neuronjai ezeknek az információknak a felhasználásával végzik az osztályozást.

7.1. ábra - Többrétegű perceptron struktúra karakterfelismerési feladathoz



Az első rejtett réteg 12 csoportból áll, ahol mindegyik csoport 64 neuront tartalmaz. A csoport minden műveletvégző egysége a bemeneti kép egy 5×5 pixelből álló négyzetével van összekötve úgy, hogy a szomszédos neuronokhoz a bemeneti kép két pixelrel eltoltt szomszédos részletei tartoznak. A csoportok mind a 64 neuronjának ugyanazok a súlyai, így ugyanazt a mintát detektálják a kép különböző részein. Így az első rejtett réteg minden neuronjának 25 bemeneti összeköttetése van. Ez a megoldás úgy biztosítja a kép egyes részeiben rejlő információ feldolgozását, hogy közben jelentősen csökkenti a tanítandó szabad paraméterek számát a teljes összeköttetéshez képest (majdnem 200 ezer kapcsolat helyett mindössze $(25+64)12=1068$, beleszámolva az összes neuronhoz tartozó független eltolás (*bias*) egységeket is).

A második rejtett réteg hasonlóan az elsőhöz egyes részletekben rejlő tulajdonságok felismerését végzi, és az ábrán látható módon az első réteg megfelelő 5×5 -ös részmatrixaival van összeköttetésben. A harmadik réteg 30 neuront tartalmaz, amelyek teljes összeköttetésben állnak a második rejtett réteggel és a 10 kimeneti neuronnal. A hálózat összesen 1256 neuront és 9760 független súlyparamétert tartalmaz.

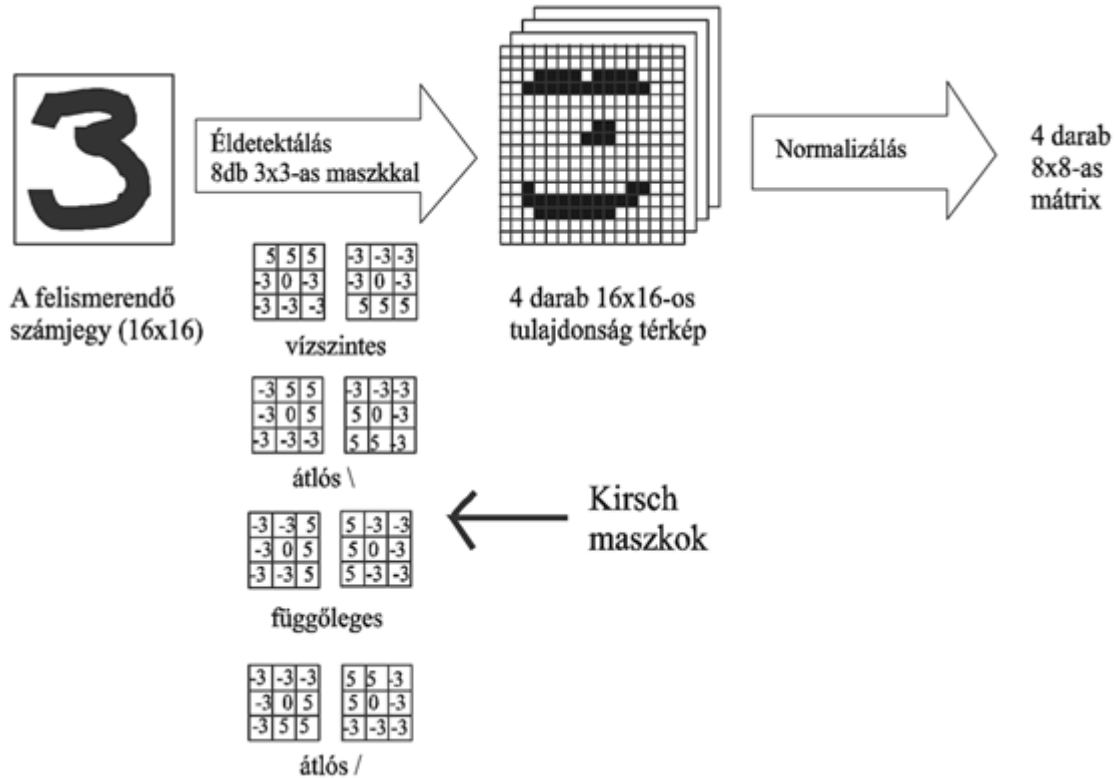
A hálózatot hibavisszaterjesztéses algoritmussal tanították, és a tanuláshoz 7000 számjegyet használtak fel. A tanított hálózat működését végül újabb 2000 számjegy felhasználásával ellenőrizték. A tanított mintahalmaz 99%-át ismerte fel helyesen a rendszer, míg a tesztelésre használt karaktereknél 95%-os eredményt értek el. Mivel az alkalmazás szempontjából előnyösebb, ha a számjegyek egy részénél a rendszer bizonytalanságot jelez, és ezzel csökkenthető a hibás felismerések száma (kisebb kárt jelent egy lassabban feldolgozott levél, mint egy rossz helyre továbbküldött), ezért azoknál az eseteknél, ahol több kimeneti neuron hasonló erősséggel jelezte, hogy a bemenet a hozzá tartozó osztályba való, ott a rendszer eredményét nem vették figyelembe. Így a hibás felismerések arányát sikerült 1% alá szorítani, miközben a levelek közel 10%-át a program kézi feldolgozásra irányította át.

1.2. Számjegyfelismerés egyrétegű hálózattal

A második bemutatásra kerülő megoldás [Kne92] is a 16×16 -os pixel képeket használta bemeneti adatként. Itt viszont az adatok nem kerültek közvetlenül a hálózat bemenetére, hanem egy előfeldolgozó egységen keresztül jutottak oda. A transzformáció alap gondolata az volt, hogy a számjegyek vonalszegmensekből állnak, és ezek

előzetes felismerése megkönnyíti az osztályozást. Ezért a képen négy éldetektáló szűrést hajtottak végre az ún. Kirsch maszkokkal való végigpásztázással, aminek eredményeképpen négy képet kaptak, amelyeken az eredeti karakter vízszintes, függőleges illetve a két átló irányú vonalszegmensei vannak kiemelve. Ezt a 4 képet 8×8 -as méretűre normalizálva alakították ki a hálózat $4 \times 8 \times 8 = 256$ elemű bemenetét.

7.2. ábra - A kép előfeldolgozása

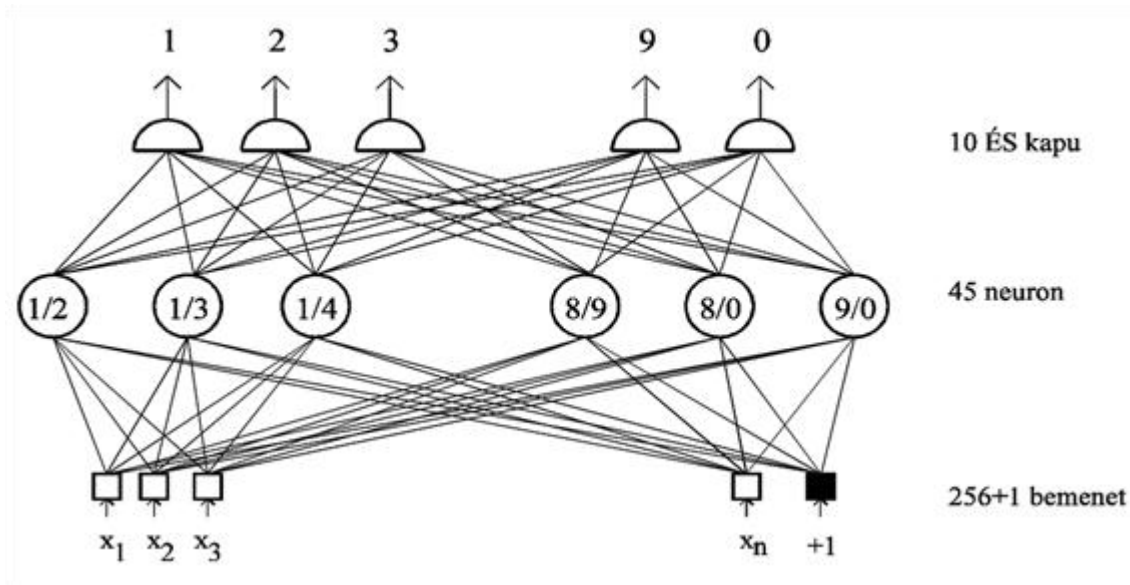


Az előfeldolgozott adat osztályozását egyrétegű perceptronnal oldották meg. Ennek a hálózatnak nagy előnye, hogy a többrétegű hálózathoz képest lényegesen gyorsabb a tanítása, viszont a megoldás létezésének feltétele az osztályok lineáris szeparálhatósága. Az egy hálózatra jutó feladat egyszerűsítése céljából (ami természetesen a hálózat teljesítményének a javulását eredményezi) és a lineáris szeparálhatóság esélyének növelése céljából az osztályozási feladatot szétbontották olyan részfeladatokká, ahol egy-egy hálózat csak két osztály között tesz különbséget. A 10 számjegy felismerését így 45 külön hálózat végezte, melyek mindegyike két osztályt különböztetett meg, rendre az $1/2$, $1/3$, $1/4$, ..., $8/9$, $8/0$, $9/0$ számjegyeket. Az osztályozás utolsó lépésében a megfelelő neuronok kimeneteit 10 darab ÉS kapura vezették, melyek kimenete jelezte az egyes számjegyek felismerését. (Ez a megoldás egy konkrét példája az olyan moduláris hálózat kialakításnak, ahol egy többosztályos osztályozási feladatot kétosztályos részfeladatokra bontunk fel, majd a részfeladatok megoldásaiból aggregáljuk a teljes megoldást. Az ilyen típusú feladat-dekomponálással általánosabban a 9. fejezetben foglalkozunk.)

Ennél a megoldásnál is felmerült, hogy szükséges a módszert úgy módosítani, hogy néhány karakter felismerésének visszautasítása árán a hibás osztályozások száma csökkenjen. Ezt itt úgy oldották meg, hogy a perceptronok döntési szintjét (küszöbét) megemelték, így a bizonytalan felismeréseket kiszűrték.

A hálózatok tanítását itt is gradiens módszerrel végezték, és ugyanazokat a karaktereket használták fel, mint az első megoldásnál. Az elért eredmények szinte teljesen megegyeztek az előző szakaszban bemutatott többrétegű perceptront alkalmazó megoldással, így a két módszer közötti választás a könnyebb implementálhatóság függvénye. Ebben az esetben ezt a kérdést csak a rendelkezésre álló eszközök és az időkorlátok ismeretében lehet eldönteni. Az elsőként bemutatott megoldás előnye, hogy egy lépésben, egy neurális hálózattal oldja meg a feladatot, és a tanított hálózat a felismerés során gyors működést tesz lehetővé. Hátránya viszont, hogy a tanulás folyamata időigényes. Ezzel szemben a második megoldás előnye, hogy egyszerűbb a rendszer és a tanítás lényegesen gyorsabb, viszont a felismerési folyamat során mindig szükség van a bemenő képek előfeldolgozására, ami időigényes.

7.3. ábra - Egyrétegű hálózat előfeldolgozott minták osztályozására



Bár itt a két megvalósítás lényegében egyforma eredményt hozott, a két módszer közti prioritás meghatározására általánosan is meg lehet fogalmazni bizonyos szempontokat. A többrétegű perceptron akkor működik jól, ha nagy mennyiségű, a lehetséges bemeneteket minél jobban reprezentáló tanító minta áll rendelkezésre. Ennek az az oka, hogy a tanítás során a hálózat nagyszámú szabad paraméterét kell beállítani. A másik megoldásnál az előzetes feldolgozás hatékonysága nem függ a mintáktól és az egyszerűbb hálózat beállításához nem szükséges a különösen nagy tanító mintahalmaz. Itt viszont határt szab a felismerő rendszer rugalmasságának az egyrétegű hálózat korlátozott általánosító képessége. Összefoglalva az első, általánosabb megközelítést akkor célszerű alkalmazni, ha a problémához kellő mennyiségű tanító példa áll rendelkezésre (az elégséges minta számát csak a feladat ismeretében lehet meghatározni), míg az előfeldolgozás és a kisebb hálózat használata akkor előnyösebb, ha viszonylag kevés minta áll rendelkezésre és a tanításra fordítható idő is rövidebb.

1.3. Számjegyfelismerés MLP-vel, újabb eredmények

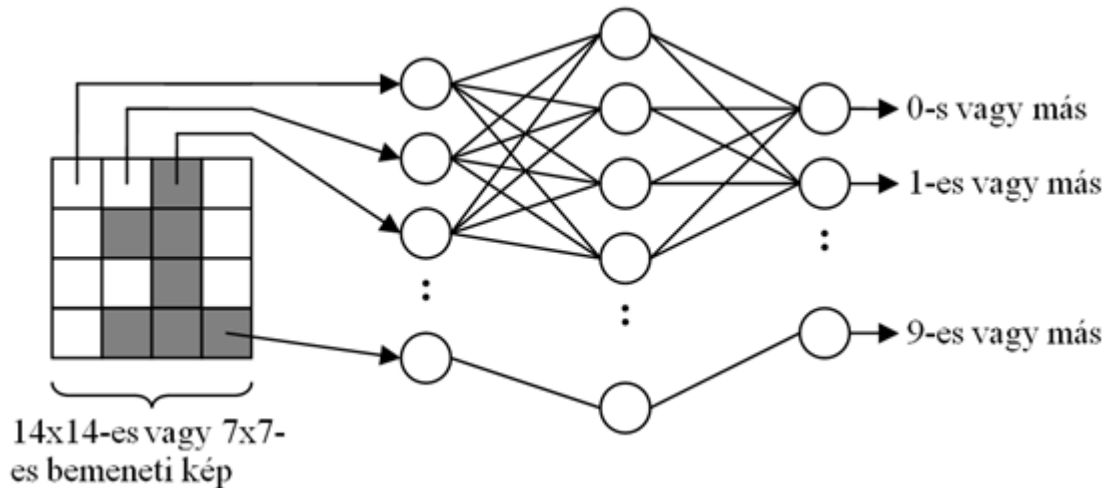
Az első megoldások megjelenése utáni években annyiféle különböző megközelítést próbáltak ki a számjegyfelismerési feladatra, hogy a probléma szinte klasszikussá vált. Bár a feladat alapvető megoldhatósága ma már nem kérdés, az egyre pontosabb és robusztusabb megoldások keresése jelenleg is nyitott terület. A tanuló algoritmusok kipróbálására, kísérletezésre is kitűnően alkalmas ez a gyakorlati színezetű, mégis kellően jól definiált probléma.

Most [Tak03] alapján egy ilyen, a többrétegű perceptron alkalmazásával megvalósult kísérlet eredményeit foglaljuk össze. Érdekes, hogy mennyire másként lehetett megközelíteni ugyanazt a feladatot az első megoldások megszületése után 10 évvel, összehasonlíthatatlanul fejlettebb számítástechnikai feltételek mellett.

Az 1990 körüli megoldásokban közös, hogy kis elemszámú mintakészleteket használtak, illetve a teljes összekötöttségű, „tisztá” MLP-hez képest erősen korlátozták a szabad paraméterek számát. Ennek egyik oka, hogy az akkor jellemző számítási teljesítmények mellett kivárhatatlanul hosszú ideig tartott volna egy $16 \times 16 = 256$ bemenetű MLP tanítása, mondjuk 500 rejtett neuronnal és fél milliós elemszámú tanító készlettel. 2002-ben már ki lehetett próbálni a nyers erő módszerét, vagyis egy hatalmas tanító készlet számjegyeit komoly előfeldolgozás nélkül rá lehetett adni egy nagy neuronszámú MLP bemenetére.

A kísérletben a hálózat a számjegyeket 14×14 -es vagy 7×7 -es felbontású változatban kapta, így 196 vagy 49 bemenettel rendelkezett. A rejtett rétegek számát illetve méretét változtatni lehetett, a kimeneti neuronok száma 10 volt (minden számjegyhez tartozott egy). Egy rejtett réteg esetén a hálózat felépítését a 7.4 ábra szemlélteti.

7.4. ábra - Számjegyfelismerés egy rejtett rétegű MLP-vel



Ez a felépítés tekinthető úgy is, hogy a rejtett réteg nemlineáris transzformációt hajt végre a bemeneten, a kimeneti neuronok pedig 10-féle lineáris szeparálást végeznek a transzformált térben. A tapasztalatok azt mutatták, hogy az egyes számjegyek szétválasztása az összes többitől nem oldható meg lineáris felülettel az eredeti térben. Ilyen helyzetekben merül fel lehetséges megoldásként a dimenziónövelő nemlineáris transzformáció alkalmazása, ami esetünkben azt jelenti, hogy a rejtett réteg nagyobb legyen a bemeneti rétegnél.

A számjegy-felismerési feladatra jellemző, hogy bizonyos típusú bemeneti torzítások kismértékű alkalmazása nem befolyásolja az osztályozás végeredményét. Ilyen például az eltolás, elforgatás, vagy vastagítás illetve vékonyítás. Ezt az a priori tudást sokféleképpen be lehet építeni a számjegyfelismerőbe. Talán a legegyszerűbb megközelítés a számjegyek torzított változatainak hozzávétele a tanító készlethez. 14×14-es vagy 7×7-es képekre a néhány pixellel való eltolás megvalósítható információvesztés nélkül (invertálható módon), ha a kép szélein található egy kis üres sáv. Forgatás vagy vastagítás esetén viszont a túl durva felbontású reprezentáció miatt a bemenet célzott módosítása mellett információvesztés is fellép. Ezért a kísérletben csak a számjegyek eltolt változataival bővítették a tanító készletet, amelynek elemszáma így módon 60.000-ről 540.000-re volt növelhető (1 pixeles eltolást alkalmaztak függőlegesen, vízszintesen és átlósan).

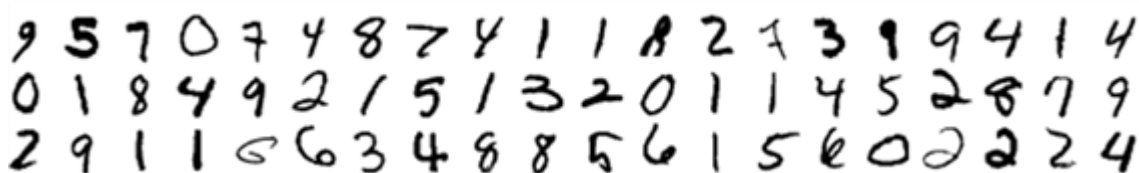
Újabb döntési pont, hogy a bővítés memória vagy számítási erőforrás felhasználásával történjen-e. A kísérletben az utóbbi mellett döntöttek, vagyis csak a számjegyek eredeti változatát tárolták, és minden tanítási lépés előtt egy torzítási transzformációt hajtottak végre (amely esetleg helybenhagyás is lehetett).

A tanítást hagyományos backpropagation algoritmussal végezték, momentum módszer alkalmazásával. A tanítás paramétereinek beállítására tapasztalati úton, kísérletezéssel alakult ki a következő recept:

- A súlyok inicializálása a $[-0,1; +0,1]$ intervallumból való, egyenletes eloszlású sorsolással történt.
- A bátorsági faktort a tanítás elején a hálózat mérete alapján állították be (nagyobb hálózathoz kisebb bátorsági faktor), majd tanítás során folyamatosan csökkentették.
- A momentum együttható értéke konstans 0,8 volt.

A tesztelést egy 10.000-es elemszámú mintakészlettel végezték, amely nem azonos a 10 évvel korábbi számjegyfelismerőknél használt készlettel. Szerencsére az új készlet bővebb és változatosabb a réginél, ezért feltételezhető, hogy az új tesztkészlettel elért eredmények nem romlanának a régi tesztkészletet használva.

7.5. ábra - Részlet a tesztelő készlet számjegyeiből



A tesztelő készlet érdekessége, hogy számjegyei más emberektől származnak, mint a tanító készlet számjegyei. A különböző beállítások mellett elért eredményeket az alábbi táblázat foglalja össze:

7.1. táblázat - A számjegyfelismerő pontossága különböző beállítások mellett

Bemenetek száma	Rejtett neuronok száma	Tanító készlet nagysága	Találati pontosság
49	80	60.000	95,6 %
49	120	60.000	96,5 %
49	180	540.000	96,6 %
196	200	60.000	96,1 %
196	300	60.000	98,0 %
196	600	60.000	98,6 %
196	600	540.000	99,1 %
196	800	540.000	98,9 %

A táblázatból leolvasható, hogy a képek finomabb reprezentálása illetve a tanító készlet bővítése általában növelte a pontosságot, de nem mindig. Ebből is látszik, hogy az MLP hálózatok tanítása némileg heurisztikus folyamat, ahol jellemzően nem egyformán jól sikerülnek az egyes tanítási menetek. Érdekes kísérletezni, a tanítást többször megismételni, mert a tanulási ráta más stratégia szerint történő változtatása, vagy egyszerűen csak eltérő kezdeti súlyok használta esetleg egy kedvezőbb végállapot megtalálásához vezethet. Az elért maximális pontosság sokkal jobb, mint a 10 évvel korábbi eredmények. Érdeklenség, hogy a kísérletben legpontosabbnak bizonyult MLP tanítása kb. 24 óráig tartott

2. Szövegosztályozás

A neurális hálózatok másik tipikus alkalmazási területe a szövegosztályozás. Az interneten található szöveges dokumentumok számának rohamos növekedésével a témakör napjainkban egyre fontosabbá kezd válni. Felmerülő konkrét feladat például a témakör szerinti dokumentum-kategorizálás, a spam szűrés, vagy az automatikus e-mail megválaszolás (ahol egy előre rögzített készletből kell kiválasztani a felhasználó kérdéséhez leginkább illeszkedő választ).

Említettük már, hogy osztályozási feladatoknál kulcskérdés az adatrepresentáció. Szövegosztályozás esetén a szokásos megközelítés, hogy veszünk egy szótárt (pl. a tanító készlet szavainak összessége), amelynek minden szava külön jellemző. Minden szöveget egy valós vektorként reprezentálunk ($\mathbf{x} = [x_1, \dots, x_n]^T$), ahol a szövegben előforduló szavaknak megfelelő komponensek nemzérus értékűek. Tehát szövegosztályozásnál a minták általában sokdimenziós, ritka vektorok. A neuronhálók egyes típusai a sokdimenziós feladatokat eltérő hatékonysággal tudják kezelni. E téren két szélsőség az Albus CMAC háló és a szupport vektor gép, illetve általában a kernel gépek. Azáltal, hogy a sokdimenziós bemeneti térről vagy a jellemzőtérről áttérünk a kernel térre, a ténylegesen megoldandó feladat dimenzióját jelentősen redukáljuk. A szupport vektor gépek tehát jól kezelik azokat a feladatokat, ahol viszonylag kevés mintapontunk van, de ezek sokdimenziósak. A szövegosztályozás tipikusan ilyen feladat, így az SVM igen elterjedt ezeknél a feladatoknál. Ha az egyes szövegek vektorok, akkor a teljes tanító készlet reprezentálható egy mátrixként, amelyet *dokumentum – term mátrix*nak hívunk.

7.2. táblázat - Példa dokumentum-term mátrix

Szó a dokumentumban	...	$(i - 1)$. dokumentum	i . dokumentum	$(i + 1)$. dokumentum	...
...					
Sphere		0	*	0	
Spice		0	0	0	
Spider		*	0	0	
Spike		0	0	*	
Spin		0	0	*	
...					

A konkrét jellemzőértékek beállítását többféle módszer szerint lehet végezni. A legegyszerűbb, ha a nemzérus komponenseket egységesen 1-re állítjuk (bináris reprezentáció). Ennél kicsit kifinomultabb a TF (Term Frequency) reprezentáció, amikor $x_i = TF(i)$, vagyis az i -edik komponens értéke az i -edik szótárszó előfordulásainak száma az adott dokumentumban. Talán a leggyakrabban használt módszer a szöveges dokumentumok reprezentálására a TF-IDF (Term Frequency – Inverse Document Frequency) séma:

$$x_i = TF(i) \cdot IDF(i) = TF(i) \cdot \log_2 \left(\frac{|D|}{DF(i)} \right) \quad (7.1)$$

ahol $|D|$ a dokumentumok száma, $DF(i)$ pedig azon dokumentumok száma, amelyekben az i -edik szó előfordul.

2.1. Spam szűrés SVM-mel

Az elektronikus levélszemét (*spam*) kiszűrése az internetes levelezésből napjainkban igen aktuális gyakorlati feladat. A levelek osztályozása spam-ekre illetve normál levelekre nem oldható meg néhány előre rögzített, egyszerű szabály segítségével. A mintákból való tanulás képessége ezen a területen is igen hasznos tulajdonság, így természetes lehetőségként kínálkozik a neurális hálózatok alkalmazása. Az alábbiakban egy szupport vektor gépes megoldást ismertetünk [Dru99].

Egy spam kategorizáló rendszer (mint minden kétosztályos osztályozó rendszer) alapvetően kétféle hibát tud követni: nem szűr ki egy spam-et, vagy a spam-ek közé helyez egy normál levelet (általánosan az első a téves negatív, a második a téves pozitív osztályozás). Jogos elvárás, hogy a felhasználó állíthassa be a szűrés szigorúságát, vagyis ő dönthesse el, hogy melyik típusú hibázást szeretné jobban elkerülni. Ehhez viszont az kell, hogy az osztályozó rendszer kimenete ne csak egy igen-nem típusú logikai érték legyen, hanem egy „valószínűség”, amely megadja, hogy az adott levél mekkora eséllyel számít spam-nek. A szupport vektor gépek automatikusan ilyen bizonyossági értéként is értelmezhető választ állítanak elő, hiszen a kimenetük az osztályozandó mintának az elválasztó hipersíktól vett (előjeles) távolságával arányos.

Az ismertetett spam kategorizáló rendszer a bináris és a TF szöveg-reprezentációt használta, néhány kis gyakorlatias módosítással. A szótárba csak azokat a szavakat vették be, amelyek tanító készlet dokumentumai közül legalább háromban szerepelnek. Így kiszűrhetők a szótárból az elírt ill. a ritka változatban használt szavak. Ezen kívül kísérleteztek tiltólista használatával a minden szövegben gyakran előforduló, kevés információt hordozó névelők, kötőszavak, stb. kiszűrésére.

Az osztályozók minősítésére a legegyszerűbb mérőszám a hibaarány. Bizonyos feladatoknál (pl. számjegyfelismerés) ez az egyetlen szám is eleget mond, mivel nem igazán érdekes a különböző típusú tévesztések aránya. Spam felismerés esetén azonban ez nincs így: itt érdemes külön lemérni a spam átengedési (téves negatív) ill. a normál levél kiszűrési (téves pozitív) arányt. Egy ésszerű megközelítés, ha a felhasználó

beállíthatja, hogy legfeljebb mekkora normál levél kiszűrési arányt tolerál. A cél a spam átengedés minimalizálása a normál levél kiszűrésre vonatkozó feltétel betartása mellett.

A teszteléséhez használt adatbázis 3000 levelet tartalmazott. Ebből 850 volt spam, 2150 pedig normál üzenet. A rendszer teljesítményének mérésére 10-szeres kereszt kiértékelést alkalmaztak. Természetesen több különböző beállítást is kipróbáltak. Lemérték, hogy mennyire romlanak le az eredmények, ha a leveleknek csak a tárgy mezőjére (*subject*) vagy csak a törzsére (*body*) támaszkodnak. Megvizsgálták, hogy milyen változást okoz a tiltólista használata. A kapott eredményeket az alábbi táblázat foglalja össze:

7.3. táblázat - Spam átengedés 5 %-os normál levél kiszűrés mellett

	TF reprezentáció	Bin. reprezentáció
Tárgy alapján, tiltólista nélkül	54,9 %	54,9 %
Tárgy alapján, tiltólistával	71,9 %	75,8 %
Törzs alapján, tiltólista nélkül	9,6 %	9,3 %
Törzs alapján, tiltólistával	12,0 %	11,5 %
Tárgy + törzs alapján, tiltólista nélkül	1,8 %	1,5 %
Tárgy + törzs alapján, tiltólistával	2,7 %	3,2 %

A legjobb eredményeket a bináris reprezentáció segítségével sikerült elérni, de volt olyan beállítás is, amelyben a TF reprezentáció bizonyult jobbnak. Érdekes, hogy a tiltólista használata minden esetben rontott az eredményeken. Ez arra enged következtetni, hogy az SVM ügyesebben választotta ki, hogy mely jellemzőket érdemes figyelembe venni a döntésnél, mint a tiltólistát összeállító emberek.

3. Ipari folyamatok modellezése

Az ipari folyamatmodellezés fogalma rendkívül széles feladatkört takar. A konkrét gyakorlati esetekben a legkülönbözőbb jellegű és nehézségű feladatok merülhetnek fel ilyen címszó alatt. A megoldás néha egy egyszerű lineáris modell, máskor emberi szakértőket kell alkalmazni, akik ráadásul nem képesek tudásukat formálisan megfogalmazni. A neurális hálózatok alkalmazása ezen a területben alapvetően két esetben merül fel:

- A bemenet és a kimenet közötti kapcsolat ismert, de olyan bonyolult, hogy jobban megéri csak a mérésekből szerzett bemenet-kimenet párokra támaszkodni, és egyszerűbb, közelítő modellt építeni.
- A bemenet és a kimenet közötti összefüggés nem ismert.

A bonyolult rendszerek modellezése a neuronhálók egyik legfontosabb alkalmazási területe. A modellezési problémák jelentős része azonban dinamikus rendszerekhez kapcsolódik, ezért a kérdést általánosabban a következő fejezetben tárgyaljuk. Az alábbiakban két konkrét feladat kapcsán néhány fontosabb tapasztalatot foglalunk össze.

3.1. Gumigyártási folyamat modellezése RBF-fel

A gumigyártás modellezése is ilyen „kemény diónak” számít. A folyamat számunkra érdekes része úgy indul, hogy az alapanyagokat egy olvasztó-préselő gépbe (ún. extruder-be) öntik. Az extruder a gumikeveréket 110-140 °C fokon megolvasztja, majd a masszát 70-140 bar nyomással egy fémmaszkon préseli keresztül. Az extruder elhagyása után a gumikeverék megszilárdul, és közben deformálódik. Az deformáció a fémmaszk alakjának, valamint egyéb paramétereknek (pl. extruder típusa, hőmérséklet, keverék sűrűsége) a bonyolult,

nemlineáris függvénye. A cél a fémmaszk megtervezése úgy, hogy a lehűlés és alakváltozás utáni gumiprofil minél közelebb legyen a kívánthoz.

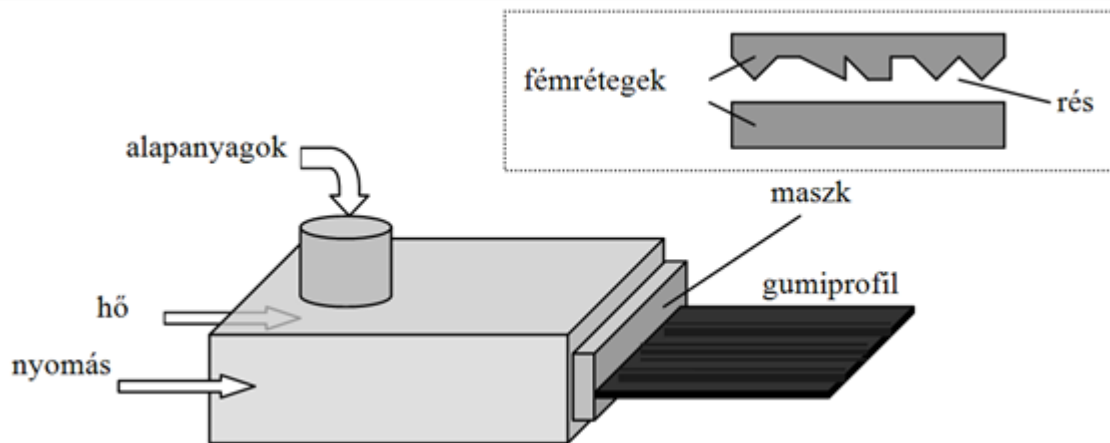
Mivel az extruder-t elhagyó anyag végső, szilárd formája egy nehezen megjósolható alakváltozás során alakul ki, minden új termék gyártásának megkezdése előtt szükség van egy több iterációból álló kísérletezési szakaszra. Elkészítik a fémmaszket, helyezik az extruder-re, végeznek vele egy próbagyártást, lemérik a kapott gumiprofil, majd a mérés eredménye alapján pontosabb maszket próbálnak készíteni. Egy ilyen iteráció költséges, ráadásul 4-5 napot is igénybe vehet, ezért egyáltalán nem mindegy, hogy hány maszkfinomítással sikerül elérni a kívánt pontosságot.

Az alkalmazottak, akik a kísérletezést végzik, teljesen hozzá vannak kötve ehhez a munkakörhöz (ami ráadásul unalmasnak számít). Mivel a „maszkfinomító” szakértelem csak lassan és elég heurisztikus módon szerezhető meg, ezen szakemberek betegsége vagy más céghez való távozása érzékenyen érintheti a gyárat.

Rendkívül hasznos lenne tehát, ha sikerülne kellően pontosan modellezni a megszilárdulás folyamatát. Még közvetlenebbül használható lenne egy inverz modell, amely megadja, hogy milyen fémmaszk paraméterek vezetnek a kívánt gumiprofil kialakulásához. Ha létezne ilyen, akkor a szakembereknek a kísérletezési fázis során rendelkezésre állna egy értékes támpontot adó szakértői vélemény. Itt merül fel a neurális hálózatok alkalmazásának ötlete, mivel a neurális hálózatok képesek modellezni olyan bonyolult, nemlineáris kapcsolatokat, amelyek nehezen írhatók le hagyományos eszközökkel.

Egy, a 90-es évek végén kifejlesztett neurális szakértői rendszerben radiális bázisfüggvényes (RBF) hálózattal próbálták megoldani a fent említett inverz modellezési feladatot [Pie99]. Az RBF hálózatok tanítására két „tisztá” megközelítés ismeretes. Az első megközelítés szerint a két réteg tanítása külön történik. A bázisfüggvény középpontokat és paramétereket egy klaszterező végeredménye alapján alakítják ki, majd a rejtett réteg súlyait rögzített bázisfüggvények mellett LMS algoritmussal tanítják. A második megközelítés szerint a hálózat összes paraméterét egyszerre tanítják, gradiens módszerrel. Az első megközelítés előnye a kis számításigény és a lokális minimumoktól való mentesség. Az egyszerűségért azonban pontosságcsökkenéssel fizethetünk. Ennek egyik oka, hogy a klaszterezés csak a tanítópontok pozícióját veszi figyelembe, így nem mintavételezi sűrűbben a közelítendő függvényt a gyors változás helyein. Az ismertetett rendszerben olyan megközelítést alkalmaztak, amely egyrészt szabályozási feladatokra van specializálva, másrészt próbálja ötvözni a két tiszta tanítási módszer előnyeit úgy, hogy közben a hátrányokból minél kevesebbet tart meg.

7.6. ábra - Az extruder működése, vázlatosan



Az alap RBF változatban a minimalizálandó mennyiség az átlagos négyzetes hiba. Szabályozási feladatok esetén viszont általában a maximális abszolút hiba minimalizálása a cél, ezért a szakértői rendszerben ezt használták kritériumfüggvényként. Egyébként azért választották ezt a hálózattípust, mivel RBF esetén a kritériumfüggvény ilyen irányú módosítása egyszerű és áttekinthető módon tehető meg.

A tanítást hálónövesztős módszerrel végezték, a következő módon: minden iterációban megkeresték, hogy melyik tanító pontnál volt a legnagyobb az abszolút hiba. Ha a hiba meghaladott egy küszöböt, akkor beillesztettek a hálóba egy új rejtett neuront úgy, hogy az új bázisfüggvény középpontja a maximális hiba helyére essen. Az új bázisfüggvény szélességparamétereit a szomszédos bázisfüggvények alapján állították be. Ezután meg lehetett határozni az új neuronhoz tartozó súlyt úgy, hogy az új bázisfüggvény középpontjánál a

hiba nulla legyen (vagy újra lehetett tanítani az egész kimeneti réteget). (A megoldás – legalábbis céljait tekintve – hasonlít az 5. fejezetben ismertetett OLS eljárásra.)

Konkrét gyakorlati feladatunk esetén a cél a fémmaszk alakjának meghatározása magasságértékek $[g_1, \dots, g_n]$ sorozataként az $[r_1, \dots, r_n]$ kívánt gumiprofilból (ami szintén magasságértékek sorozata) és a gumikeverék sűrűségéből. Feltételezhetjük, hogy a keresztmetszet egy adott pozíciójánál létrejövő alakváltozásban csak a környező részekben található gumi magassága játszik szerepet. Ezért a g_i meghatározásához csak a szomszédos területekhez tartozó $[r_{i-k}, \dots, r_{i+k}]$ gumiprofil értékeket érdemes felhasználni. (Valójában az extruder működéséből adódóan a rés mentén nem egyenletes a nyomás, ezért az alakváltozás az i abszolút pozíciótól is függ, de azt tapasztalták, hogy ez a hatás elhanyagolható.)

A kutatás eredménye félsiker volt. Az elért pontosság arra még nem alkalmas, hogy kiiktassa a kísérletezési fázist a terméktervezésnél, de a kapott neurális modell segítheti, felgyorsíthatja a szakemberek munkáját. Érdekes tapasztalat, hogy az általános helyzetű elliptikus bázisfüggvények alkalmazása ennél a feladatnál jelentős javulást hozott a körszimmetrikus bázisfüggvények használatához képest.

3.2. Acélgyártási folyamat modellezése

A Linz-Donawitz (LD) konverteres technológiával végzett acélgyártási folyamat neurális hálós modellezése is tipikus példája a neuronhálók ipari alkalmazásának. A konverteres acélgyártás az acélgyártási technológiák közül hatékonysága miatt az egyik legelterjedtebben alkalmazott eljárás. Az LD konverteres acélgyártás során egy nagy tartályban – ezt nevezik konverternek – történik a nyersvasból az acél készítése. Itt zajlanak le azok a fizikai és kémiai folyamatok, melynek során a kb. 4%-os szénttartalmú, mintegy 1400 °C-os folyékony nyersvasból kb. 0,05% szénttartalmú, mintegy 1670 °C-os acél készül, miközben a nyersvasban meglévő számos egyéb szennyezőanyag is kiég. Az eljárás hatékonysága abból ered, hogy a gyártási folyamathoz külső energiára nincs szükség, ugyanis a szén és a további szennyezőanyagok kiégetése hőtermelő folyamat. Így maga a kémiai folyamat biztosítja, hogy a vég hőmérséklet magasabb lesz, mint a kiinduló hőmérséklet. Az eljárás legfontosabb fázisa az ún. fúvatás, amikor a konverterben lévő olvadékok nagy tisztaságú oxigént fúvatnak. A gyártási folyamat során a megfelelő minőségű acél elérése érdekében a nyersvason kívül hulladékvas és számos adalékanyag felhasználása szükséges.

A gyártási folyamat főbb lépései a következők:

1. A megfelelő összetevők betöltése a konverterbe. Ezt nevezik betétösszeállításnak. A konverterbe hulladékvas, folyékony nyersvas és a kémiai folyamatokat segítő adalékanyagok kerülnek. A betét összeállítása, alapvető fontosságú a legyártott acél minősége szempontjából. A magas hőmérsékletű nyersvas felolvasztja az eredetileg szilárd hulladékvasat is, így a konverterben olvadék alakul ki.
2. Fúvatás. A fúvatás nagy tisztaságú (tipikusan 99,5%-os tisztaságú), nagy nyomású oxigénnel történik az olvadék felett elhelyezkedő ún. lándzsa (vízhűtéses cső) segítségével. A fúvatás eredményeképp beindulnak a megfelelő kémiai és fizikai folyamatok, kiégnek a szennyezők és a felesleges szén, valamint az egész olvadék hőmérséklete megemelkedik. A fúvatás az acélgyártás lényegi fázisa, ennek eredményeképp alakul ki adott kémiai összetételű és hőmérsékletű olvadék, ami a további feldolgozás (pl. ötvöztetés, stb.) kiinduló alapja. A fúvatás időtartama kb. 20-25 perc, mely során mérések végzésére nincs lehetőség.
3. Hőfokmérés, próbavétel. A fúvatás befejeztével egy gyors kémiai elemzéssel és hőmérsékletméréssel történik az elkészült acél legfontosabb jellemzőinek meghatározása.
4. Csapolás.

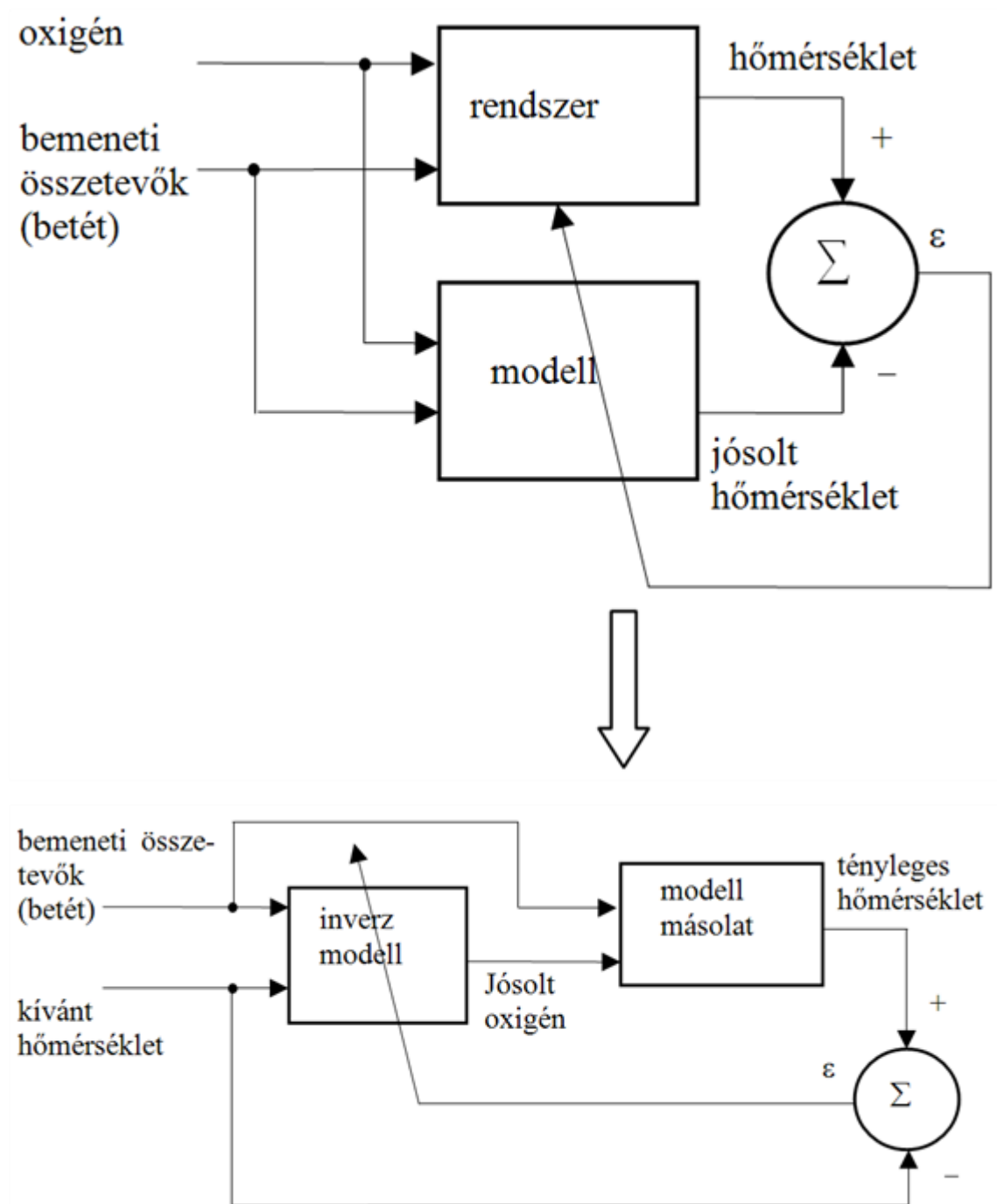
Az alapfeladat a betét megfelelő összeállítása és a fúvatási paraméterek, alapvetően a fúvatásnál felhasznált oxigén mennyiségének meghatározása. Minthogy a technológiából adódóan a gyártási folyamat közben mérések elvégzésére nincs lehetőség, az acél minőségét csak a fúvatás végén lehet megállapítani. A végtermék minőségét elsősorban a szénttartalom és hőmérséklet jellemzi, tehát ez az a két minőségjellemző paraméter, melyeknek megfelelően szűk tartományon belül kell lenniük. A végtermék minőségét – ha adott betétből indulunk ki – elsődlegesen a fúvatási fázis lezajlása és ezen belül is a felhasznált oxigén mennyisége határozza meg. A megfelelő minőségű acél előállításának biztosításához tehát a felhasználandó oxigén mennyiségét kell meghatározni.

Bár a végtermék összetételét és hőmérsékletét mind a kiinduló betétösszetétellel, mind a fúvatással befolyásolhatjuk, a feladat komplexitása és számos nem, vagy nehezen kézbe tartható jelenség miatt a

betétösszeállítás és a fűtési folyamat együttes optimalizálása nem járható út. Ezért a betét összeállítását adottnak vesszük (a valóságban általában ez a helyzet). A feladat tehát a betétösszeállítás többé-kevésbé pontos ismerete és a célul kitűzött kémiai összetétel és véghőmérséklet ismeretében a felhasználandó oxigén mennyiségének a meghatározása.

Az adagolandó oxigén minél pontosabb meghatározása alapvető gazdasági érdek. Ha a fűtásvégi paraméterek nem megfelelőek (nem megfelelő hőmérsékletű és/vagy karbontartalmú az acél), a minőséget jellemző paraméterek csak jelentős idő- és költségigényes korrekciós műveletekkel módosíthatók.

7.7. ábra - Az előrejelzés, mint modellezési és inverz modellezési feladat.



A megoldandó feladat ennek megfelelően egy olyan tanácsadó rendszer létrehozása, mely a rendelkezésre álló adatok (kiinduló betétösszetétel, a konverter állapotjellemzői és a célparaméterek: végkarbon és véghőmérséklet) alapján javaslatot ad a fűtás során felhasználandó oxigén mennyiségére. Lényegében tehát

folyamatvezérlési feladatot kell megoldani, melynek elengedhetetlen feltétele a kérdéses folyamat valamilyen modelljének felállítása. A modell a vizsgálandó folyamat bemenetei és kimenetei közötti kapcsolat minél pontosabb reprodukálását kell, hogy megvalósítsa. A modell ismeretében hozható létre a fűtési oxigénre javaslatot tevő előrejelző, amely valójában a folyamat inverz modellje, mint ahogy ez az 7.7 ábrán látható [Hor99]. Az inverz modell kimenete állítja elő a folyamat vezérlőjelét (az ábrán a folyamat modelljének vezérlőjelét), ami az adott feladatban a javasolt oxigén mennyisége.

Egy folyamatmodell többféle megközelítésben is létrehozható. Lehet fizikai-kémiai modellt létrehozni, ahol a modell a rendszer felépítésének és működésének mechanizmusát is igyekszik minél pontosabban visszaadni, illetve készíthetünk viselkedési modellt, amikor csak az a cél, hogy a modell kívülről nézve minél inkább úgy viselkedjen, mint a modellezendő rendszer. Ilyenkor fekete doboz modellezésről beszélünk. Viselkedési vagy fekete doboz modell konstrukciójához bemeneti-kimeneti mintapontokra van szükségünk.

Fekete doboz modell létrehozását két fő szempont indokolhatja. Lehetséges, hogy a modellezendő rendszerről nem áll rendelkezésünkre annyi információ, ami fizikai-kémiai modell létrehozását lehetővé tenné, de az is lehet, hogy egy ilyen fizikai-kémiai modell létrehozása nagyságrendekkel bonyolultabb lenne, mint egy viselkedési modell felállítása. A konverteres acélgyártásnál valójában mindkét ok fennáll. Egyrészt a gyártási folyamat tényleges lezajlásáról nem szerezhető annyi információ, ami egy megfelelő pontosságú fizikai-kémiai modell létrehozását biztosítaná, másrészt, ha ilyen információ rendelkezésünkre is állna, akkor is túl komplex lenne a modellezési feladat. Ugyanakkor rendelkezésünkre állnak a gyártási folyamat során összegyűjtött bemeneti-kimeneti mintapárok. Az adott probléma megoldásánál ezért a fekete doboz felállítása volt a cél, ahol a fő modellezési eszköz a neuronháló volt. (A neuronhálók rendszermodellezésre történő alkalmazásával a 8. fejezetben általánosabban is foglalkozunk.)

Mint a gyakorlati modellezési feladatok többségében, a probléma megoldásának egyik kulcskérdése a megfelelő adatbázis létrehozása és alkalmazása. Ez egyrészt azt jelenti, hogy a problémateret kellő mértékben le kell fednünk tanítópontokkal, másrészt azt, hogy ezen tanító pontok kellő "megbízhatóságuk" kell legyenek, továbbá, hogy az adatokat megfelelő módon kell reprezentálnunk. Az ipari környezetből adódóan az acélgyártási folyamat leírására egy meglehetősen nagy, ugyanakkor pontatlan, zajos és hiányos adatbázis áll rendelkezésre. Ennek megfelelően a munka lényegi részét képezte az adatbázis megfelelő kialakítása, ahol az alábbi problémákat kell megoldani (itt csak az adott alkalmazási feladatban felmerült speciális problémákat említjük meg, az adatbázis kialakítással kapcsolatos általános szempontokat – melyek jórészt ennek a munkának a kapcsán merültek fel – a 13. fejezetben tárgyaljuk):

- A túl nagy adatdimenzió által okozott problémák.
- Az adatok "megbízhatóságának" vizsgálata. Zajos és torzított adatok kezelése.
- A bemeneti adatok normalizálása.
- Hiányos adatok kezelése, hiányzó adatok pótlása.

Az adatbázis létrehozásának eredményeként egy kb. 4000 mintapontból álló adatkészlet állt elő (melynek kb. 60%-a tanításra, 40%-a tesztelésre volt felhasználva), mely 30-dimenziós bemeneti és kétdimenziós kimeneti adatokat tartalmazott. Ennek az adatkészletnek a felhasználásával történt a neurális modellek létrehozása és tesztelése. Az alkalmazásnál különböző méretű MLP-k kerültek felhasználásra. A legjobb eredményt két rejtett rétegű hálózattal sikerült elérni, ahol az első rejtett rétegben 10, a másodikban 5 neuron volt.

A tesztkészleten történő kiértékelés szerint a neurális háló által javasolt oxigénmennyiség alapján az esetek mintegy 75-80%-ában a leggyártott acél kimeneti paraméterei az előírt toleranciasávon belül voltak, ami az adott problémát tekintve igen jó eredmény. Meg kell azonban jegyezni, hogy a feladat komplexitása, a rendelkezésre álló viszonylag kisméretű mintakészlet és az adatok nagymértékű zajossága miatt, valamint abból adódóan, hogy ritkán speciális minőségű acél gyártására is szükség van, a feladat tisztán neurális hálózattal történő megoldása megfelelő minőségben nem lehetséges (a speciális minőségű gyártási esetekről nagyon kevés mintapont áll rendelkezésre, így ezen esetekre a neuronháló meglehetősen rossz eredményre képes csak). A ténylegesen kialakított modell ezért egy hibrid-neurális megközelítés eredményeként született, amely arra is képes, hogy a mintapontokon túl, más formában (pl. szabályok formájában) rendelkezésre álló információt is fel tudjon használni. A hibrid-neurális rendszerek kialakításának néhány fontos szempontjával a 12. fejezet foglalkozik, a konverteres acélgyártási folyamat hibrid-neurális modellezésének részletei pedig a megfelelő irodalmakban megtalálhatók: [Pat98], [Str98], [Hor99], [Pat00], [Ber01].

3.3. Néhány egyéb alkalmazás

Az előbb ismertetett problémákon kívül az elmúlt években számos érdekes és/vagy gyakorlatilag fontos alkalmazás született. Az érdekes problémák között kell megemlíteni néhány olyan, a 80-as évek végén, 90-es évek elején készült alkalmazást, melyek célja inkább a figyelemfelkeltés, mint az igazi gyakorlati alkalmazás volt, bár természetesen az elkészült rendszerek gyakorlati feladatok megoldására is használhatók.

Az egyik első ilyen alkalmazás volt a NETtalk, amely egy angol szöveg-beszéd konverter [Sej87]. A hálót (MLP) egy olyan rendszerben alkalmazták, amely a bemenetére kerülő szöveget a rendszerben szintén meglévő beszéd szintézer segítségével alakította beszéddé. Ehhez az kellett, hogy a szöveg egyes karaktereihez olyan kódokat rendeljen egy neurális háló, mely kódok a beszéd szintézer meghajtására alkalmasak. A teljes rendszer ezért két fő modulból áll: egy karakter-fonéma kód leképező neurális hálóból és egy hozzá kapcsolódó szintézerből. A megfelelő fonéma előállítása nem végezhető el környezetfüggetlen módon, ezért a karaktereket egymagukban nem dolgozhatja fel a neuronháló. A neuronháló bemenetére 7 egymást követő karakter bináris reprezentációja került, melyek közül a középsőnek megfelelő fonéma kód előállítása volt a neuronháló feladata. A leképezést egy 203 bemenetű 80 rejtett neuronnal felépülő 26 kimenetű hálónak tanították meg. Megemlítjük, hogy a NETtalk egy szabály alapú, hasonló célú és kereskedelmi forgalomban kapható rendszer, a DECtalk mintájára készült. A neurális megközelítés előnye a viszonylag rövid fejlesztési idő. A háló tanítása nagyságrendekkel rövidebb idő alatt megtörtént, mint a szabály alapú rendszer kifejlesztése.

Ugyancsak nevezetes alkalmazás az autonóm autóvezető rendszer (*autonomous land vehicle in a neural network, ALVINN*), melyet Carnegie-Mellon Egyetemen fejlesztettek ki, és amely egy gépkocsivezető cselekvése alapján tanít meg egy neurális hálót az autóvezetésre [Pom91]. A rendszer durva felbontású (30x32 képpontos) videoképet és a hozzá tartozó, sofőr által beállított kormánybeállítást kapott bemenetként. A feladatra 5 rejtett neuronú, 30 kimenetű MLP-t alkalmaztak. A 30 kimenet 30 lehetséges kormánybeállítást jelentett. A hálót vagy 1 a 30-ból kódolású kívánt válasszal tanították, vagy a kívánt választ olyan Gauss jelként fogalmazták meg, melynek maximális értéke a kívánt kormánybeállítást képviselő neuronhoz került. A feladat nehézségét a kellő számú tanító kép (mintapont) létrehozása jelentette. A megfelelő mennyiségű tanítómintához (összerendelt kép és kormánybeállítás) a valóságosan felvett képek transzformált változatait is felhasználták, sőt a tanítókészletet az egyes tanító ciklusok során részben cserélték is, hogy a háló ne alkalmazkodjon túlzottan egy adott tanítókészlethez. Közel 50 tanítási ciklus után, miközben a kocsit kb. 4 mérföld/óra sebességgel vezették a kijelölt úton, a háló nemcsak a megtanított környezetben, hanem attól eltérő úton, a tanítási körülményektől eltérő környezeti feltételek pl. más időjárási viszonyok között is képes volt a kocsit automatikus kormányozására mintegy 25 mérföld/óra max. sebességig.

A neurális hálózatok alkalmazása olyan esetekben is felmerül, amikor a szokásostól jelentősen eltérő, extrém körülmények miatt a jól bevált, hagyományos megoldás valamelyik infrastrukturális feltétele (pl. különböző mérőszensorok megléte) nem, vagy csak nehezen biztosítható. Ez a helyzet például a kis teljesítményű (max. 15 kW-os) szélturbinák esetén, amelyeknek egyik fő alkalmazási területe a nehezen megközelíthető helyek árammal való ellátása.

Egy ilyen rendszer hatékony működtetéséhez a szélesebb és a lapátvég-sebesség arányát (tip-speed ratio) lehetőleg konstans szinten kell tartani. Ez a generátorban található forgórész (ún. rotor) szögsebességének szabályozásával érhető el. A maximális energiakinyerést biztosító optimális szögsebesség a szélesebb, és a pillanatnyilag leadott teljesítmény ismert függvénye. A hagyományos megközelítés szerint szélesebb mérő szenzorokat helyeznének el a turbina környezetében, de ez az előbb említettek miatt költséges és nehezen karbantartható lenne.

A Floridai Egyetemen e helyett neurális megoldást fejlesztettek ki [Li05], amely szenzorok nélkül próbálja megbecsülni a szélesebb és a rotor szögsebességéből és a pillanatnyilag leadott teljesítményből. Konkrétan 2 bemenetű, 5 rejtett neuronnal rendelkező, 1 kimenetű MLP-t használtak. A tanításhoz 10.000 mintahármast [szögsebesség, teljesítmény, szélesebb] gyűjtöttek össze mérésekből. A neurális hálózat adaptivitását kihasználva a rendszert később olyan irányban is továbbfejlesztették, hogy képes legyen érzékelni, ha idővel megváltozik a maximális hatékonyságot biztosító tip-speed arány. A szélturbinát először szoftveres, majd hardware-in-loop szimulátorban valósították meg. A teszteredmények azt mutatták, hogy a szélesebb-becsülés megoldás hatékonysága nem sokkal marad el a szenzorokkal elérhetőhöz képest. (Pl. az egyik kísérleti szélprofil maximumánál, egy 11 m/s-os széllökés közepén a rendszer 3.7 helyett 3.5 kW teljesítmény leadására volt képes.)

Szintén érdekes példa a neurális hálózatok szélsőséges körülmények között történő alkalmazására a Sao Paolo-i Egyetemen kifejlesztett tengeri olajkút monitorozó rendszer [Sim00]. Egy olajkút megfelelő üzemeltetéséhez

rendszeresen mérni kell a hőmérsékletet és a nyomást a kút alján (ami néha jóval a tengerfenék alatt található). A problémát a mérési eredmények továbbítása jelenti. A hagyományos megközelítés szerint vezetékes átviteli közeget építenek ki a felszín és a kútfének között, de ez a megoldás igen drága, és sérülékeny. A rádióhullámok ebben az esetben nehezen használhatók, mert a vízben rendkívül nagy az elnyelődés. Olcsó és robusztus alternatív megoldás az akusztikus adatátvitel használata, az acél olajszállító csövet használva átviteli közegként. A bináris adatok átvitele 2 különböző frekvencián történik, frekvenciamodulációs technika segítségével. A szokatlan átviteli közeg miatt persze számos probléma merül fel (pl. zaj, elnyelődés, ill. a csőelemek kapcsolódásánál többszörös visszaverődés, ill. nemlineáris torzulás), így a hagyományos demodulációs technikák nem alkalmazhatók. Az ismertetett megközelítés feladathoz igazított előfeldolgozást, és counterpropagation hálózatot [Hec89] használt¹. Az előfeldolgozás első része még nem feladat-specifikus: minden bit dekódolásánál meghatározzák a jelet tartalmazó időablakot, és mintavételezik a jelet. A neurális hálózatnál alkalmazott recepteket, stratégiákat neurális hálózat szimulátor program használatával fejlesztették ki, majd a legcélszerűbbnek bizonyult hálót C++-ban leprogramozták.

A rendszert laboratóriumban megépítették, és 100 méteres csővezetékkel tesztelték. A bemeneti jeleket szándékosan Gauss-zajjal torzították, hogy valamennyire felkészüljenek a laboratóriumtól eltérő, valós körülményekre. A vevő így a bitek 97,5 %-át dekódolta helyesen.

A neuronhálókat számos egyéb ipari probléma megoldásánál is alkalmazzák. Az előbbieken említett acélipari alkalmazáson túl számos további acélipari alkalmazásra találunk példát. [Hao04] pl. egy nagykohó neurális, illetve hibrid-neurális modellezéséről számol be, míg [Zar00] az acélgyártásban alkalmazott hideghengerműben a hengerlési erő meghatározására szolgáló neurális megoldást ismertet. Egyéb alkalmazásokról szóló beszámolók a bőséges irodalomban találhatók.

¹ A counterpropagation (CPN) hálózat egy olyan kétrétegű háló, melynek első rétege a bemeneti tér klaszterezését végzi egy Kohonen hálóval (ld. 11. fejezet), a kimeneti rétege pedig egy felügyelt tanítású lineáris réteg. Fő előnye a gyors tanulás.

8. fejezet - Időfüggő (szekvenciális) hálók

Az eddig bemutatott hálók legfontosabb jellemzője, hogy a bemenetükre kerülő adatok és az ezekhez tartozó kívánt kimenetek közötti *statikus* leképezés megtanulására képesek. Számos olyan feladattal találkozunk azonban, amikor a statikus modell nem megfelelő, a bemeneti és a kimeneti adatok közötti kapcsolat az adatok sorrendjétől is függ. Ilyenkor olyan hálókra van szükségünk, ahol a hálózat válasza nemcsak a pillanatnyi bemenettől, hanem a mintasorozat időben előbbi értékeitől is függ. Ilyen feladattal találkozunk, ha nemlineáris dinamikus rendszerek modellezését, irányítását kell megoldanunk, illetve olyan előrejelzési, jóslási problémáknál, ahol egy folyamat régebbi viselkedésének ismeretében kívánjuk megjósolni a folyamat viselkedésének további alakulását.

Az időfüggő hálók a statikus hálóból származtatott olyan rendszerek, amelyek emlékezettel, memóriával rendelkeznek. Ez azt jelenti, hogy a hálózat kimenete nem csupán a pillanatnyi bemeneti értéktől, hanem régebbi bemeneti és/vagy a régebbi kimeneti értékektől is függ.

Az időfüggő hálózatok általánosságban nemlineáris dinamikus viselkedést mutatnak, tehát mint nemlineáris dinamikus rendszermodellek tekinthetők. Ebben a megközelítésben az időfüggő hálók a lineáris dinamikus rendszermodellek nemlineáris általánosításaként is származtathatók, illetve úgy, mint az általánosított nemlineáris regressziós feladat megoldására alkalmas architektúrák.

Egy általános nemlineáris dinamikus rendszermodell által megvalósított be-kimeneti kapcsolat – diszkrét idejű rendszerek esetében – az alábbi általános formában adható meg:

$$y(k) = f(\Theta, \varphi(k)), \quad (8.1)$$

Az $f(\Theta, \varphi(k))$ kapcsolat a modell struktúráját rögzíti, ahol $\varphi(k)$ az ún. regresszorvektor, k az időindex, Θ pedig a rendszer paramétereit összefoglaló vektor. A regresszorvektor feladata megadni, hogy a kimenet előállításában a modellezendő rendszer, folyamat milyen régebbi bemeneti és kimeneti adatait használjuk fel.

A modell felállításával az a cél, hogy a modell $y(k)$ kimenete, valamely $d(k)$ kívánt válasz (a modellezendő rendszer válasza) minél pontosabb közelítését adja meg egy adott kritérium értelmében (ld. még a 2. fejezetben említett modell-illesztési alapfeladatot). A modell általában tökéletesen nem fogja előállítani a kívánt választ; a modell kimenete és a kívánt válasz közötti különbséget, a hibát egy $\varepsilon(k)$ zaj folyamattal vehetjük figyelembe.

$$d(k) = y(k) + \varepsilon(k) \quad (8.2)$$

A modellezésnél arra törekszünk, hogy olyan regresszorvektort ($\varphi(k)$), paramétervektort (Θ) és nemlineáris leképezést (f) találjunk, amely $\varepsilon(k)$ valamilyen jellemzőjét, rendszerint az átlagos négyzetes értéket minimalizálja.

A felmerülő kérdések részben megtalálhatók a lineáris modellezési feladatok körében is. A $\varphi(k)$ regresszort – tehát a modell struktúráját – lineáris esetben is meg kell választani; $\varphi(k)$ megválasztása azonban lineáris esetben a feladatot teljesen specifikálja, hiszen ezt követően már csak a paraméterek meghatározása marad hátra. Nemlineáris modellezési feladatnál ezen kívül még $f(\cdot)$ megválasztásáról is gondoskodni kell. Amennyiben a nemlineáris dinamikus modellezési feladat megoldására neurális hálózatot használunk, a feladatmegoldás tipikusan ún. fekete doboz (*black box*) modellezési feladat. Fekete doboz modellezésnél a modell struktúrájáról nincs a priori ismeretünk, a feladatot pusztán a rendelkezésre álló összetartozó be-kimeneti adatok (adatszekvenciák) alapján kell megoldanunk. A (8.2) összefüggéssel kapcsolatban meg kell jegyezzük, hogy lineáris rendszerek modellezésénél $\varepsilon(k)$ általában valóban a zajt jelenti, nemlineáris rendszereknél azonban $\varepsilon(k)$ -ba beleértjük az ún. modellezési hibát is, ami abból eredhet, hogy a nemlineáris modellünk struktúrája nem feltétlenül felel meg a modellezendő rendszer struktúrájának.

A rendelkezésre álló adatokhoz illeszkedő nemlineáris dinamikus modell általában nem egyértelmű, tehát – hasonlóan a statikus regressziós feladatokhoz – itt is rosszul definiált (*ill-posed*) problémákkal állunk szemben.

Ugyanahhoz az adatkészlethez különböző struktúrájú, más nemlineáris függvényt, más paramétereket alkalmazó megoldások is találhatók. Sok esetben célszerű, ha nem a legáltalánosabb modellstruktúrát alkalmazzuk. A nemlineáris függvény lehetséges megválasztását is érdemes bizonyos függvényosztályra korlátozni, mivel így jobban kezelhető, könnyebben megtalálható modellt nyerhetünk. A nemlineáris dinamikus rendszer-modellezési feladat tehát jóval nehezebb a lineáris megfelelőjénél. Az alábbiakban mielőtt néhány általános neurális dinamikus struktúrával foglalkoznánk, előbb a nemlineáris rendszermodellek egyfajta csoportosítását tekintjük át.

1. Regresszorválasztás, modellstruktúra választás

A fekete doboz modellstruktúra választása egyrészt a modellstruktúra-osztály megválasztását, másrészt az osztályon belüli tényleges struktúra kijelölését jelenti. A struktúra-osztály rögzítésekor azt határozzuk meg, hogy a regresszor milyen argumentumokból áll. Beszélhetünk olyan rendszerről, ahol a regresszor csak a régebbi bemeneti értékekből áll, de olyan rendszerről is, ahol a régebbi bemeneti értékek mellett figyelembe vehetjük a régebbi kimeneti értékeket is. A lineáris dinamikus rendszerek bemenet-kimenet reprezentáció (*input-output representation*) alapján történő csoportosításához [Lju99], [Sjö95] hasonlóan nemlineáris dinamikus rendszereknél az alábbi főbb modellstruktúra-osztályok megkülönböztetése célszerű:

- **NFIR modellek**, ahol a regresszor csak a régebbi bemeneteket tartalmazza:

$$\Phi(k) = [x(k-1), x(k-2), \dots, x(k-N)] \quad (8.3)$$

Az NFIR modellek a lineáris véges impulzusválaszú (FIR) szűrők nemlineáris megfelelőjeként olyan előreccsatolt modellek, ahol az emlékezetet a bemenetek régebbi értékeinek tárolása biztosítja.

- **NARX modellek**, ahol a regresszor mind a régebbi bemeneteket, mind a régebbi rendszerkimeneteket magában foglalja:

$$\Phi(k) = [x(k-1), x(k-2), \dots, x(k-N), d(k-1), d(k-2), \dots, d(k-M)] \quad (8.4)$$

A NARX modell a lineáris külső gerjesztéssel ellátott autoregresszív (*ARX, Auto-Regressive with exogenous inputs*) modell nemlineáris megfelelője. Valójában a NARX modell is előreccsatolt struktúra annak ellenére, hogy a leképezésben a régebbi kimeneti értékek is szerepelnek bemenetként. A kimeneti értékek azonban nem a modell régebbi kimenetei, hanem a kívánt válasz, a modellezendő rendszer kimenetének régebbi értékei. Ezért ezt a modell-struktúrát szokás soros-párhuzamos (*series-parallel*) struktúrának is nevezni [Nar89].

- **NOE modellek**. Itt a regresszor a régebbi bemenetek mellett a modell régebbi kimeneteit is tartalmazza:

$$\Phi(k) = [x(k-1), x(k-2), \dots, x(k-N), y(k-1), y(k-2), \dots, y(k-M)] \quad (8.5)$$

A NOE modell már igazi visszacsatolt rendszer, hiszen itt a kimenet pillanatnyi értékének meghatározásánál a modell-kimenet régebbi értékeit is felhasználjuk. A NOE elnevezés a lineáris OE (*output error*) struktúra elnevezéséből származik. A NOE struktúraosztályt a NARX modellnél használt soros-párhuzamos elnevezés mintájára párhuzamos (*parallel*) modell-osztálynak is szokás nevezni [Nar89].

- **NARMAX modellek**. A NARMAX modellek a NARX modellek olyan bővítései, ahol a regresszor a régebbi bemenetek és kívánt válaszok mellett a modellezési hiba régebbi értékeit is tartalmazza:

$$\Phi(k) = [x(k-1), \dots, x(k-N), d(k-1), \dots, d(k-M), e(k-1), \dots, e(k-L)] \quad (8.6)$$

A hiba régebbi értékeinek figyelembevétele miatt ez is visszacsatolt modell annak ellenére, hogy a modell régebbi kimeneti értékei bemenetként nem szerepelnek.

- **NBJ modellek**. A nemlineáris Box-Jenkins modelleknél a NOE modellhez hasonlóan a modell régebbi kimeneti értékeit csatoljuk vissza, továbbá kétfajta hiba régebbi értékeit is felhasználjuk:

$$\Phi(k) = [x(k-1), \dots, x(k-N), y(k-1), \dots, y(k-M), e(k-1), \dots, e(k-L), \varepsilon_x(k-1), \dots, \varepsilon_x(k-K)] \quad (8.7)$$

$\varepsilon_x(k-i) = d(k-i) - y_x(k-i)$ a $y_x(k-i)$ definiált $\varepsilon(k-i)$ értékeken kívül újabb hibaértékek is szerepelnek: $\varepsilon(k-i)$ -t egy olyan modell kimenetén nyerjük, ahol a (8.7) által megadott regresszorvektorban ε és ε_x helyére nullákat írunk.

A fenti modellosztályok mind bemenet-kimenet reprezentációt használnak, továbbá egy bemenetet és egy kimenetet tételeznek fel (*single-input-single-output*, *SISO* rendszerek). Kiterjesztésük több-bemenetű-több-kimenetű rendszerekre viszonylag természetes módon megtörténhet.

Rendszermodellezésnél a bemenet-kimenet reprezentációkon kívül gyakran alkalmazzuk az állapotváltozós leírást (*state-space representation*). Több-bemenetű-több-kimenetű rendszert feltételezve ekkor a nemlineáris dinamikus kapcsolatot az alábbi formában adjuk meg:

$$\begin{aligned} \mathbf{u}(k) &= \mathbf{f}(\mathbf{u}(k-1), \mathbf{x}(k-1), \mathbf{b}_1(k-1)) \\ \mathbf{y}(k) &= \mathbf{g}(\mathbf{u}(k), \mathbf{b}_2(k)) \end{aligned} \quad (8.8)$$

ahol $\mathbf{u}(k)$ az állapotvektor (melynek komponensei az állapotváltozók), $\mathbf{x}(k)$ a bemeneti vektor, $\mathbf{f}(\cdot)$ a következő állapotot meghatározó nemlineáris vektorfüggvény, $\mathbf{y}(k)$ a nemlineáris dinamikus rendszer kimeneti vektora, $\mathbf{g}(\cdot)$ a kimenetet meghatározó nemlineáris függvény, $\mathbf{b}_1(k)$ és $\mathbf{b}_2(k)$ pedig zajvektorok.

Sok esetben a fenti általánosabb állapotváltozós kapcsolat helyett az alábbi egyszerűbbet tételezzük fel:

$$\begin{aligned} \mathbf{u}(k) &= \mathbf{f}(\mathbf{u}(k-1), \mathbf{x}(k-1)) + \mathbf{b}_1(k-1) \\ \mathbf{y}(k) &= \mathbf{C}\mathbf{u}(k) + \mathbf{b}_2(k) \end{aligned} \quad (8.9)$$

ahol $\mathbf{C}$ a kimenetet meghatározó mátrix. Ez utóbbi esetben a kimenetek a belső állapotok lineáris függvényei¹.

A fenti modellosztályok eltérő szerepű és számú paramétert tartalmaznak. Az általános regressziós kapcsolatot megfogalmazó (8.1) összefüggésben Θ egy adott modellosztály összes paraméterét képviseli.

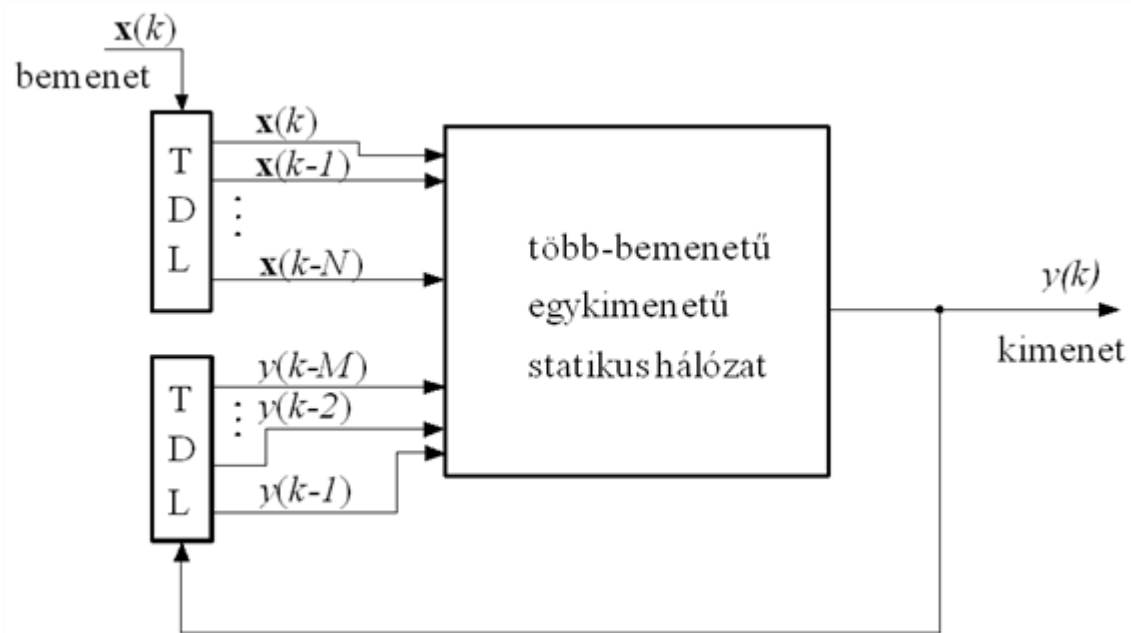
A bemutatott reprezentációs lehetőségek közötti választás csak a modellstruktúra-osztály megválasztását jelenti. Egy konkrét feladatnál a modellosztályon belül a tényleges modellstruktúra megválasztására, tehát a modellben szereplő paraméterek számának a meghatározására is szükség van. Ez input-output reprezentációnál a regresszorvektorban a Θ paramétervektor komponensei számának, tehát a régebbi bemeneti minták és megfelelő esetben a régebbi kimeneti értékek és hibaértékek számának meghatározását jelenti. Nemlineáris dinamikus rendszerek modellezésénél – de lineáris rendszerek modellezésénél is – talán a legnehezebb feladat a modellstruktúra-osztály és a modell fokszáma, vagyis a paraméterek számának a meghatározása. A modellstruktúra megválasztásának szempontjait és a modell-fokszám meghatározásnak néhány fontos módszerét a fejezet későbbi részében foglaljuk össze röviden.

2. Dinamikus neurális modellek

A neurális dinamikus modellek kialakításának több útja lehet. A legkézenfekvőbb megoldás, ha egy eredetileg statikus neuronhálót dinamikus elemekkel egészítünk ki. A dinamikus elemek megjelenhetnek a statikus háló bemeneteinél, de a háló "belsejében" is [Ner93], [Dre05]. A nemlineáris statikus és a lineáris dinamikus hálók ötvözeteként létrehozott nemlineáris dinamikus rendszerekre további példát jelentenek azok a kaszkád hálózatok, ahol lineáris dinamikus és nemlineáris statikus rendszerek szerepelnek valamilyen sorrendben a jelútban. Ilyen rendszerek pl. a Wiener, a Hammerstein, illetve a Wiener-Hammerstein modellek [Bal00]. E speciális felépítésű "hálók" ugyan korlátozott modellező képességgel rendelkeznek, de tárgyalásuk az általános esetről egyszerűbb lehet, és számos gyakorlati feladatnál a "valóság" megfelelő pontosságú modelljének bizonyulhatnak [Nar90], [Bou95], [Pin01].

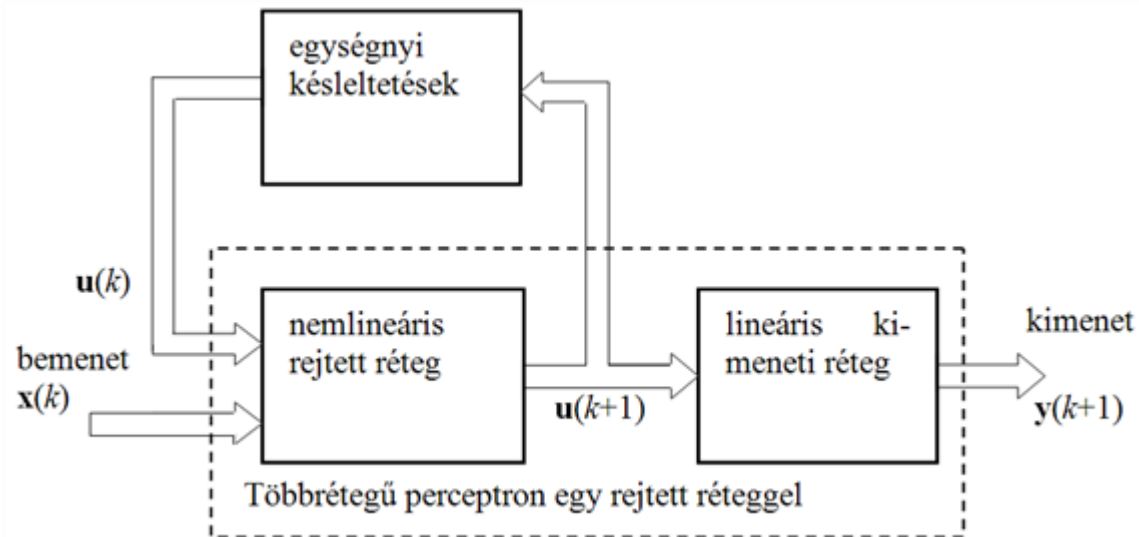
8.1. ábra - Általános időfüggő neurális hálózat többdimenziós bemenet és egydimenziós kimenet mellett

¹ Az és összefüggésben a rendszermodellezésnél általánosan használatos jelölésektől – ahol $\mathbf{x}(k)$ az állapotvektort jelöli – eltértünk. Ennek oka, hogy a könyv eddigi jelölésrendszerének megfelelően $\mathbf{x}$ -szel most is a bemenetet jelöljük.



Az időfüggés beépítésének triviális módja, ha a hálózat bemeneteinek számát megnöveljük és az egyes bemenetekre a tényleges be- és/vagy kimenőjelek késleltetett értékeit adjuk megfelelő késleltető láncokon (*tapped delay line*: TDL) keresztül (8.1 ábra). Az ábra valójában egy NOE modellstruktúrát mutat, de ebből a felépítésből egyszerűsítéssel vagy kis módosítással a többi bemenet-kimenet reprezentáción alapuló modellosztálynak megfelelő neurális architektúra is származtatható. Amennyiben csak az $\mathbf{x}$ bemenőjelnél alkalmazzuk a késleltető láncot és a háló nem tartalmaz visszacsatolást, a dinamikussá tett háló NFIR struktúrát valósít meg. Ha az ábra szerinti visszacsatolás is jelen van NOE, vagy párhuzamos modellt kapunk. A kimenetről történő visszacsatolás helyett a kívánt választ "csatolva vissza" NARX vagy soros-párhuzamos modellt nyerünk.

8.2. ábra - Állapotváltozós modell neurális implementációja

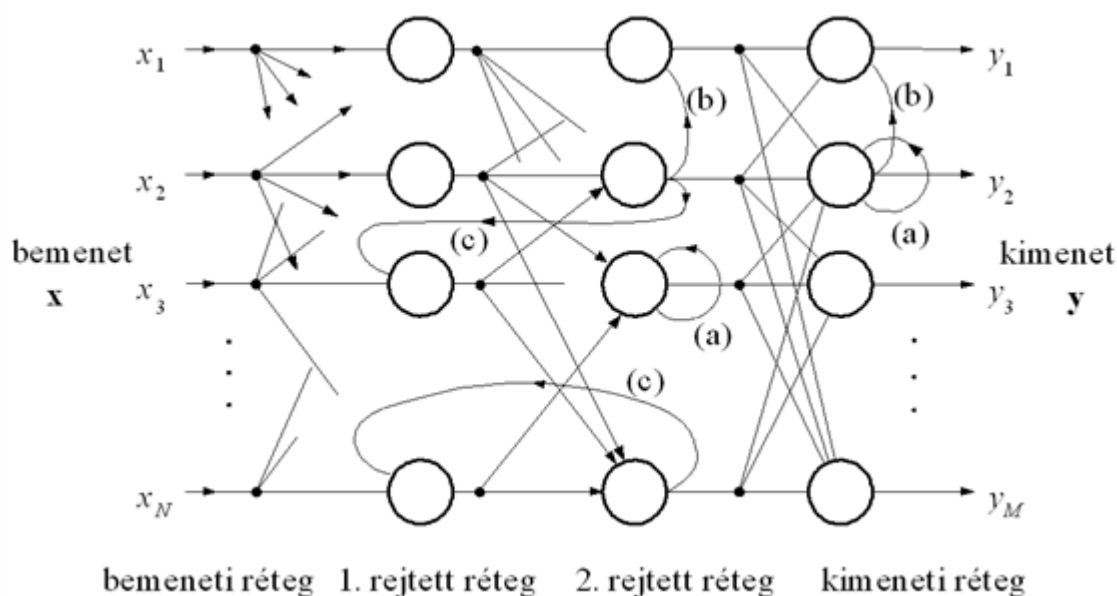


Látható, hogy minden esetben valójában továbbra is statikus hálózattal van dolgunk, az időfüggő viselkedést a hálózat megfelelően képzett bemenetei biztosítják. Az 8.1 ábrán bemutatott általános háló-felépítés az input-output reprezentációnak felel meg. Hasonló általános neurális struktúrát adhatunk meg az állapotváltozós leíráshoz. Ezt mutatja a 8.2 ábra.

Az előbbi ábrákon bemutatott megoldások gyakorlati alkalmazása a bemenetek számának esetleges nagymértékű növelése miatt nem mindig célszerű, bár az így kapott hálók általános időfüggő hálózatok

kialakítására alkalmasak. Más megoldásokkal nem a bemeneteken keresztül vezetünk be időfüggést, hanem a hálózat belső felépítését módosítjuk olyan módon, hogy az időfüggő viselkedést mutasson.

8.3. ábra - Példa lokális visszacsatolásokra



A belső felépítés módosítása jelentheti dinamikus, memóriával rendelkező neuronok alkalmazását (ld. 1.1. ábra), lineáris dinamikus rendszerek, lineáris szűrők beépítését a hálóba, illetve lokális visszacsatolások alkalmazását. A lokális visszacsatolás(oka)t tartalmazó hálókat számos változatát dolgozták ki. A 8.3 ábrán néhány visszacsatolási lehetőségre mutatunk példát.

A többrétegű perceptronnál a szokásos előrecsatoló összeköttetéseket csak vázlatosan jelöltük. Egy processzáló elem lehet önmagában visszacsatolt (a), de lehet az egy rétegen belüli processzáló elemek között oldalirányú kapcsolat (b), ill. kialakíthatunk visszacsatolást egy későbbi rétegről egy valamelyik előbbire is (c).

A visszacsatolt hálózatok modellező képessége a csak előrecsatolást tartalmazó dinamikus hálózatokénál nagyobb, de használatuknál számos nehézség merül fel. Egyrészt a visszacsatolás miatt stabilitási problémák léphetnek fel, másrészt a hálózatok tanítása lényegesen bonyolultabb lehet.

Az időfüggő hálózatokat, ha jel-szekvencia felismerésre, szekvencia reprodukálására és folytatására (jóslás, előrejelzés) vagy egy bemeneti szekvenciára kimeneti válasz-szekvencia generálására alkalmazzuk, szokás szekvenciális hálózatoknak is nevezni. (A visszacsatolt hálózatok egy másik csoportjánál a bemenetre kerülő minta hatására egy tranzienst indul el, miközben a hálózat bemenete nem változik, és a tranzienst lezajlása utáni állandósult állapothoz (egyensúlyi állapot) tartozó kimenetet tekintjük a hálózat válaszának. Ilyen pl. a Hopfield hálózat, mellyel nem itt, hanem a 11. fejezetben foglalkozunk.)

Az időfüggő hálózatok és a statikus hálózatok közötti legjelentősebb különbség a hálózat válaszáinak minősítése és a hálózat tanítása szempontjából, hogy míg a statikus hálónál egy bemeneti mintára adott kimeneti válasz önmagában értékelhető, addig az időfüggő hálózatok értékelése bemeneti szekvenciákra adott kimeneti szekvenciák alapján történhet csak. A hálózat viselkedése ezért általában egy adott időpillanathoz tartozó kimeneti hiba alapján nem minősíthető, hanem a kimeneti hiba valamely időintervallumra vett átlagának vagy összegének meghatározására van szükség, és azt kell vizsgálnunk, hogy ezen eredő hiba hogyan csökkenthető a hálózat paramétereinek megválasztásával.

Ha az eddigiekhez hasonlóan a kimeneti pillanatnyi hibát $e(k)$ -val jelöljük, akkor az eredő négyzetes hiba:

$$e^2 = \sum_{k=1}^K e^2(k) = \sum_{k=1}^K |z(k) - y(k)|^2, \quad (8.10)$$

ahol k most is a diszkrét időindexet, K pedig a háló működésének szempontjából fontos időablak méretét jelöli. A hálózat gradiens alapú eljárással történő tanításához tehát az eredő hibának a megfelelő súly vagy súlyvektor

szerinti deriváltját $\left(\frac{\partial \varepsilon^2}{\partial w_y}\right)$ -t kell meghatározni. Az egyes hálózatoknál alkalmazott tanítóeljárások, a hálózat struktúrájából adódó eltéréseken kívül, abban is különböznek, hogy e deriváltak meghatározása hogyan történik, illetve, hogy a tanító eljárások a derivált pontos vagy közelítő meghatározásán alapulnak-e.

Az eddigiekből látható, hogy dinamikus feladatra igen sokféle neuronháló architektúra alkalmazható. Bár e háló többé-kevésbé besorolhatók valamely általános modellosztályba, a különböző háló analízise, tanítási összefüggéseinek meghatározása mind-mind egyedi problémákat is felvet. A sokféle dinamikus neurális architektúra áttekintésére nincs módunk, a részletes bemutatástól pedig mindenképpen el kell tekintenünk. A továbbiakban ezért néhány, a gyakorlati alkalmazás szempontjából fontos háló-architektúrát választunk ki és csak ezek felépítését, tanítási összefüggéseit tárgyaljuk. Az érdeklődők a témakört részletesebben az igen bő irodalomból ismerhetik meg (ld. pl. [Gup03], [Dre05]).

3. Előrecsatolt időfüggő hálózatok

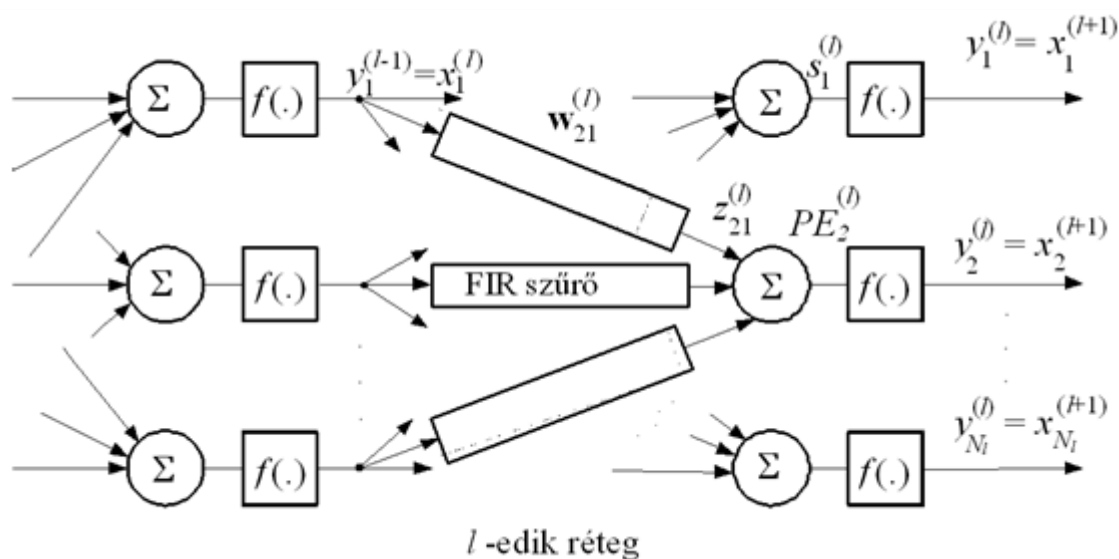
Az előrecsatolt időfüggő hálózatok jellemzője, hogy vagy a hálózat bemenetén vagy az egyes processzáló elemeket összekötő súlyok helyén súlyozott megcsapolásokkal rendelkező késleltető lánc szerepel. Ez valójában azt jelenti, hogy vagy a bemeneten vagy az egyes súlyok helyett FIR szűrőket építünk a hálózatba. Ebből adódik e hálózatok elnevezése is: FIR-MLP, FIR-CMAC, stb. A hálózatok nem tartalmaznak visszacsatolást, a konkrét elnevezések is utalnak arra, hogy ezen hálózatok az NFIR modell-osztályba tartoznak.

A következőkben előbb a hibavisszaterjesztéssel tanítható többrétegű perceptron kiterjesztésével kapott FIR-MLP hálót, majd a kiterjesztett bázisfüggvényes hálózatokat (FIR-RBF, FIR-CMAC) mutatjuk be.

3.1. A FIR-MLP háló

A FIR-MLP hálózat felépítését a 8.4 ábra mutatja. Látható, hogy ez a dinamikus struktúra egy olyan MLP hálózat, ahol minden súly helyett egy lineáris FIR szűrő szerepel. Ez alól csak az egyes neuronok eltolás (*bias*) bemenetei a kivételek, amelyeknél meghagyjuk a statikus kapcsolatot biztosító súlyértéket. Az ábrán azonban ezek az eltolás bemenetek a jobb áttekinthetőség érdekében nem szerepelnek. A struktúra előnye a bemeneti késleltető lánc alkalmazásával szemben, hogy miközben itt a bemenetek számát nem növeltük meg, és ezáltal várhatóan nincs szükség a rejtett neuronok számának megnövelésére sem, az előrecsatolt dinamikus működést biztosítjuk. A szabad paraméterek száma természetesen egy hasonló méretű (hasonló neuronszámú) statikus MLP-hez képest megsokszorozódik.

8.4. ábra - A FIR-MLP háló felépítése



3.1.1. A FIR-MLP háló tanítása

A hálózat tanításához először a kimeneti hiba (8.10) szerinti összefüggését kell felírni. Az eredő hiba a háló összes kimenetén értelmezhető, adott időpillanathoz tartozó négyzetes hibakomponensek mind a kimenetek, mind a K időablak szerinti összege (N_l a kimeneti réteg neuronjainak száma). Jelöljük ezt a hibát ε^2 -tel.

$$\varepsilon^2 = \sum_{k=1}^K \sum_{i=1}^{n_k} \varepsilon_i^2(k), \quad (8.11)$$

Egy adott súly módosításához ennek az eredő négyzetes hibának az adott súly szerinti deriváltját kell meghatározni. Célszerű azonban, ha közvetlenül egy összeköttetés-ágban szereplő FIR szűrő összes együttthatóját összefogó súlyvektor tanító összefüggését írjuk fel.

Vezessük be ehhez a következő jelölést. Legyen

$$\mathbf{w}_y^{(l)} = \left[w_{y,0}^{(l)}, w_{y,1}^{(l)}, \dots, w_{y,M_y^{(l)}}^{(l)} \right]^T \quad (8.12)$$

az l -edik réteg i -edik és az előző réteg j -edik processzáló eleme közötti szűrő együttthatóinak vektora. A vektor felírásából látható, hogy a szűrőágban $M_y^{(l)}$ egységnyi késleltetés és $M_y^{(l)} + 1$ szűrőegyütttható van. Bár elvileg nincs akadálya, hogy az egyes szűrők eltérő számú késleltetést tartalmazzanak, ennek gyakorlati haszna nincs, ezért most is feltételezzük, hogy egy adott réteg összes neuronjának minden bemeneti szűrője azonos számú késleltetéssel és ennek megfelelően azonos számú szűrő együttthatóval rendelkezik. Ezért $M_y^{(l)} = M^{(l)}$ minden $i=1, 2, \dots, N_{l-1}$ és minden $j=1, 2, \dots, N_l$ -re (N_{l-1} és N_l az $(l-1)$ -edik, illetve az l -edik réteg neurojainak a száma).

A fentiek figyelembevételével a súlymódosító összefüggéshez a $\frac{\partial \varepsilon^2}{\partial \mathbf{w}_y^{(l)}}$ gradiens meghatározása szükséges. A

háló tanítási összefüggéseinek meghatározása többféle megközelítésben lehetséges. Egy lehetőség, ha a $\frac{\partial \varepsilon^2}{\partial \mathbf{w}_y^{(l)}}$

gradienst az egyes időpillanatokhoz tartozó $\frac{\partial \varepsilon^2(k)}{\partial \mathbf{w}_y^{(l)}}$ gradiensek segítségével írjuk fel:

$$\frac{\partial \varepsilon^2}{\partial \mathbf{w}_y^{(l)}} = \sum_k \frac{\partial \varepsilon^2(k)}{\partial \mathbf{w}_y^{(l)}}. \quad (8.13)$$

Itt tehát $\varepsilon^2 = \sum_k \varepsilon^2(k)$, vagyis az eredő négyzetes hiba az egyes időpillanatokbeli négyzetes hibaértékek megfelelő időablakra vett összegét jelenti. E megközelítés szerint végezve a vizsgálatot valójában időben kiterítjük a hálózatot és így egy nagyobb, de statikus hálót kapunk, ami már a hagyományos hibavisszaterjesztés algoritmusával tanítható. A kiterítés következtében kapott virtuális háló azonban igen nagyméretű is lehet a FIR szűrőkben található késleltető láncok méretétől függően, továbbá a virtuális háló struktúrája el fog térni a szokásos többrétegű perceptron felépítésétől: az egyes rétegek processzáló elemei nincsenek a következő réteg összes processzáló elemével összekötve. További eltérés, hogy a virtuális háló az időbeli kiterítés következtében jóval több súlyt tartalmaz, mint a fizikailag létező valóságos; egy valóságos súlynak a virtuális hálóban több megfelelője van, melyek a háló működésének különböző időpillanataihoz tartoznak és a kiterített hálóban más-más rétegben szerepelnek. A hagyományos hibavisszaterjesztés algoritmus alkalmazásánál az azonos fizikai súly különböző virtuális megfelelőihez eltérő súlymódosító összefüggéseket kapunk, mivel a hibavisszaterjesztés útja a különböző virtuális megfelelőikig eltérő. Ennek ellenére csak egy fizikai súly létezik, tehát ezt különbözőképpen nem módosíthatjuk. A megoldás csak az lehet, hogy a különböző módon visszaterjesztett "hiba" figyelembevételével nyert súlymódosítások eredőjével kell módosítanunk a fizikai súly értékét, ami azt jelenti, hogy a megfelelő virtuális súlyok azonos mértékben és egyszerre módosulnak.

E nehézségek miatt, különösen nagyobb rendszereknél az időbeli kiterítés alapján történő FIR-MLP tanítás – bár elvileg lehetséges – nem célszerű. A következőkben egy más megközelítést mutatunk be.

3.1.2. Temporális hibavisszaterjesztéses eljárás

A temporális hibavisszaterjesztéses (*temporal back-propagation*) eljárás a FIR-MLP háló tanítására egy hatékonyabb lehetőség [Wan90]. A kimeneti eredő hiba gradiense ugyanis a (8.13)-tól eltérően másféle

felbontásban is felírható (a temporális back-propagation eljárás származtatásánál a 8.4 ábra jelöléseit illetve az MLP tanításánál, a standard hibavisszaterjesztéses eljárásnál alkalmazott jelöléseket használjuk):

$$\frac{\partial \varepsilon^2}{\partial \mathbf{w}_y^{(l)}} = \sum_k \frac{\partial \varepsilon^2}{\partial s_i^{(l)}(k)} \frac{\partial s_i^{(l)}(k)}{\partial \mathbf{w}_y^{(l)}} \quad (8.14)$$

Itt tehát nem a pillanatnyi négyzetes hiba adott súlyvektor szerinti gradiensét határozzuk meg először, amiből a megfelelő időablakra vonatkozó összegzéssel határozható meg az eredő gradiens, hanem az eredő hibának a megfelelő kimenet adott időpillanatbeli értéke szerinti parciális deriváltján keresztül kapjuk meg az eredő gradienst. Vegyük észre, hogy

$$\frac{\partial \varepsilon^2(k)}{\partial \mathbf{w}_y^{(l)}} \neq \frac{\partial \varepsilon^2}{\partial s_i^{(l)}(k)} \frac{\partial s_i^{(l)}(k)}{\partial \mathbf{w}_y^{(l)}} \quad (8.15)$$

vagyis bár (8.13) és (8.14) egyenlők, adott k időindexhez tartozó komponenseikre az egyenlőség már nem áll fenn. Megmutatható, hogy ez utóbbi megközelítést választva a FIR-többrétegű perceptron tanítása a standard BP eljáráshoz hasonlóan történhet, vagyis a hibavisszaterjesztés itt is a jel előreterjesztését végző hálózathoz hasonló hálózaton, most tehát FIR szűrőkön keresztül történik.

A (8.14) szerinti felbontás alapján a következő súlymódosító összefüggést nyerjük:

$$\mathbf{w}_y^{(l)}(k+1) = \mathbf{w}_y^{(l)}(k) - \mu \frac{\partial \varepsilon^2}{\partial s_i^{(l)}(k)} \frac{\partial s_i^{(l)}(k)}{\partial \mathbf{w}_y^{(l)}} \quad (8.16)$$

A (8.16)-nak megfelelő súlymódosítás természetesen nem pontosan az igazi gradiens szerint halad a megoldás térben, hiszen egy módosító lépésben az eredő teljes hiba gradiense helyett egy gradiens-komponenssel dolgozunk. Nem túl nagyra választva azonban μ értékét, a súlymódosítás közel a valódi gradiensnek megfelelően történik. Megjegyezzük azonban, hogy a gradiens-komponens nem csak egy időpillanatbeli hiba alapján számítható, hiszen a súlymódosításhoz ε^2 megfelelő deriváltját használjuk fel.

Mivel a standard hibavisszaterjesztéses algoritmushoz hasonlóan $\frac{\partial s_i^{(l)}(k)}{\partial \mathbf{w}_y^{(l)}} = \mathbf{x}_j^{(l)}(k)$, továbbá, ha most is használjuk a $\delta_i^{(l)}(k) = -\frac{1}{2} \frac{\partial \varepsilon^2}{\partial s_i^{(l)}(k)}$ jelölést, akkor egy adott súly helyén található szűrőegyütthatókból álló vektor módosítására az alábbi, a statikus MLP tanításával formailag azonos összefüggést kapjuk:

$$\mathbf{w}_y^{(l)}(k+1) = \mathbf{w}_y^{(l)}(k) + 2\mu \delta_i^{(l)}(k) \mathbf{x}_j^{(l)}(k) \quad (8.17)$$

A visszaterjesztett, általánosított hiba (valójában a kimeneti négyzetes hibának egy adott neuron lineáris kimenete szerinti érzékenysége) meghatározása attól függ, hogy az adott neuron melyik rétegben található.

A kimeneti réteg esetén $\delta_i^{(l)}(k) = \delta_i^{(L)}(k)$ egyszerűen meghatározható, hiszen:

$$\begin{aligned} \delta_i^{(L)}(k) &= -\frac{1}{2} \frac{\partial \varepsilon^2}{\partial s_i^{(L)}(k)} \\ &= -\frac{1}{2} \frac{\partial \varepsilon_i^2}{\partial s_i^{(L)}(k)} \\ &= -\varepsilon_i \frac{\partial \varepsilon_i}{\partial y_i(k)} \frac{\partial y_i(k)}{\partial s_i^{(L)}(k)} \\ &= \varepsilon_i f'(s_i^{(L)}(k)) \end{aligned} \quad (8.18)$$

Így a kimeneti réteg neuronjainál a súlyok módosítására formálisan a standard back-propagation eljárás kimeneti rétegre vonatkozó összefüggését kapjuk:

$$\mathbf{w}_y^{(L)}(k+1) = \mathbf{w}_y^{(L)}(k) + 2\mu \varepsilon_i f'(s_i^{(L)}(k)) \mathbf{x}_j^{(L)}(k) \quad (8.19)$$

Az egyetlen különbség, hogy itt $\mathbf{w}_y^{(l)}$ nem a kimeneti réteg összes sűrűségi vektor, hanem a megfelelő szűrőág együtthatóiból álló, (8.12) által definiált vektor. Ugyancsak nem az L -edik réteg teljes bemeneti vektora, hanem minden kimeneti neuron j -edik szűrőágában a késleltetett bemenetekből képezett vektor:

$$\mathbf{x}_j^{(l)}(k) = \left[x_j^{(l)}(k), x_j^{(l)}(k-1), \dots, x_j^{(l)}(k-M^{(l)}) \right]^T \quad (8.20)$$

Ez a vektor természetesen megegyezik az előző réteg j -edik neuronjának késleltetett kimeneteiből képezett vektorral.

Az adott szűrő tehát $M^{(l)}$ késleltető tagot tartalmaz, és a szűrő kimenetét a

$$z_y^{(l)}(k) = \mathbf{w}_y^{(l)T} \mathbf{x}_j^{(l)}(k) = \sum_{r=0}^{M^{(l)}} w_{y,r}^{(l)} x_j^{(l)}(k-r) \quad (8.21)$$

kapcsolat adja meg. A rejtett rétegeknél $\delta_i^{(l)}(k)$ meghatározásában a statikus hálónál származtatott hibavisszaterjesztési algoritmushoz hasonlóan szerepet játszik az összes olyan neuron, mely összeköttetésben van az l -edik réteg i -edik neuronjának kimenetével, vagyis az $(l+1)$ -edik réteg összes neuronja. A FIR szűrőben lévő késleltető lánc miatt azonban itt az időt is figyelembe kell venni. Ugyanis $s_i^{(l)}(k)$ a következő rétegbeli neuronok kimeneteinek nemcsak a k -adik időpillanatbeli értékét, hanem több, a k -adikat követő időpillanatbeli értékét is befolyásolja.

Mivel $\delta_i^{(l)}(k)$ a teljes négyzetes hiba $s_i^{(l)}(k)$ szerinti deriváltja, ezt az időfüggést nem hagyhatjuk figyelmen kívül. $\delta_i^{(l)}(k)$ meghatározása most is a láncszabály alkalmazásával lehetséges:

$$\delta_i^{(l)}(k) = -\frac{1}{2} \frac{\partial \varepsilon^2}{\partial s_i^{(l)}(k)} = -\frac{1}{2} \sum_m \sum_t \frac{\partial \varepsilon^2}{\partial s_m^{(l+1)}(t)} \frac{\partial s_m^{(l+1)}(t)}{\partial s_i^{(l)}(k)} \quad (8.22)$$

Itt t szintén diszkrét időindex, és azokat az időpillanatokat jelöli, amelyekben az $(l+1)$ -edik réteg kimenetei függnek az l -edik réteg i -edik neuronjának lineáris kimenetétől, $s_i^{(l)}(k)$ -től. Mivel teljesen összekötött hálót tételezünk föl, egy adott réteg minden kimenete kapcsolódik a következő réteg összes neuronjához. Ebből adódóan m végigfut az $1, \dots, N_{(l+1)}$ értékeken.

A (8.22) kifejezés első tényezőjét nézve azonnal látható, hogy $\frac{\partial \varepsilon^2}{\partial s_m^{(l+1)}(t)} = -2\delta_m^{(l+1)}(t)$. A második tényezőt tovább írva:

$$\frac{\partial s_m^{(l+1)}(t)}{\partial s_i^{(l)}(k)} = \frac{\partial s_m^{(l+1)}(t)}{\partial y_i^{(l)}(k)} \frac{\partial y_i^{(l)}(k)}{\partial s_i^{(l)}(k)} = \frac{\partial s_m^{(l+1)}(t)}{\partial y_i^{(l)}(k)} f' \left(s_i^{(l)}(k) \right) \quad (8.23)$$

Az egyenlet tovább alakításához figyelembe kell vennünk $s_m^{(l+1)}(t)$ keletkezését. Ez az $(l+1)$ -edik réteg m -edik neuronjának lineáris kimenete, amit nyilvánvalóan a neuron bemenetére csatlakozó szűrők kimeneteinek összegeként kapunk:

$$s_m^{(l+1)}(t) = \sum_{j=1}^{N_l} z_{mj}^{(l+1)}(t) \quad (8.24)$$

ahol $z_{mj}^{(l+1)}(t)$ az $(l+1)$ -edik réteg m -edik neuronja j -edik bemeneti szűrőágának kimenete:

$$z_{mj}^{(l+1)}(t) = \mathbf{w}_{mj}^{(l+1)T} \mathbf{x}_j^{(l+1)}(t) \quad (8.25)$$

Itt $\mathbf{x}_j^{(l+1)}(t)$ a j -edik szűrőág bemenetére kerülő késleltetett értékekből álló vektor:

$$\mathbf{x}_j^{(l+1)}(t) = \left[x_j^{(l+1)}(t), x_j^{(l+1)}(t-1), \dots, x_j^{(l+1)}(t-M^{(l+1)}) \right]^T \quad (8.26)$$

Egy neuron bemenetén annyi szűrőápré, ahány neuron van az előző rétegben. A (8.23) összefüggésben az i -edik szűrőág szerepel, ugyanis a i -edik neuron (az $(l+1)$ -edik réteg m -edik neuronja) bemeneti szűrői közül csak az i -edikre jut. Ennek kimenete:

$$z_m^{(l+1)}(t) = \sum_{p=0}^{d^{(l+1)}_{i,t}} w_{m,i,p}^{(l+1)} y_i^{(l)}(t-p), \quad (8.27)$$

melynek felhasználásával:

$$\frac{\partial z_m^{(l+1)}(t)}{\partial y_i^{(l)}(k)} = \frac{\partial z_m^{(l+1)}(t)}{\partial y_i^{(l)}(k)} = w_{m,i,p}^{(l+1)}, \quad (8.28)$$

ha $k=t-p$. Figyelembe véve, hogy az $(l+1)$ -edik réteg minden neuronjának szűrőiben $M^{(l+1)}$ egységnyi késleltetés van, vagyis a szűrőegyütthatók száma $M^{(l+1)}+1$, (8.28)-ra a következőt kapjuk:

$$\frac{\partial z_m^{(l+1)}(t)}{\partial y_i^{(l)}(k)} = \begin{cases} w_{m,i,t-k}^{(l+1)} & \text{ha } 0 \leq t-k \leq M^{(l+1)} \\ 0 & \text{egyébként} \end{cases}. \quad (8.29)$$

Ennek figyelembevételével a rejtett rétegre vonatkozó $\delta_i^{(l)}(k)$ -ra

$$\delta_i^{(l)}(k) = f' \left(z_i^{(l)}(k) \right) \sum_m \sum_{t=k}^{d^{(l+1)}_{i,t}} \delta_m^{(l+1)}(t) w_{m,i,t-k}^{(l+1)} \quad (8.30)$$

adódik. Alkalmazva a $p=t-k$ jelölést

$$\begin{aligned} \delta_i^{(l)}(k) &= f' \left(z_i^{(l)}(k) \right) \sum_m \sum_{p=0}^{d^{(l+1)}_{i,t}} w_{m,i,n}^{(l+1)} \delta_m^{(l+1)}(k+p) \\ &= f' \left(z_i^{(l)}(k) \right) \sum_m w_{m,i}^{(l+1)} \delta_m^{(l+1)}(k) \end{aligned} \quad (8.31)$$

ahol

$$\delta_m^{(l+1)}(k) = \left[\delta_m^{(l+1)}(k), \delta_m^{(l+1)}(k+1), \dots, \delta_m^{(l+1)}(k+M^{(l+1)}) \right]. \quad (8.32)$$

Összefoglalva:

$$\delta_i^{(l)}(k) = \begin{cases} \delta_i^{(l)}(k) f' \left(z_i^{(l)}(k) \right), & \text{ha } l = L, \text{ vagyis a kimeneti rétegnél} \\ f' \left(z_i^{(l)}(k) \right) \sum_m w_{m,i}^{(l+1)} \delta_m^{(l+1)}(k) & \text{a rejtett rétegeknél} \end{cases} \quad (8.33)$$

A (8.31) és a (8.32) összefüggésekből viszont látható, hogy $\delta_i^{(l)}(k)$ meghatározásához a rejtett rétegnél olyan szűrő szükséges, amelynek együtthatói a megfelelő előrevezető ág szűrőegyütthatói, a szűrt értékek pedig jövőbeli $\delta_m^{(l+1)}(k+n)$ értékek ($n=1, 2, \dots, M^{(l+1)}$). A súlymódosítást meghatározó összefüggés tehát nem kauzális.

A számítás kauzáliássá tehető, ha a hálózat megfelelő pontjain alkalmas késleltető fokozatokat iktatunk be. A késleltetések közbeiktatása azt eredményezi, hogy a súlyvektorok módosításához csak az aktuális és a korábbi hibák ismerete szükséges.

A kimeneti réteg súlyvektorainak módosítása a fenti összefüggések szerint történhet, hiszen ott eddig sem merült föl nem kauzális számítási igény. A rejtett rétegek számítása pedig az alábbiak szerint lehetséges: $\delta_i^{(l)}(k)$ helyett a $\delta_i^{(l)}(k-M^{(l+1)})$ visszaterjesztett "hibát" határozzuk meg, tehát a k indexet $(k-M^{(l+1)})$ -re cseréljük és ezzel a visszaterjesztett hibával határozzuk meg a rejtett réteg súlyvektorainak módosítását. Azonban a (8.17) összefüggés szerinti súlymódosításhoz $\mathbf{x}_j^{(l)}(k)$ helyett is $\mathbf{x}_j^{(l)}(k-M^{(l+1)})$ szükséges, vagyis a megfelelő bemeneti vektorokat tárolni kell.

Összefoglalva: a kauzális számítás a következő összefüggések alapján végezhető:

– a kimeneti réteg i -edik neuronjához vezető súlyvektoroknál:

$$\mathbf{w}_y^{(i)}(k+1) = \mathbf{w}_y^{(i)}(k) + 2\mu\delta_i^{(i)}(k)\mathbf{x}_y^{(i)}(k), \quad (8.34)$$

ahol

$$\delta_i^{(i)}(k) = \varepsilon_i(k) f' \left(s_i^{(i)}(k) \right), \quad (8.35)$$

– az l -edik (rejtett) réteg i -edik neuronjánál:

$$\mathbf{w}_y^{(i)}(k+1) = \mathbf{w}_y^{(i)}(k) + 2\mu\delta_i^{(i)}(k - M^{(i+1)})\mathbf{x}_y^{(i)}(k - M^{(i+1)}), \quad (8.36)$$

ahol

$$\delta_i^{(i)}(k - M^{(i+1)}) = f' \left(s_i^{(i)}(k - M^{(i+1)}) \right) \sum_m \delta_m^{(i+1)}(k - M^{(i+1)}) \mathbf{w}_{mi}^{(i)}, \quad (8.37)$$

és

$$\delta_m^{(i+1)}(k - M^{(i+1)}) = \left[\delta_m^{(i+1)}(k - M^{(i+1)}), \delta_m^{(i+1)}(k - M^{(i+1)} + 1), \dots, \delta_m^{(i+1)}(k) \right], \quad (8.38)$$

Ha a hálóban több rejtett réteg található, akkor a fenti összefüggések a kimenethez legközelebbi rejtett rétegre vonatkoznak, a kimenettől egyre távolabbi rejtett rétegeknél az indexeltolás egyre nagyobb mértékű, a FIR szűrő késleltető-láncok hosszának összege kell legyen.

A fenti összefüggések azt mutatják, hogy a FIR-MLP hálózat temporális hibavisszaterjesztő tanításánál a hibavisszaterjesztő hálózat felépítése az előreterjesztő hálózatstruktúrához hasonló: a visszaterjesztett általánosított hiba a megfelelő előreterjesztő ágakban lévő szűrőkön keresztül jut az egyre előbbi rétegekhez. A visszaterjesztő hálózatnál tehát itt is megtaláljuk a klasszikus statikus MLP hálózatnál származtatott BP eljárás reciprok voltát, a hálózat eredeti struktúrája is megmaradt szemben az időbeli kiterítés módszerének alkalmazásával.

A FIR-MLP háló képességei igen kedvezőnek bizonyultak idősor modellezési feladatoknál, továbbá nemlineáris dinamikus rendszerek modellezésénél, identifikációjánál [Wan94].

Megjegyezzük, hogy a FIR-MLP struktúrához némileg hasonló az ún. késleltetési neuronháló (*Time Delay Neural Network*, TDNN), melyet elsődlegesen beszédjelek osztályozására dolgoztak ki [Lan88], [Wai89]. A TDNN nem szűrőket alkalmaz az MLP súlyai helyén, hanem az egyes neuronokat többszörözi meg úgy, hogy az egyes ismétlések eltérő időpillanatbeli értékekkel dolgoznak. A rétegek között nem teljes az összeköttetési rendszer. Egy későbbi (pl. rejtett) réteg adott időpillanathoz tartozó minden neuronja az előbbi (pl. bemeneti) réteg neuronjainak (a bemeneteknek) egy keskeny időszávhhoz tartozó neuron-másolataival (értékeivel) van összeköttetésben mégpedig úgy, hogy az eltérő időpillanatokhoz eltérő időszávok tartoznak, miközben az eltérő időpillanatokhoz tartozó neuron-másolatok azonos súlyokkal rendelkeznek.

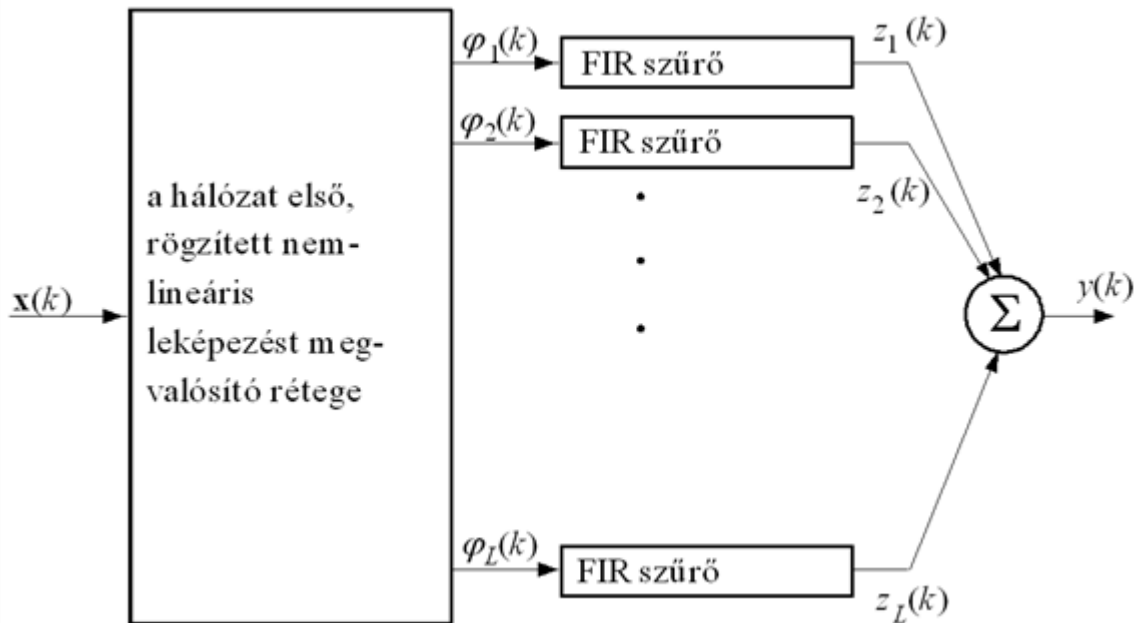
3.2. Egyéb NFIR hálóstruktúrák

A FIR-MLP háló alkalmazásánál nehézséget jelent, hogy e hálózatok tanítása az amúgy is lassan tanuló BP hálózatok tanításához képest is lassú. Ezért itt különös jelentősége van olyan hálók alkalmazásának, ahol a tanítható rétegek száma kicsi, lehetőleg egy. Ilyen lehetőséget jelentenek, ha a bázisfüggvényes hálókat, pl. az RBF-et vagy a CMAC-t terjesztjük ki. Így kaphatjuk a FIR-RBF, ill. a FIR-CMAC hálót [Dun93a], [Dun93b], melyek felépítése a ábrán látható. Mind az RBF, mind a CMAC hálózat fő jellemzője, hogy csak egy tanítható réteget tartalmaznak, és ez a tanítható réteg egyben a kimeneti réteg is. Ha tehát e réteg súlyait helyettesítjük FIR szűrőkkel, akkor ennek tanítása során egyrészt nem merülnek fel kauzalitási problémák, másrészt nincs szükség több rétegen keresztül hibavisszaterjesztésre, ami a tanítás gyorsaságát biztosítja. E hálók kimeneti rétegének FIR együtthatóit (súlyvektorait) egyébként a FIR-MLP háló kimeneti rétegének tanításával megegyező módon taníthatjuk ((8.19) összefüggés) azzal a megjegyzéssel, hogy mivel e hálók kimeneti rétege

lineáris, még a szigmoid deriváltra sincs szükség. Tehát az (8.17) általános súlymódosító összefüggésben $\delta_i(k) = \varepsilon_i(k)$.

Dinamikus működésre képes kiterjesztést nemcsak a klasszikus neuronhálónál alkalmazhatunk, hanem a kernel gépeknél is. Mind az SVM, mind az LS-SVM és a ridge regresszió is felhasználhatók dinamikusan feladatok megoldására. A kiterjesztés természetes módja, ha a 8.5 ábrának megfelelő architektúrában a statikus háló egy megfelelő kernel gép [Muk97], [Mül97], [Ral04]. Ebben az esetben a dinamikusan történő alkalmazás annyit jelent, hogy az aktuálisan használt regresszornak megfelelően kell a kernel gép bemeneti vektorait előállítani. Megjegyezzük, hogy míg egy klasszikus hálónál (MLP, CMAC, stb.) a bemeneti dimenzió késleltetések miatti növelése jelentősen nehezítheti a dinamikusan háló konstrukcióját, addig a kernel gépeknél a kernel trükknek köszönhetően ez a hátrányos hatás nem jelentkezik, hiszen a kernel mátrix méretét nem a bemenetek dimenziója, hanem a mintapontok száma határozza meg.

8.5. ábra - A FIR-CMAC háló



4. Visszacsatolt (rekurzív) hálózatok

Az előzőekben láttuk, hogy a dinamikusan háló az előreecsatolt struktúrákon kívül különféle visszacsatolt struktúrákkal is létrehozhatók. Sőt a visszacsatolt struktúra-osztályon belül tulajdonképpen lényegesen több lehetőségünk van konkrét visszacsatolt háló-architektúrák konstrukciójára.

A visszacsatolt architektúrák vizsgálatának, – a stabilitási kérdések vizsgálatán túl – az egyik legnehezebb, a háló tanításával is összefüggő kérdése, hogy miként oldjuk meg az időkezelést. A következőkben a rekurzív hálózatok két különböző tárgyalási megközelítésével foglalkozunk, ahol az időkezelés alapvetően eltérő módon történik. Az egyik megközelítés – hasonlóan az előreecsatolt időfüggő hálózatoknál alkalmazható eljárashoz – a hálózat működésének időbeli kiterítésén alapul (*back-propagation through time, BPTT*) [Bea94], ahol a kiterített hálózat a standard statikus hibavisszaterjesztési eljárással tanítható; a másik megközelítés esetén a tanítás a pillanatnyi gradiens felhasználásával lehetséges. Minthogy ez esetben minden lépésben végzünk tanítást, az eljárást valós idejű rekurzív tanításnak (*real-time recurrent learning, RTRL*) is nevezik.

4.1. Rekurzív háló időbeli kiterítése

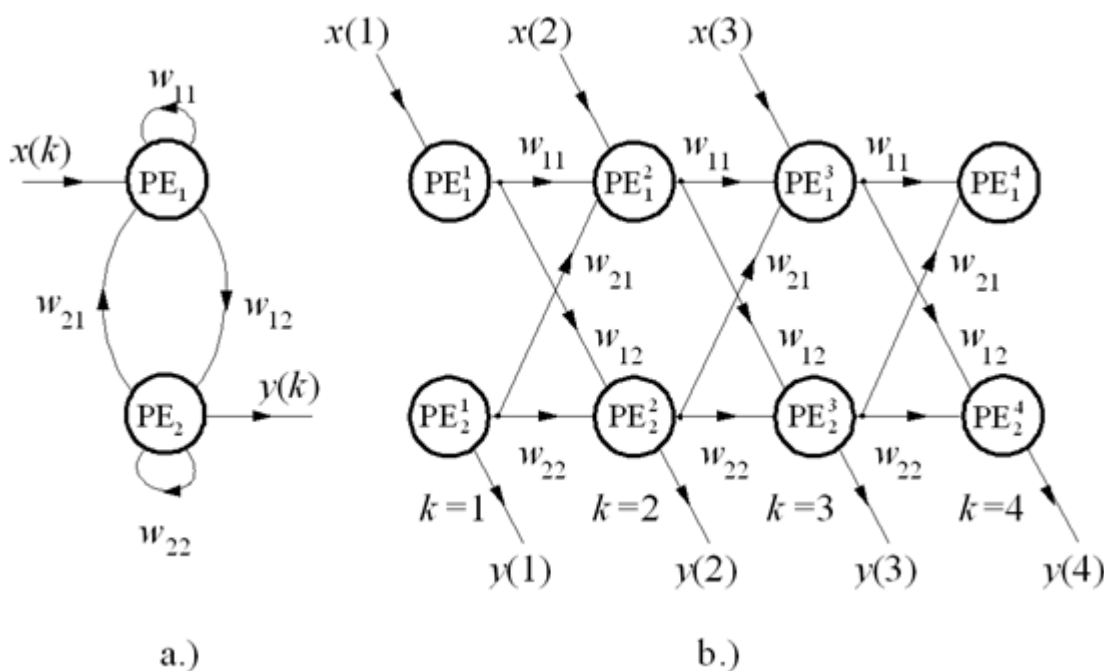
A rekurzív háló időbeli kiterítése egy olyan módszer, amely lehetővé teszi, hogy egy visszacsatolt háló tanítására a hagyományos hibavisszaterjesztési algoritmust alkalmazzuk [Rum86]. Minden visszacsatolásokat tartalmazó háló megfeleltethető egy tisztán előreecsatolt hálónak olyan módon, hogy az egyes processzáló elemek egymást követő időpillanatokbeli viselkedését egy processzáló elem-sorozattal helyettesítjük. E sorozat egyes elemei tehát ugyanannak a processzáló elemnek eltérő időpillanatokhoz tartozó állapotait reprezentálják,

mintegy kiterítve a visszacsatolt hálózat működését az időben. A módszert, amely kisebb hálózatoknál jól alkalmazható, egy egyszerű példán keresztül mutatjuk be.

Legyen egy két processzáló elemből felépülő, teljesen visszacsatolt hálózatunk (8.6 (a) ábra). Az időbeli kiterítés eredményeképpen egy többrétegű, előrecsatolt hálót nyerünk (8.6 (b) ábra), amelynek minden egyes rétege egy-egy időpillanathoz tartozik ($k=1,2,\dots$). A háló sajátossága, hogy bár (az időablak méretétől függően) sok réteget és ennek megfelelő számú súlyt tartalmaz, a különböző rétegekhez tartozó súlyok azonos értékűek, hiszen fizikailag azonos súlyokról van szó.

Az előrecsatolt háló a standard hibavisszaterjesztési eljárással tanítható azzal az eltéréssel, hogy minden lépésben nem módosíthatjuk a súlyokat, és a fizikai azonosság folytán a különböző rétegekbeli súlyokat sem módosíthatjuk egymástól függetlenül. Az egyes időpillanatokhoz tartozó kívánt válaszok azonban összevethetők a háló megfelelő időpillanatbeli tényleges kimenetével, így lépésenként meghatározható a hiba. A lépésenkénti hiba a kiterített hálónál rétegenkénti hibát jelent, tehát most hiba nem csak az utolsó réteg kimenetén, hanem minden réteg kimenetén megadható. A súlymódosításhoz ezeket a hibákat kell a keletkezési helyüknek megfelelő rétegtől visszaterjesztetni. Szemben azonban a normál hibavisszaterjesztési eljárással, ahol az egyes visszaterjesztések eredményeképpen eltérő súlyok módosítását végezzük, itt ugyanazon súlyokról van szó. Ezért a súlyok módosítása az összes rétegen csak egyszerre és azonos mértékben lehetséges. A szokásos eljárás, hogy a visszaterjesztésekből származó súlymódosítás-értékeket összegezzük, és az eredménnyel módosítjuk a súlyokat: az azonos indexű súlyokat egyformán, bármely rétegben is találhatók.

8.6. ábra - Egy két processzáló elemből álló teljesen visszacsatolt hálózat (a), és időben kiterített előrecsatolt megfelelője (BPTT) (b).



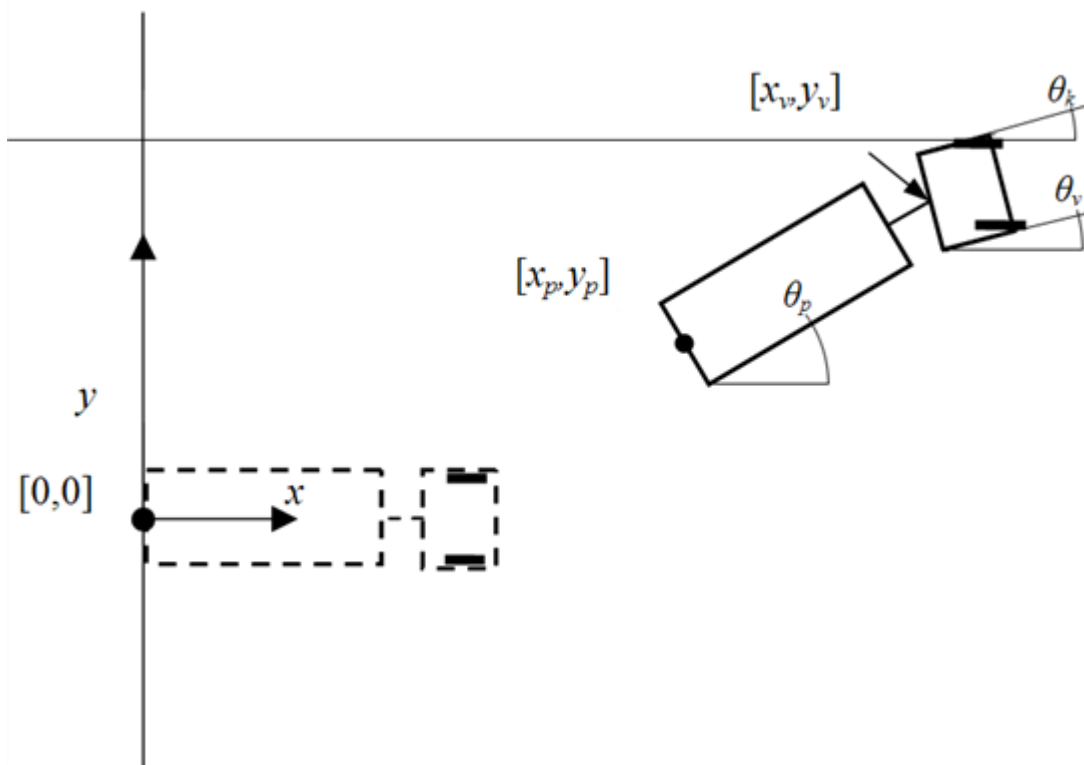
Az eljárás fő hátránya a viszonylag nagy számítás- és tárolási kapacitás-igény. Az eredeti háló-méret annyszorosával kell ugyanis dolgoznunk, ahány időlépésig kiterítettük a hálót. Természetesen a processzáló elemek többszörözése csak a tanítási fázisban szükséges, a megtanított hálót már mint rekurzív hálót működtetjük. További hátrány, hogy a hálózat valós időben nem tanítható, az egyes súlykorrekciók mindig az időablaknak megfelelő számú lépés után végezhetők el.

8.1 példa

A következőkben a rekurzív háló időbeli kiterítésének egy klasszikus alkalmazási példáját mutatjuk be. A megoldandó feladat része annak a példasorozatnak, amelyet Bernard Widrow és munkatársai dolgoztak ki annak bemutatására, hogy különböző egyszerűbb irányítási feladatokat hogyan lehet neuronhálóval megoldani.

A megoldandó feladat, hogy egy pótkocsi teherautót adott kezdeti pozícióból hátramenetben adott végpozícióba juttassunk, mégpedig úgy, hogy az autó kormányát egy neurális kontrollert vezérli. Az autótolatós feladat (*truck backer-upper problem*) a 8.7 ábrán követhető [Ngu89]. Az ábrán az $[x_p, y_p]$ pont jelöli a pótkocsi kezdeti referencia pontját. Ebből a pontból kell kizárólag hátramenetben eljuttatni az autót az $[0,0]$ koordinátájú pontba (az ábrán szaggatott vonallal jelzett pozíció), azt is biztosítva, hogy a végpozícióban legalább a pótkocsi álljon az x tengely irányában (a végpozícióban a θ_p szög legyen nulla).

8.7. ábra - Az autótolatós feladat vázlata



A kontrollert bemenetként a kiinduló, illetve az aktuális pozíció paramétereit kapja $\mathbf{x}=[x_p, y_p, x_v, y_v, \theta_p, \theta_v]^T$, ahol x_v, y_v a vezetőfülke pozíciója és θ_v a vezetőfülke szöge. A kontrollert feladata, hogy a kormányt a megfelelő szögbe állítsa be, vagyis θ_k értékét kell, hogy meghatározza. Feltételezzük, hogy a teherautó elemi tolató lépéseket tesz, és egy elemi lépés alatt a kormány helyzete rögzített. Egy elemi lépés tehát adott kiinduló pozícióból egy következő pozíciót eredményez. Az eredeti feladat-megfogalmazásban egy autó-pozíció jellemzésére a fenti hatelemű $\mathbf{x}$ vektor szolgált.

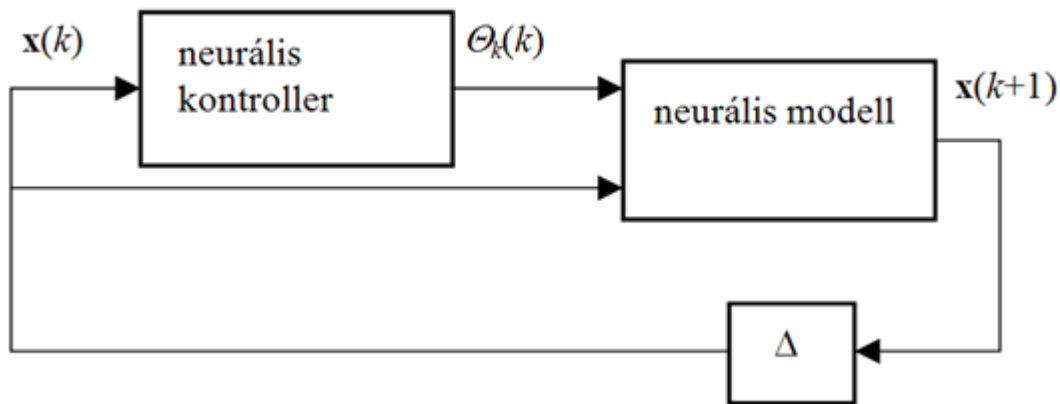
A kontrollert létrehozásához előbb a kocsit modelljé kell létrehozni. Egy neurális modellhez egy nyolc-bemenetű (a pozíciót megadó hat állapotjellemző, a kormány-beállítás szöge és a szokásos eltolásérték (*bias*)), hat-kimenetű hálóra van szükség. A kocsit modelljének birtokában már létrehozható a kontrollert-modell együttes (8.8 ábra). A kontrollert feladata, hogy az $\mathbf{x}(k)$ pillanatnyi állapot alapján meghatározza a következő lépésben szükséges θ_k kormánybeállítást, a modell pedig a pillanatnyi állapot és a kormánysszög alapján meghatározza az $\mathbf{x}(k+1)$ következő állapotot. A következő lépésnél ez képezi a kiinduló állapotot, tehát $\mathbf{x}(k+1)$ -et egy elemi késleltetés közbeiktatásával vissza kell csatolni a kontrollert bemenetére. A teljes elrendezés egy olyan visszacsatolt hálózat, melyben a modell egy már megtanított fix hálózat, a kontrollert pedig egy olyan háló, melyet még meg kell tanítani a megfelelő működésre. A tanítás az időbeli kiterítés elvét alkalmazva, tehát BPTT megközelítésben lehetséges.

Ha K időlépésig működtetjük a tanításhoz a modell-kontrollert együttest, akkor az időbeli kiterítés szerint a fenti elrendezést K -szor meg kell ismételni, ahogy ezt a 8.9 ábra mutatja. Az ábrán a C-vel jelölt blokkok jelentik a kontrollert az M-mel jelöltek pedig a modellt.

A BPTT itteni alkalmazása csak annyiban tér el az általános megoldástól, hogy míg az általános esetben minden lépésben, az autótolatós feladatnál csak az utolsó, K -adik lépésben tudunk hibát meghatározni. Ezt kell

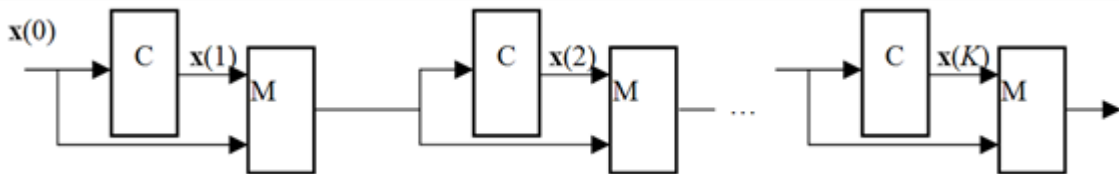
visszaterjeszteni a teljes láncon. A súlymódosítás tényleges elvégzése az időbeli kiterítés elvének bemutatásánál mondottak figyelembevételével lehetséges.

8.8. ábra - A modell-kontroller együttes. Az ábrán Δ egy lépéses késleltetést jelöl.



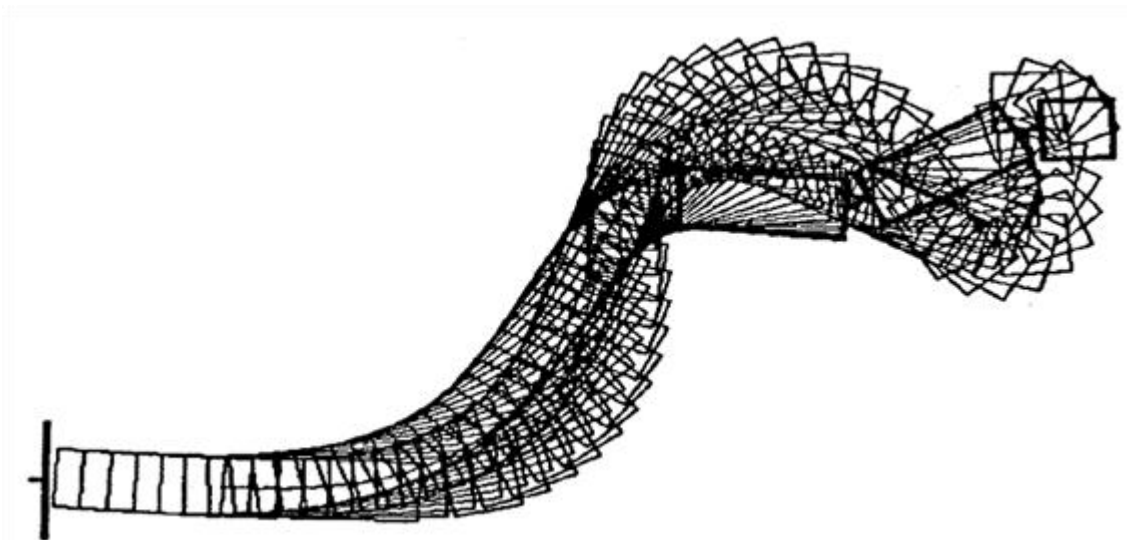
A kontroller tanítása úgy történik, hogy egy kezdeti helyzetből indulva megteszünk K tolató lépést, majd meghatározzuk a végpozíció hibáját. Ezt a hibát visszaterjesztve a teljes láncon egy tolatási szekvenciához tartozó súlymódosítás elvégezhető. Kellő számú tolatási szekvenciával történő tanítás után a háló adott tartományon belül bármely kezdeti helyzetből képes a kocsit a végpozícióba vezérelni.

8.9. ábra - A BPTT megközelítés alkalmazása az autótoltós feladatra



A 8.10 ábra a feladat bonyolultabb változatának egy tipikus eredményét mutatja [Wid90b]. A bonyolultabb változatban a vezetőfülke mögött két pótkocsi van.

8.10. ábra - A két pótkocsis teherautó tolatási feladat egy tipikus eredménye



4.2. Rekurzív hálók tanítása pillanatnyi gradiens alapján

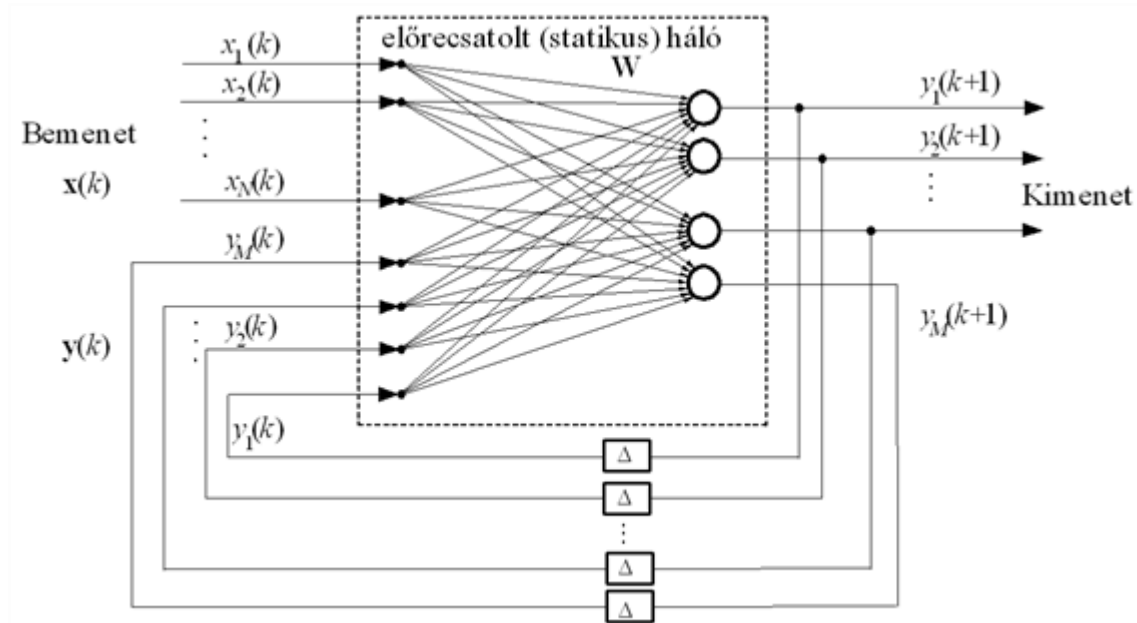
Az alábbiakban olyan, visszacsatolt háló-architektúrára alkalmazható tanulási eljárást mutatunk be, mely a pillanatnyi gradiens alapján valós idejű, lépésenkénti tanítást tesz lehetővé [Wil89].

A háló felépítését a 8.11 ábra mutatja. Látható, hogy ez valójában egy globális visszacsatolást tartalmazó háló, hiszen a háló kimenetéről is van visszacsatolás a háló bemenetére, ugyanakkor a háló lokális visszacsatolásokat is tartalmazhat, mivel a statikus háló-kimenetek nem mindegyike képezi a teljes háló kimenetét. Ha az általános modell-struktúrák szerint kategorizáljuk a hálózatot, akkor az alapján egy NOE struktúrának tekinthető. Ha azonban figyelembe vesszük, hogy van olyan processzálo eleme is, amelyről történik visszacsatolás, de amely közben nem képezi a háló kimenetét, akkor a hálót egy más modell-osztályba tartozónak is tekinthetjük. A visszacsatolt, de a kimenetre ki nem jutó PE-k a háló belső állapotait reprezentálják, így egy nemlineáris állapotteret modellező látszik e struktúra jellemzésére a legalkalmasabbnak.

A rekurzív háló emlékezzettel rendelkezik, tehát egy bemeneti szekvenciára egy kimeneti szekvenciával válaszol. A kimenet a $(k+1)$ -edik időpillanatban a visszacsatolás miatt a következő:

$$y_i(k+1) = f[x_i(k), y(k)]. \quad (8.39)$$

8.11. ábra - Rekurzív háló architektúra



Az egységes kezelés érdekében vezessük be a következő jelölést. Jelöljük az előrecsatolt háló-rész i -edik bemeneti komponensét $u_i(k)$ -val:

$$u_i(k) = \begin{cases} x_i(k) & \text{ha } i \in \mathbf{A} \\ y_i(k) & \text{ha } i \in \mathbf{B} \end{cases} \quad (8.40)$$

ahol $\mathbf{A}$ jelöli a bemeneti indexek halmazát, $\mathbf{B}$ a háló azon processzálo elemeinek index-halmazát, mely processzálo elemek kimenetét visszacsatoljuk. Ezzel a háló kimenete

$$y_i(k+1) = f[s_i(k)] = f\left[\sum_{j \in \mathbf{A} \cup \mathbf{B}} w_{ij} u_j(k)\right], \quad (8.41)$$

vagy az eredeti jelölésekkel

$$y_i(k+1) = f\left[\sum_{j \in \mathbf{A}} w_{ij} x_j(k) + \sum_{j \in \mathbf{B}} w_{ij} y_j(k)\right] \quad (8.42)$$

A hiba a háló azon kimenetein értelmezhető, amelyeknél kívánt választ specifikálunk. Vagyis a hibakomponensek a k -adik időpillanatban:

$$\varepsilon_i^2(k) = \begin{cases} (d_i(k) - y_i(k))^2 & \text{ha } i \in C(k) \\ 0 & \text{egyébként} \end{cases} \quad (8.43)$$

Itt $C(k)$ jelöli azon processzáló elem-indexek halmazát, amelyekre kívánt választ specifikálunk a k -adik lépésben. A halmaz lépésfüggő megadása azt jelenti, hogy időlépésenként eltérő neuronokhoz rendelhetünk kívánt választ.

Az eredő pillanatnyi hiba a hibakomponensek összege:

$$\varepsilon^2(k) = \sum_{i \in C(k)} \varepsilon_i^2(k) \quad (8.44)$$

A megfelelő működés eléréséhez a tanítás során a teljes szekvencia alatti eredő hibát, vagyis a pillanatnyi hibák teljes szekvenciára, mint időablakra vett összegét kell minimalizálnunk.

$$\varepsilon^2 = \sum_{k=1}^K \varepsilon^2(k) \quad (8.45)$$

A gradiens módszer alkalmazásakor ennek a teljes hibának a súlymátrix szerinti gradiensét:

$$\frac{\partial \varepsilon^2}{\partial \mathbf{W}} = \sum_k \frac{\partial \varepsilon^2(k)}{\partial \mathbf{W}} \quad (8.46)$$

kell meghatároznunk és a szokásos módon, a negatív gradiens irányában kell változtatnunk a súlyokat. A gradiens számítás az egyes időpillanatokbeli gradiens számítás felhasználásával elvégezhető. Ennek megfelelően minden egyes időpillanathoz tartozik egy súlymódosítás, és ezen súlymódosítások adott időablakra vett összege adja a tényleges súlymódosítás értékét. Itt súlykorrekció nyilvánvalóan nem történhet minden időpillanatban, hiszen az időablakon belüli hibameghatározás és súlykorrekció-számítás feltételezi, hogy közben a súlyok nem változnak, tehát a háló ezen időlépések alatt rögzített. A tanítást szakaszosan végezzük, egy szakaszban tulajdonképpen az összes tanítópontot, tehát a teljes tanítószekvenciát egyszer felhasználjuk.

Az időablakra számított teljes hiba alapján történő tanítás tehát – hasonlóan az előző pontban bemutatott tanításhoz – nem végezhető el valós időben. A valós idejű hálózat-módosításhoz a pillanatnyi hiba gradiensét minden egyes lépésben kiszámítjuk és minden lépésben el is végezzük a súlyok módosítását. Ez azt jelenti, hogy a háló "emlékezetének" ideje alatt is változik maga a háló, igazában tehát nem a valódi gradiens irányában történik a súlymódosítás. E módszer a legmeredekebb lejtő eljárásnak csupán közelítését adja. Az eltérés a "valódi" és a közelítő gradiens között annál kisebb, minél óvatosabb a lépésenkénti súlymódosítás, tehát minél kisebb a tanulási tényező. Megfelelően kis μ alkalmazása esetén a valós idejű tanítás a valódi gradiens alapú tanítás jó közelítése lesz. Az eljárás előnye, hogy egyszerűbb, minden újabb tanítópont felhasználásánál azonos módon kell eljárunk, nem kell tárolnunk, illetve akumulálnunk az egyes lépésekhez tartozó súlymódosító értékeket, nem kell blokkosan végezzük a tanítást. A súlymódosítás alapösszefüggései a kétféle megközelítésben azonosak, különbség tehát csak az elemi súlymódosítások felhasználásában van. A módszer alkalmazhatóságát szimulációs vizsgálatok és gyakorlati alkalmazások igazolják. Az alábbiakban a tanítási összefüggések származtatását mutatjuk be.

A súlykomponensekre vonatkoztatott pillanatnyi gradiens az alábbi formában határozható meg:

$$\frac{\partial \varepsilon^2(k)}{\partial w_y} = \sum_{i \in C(k)} \frac{\partial \varepsilon_i^2(k)}{\partial w_y} \quad (8.47)$$

A megfelelő súly módosítását meghatározó összefüggés:

$$\Delta w_y(k) = -\mu \frac{\partial \varepsilon^2(k)}{\partial w_y}, \quad (8.48)$$

ahol μ most is a tanulási tényező (bátorsági faktor). A gradienst továbbírva kapjuk, hogy:

$$\begin{aligned}\frac{\partial \mathcal{E}^2(k)}{\partial w_{ij}} &= 2 \sum_{i \in \mathcal{C}} e_i(k) \frac{\partial e_i(k)}{\partial w_{ij}} \\ &= -2 \sum_{i \in \mathcal{C}} e_i(k) \frac{\partial y_i(k)}{\partial w_{ij}}\end{aligned}\quad (8.49)$$

A $\frac{\partial y_i(k)}{\partial w_{ij}}$ parciális deriváltat a háló dinamikáját leíró (8.41), ill. (8.42) egyenleteket figyelembe véve kapjuk meg:

$$\frac{\partial y_i(k+1)}{\partial w_{ij}} = f'(s_i(k)) \frac{\partial s_i(k)}{\partial w_{ij}}, \quad (8.50)$$

ahol

$$s_i(k) = \sum_{r \in \mathcal{A}} w_{ir} x_r(k) + \sum_{r \in \mathcal{B}} w_{ir} y_r(k) = \sum_{r \in \mathcal{A} \cup \mathcal{B}} w_{ir} u_r(k). \quad (8.51)$$

Tehát

$$\begin{aligned}\frac{\partial y_i(k+1)}{\partial w_{ij}} &= f'(s_i(k)) \sum_{r \in \mathcal{A} \cup \mathcal{B}} \frac{\partial [w_{ir} u_r(k)]}{\partial w_{ij}} \\ &= f'(s_i(k)) \sum_{r \in \mathcal{A} \cup \mathcal{B}} \left[\frac{\partial w_{ir}}{\partial w_{ij}} u_r(k) + w_{ir} \frac{\partial u_r(k)}{\partial w_{ij}} \right]\end{aligned}\quad (8.52)$$

Felhasználva, hogy $\frac{\partial w_{ir}}{\partial w_{ij}} = 1$, ha $i=r$ és $r=j$, egyébként a két súly között nincs kapcsolat, tehát a derivált értéke 0:

$$\frac{\partial w_{ir}}{\partial w_{ij}} = \delta_{ij}, \quad (8.53)$$

ha $r=j$, (itt δ_{ij} a Kronecker delta) és hogy

$$\frac{\partial u_r(k)}{\partial w_{ij}} = \begin{cases} 0 & \text{ha } r \in \mathcal{A} \\ \frac{\partial y_r(k)}{\partial w_{ij}} & \text{ha } r \in \mathcal{B}, \end{cases} \quad (8.54)$$

$$\frac{\partial y_i(k+1)}{\partial w_{ij}} = f'(s_i(k)) \left[\sum_{r \in \mathcal{B}} w_{ir} \frac{\partial y_r(k)}{\partial w_{ij}} + \delta_{ij} u_j(k) \right]. \quad (8.55)$$

Fenti összefüggés a parciális deriváltra egy rekurzív összefüggés, amelynek kiértékeléséhez természetes feltételezés, hogy a kiinduló érték

$$\frac{\partial y_i(0)}{\partial w_{ij}} = 0. \quad (8.56)$$

A lépésenkénti súlymódosítás a fentiek felhasználásával tehát:

$$w_{ij}(k+1) = w_{ij}(k) + 2\mu \sum_{i \in \mathcal{C}} \left\{ e_i(k) f'(s_i(k)) \left[\sum_{r \in \mathcal{B}} w_{ir} \frac{\partial y_r(k)}{\partial w_{ij}} + \delta_{ij} u_j(k) \right] \right\}, \quad (8.57)$$

ahol $i \in \mathcal{B}, j \in \mathcal{A} \cup \mathcal{B}, l \in \mathcal{C}$ és $r \in \mathcal{B}$.

A valós idejű eljárás hátránya, mint említettük, hogy nem pontosan a negatív gradiens mentén történnek a korrekciós lépések. Ez nagyon hasonló a standard hibavisszaterjesztéses eljárásnál alkalmazott módszerhez akkor, ha minden egyes tanítópont után végzünk súlymódosítást és nem csak a batch tanításon szokásos módon, amikor csak a teljes tanító sorozatokat (epochokat) követően történik súlymódosítás.

A Ronald Williams és David Zipser által javasolt eljárásnak több módosított verzióját is kidolgozták. A valódi gradiens és a pillanatnyi gradiens alapján történő számítások közötti átmenet különösen akkor gyorsítja a konvergenciát, ha a hálóval feldolgozandó bemeneti jel-szekvencia nagyon hosszú. Tony Robinson és Frank Fallside [Rob91] olyan hálózat tanítására adott a fentiektől eltérő megoldást, ahol visszacsatolás a valódi kimenetekről – tehát amely neuronokhoz kívánt válaszokat is rendelünk – nincs, csupán a háló többi – valójában rejtett – neuronjának kimeneteit csatoljuk vissza.

5. Dinamikus hálózatok kialakításának néhány további lehetősége

Az előzőekben bemutatott konkrét neuronháló architektúrákon túl számos további lehetőség áll rendelkezésünkre neuronháló alapú dinamikus rendszerek létrehozására. Ezek a lehetőségek azon a már említett megközelítésen alapulnak, hogy statikus nemlineáris és dinamikus lineáris rendszerek különböző struktúrájú összekapcsolásával különböző nemlineáris dinamikus modellek nyerhetők.

Láttuk, hogy a statikus tulajdonságú hálózatok dinamikussá tétele során a leggyakrabban használt lehetőség késleltető sorok (*tapped delay line*), illetve visszacsatolások alkalmazása. Természetesen az így keletkező struktúrák tanításához legtöbbször egy új tanító algoritmus megalkotása is szükséges. Erre az előzőekben több példát is láttunk. A módszereket alapvetően három csoportba sorolhatjuk:

- dinamika beépítése a feldolgozó elemekbe (tulajdonképpen lokális visszacsatolás),
- visszacsatolások kialakítása a hálózaton belül (globális vagy lokális visszacsatolás),
- egybeépítés hálózaton kívüli dinamikával.

Valójában mindhárom fenti lehetőség statikus-nemlineáris és dinamikus-lineáris részek – különféle kombinációkban történő – összekapcsolását jelenti.

Az első lehetőséget sokan vizsgálták, elsősorban a hibavisszaterjesztéses algoritmussal tanított többrétegű perceptron hálózathoz kiindulva (ld. pl. a FIR-MLP hálót). Ennek a módszernek a hátránya az, hogy a többrétegű háló önmagában is nagy számításigényű tanítása a dinamikus elemek beépítése miatt még nagyobb számítási kapacitást igényel, az így kapott háló tanítása meglehetősen lassú lesz.

A második lehetőségre jellegzetes példaként két rekurzív hálózat-architektúrát a megfelelő tanítási algoritmusokkal szintén bemutatunk.

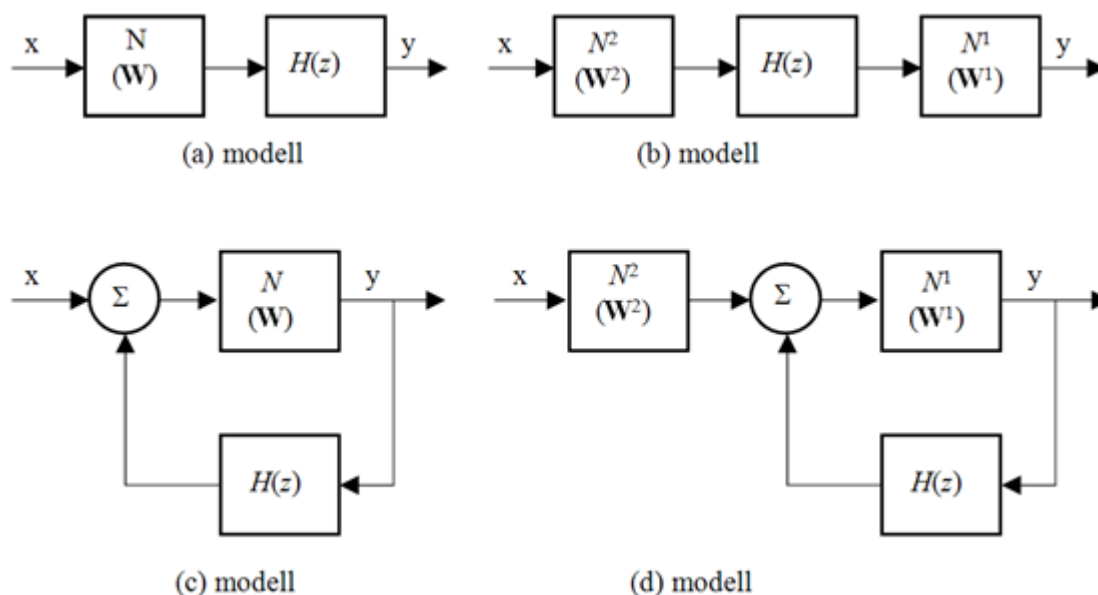
A továbbiakban az utolsó lehetőséget vizsgáljuk, mivel az így felépíthető rendszerek igen bonyolultak is lehetnek, ugyanakkor strukturált felépítésük miatt viszonylag könnyebben értelmezhetők. Bár ezekre a lehetőségekre már utaltunk, a lineáris dinamikus és nemlineáris statikus rendszerek kombinációjaként származtatott elrendezések vizsgálata az eddigiektől kissé eltérő, általános megközelítés bevezetését teszi lehetővé: a tanítást, mint az általános érzékenységvizsgálat eredményét felhasználó eljárást értelmezi. Ezt a megközelítést Kumpati Narendra cikkeit [Nar90] és [Nar91] követve a 8.12 ábrán látható elrendezésekhez kapcsolódóan fogjuk bemutatni.

Az ábra négy elrendezést mutat, melyek mindegyike statikus nemlineáris leképezést megvalósító neuronháló(k)ból és lineáris dinamikus rendszerből áll. Az ábrán $H(z)$ jelöli a lineáris dinamikus részszerkezetet, N , M^1 , M^2 pedig neurális hálózattal megvalósított nemlineáris statikus elemeket, ahol $\mathbf{W}$ jelöli a hálózat súlymátrixát. Ezeket az összetett struktúrákat általánosított neurális hálózatoknak (*generalized neural network*) nevezzük.

5.1. A parciális deriváltak számítása

A statikus MLP hálózatok tanítására alkalmazott hibavisszaterjesztéses algoritmus (5. fejezet) kiszámítja a hálózat négyzetes kimeneti hibájának a hálózat paraméterei (súlyai) szerinti parciális deriváltjait. Erre azért van szükség, mert az alkalmazott gradiens alapú algoritmus ez alapján számítja a súlymódosításokat. Az algoritmus a rétegeken keresztül visszafelé haladva a láncszabály alkalmazásával számítja ki az egyes deriváltakat, ami tulajdonképpen azt jelenti, hogy mindig kiszámolja az egyes rétegek kimeneteinek a réteg bemenetei szerinti deriváltjait is.

8.12. ábra - A Narendra által javasolt struktúrák



Amikor a statikus hálózatot egy összetett struktúra részeként akarjuk használni és az egész rendszert akarjuk tanítani, akkor – amint a következő részben majd látjuk – szükség lehet a neurális alaprész kimeneteinek a bemenetei szerinti parciális deriváltjaira is. Ezeket a deriváltakat a hibavisszaterjesztéses algoritmushoz nagyon hasonló módon, a hálózat rétegein visszafelé haladva tudjuk kiszámítani.

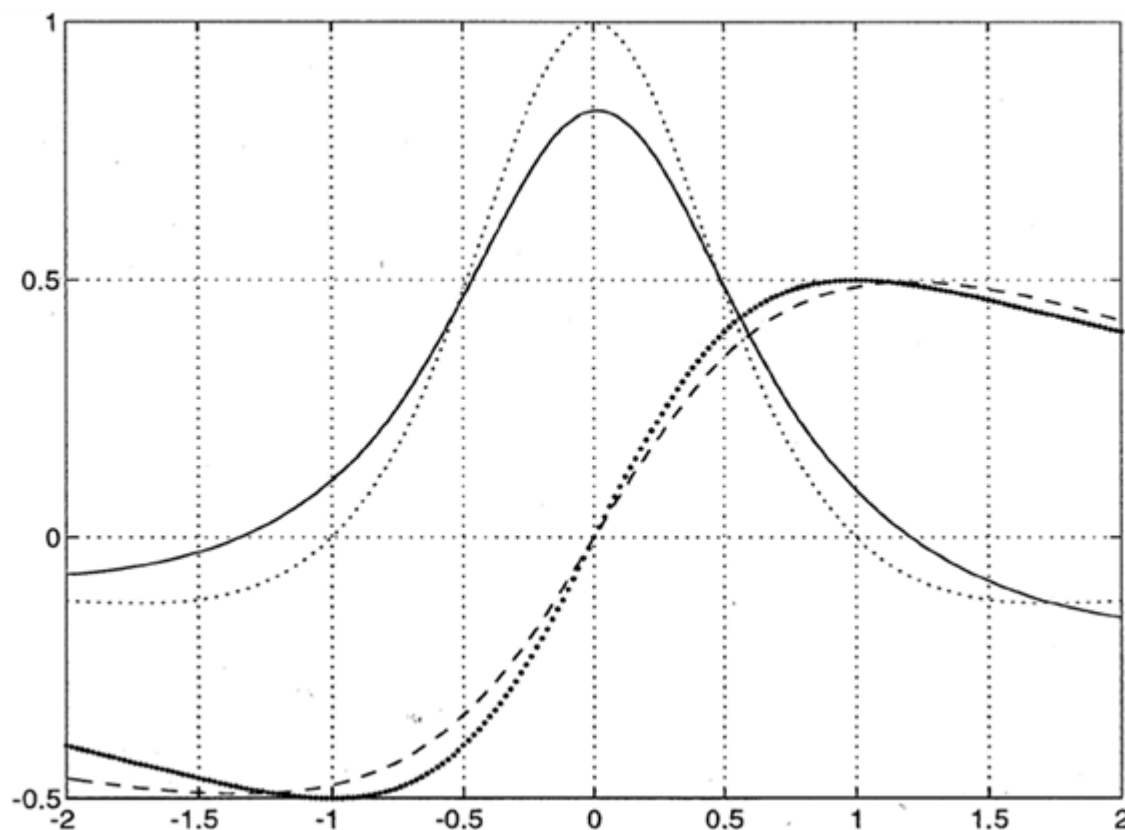
A módszert a hibavisszaterjesztéses algoritmushoz is használt, a 4.1. ábrán látható kétrétegű, két kimenettel és

egy háromelemű rejtett réteggel rendelkező hálózaton demonstráljuk. Példaként a $\frac{\partial y_1}{\partial x_2}$ parciális derivált számítását végezzük el:

$$\begin{aligned}
 \frac{\partial y_1}{\partial x_2} &= \frac{\partial y_1}{\partial x_1^{(1)}} = \frac{\partial y_1}{\partial s_1^{(2)}} \frac{\partial s_1^{(2)}}{\partial x_1^{(1)}} = f'(s_1^{(2)}) \sum_{i=1}^3 \frac{\partial s_1^{(2)}}{\partial y_i^{(1)}} \frac{\partial y_i^{(1)}}{\partial x_1^{(1)}} = f'(s_1^{(2)}) \sum_{i=1}^3 w_{1i}^{(2)} \frac{\partial y_i^{(1)}}{\partial x_1^{(1)}} \\
 &= f'(s_1^{(2)}) \sum_{i=1}^3 w_{1i}^{(2)} \frac{\partial y_i^{(1)}}{\partial x_1^{(1)}} = f'(s_1^{(2)}) \sum_{i=1}^3 w_{1i}^{(2)} \frac{\partial y_i^{(1)}}{\partial s_i^{(1)}} \frac{\partial s_i^{(1)}}{\partial x_1^{(1)}} \\
 &= f'(s_1^{(2)}) \sum_{i=1}^3 w_{1i}^{(2)} f'(s_i^{(1)}) w_{i2}^{(1)}
 \end{aligned} \tag{8.58}$$

A 4.2. alfejezetben bemutatotthoz hasonló módon itt is bevezethetők az egyes rétegek neuronjainak lineáris kimeneteihez tartozó kimeneti deriváltak (a hálózat kimenetének a neuronok lineáris kimenetei szerinti deriváltjai). Ha a (4.4)-es képlettel definiált származtatott hibagradiens számításakor a kimeneti hiba (δ_1) helyére egységet helyettesítünk, akkor a további visszaterjesztő lépések során a $\delta_i^{(r)}$ mennyiségek sorra a kimenet parciális deriváltjait fogják tartalmazni az adott neuron lineáris kimenete szerint. Ezért a közönséges statikus hibavisszaterjesztéses algoritmus felhasználható a kimenet parciális deriváltjainak számításánál (közvetlenül azonban csak egy kimenet esetén). A j -edik bemenethez tartozó parciális derivált az első réteg neuronjainak az adott bemenethez tartozó súlyai ($w_{ij}^{(1)}$) és a $\delta_i^{(1)}$ mennyiségek lineáris kombinációja lesz.

8.13. ábra - A kimenet bemenet szerinti gradiensének számítása egybemenetű - egykimenetű hálózaton



A 8.13 ábra a fenti módszerrel meghatározott parciális derivált számítását mutatja egybemenetű és egykimenetű hálózat esetén (ekkor persze a parciális derivált közösleges deriváltat jelent). Az ábrán a tanító jel és a hálózat válasza is látható 200 tanító ciklus után. (A konkrét példánál a 200 tanító ciklus még kevés). A sűrűn pontozott vastagabb vonal jelöli a megtanulandó függvényt, a szaggatott a hálózat válaszát. A folytonos vonal a fenti módon számított gradiens, míg a ritkán pontozott vékonyabb vonal a megtanulandó jel valódi deriváltja.

5.2. Neurális hálózatot tartalmazó összetett struktúrák tanítása

Elsőként a 8.12 ábrán felvetett legegyszerűbb, (a) modellt vizsgáljuk. Ez egy statikus neurális hálózatot és ennek kimenetére kapcsolt (ismert) lineáris dinamikus elemet (szűrőt) tartalmaz. Mivel már ebben az egyszerű modellben sem a neurális rész kimenetén keletkezik a hiba, a statikus hibavisszaterjesztés nem használható. (A kimeneten kapott hibát közvetlenül nem használhatjuk föl a hálózat tanítására. Gondoljunk csak a legegyszerűbb esetre, ha a kimeneti tag nem is dinamikus, hanem csak egy konstans szorzó. Ha a konstans értéke negatív és a kimeneti hibát használva statikus tanító algoritmussal próbáljuk tanítani a hálózatot, akkor az mindig pontosan a helyi gradienssel megegyező irányban, azaz az optimumtól távolodva fogja változtatni a hálózat súlyait.) Erre az esetre dolgozták ki az ún. dinamikus hibavisszaterjesztéses algoritmust.

A statikus esettől eltérően most mind a kimeneti kívánt értékek, mind a kimeneti hibavektor időfüggvények. Célunk most is a kimeneti négyzetes hiba minimalizálása, és most is gradiens alapú módszert használunk.

Az 8.14 ábra több-bemenetű-többkimenetű hálózatra mutatja a gradiensszámítás elvét. Az ábra jelöléseivel az alábbi eredményre juthatunk:

$$\frac{\partial \mathbf{e}(k)}{\partial w_{ij}} = -\frac{\partial y(k)}{\partial w_{ij}} = -H(z) \frac{\partial v}{\partial w_{ij}} \quad (8.59)$$

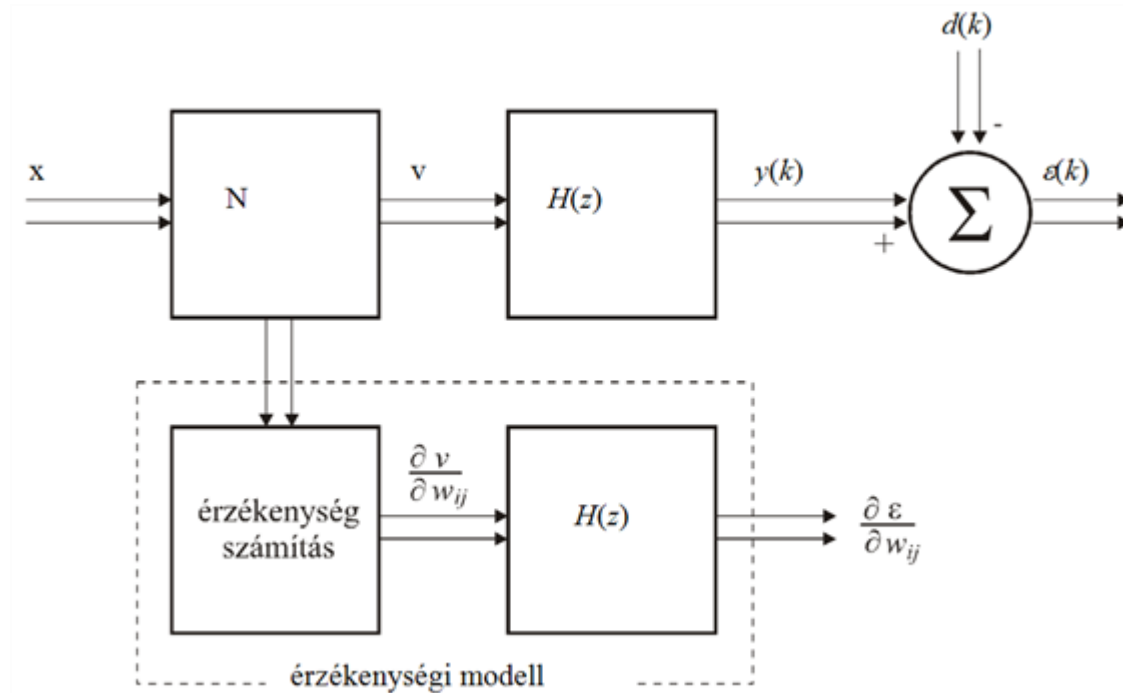
A $\frac{\partial v}{\partial w_{ij}}$ parciális deriváltak minden lépésben számíthatók a statikus hibavisszaterjesztéses módszerrel. A képletet értelmezve azt láthatjuk, hogy az egymás után, statikus visszaterjesztéssel kapott súlyváltoztatásokat át kell engedni (szűrni) a kimeneti lineáris dinamikán. Ezt mutatja az 8.14 ábrán az érzékenységi modellnek nevezett rész. Ha az előbbi egyszerű dinamikamentes példára vizsgáljuk az eredményt, láthatjuk, hogy a kimeneti elemén átszűrve a súlyváltoztatásokat, valóban jó eredményre jutunk.

A 8.12 (b) ábrán felvett N^* rendszer N^l jelű neurális hálózata a kimeneten helyezkedik el, így statikus algoritmussal tanítható. Az N^l jelű hálózat gradiense az alábbi összefüggés szerint határozható meg:

$$\frac{\partial \varepsilon(k)}{\partial w_{ij}} = \frac{\partial y(k)}{\partial w_{ij}} = \sum_i \frac{\partial y(k)}{\partial v_i} \frac{\partial v_i}{\partial w_{ij}} \quad (8.60)$$

A fenti kifejezésben a $\frac{\partial y(k)}{\partial v_i}$ differenciáhányados az előző részben meghatározott módon (ld. (8.58) összefüggés), míg a $\frac{\partial v_i}{\partial w_{ij}}$ derivált az (a) rendszernél bemutatott módon számítható.

8.14. ábra - A gradiens származtatása az 8.12 (a) ábrán látható modellben



A 8.12 (c) modell egy lineáris rendszerrel visszacsatolt neurális hálózatot tartalmaz. A modell kimenete ennek megfelelően:

$$y = N(x + H(z)y) \quad (8.61)$$

Az eddigiekhez hasonlóan most is igaz, hogy:

$$\frac{\partial \varepsilon(k)}{\partial w_{ij}} = - \frac{\partial y(k)}{\partial w_{ij}}, \quad (8.62)$$

A visszacsatolás miatt az itt adódó egyenlet egy differenciaegyenlet; a keresett deriváltak függnek saját korábbi értékeiktől is.

$$\frac{\partial \varepsilon(k+1)}{\partial w_{ij}} = - \frac{\partial N(v)}{\partial v} H(z) \frac{\partial y(k)}{\partial w_{ij}} + \frac{\partial N(v)}{\partial w_{ij}}, \quad (8.63)$$

ahol a $\frac{\partial N(\mathbf{v})}{\partial w_y}$ értékek statikus visszaterjesztéssel, a $\frac{\partial N(\mathbf{v})}{\partial \mathbf{v}}$ Jacobi mátrix (enr $\frac{\partial N(\mathbf{v})}{\partial w_y}$ me) a kimenetek deriváltjai a bemen $\frac{\partial y(k)}{\partial w_y}$ zerint) pedig az (a) modellnél bemutatott módon számíthatók (a pillanatnyi deriváltat jelöli, míg a az "időben" számítottat).

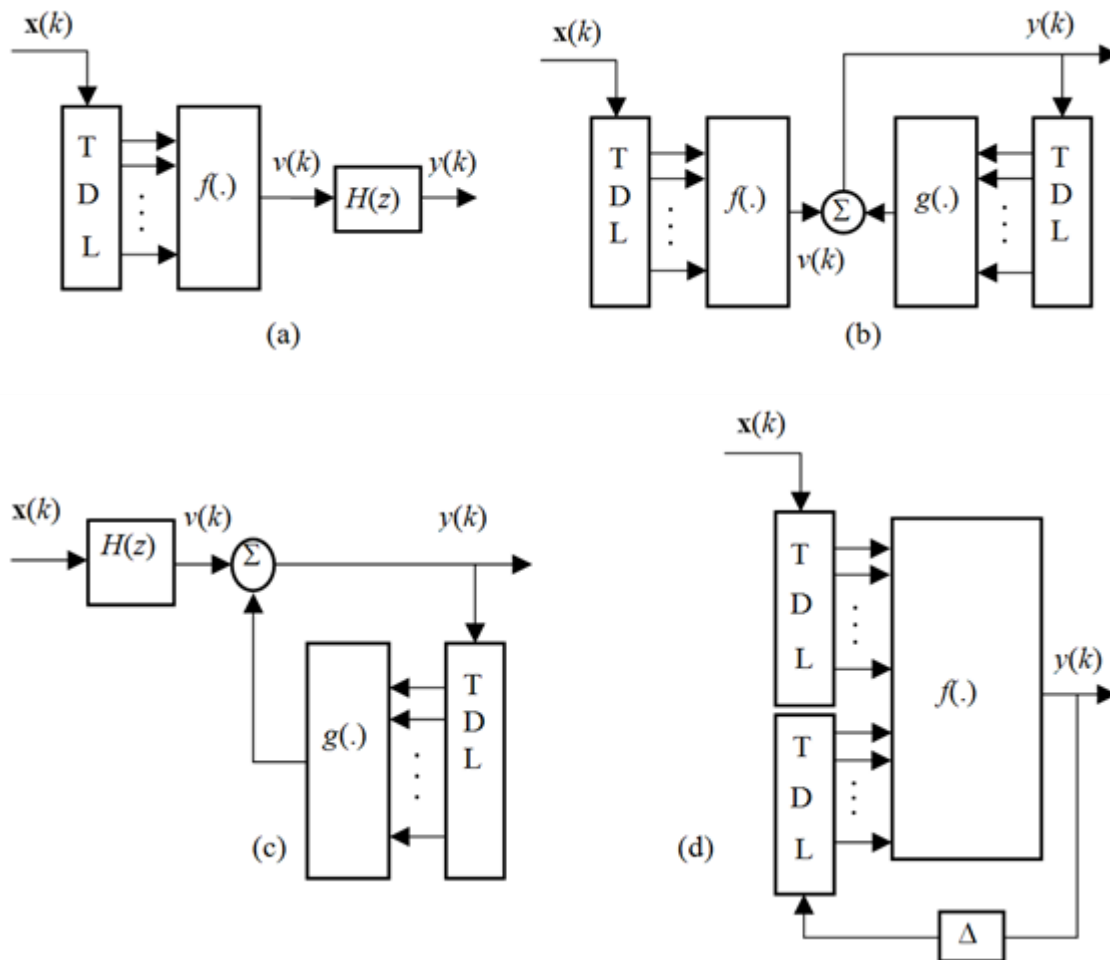
A (d) modell a legösszetettebb a négy közül. Az N^1 jelű hálózat a (c) modellnél bemutatott módon tanítható, míg az N^2 tanításához szükséges hibaderiváltakra a következő formula származtatható:

$$\frac{\partial \varepsilon(k)}{\partial w_y} = \frac{\partial y(k)}{\partial w_y} = \frac{\partial N^1(\mathbf{v})}{\partial \mathbf{v}} \left(\frac{\partial N^2(x)}{\partial w_y} + H(z) \frac{\partial y(k)}{\partial w_y} \right) \quad (8.64)$$

Mind a (c), mind a (d) modell tanítására kapott eredmények úgy is értékelhetők, hogy a statikus visszaterjesztéssel kapott deriváltakat át kell engedni egy, a beágyazó rendszer által meghatározott dinamikus rendszeren [Nar90].

A 8.15 ábra négy konkrét rendszert mutat, amelyek tanítása az előzőekben bemutatott elvek alkalmazásával elvégezhető.

8.15. ábra - Négy dinamikus struktúra.



Az egyes architektúráknál $f(\cdot)$ és $g(\cdot)$ statikus nemlineáris leképezéseket jelöl, melyeket valamilyen neuronhálóval valósíthatunk meg. A TDL blokkok késleltető láncot jelölnek a késleltető fokozatok közötti megcsapolásokkal, $H(z)$ pedig egy lineáris dinamikus rendszert, amely lehet előrecsatolt (FIR) vagy visszacsatolt (IIR) típusú is. E struktúráknak az az érdekessége, hogy a nemlineáris leképezéseket statikus

hálókkal megvalósító $f(\cdot)$ és $g(\cdot)$ blokkok bemenetén megcsapolt késleltető sorokat találunk, tehát ezek a hálórészek önmagukban is dinamikus nemlineáris rendszerek (pl. FIR-MLP-k). Ennek megfelelően az (a) architektúra egy NFIR típusú dinamikus neuronháló és egy lineáris szűrő sorbakapcsolásából áll. A (b) elrendezésben két NFIR típusú háló található; az egyik az előrecsatoló, a másik a visszacsatoló ágban. A (c) elrendezés hasonló a (b)-hez, azzal a különbséggel, hogy az előrecsatoló ágban lévő NFIR háló helyett egy lineáris szűrő szerepel, míg a (d) elrendezés a fejezet elején bemutatott NOE modell. Ezek az architektúrák is azt illusztrálják, hogy neuronháló alapú nemlineáris dinamikus rendszermodelleket nagyon sok változtában tudunk konstruálni. Az itt bemutatott architektúrák tanítása a megfelelő dinamikus neuronhálók és az összetett struktúrák tanításánál bemutatott eljárások kombinálásával végezhető el.

6. Dinamikus hálók alkalmazása

A dinamikus neurális hálózatok alkalmazása igen szerteágazó. A hálók legfontosabb alkalmazási területei bonyolult ipari, gazdasági, pénzügyi, stb. rendszerek modellezéséhez, irányításához, illetve az ilyen komplex rendszerek viselkedésének előrejelzéséhez kapcsolódnak. Általánosságban elmondható, hogy a dinamikus neuronhálók mindazon területeken sikerrel alkalmazhatók, ahol a hagyományos eszközökkel történő feladatmegoldás nem vagy nehezen lehetséges, és ahol a neuronhálók alkalmazásához elengedhetetlenül szükséges megfelelő mintakészlet rendelkezésre áll vagy megszerezhető. Egyfelől tehát a rendelkezésre álló ismeret, tudás jellege, másfelől a feladat komplexitása szabja meg, hogy a neuronhálók adott feladat megoldásában sikerrel alkalmazhatók-e vagy sem.

A dinamikus hálók alkalmazási körét két fő szempont határozza meg: egyrészt a dinamikus hálók képessége, másrészt az, hogy milyen előnyök várhatók a neuronhálók alkalmazásától. Láttuk, hogy a statikus neuronhálók több típusára igaz, hogy univerzális approximátorok, tehát képesek tetszőleges folytonos nemlineáris leképezés tetszőleges pontosságú közelítésére. A dinamikus neuronhálók – bármilyen alkalmazási területet is nézünk – sikerüket az általános modellező képességüknek köszönhetik. Előnyük – és részben hátrányuk is –, hogy a megoldandó feladatról viszonylag kevés háttérismeret birtokában is alkalmazhatók. Téves nézet azonban az, hogy mivel egy neuronháló fekete doboz modellt valósít meg, itt nincs is szükség a megoldandó feladat részletes megismerésére. Miközben a fekete doboz modell a bemeneti-kimeneti mintapontok vagy mintaszekvenciák alapján létrehozható, a megoldás minőségét és a megfelelő minőségű megoldás elérésének nehézségét is döntően befolyásolja, hogy milyen jellegű és mennyi ismeret áll rendelkezésünkre.

A dinamikus hálók alapvetően a statikus hálók kiterjesztései révén származtathatók, ahol a fő képességeket a regresszor és a nemlineáris leképezés együttesen határozzák meg. Megfelelően megválasztott regresszor és az univerzális approximátor képesség biztosítja, hogy a dinamikus neuronhálót, mint univerzális nemlineáris dinamikus rendszermodellező eszközöket tekintsük.

A következőkben előbb a dinamikus rendszerek modellezésének néhány általános kérdését foglaljuk össze röviden, majd a neurális hálózatok dinamikus feladatokra történő alkalmazásánál felmerülő fontosabb kérdésekkel foglalkozunk, végül néhány mintapéldát mutatunk be.

6.1. Identifikáció, rendszermodellezés

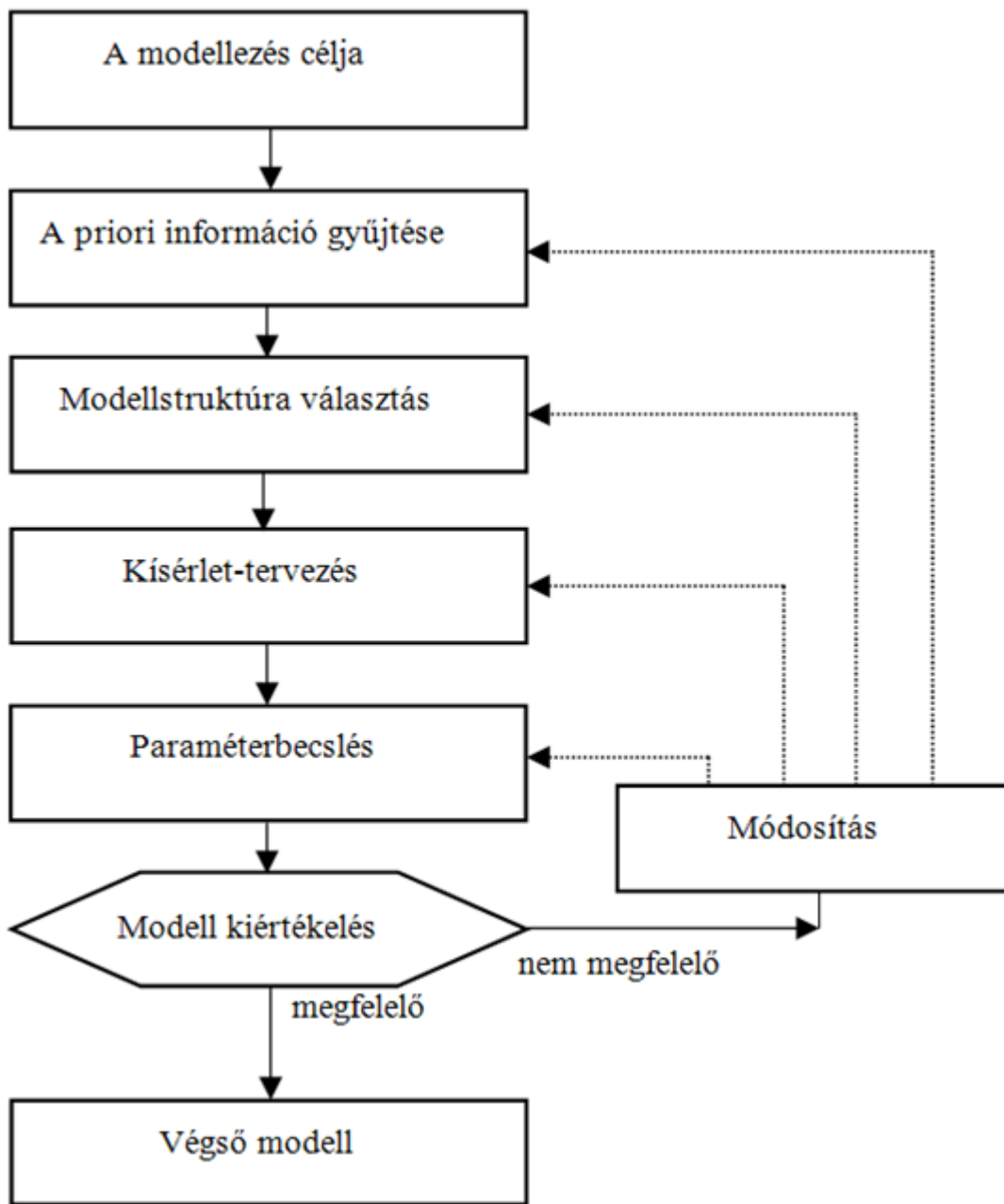
Az identifikáció, mint alapeladat célja létrehozni a vizsgált rendszer egy modelljét, amely adott feltételek mellett, adott pontossággal közelíti az eredeti rendszer viselkedését.

Egy rendszermodellezési feladat megoldásánál a következő főbb lépések különböztethetők meg:

- a modellezés céljának meghatározása,
- kiinduló a priori információ begyűjtése a modellezendő rendszerről,
- a modellstruktúra megválasztása,
- kísérlet-tervezés és adatgyűjtés,
- modell paraméterbecslés,
- a modell kiértékelése.

A 8.16 ábra egyrészt a fenti lépések kapcsolatát, másrészt azt mutatja, hogy a modellezés iteratív folyamat.

8.16. ábra - A rendszermodellezés folyamata



A struktúra rögzítését követően, a kísérletek eredményeképpen kapott további ismeretek felhasználásával tudjuk a modell szabad paramétereinek értékét meghatározni, a paraméterbecslési feladatot megoldani. Az így kapott modell kiértékelése alapján dönthetünk, hogy a modellünk a kitűzött célnak megfelel-e vagy finomítani, pontosítani kell. Ez utóbbi esetben a modellezési folyamat valamely korábbi lépéséhez vissza kell térni. Ez jelentheti a modell struktúrájának módosítását, további ismeret begyűjtését – melyhez akár új kísérlettervezésre is szükség lehet –, vagy a paraméterbecslési eljárás módosítását.

Azzal, hogy neuronhálós modellt építünk a fenti kérdések egy részét eleve eldöntöttük, hiszen a neuronhálós modell struktúráját a neuronháló típusának meghatározásával részben rögzítjük is. Neuronhálós modellezésnél azt azonban figyelembe kell venni, hogy a modell struktúráját döntően a modellező eszköz és nem a modellezendő rendszer szabja meg. Ezért strukturális modell létrehozására – ahol cél, hogy a modell struktúrája minél inkább hasonló legyen a modellezendő rendszer struktúrájához – lényegében nincs lehetőség; a neuronhálós modell viselkedési modell. A viselkedési modell a modellezendő rendszer bemeneti-kimeneti viselkedését tudja minél pontosabban leírni anélkül, hogy a modell struktúrája akárcsak hasonlítana is a modellezendő rendszer struktúrájához. Ez azt jelenti, hogy neuronhálós modellezésnél még akkor sem tudunk

strukturális modellt létrehozni, ha egyébként az ehhez szükséges ismeret birtokában vagyunk. A neuronhálós modellek fekete doboz (*black box*) modellek, melyek csupán kívülről nézve jellemzik a modellezendő rendszer viselkedését.

6.1.1. Struktúraválasztás dinamikus neurális hálózatoknál

Neuronhálós modellezésnél a modellstruktúra választás alapvetően arra a kérdésre ad választ, hogy a lehetséges általános struktúrák melyikét alkalmazzuk. A struktúraválasztás első kérdése, hogy a modellezendő rendszer statikus vagy dinamikus modellel írható-e le kellő pontossággal.

Amennyiben statikus rendszermodellezési feladattal állunk szemben, az előző fejezetekben bemutatott neuronháló típusok közül választhatunk. A struktúraválasztás a típusválasztáson túl a háló méretének – pl. egy MLP rejtett rétegében a neuronok számának – a megválasztását is jelenti. Az ezzel kapcsolatos nehézségekkel és lehetőségekkel az előző fejezetekben foglalkoztunk.

Dinamikus feladatoknál a struktúraválasztás jóval nehezebb: egy dinamikus neurális hálózat által biztosított $y(k) = f(\Theta, \varphi(k))$ leképezésben mind a regresszorvektort, mind a dinamikus háló komplexitását (pl. a rejtett rétegek számát, a rejtett réteg(ek) méretét vagy a bázisfüggvények számát, stb.) meg kell határozni. A regresszorvektor-választás egyfelől a nemlineáris dinamikus modellosztály megválasztását jelenti, másfelől a modell-fokszám meghatározását is igényli.

A modellosztály megválasztása elsődlegesen a priori információ alapján lehetséges. Amennyiben van olyan előzetes ismeretünk, amely alapján kiválasztható a modellosztály, valójában már nem fekete doboz, hanem sokkal inkább szürke doboz (*grey box*) modellezéssel van dolgunk. Megfelelő a priori információ nélkül azt az általános elvet követhetjük, hogy mindig a lehető legegyszerűbb, a feladatot még megfelelő minőségben megoldó modellosztályt célszerű választanunk. Ez azt jelenti, hogy amennyiben dinamikus nemlineáris modellre van szükségünk, célszerű az NFIR struktúrával kezdeni és csak akkor áttérni komplexebb modellosztályra, ha az egyszerűbbel a feladat megfelelő minőségben nem oldható meg.

A fejezet elején áttekintett modellosztályok közül az NFIR és a NARX struktúrák kitüntetettek, mivel ezek előrecsatolt architektúrák, így alkalmazásukkal a visszacsatolt struktúráknál jelentkező stabilitási kérdések nem merülnek fel. A visszacsatolt nemlineáris rendszerek stabilitási kérdéseivel a könyvben nem foglalkozunk. Az érdeklődő olvasó az irodalomban (pl. [Nar05], [Suy99], [Hay99], [Gup03], stb.) tájékozódhat.

A modellosztály megválasztását követően a struktúra meghatározása a modell-fokszám meghatározását igényli. A bemenet-kimenet reprezentációk mellett a modell-fokszám a regresszorvektor konstrukciójánál figyelembe vett régebbi bemeneti és/vagy kimeneti, stb. értékek számát jelenti. Mivel a modell-fokszám előzetesen általában nem ismert, célszerű különböző modell-fokszámok mellett különböző komplexitású modellek létrehozása és valamilyen kritérium szerinti kiértékelése. A kérdés egy lehetséges intuitív megközelítése, ha egyre növekvő komplexitású modelleket hozunk létre és értékelünk ki, például az átlagos négyzetes hiba kritérium alapján. Már a statikus hálókban láttuk azonban, hogy a modell komplexitásának növelésével monoton módon csökkenthető (a tanítópontokban) a modellezés hibája, miközben az általánosítóképesség romolhat. A túl nagy komplexitás dinamikus hálókban is a tanítópontokhoz való túlzott illeszkedést eredményezheti, miközben az általánosítóképességet ronthatja.

A tanítópontokban meghatározott hiba önmagában nem elegendő a megfelelő modellkomplexitás meghatározására. Ehhez olyan kritériumfüggvények alkalmazása szükséges, melyek a tanítópontokban meghatározható modellezési hiba mellett a modell komplexitását is figyelembe veszik: a modell komplexitást büntetik, elkerülve így a túl komplex modellek alkalmazását.

Számos olyan kritériumot definiáltak, melyek ezt az elvet követik. Ezek között a legfontosabbak a különböző információs kritériumok, elsősorban az Akaike információs kritérium (*Akaike information criterion, AIC*) [Aka74], valamint a végső előrejelzési hiba (*final prediction error, FPE*) [Aka69] vagy a minimális leíró hossz (*minimum description length, MDL*) [Ris78]. Ezeket a kritériumokat elsődlegesen lineáris rendszerek komplexitásának meghatározására dolgozták ki, de korlátozottan alkalmazták őket nemlineáris dinamikus rendszereknél, így dinamikus neuronhálókban is (ld. pl. [Nør00]). Ezen kritériumok közös jellemzője, hogy két tagból állnak: a kritérium első tagja a tanítópontokbeli hibát veszi számba (tapasztalati hiba), míg a második tag egy büntető tag, melynek értéke annál nagyobb, minél nagyobb komplexitású a modell. Az általános forma:

Kritérium = tapasztalati hiba + modell-komplexitást büntető tag

Vegyük észre, hogy ez az általános forma rokon a 2. fejezetben bemutatott strukturális kockázat minimalizálási (SRM) elvvel alkalmazott általános formával. Lényeges különbség azonban, hogy míg az SRM elvhez kapcsolódó felső korlátok (pl. (2.31) összefüggés) a komplexitás mértékeként a VC-dimenziót használják, addig itt a komplexitást a modellstruktúra szabad paramétereinek a száma fejezi ki.

Az előbbi kritériumok alkalmazása olyan modell kialakítására próbál kényszeríteni, ahol nemcsak a hibát, hanem a komplexitást is minimalizáljuk. Így valójában itt is az ún. Ockham borotvája (*Ockham's razor*) elvet alkalmazzuk, amely azt mondja ki, hogy két azonos minőségű megoldás közül mindig az egyszerűbbet válasszuk.²

Ezek a kritériumokon kívül kidolgoztak más eljárásokat is a modell-fokszám meghatározására. Dinamikus neurális hálók fokszám-meghatározására született a Lipschitz indexen alapuló heurisztikus eljárás [He93], amelyet a következőkben foglalunk össze. A fokszám meghatározás valójában azt adja meg, hogy a regresszorvektorban (ld. (8.3)-(8.7) összefüggések) hány komponens szerepel és melyek ezek a komponensek. Egy nemlineáris dinamikus rendszermodell (8.1) szerint a regresszorvektor (nemlineáris) függvénye. Ebben a felfogásban a fokszám azt adja meg, hogy ennek a függvénynek hány bemeneti változója van. A Lipschitz indexen alapuló eljárás a bemeneti változók megfelelő számát és a megfelelő bemeneti változókat próbálja meghatározni.

6.1.1.1. Lipschitz index

Az eljárás azon alapul, hogy feltételezhetjük a nemlineáris dinamikus rendszer által megvalósított leképezés folytonosságát. A módszer érdekessége és a gyakorlati alkalmazás szempontjából az előnye, hogy kizárólag a tanító adatokból dolgozik.

A módszer bemutatásához induljunk ki abból, hogy adott egy nemlineáris bemeneti-kimeneti leképezés:

$$y = f(\mathbf{x}) = f(x_1, x_2, \dots, x_N), \quad (8.65)$$

amely egy folytonos, sima, többváltozós függvény valamilyen értelmezési tartományon, és ahol az $x_1, x_2, \dots, x_N$ változók a függvény bemeneti változói. Az f függvény az $x_1, x_2, \dots, x_N$ független változók mindegyikétől, de csak ezekről a független változóktól függ. A megfelelő számú bemeneti változó meghatározása a következő, Lipschitz indexnek nevezett mennyiség alapján lehetséges:

$$L_{q^{(w)}} = \left(\prod_{i=1}^P \sqrt{N} \cdot q^{(w)}(k_i) \right)^{1/P} \quad (8.66)$$

ahol $q^{(w)}(k_i)$ a

$$q_y^{(w)} = \frac{|y(i) - y(j)|}{\|\mathbf{x}(i) - \mathbf{x}(j)\|} \quad i \neq j; \quad i, j = 1, 2, \dots, P \quad (8.67)$$

Lipschitz hányadosok közül a k -adik legnagyobb érték, N a bemeneti változók száma (a regresszorvektor dimenziója), p pedig egy alkalmasan megválasztott pozitív szám, rendszerint $p = 0,01P \sim 0,02P$. (P a szokásos jelölésnek megfelelően a tanítópontok száma).

Tételezzük föl, hogy az $f(\mathbf{x})$ függvény bemeneti változók szerinti deriváltjai korlátosak, vagyis

$$\left| f'_i \right| = \left| \frac{\partial f(\mathbf{x})}{\partial x_i} \right| \leq M \quad i = 1, 2, \dots, N, \quad (8.68)$$

ahol M egy véges pozitív érték.

Ha a függvény folytonos, a Lipschitz hányados korlátos kell legyen, vagyis

$$0 \leq q_y^{(w)} \leq L \quad (8.69)$$

² Az Ockham borotvája elvet William of Ockham (vagy Occam) 14. századi angol filozófusnak tulajdonítják

bármely i, j adatpárra. A sok adatpár közül azokat vizsgáljuk, ahol a Lipschitz hányados értéke nagy.

Határozzuk meg a Lipschitz hányados felső korlátját. A függvénykapcsolat érzékenységi vizsgálata alapján a kimenet megváltozása a bemenetek megváltozásai alapján az alábbi formában írható fel:

$$\begin{aligned}\Delta y &= \frac{\partial f}{\partial x_1} \Delta x_1 + \frac{\partial f}{\partial x_2} \Delta x_2 + \dots + \frac{\partial f}{\partial x_n} \Delta x_n \\ &= f'_1 \Delta x_1 + f'_2 \Delta x_2 + \dots + f'_n \Delta x_n\end{aligned}\quad (8.70)$$

ahol f'_i az $f(\mathbf{x})$ függvény x_i szerinti parciális deriváltja, $\Delta x_l = x_l(i) - x_l(j)$ $l=1, \dots, N$ és $\Delta y = y(i) - y(j)$. Felhasználva a deriváltra vonatkozó (8.68) felső korlátot és a Schwarz egyenlőtlenséget, továbbá, ha a bemeneti térben az euklideszi normát használjuk, a Lipschitz hányadosra az alábbi felső korlátot kapjuk:

$$\begin{aligned}q_y^{(n)} &= \frac{|\Delta y|}{\sqrt{(\Delta x_1)^2 + (\Delta x_2)^2 + \dots + (\Delta x_n)^2}} \\ &= \frac{|f'_1 \Delta x_1 + f'_2 \Delta x_2 + \dots + f'_n \Delta x_n|}{\sqrt{(\Delta x_1)^2 + (\Delta x_2)^2 + \dots + (\Delta x_n)^2}} \\ &\leq \frac{|\Delta x_1 + \Delta x_2 + \dots + \Delta x_n|}{\sqrt{(\Delta x_1)^2 + (\Delta x_2)^2 + \dots + (\Delta x_n)^2}} M \leq \sqrt{N} M\end{aligned}\quad (8.71)$$

Az $f(\mathbf{x})$ függvény rekonstrukcióját akarjuk mintapontok alapján megkapni. Ha a függvény valóban kielégíti a (8.68) összefüggést, és az összes bemeneti változó szerepel az $f(\mathbf{x})$ függvény rekonstrukciójában, akkor az összes mintapárra számított Lipschitz hányadosra érvényes kell legyen a (8.71) felső korlát.

Tételezzük most fel, hogy valamelyik bemeneti változó, pl. x_N hiányzik a függvény rekonstrukciójánál, vagyis a mintapontoknál az x_N komponens hiányzik. Ekkor két bemeneti pont közötti távolságra $\sqrt{(\Delta x_1)^2 + (\Delta x_2)^2 + \dots + (\Delta x_{n-1})^2}$ adódik, miközben a megfelelő kimenet-megváltozás változatlan marad, vagyis a Lipschitz hányados

$$\begin{aligned}q_y^{(n-1)} &= \frac{|\Delta y|}{\sqrt{(\Delta x_1)^2 + (\Delta x_2)^2 + \dots + (\Delta x_{n-1})^2}} \\ &= \frac{|f'_1 \Delta x_1 + f'_2 \Delta x_2 + \dots + f'_n \Delta x_n|}{\sqrt{(\Delta x_1)^2 + (\Delta x_2)^2 + \dots + (\Delta x_{n-1})^2}}\end{aligned}\quad (8.72)$$

lesz. Ha x_N független a többi bemeneti változótól, $q_y^{(n-1)}$ extrém nagy, sőt végtelen értéket is felvehet. Ilyen eset állhat elő, ha $\Delta x_1 = \Delta x_2 = \dots = \Delta x_{n-1} = 0$, és Δx_n tetszőlegesen kicsi, de nem nulla. Ha x_N nem teljesen független az $x_1, x_2, \dots, x_{n-1}$ bemeneti változóktól (ilyen helyzet lehet pl. ha $x_1, x_2, \dots, x_n$ egy jel késleltetett értékei), akkor $q_y^{(n-1)}$ sokkal nagyobb lesz, mintha a bemeneti változók között x_N is szerepelne. Ebből a vizsgálatból az következik, hogy ha egy szükséges bemeneti változó hiányzik a bemeneti vektorból, a Lipschitz hányados nagyon nagy értéket is felvehet. Minél több változó hiányzik, annál nagyobb lehet a Lipschitz hányados. Ha ez a helyzet áll elő, semmilyen nemlineáris approximációs eszköz sem lesz képes megtanulni a leképezést kielégítő minőségben.

Ha a bemenetek közé egy redundáns változót, pl. x_{N+1} -et is bevesszük a Lipschitz hányadosra a következő adódik:

$$\begin{aligned}q_y^{(n+1)} &= \frac{|\Delta y|}{\sqrt{(\Delta x_1)^2 + (\Delta x_2)^2 + \dots + (\Delta x_{n+1})^2}} \\ &= \frac{|f'_1 \Delta x_1 + f'_2 \Delta x_2 + \dots + f'_n \Delta x_n|}{\sqrt{(\Delta x_1)^2 + (\Delta x_2)^2 + \dots + (\Delta x_{n+1})^2}}.\end{aligned}\quad (8.73)$$

$q^{(n+1)}$ csak kismértékben lehet kisebb, mint $q^{(n)}$, vagyis $q^{(n+1)} \approx q^{(n)}$. Ha további redundáns változók kerülnek a bemeneti vektorba a Lipschitz hányadosok jelentősen nem fognak változni. Egy bemeneti változó az $f(\mathbf{x})$ függvénykapcsolat szempontjából akkor redundáns, ha ettől a változótól – jelen esetben x_{N+1} -től – a függvényérték, y nem függ.

Fentie $L_q^{(N+1)}$ -ján látható, hogy $L_q^{(N)}$ Lipschitz index tulajdonságai $L_q^{(N-1)}$ -gyel ha N a $L_q^{(N-1)}$ -eti változók megfelelő $L_q^{(N-2)}$, akkor $L_q^{(N+1)}$ -el azonos $L_q^{(N+1)}$ jóval kisebb, mint $L_q^{(N)}$. Továbbá $L_q^{(N+1)}$ is sokkal kisebb $L_q^{(N)}$ -int, miközben $L_q^{(N+1)}$ közel azonos $L_q^{(N)}$ -gyel és így tovább. Ebből adódóan, ha felrajzoljuk az $L_q^{(N)}$ Lipschitz index alakulását N függvényében, azt figyelhetjük meg, hogy az $L_q^{(N)}$ értéktől kezdve N -tól egyre kevésbé fog függni, viszont N_0 -nál kisebb számú bemeneti változó esetén $L_q^{(N)}$ növelésével meredeken csökken. Ez az N_0 érték lesz a bemeneti változók megfelelő száma.

A 8.17 (a) ábra jól mutatja, hogy $N_0=3$ -nál a Lipschitz index minimumot vesz fel. Az ábra azt az ideális helyzetet tükrözi, amikor a megfigyeléseink zajmentesek. A görbe közel hasonlóan alakul akkor is, ha a mintapontok kismértékben zajosak (8.17 (b) ábra). Ha a zajszint növekszik a görbén látható határozott töréspont egyre inkább elmosódik. Erre mutat példát a 8.17 (d) ábra.

A (8.66) összefüggésből látható, hogy a Lipschitz index a $q^{(N)}(1), q^{(N)}(2), \dots, q^{(N)}(p)$ Lipschitz hányados-sorozat $\sqrt{N}$ -nel szorzott értékeinek mértani középértéke. A mértani közép alkalmazásának oka, hogy így az index alakulása kevésbé érzékeny a tanító adatokat terhelő mérési zajra.

Ha a fenti eredményeket a fokszámmeghatározásra alkalmazzuk, akkor valójában azt határozzuk meg, hogy a regresszorvektorban hány komponensnek kell szerepelnie (és melyek ezek a komponensek). A bemeneti komponensek most a dinamikus modellstruktúra osztály függvényében a rendszer régebbi bemeneti (NFIR struktúra), régebbi bemenetei és a régebbi rendszerkimenetek (NARX struktúra), stb. lesznek.

NFIR modellstruktúra esetén az eljárás a következő: Az $L_q^{(N)}$ Lipschitz indexet több különböző modell-fokszám érték mellett kell meghatározni, ahol N jelöli az előrecsatoló ág fokszámát. $N=1$ -gyel kezdve csak $x(k-1)$ -et vesszük be a regresszorvektorba. Ezzel számíthatjuk ki $L_q^{(1)}$ értékét. $N=2$ esetén $x(k-1)$ mellett a regresszorvektor már $x(k-2)$ -t is tartalmazza, így számítható $L_q^{(2)}$. Ha $L_q^{(2)}$ közel azonos $L_q^{(1)}$ -gyel, $x(k-2)$ -re nincs szükség. Ezzel szemben, ha $L_q^{(2)}$ jelentősen kisebb, mint $L_q^{(1)}$, $x(k-2)$ része marad a regresszorvektornak és a fokszámnövelést innen folytatjuk $x(k-3)$ -mal. Ezt a stratégiát folytatva lépésről-lépésre növeljük az előrecsatoló ág késleltetéseinek számát, amíg a megfelelő fokszámot el nem érjük.

A Lipschitz index nemcsak NFIR esetben, hanem NARX hálózatoknál is alkalmazható, ahol két fokszámérték meghatározása a cél. A NARX modelleknél az előrecsatoló ágban lévő késleltetések száma mellett a visszacsatoló ág késleltetett értékeinek számát is meg kell határozni.

NARX modell esetén a következő stratégia szerint járhatunk el. Az $L_q^{(R)} = L_q^{(N+R)}$ Lipschitz indexet több különböző modell-fokszám érték mellett kell meghatározni, ahol R az eredő fokszám és ezen belül M jelöli a visszacsatoló ág, N pedig az előrecsatoló ág fokszámát. Ha $R=1$, akkor csak $d(k-1)$ -et vesszük be a regresszorvektorba. Ezzel számíthatjuk ki $L_q^{(1+0)}$ értékét. $R=2$ esetén $d(k-1)$ mellett már $x(k-1)$ -et is szerepeltetjük a regresszorvektorban, így számítható $L_q^{(1+1)}$. $R=3$ mellett $d(k-2)$ is felhasználásra kerül, vagyis $L_q^{(1+2)}$ számítható. Ha azonban $L_q^{(1+2)}$ közel azonos $L_q^{(1+1)}$ -gyel, a visszacsatoló ágban nincs szükség két késleltetésre. Helyette az $R=3$ -hoz a bemeneti ágban kell egy újabb késleltetett értéket, $x(k-2)$ -t felhasználni és $L_q^{(1+2)}$ -t kell meghatározni. Ezzel szemben, ha $L_q^{(1+2)}$ jelentősen kisebb, mint $L_q^{(1+1)}$, $d(k-2)$ része marad a regresszorvektornak és a fokszámnövelést innen folytatjuk $x(k-2)$ -vel. Így juthatunk el $L_q^{(1+2)}$ -höz, amit ismét össze kell hasonlítanunk $L_q^{(1+1)}$ -gyel. A továbblépés az összehasonlítás eredményétől függően vagy az $(M,N)=(2,2)$ struktúra megtartását jelenti, vagy vissza kell térni az $(M,N)=(2,1)$ felépítéshez, ahonnan a továbblépés ismét M növelésével az $L_q^{(1+2)}$ index meghatározásához vezet. Ezt a stratégiát folytatva lépésről-lépésre növeljük a visszacsatoló és az előrecsatoló ág késleltetéseinek számát, amíg a megfelelő fokszámot el nem érjük.

A Lipschitz index alapú eljárás előnye, hogy közvetlenül a mintapontokból dolgozik, anélkül, hogy különböző komplexitású modellek előzetes megépítésére szükség lenne. Hátránya a viszonylag nagyfokú zajérzékenység.

8.2 példa

Példaként határozzuk meg a Mackey-Glass kaotikus folyamatmodellezésénél a modell-fokszámot. A Mackey-Glass kaotikus folyamatot a

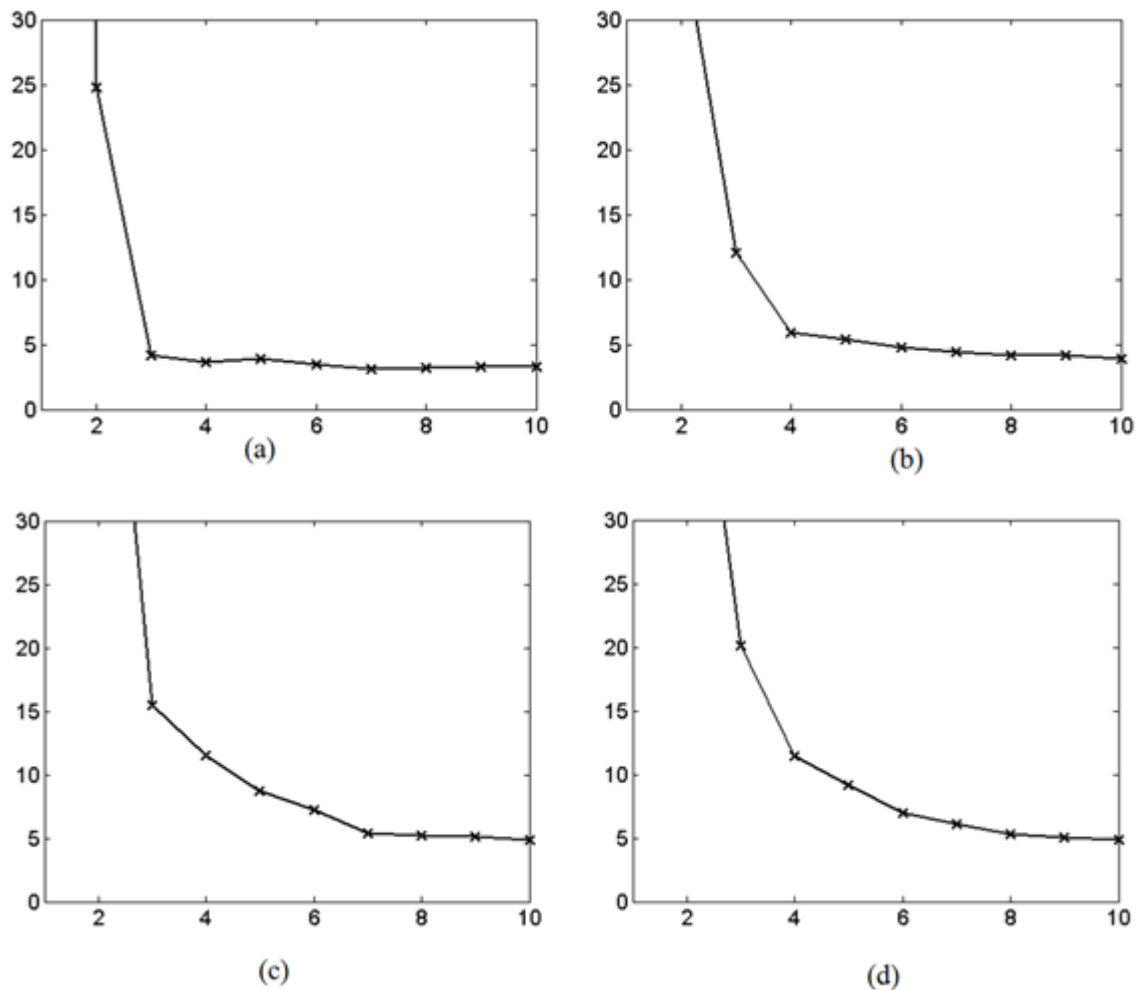
$$\frac{dx(t)}{dt} = -0,1x(t) + \frac{0,2x(t-\tau)}{1+x^{10}(t-\tau)} \quad (8.74)$$

differentiálegyenlet írja le. (A felhasznált idősort a differenciálegyenletből $\tau=17$ mellett negyedrendű Runge-Kutta integrálással és (6:1)-es alulmintavételezéssel kaphatjuk. Az az így kapott idősort gyakran alkalmazzák különböző dinamikus modellezési megközelítések vizsgálatánál mintapéldaként, mint *benchmark* feladatot.) A Mackey-Glass folyamatból kapott idősor egy olyan kaotikus idősort definiál, ahol a folyamat korábbi értékei alapján határozható meg egy későbbi érték.

$$\varphi(k) = [d(k-1), \dots, d(k-M)] \quad (8.75)$$

A folyamat modellezésére tehát egy olyan visszacsatolt rendszer lehet alkalmas, ahol külső gerjesztés nincs. NARX modellt alkalmazva a folyamat mintavételi értékei lesznek a kívánt $d(k)$ válaszok, vagyis a regresszorvektor (8.75) alakú lesz és a modell-fokszám meghatározása M értékének meghatározását jelenti. $P=1000$ mintapont alapján, továbbá, ha a Lipschitz index (8.66) összefüggésében $p=10$, a 8.17 ábrán látható eredményt kapjuk. Az ábra jól mutatja, hogy a zaj növelése milyen mértékben teszi bizonytalanná a Lipschitz index alkalmazását a fokszám meghatározására. Az ábrán bemutatott zajos esetekben a mintapontokat egyre nagyobb szórású additív Gauss zaj terheli.

8.17. ábra - A Lipschitz index alakulása a fokszám függvényében Mackey-Glass folyamat esetén zajmentes (a) és zajos (b-d) esetekben. A Gauss zaj szórása: (b) $\sigma=0,05$; (c) $\sigma=0,1$; (d) $\sigma=0,3$.



6.1.1.2. További lehetőségek a modell komplexitás meghatározására

A fenti eljárásokon túl a gyakorlatban számos további módszer alkalmazható a megfelelő modellkomplexitás meghatározására. A keresztkiértékelés (*cross validation*) szerepét a statikus neuronhálók minősítésénél már említettük. A módszer dinamikus hálónál szintén alkalmazható, amikor a létrehozott modellt olyan mintákkal teszteljük, melyeket a modell kialakításánál nem használtunk. A valóságos, megfigyelt adatok és a modell kimenete közötti különbség – a maradék hiba – vizsgálata alapján értékelhető a modell minősége. A keresztkiértékelés eredménye tehát nem közvetlenül a foks szám megfelelő megválasztására utal, hanem a teljes megoldást minősíti, de a minősítés eredményét felhasználva meghatározható a megfelelő modell-foks szám is.

A maradék hiba vizsgálata (*residual test*) alapján eldönthető, hogy a modell kellően pontosan leírja-e a modellezendő rendszert. A modell akkor tekinthető megfelelőnek, ha az

$$\varepsilon(k) = d(k) - y(k) \quad (8.76)$$

maradék hiba „struktúramentes”, semmiféle „mintázatot” nem mutat, sem a bemenettel sem a kimenettel nem korrelált.

Precízebben megfogalmazva, az $\{\varepsilon(k)\}$ hibaminta sorozat

- nulla középértékű, diszkrét fehér zaj folyamat,
- Gauss eloszlású folyamat,
- a hibasorozat elemei és az előző bemeneti értékek korrelálatlanok:

$$E\{\varepsilon(k)x(j)\} = 0; \quad k > j$$

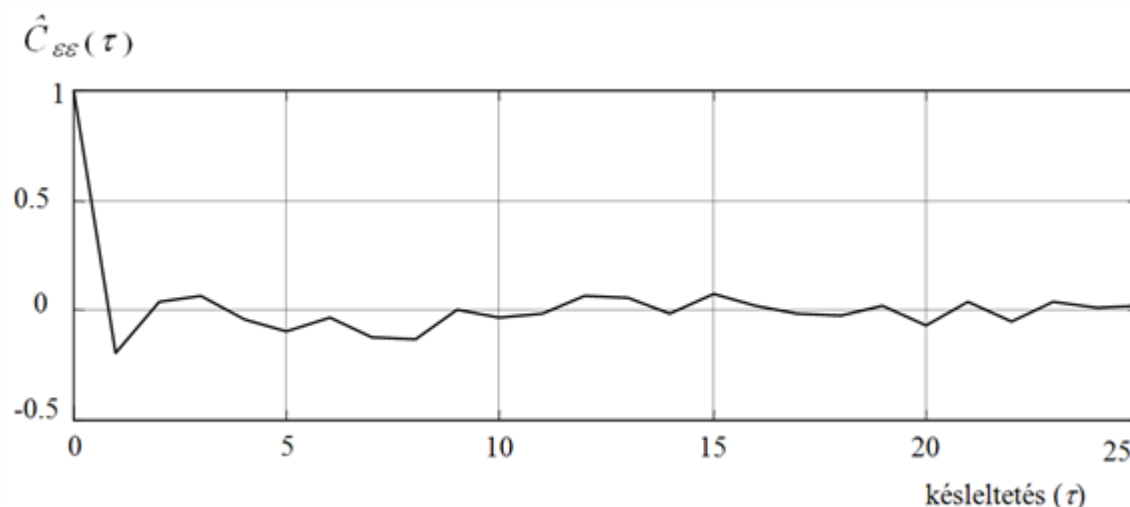
A maradék hiba vizsgálatára szolgál az autókorrelációs és a keresztkorrelációs teszt, melynek során a maradék hiba autókorrelációs illetve a hibának és a bemenetnek a keresztkorrelációs függvényét becsüljük.

Az autókorrelációs függvényt véges számú mintapont alapján az alábbi módon becsülhetjük:

$$\hat{C}_{\varepsilon\varepsilon}(\tau) = \frac{1}{N-\tau} \sum_{k=\tau+1}^N \varepsilon(k)\varepsilon(k-\tau) \quad (8.77)$$

Minél inkább fehér zaj a maradék hiba (vagyis minél inkább „struktúramentes”), annál inkább az impulzusfüggvényhöz tart az autókorrelációs függvény. A 8.18 ábra egy tipikus példát mutat a maradék hiba autókorrelációs függvényére.

8.18. ábra - A maradék hiba autókorrelációs függvénye

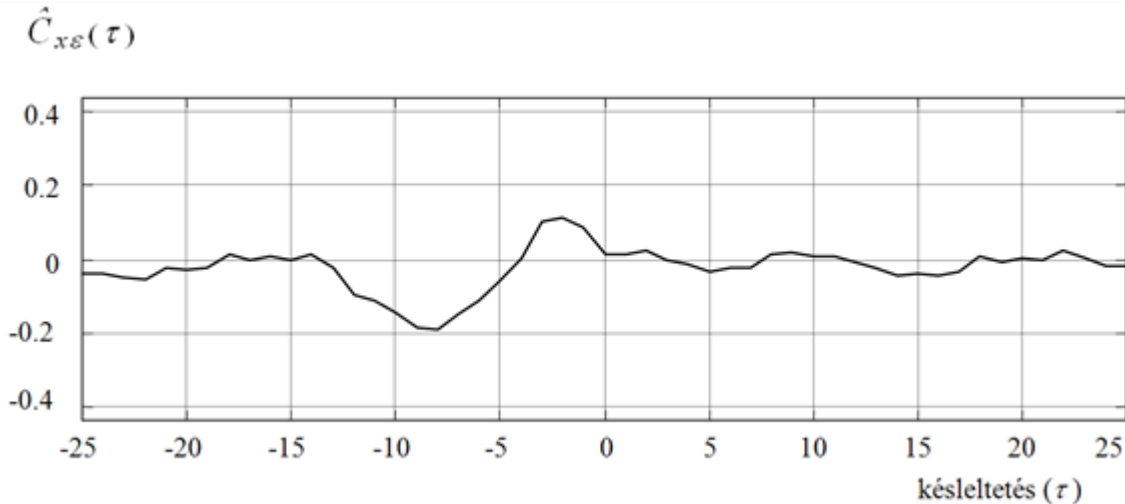


A keresztkorreláció a maradék hiba és a bemenet közötti kapcsolat vizsgálatára szolgál. A keresztkorrelációs függvény az alábbi összefüggés szerint becsülhető:

$$\hat{C}_{\varepsilon\varepsilon}(\tau) = \frac{1}{N-\tau} \sum_{k=-\tau}^N \varepsilon(k)\varepsilon(k-\tau) \quad (8.78)$$

Megfelelő pontosságú modell esetén a maradék hiba és a bemenet korrelálatlan, vagyis a keresztkorrelációs függvény értéke közel nulla bármilyen τ késleltetés mellett is. Egy tipikus keresztkorrelációs kapcsolatot mutat a 8.19 ábra.

8.19. ábra - A maradék hiba és a bemenet keresztkorrelációs függvénye



6.1.2. Kísérlettervezés

A kísérlettervezés célja olyan feltételeket teremteni, melyek mellett kellő számú és a működést kellő mértékben reprezentáló minta begyűjtése lehetséges a modellezendő rendszerről. A feladatok egy jelentős részében a kísérlettervező kezében van, hogy a modellezendő rendszer milyen gerjesztő jelet kapjon, vagyis a bemeneti jelet a kísérlettervező határozhatja meg. Más feladatoknál a tanítás céljára történő adatok gyűjtésénél a bemenet megtervezésére nincs lehetőség, azt a modellezendő rendszer normál működése megszabja.

A megfelelő gerjesztőjel, az ún. perzisztens gerjesztés (*persistent excitation*) megtervezése önálló tudomány. A gerjesztőjel megválasztásánál, még lineáris rendszereknél is sokféle szempont figyelembevétele szükséges [Nar89] [Lju99], [Pin01]. A gerjesztőjel megtervezésekor kell döntenünk, hogy a standard vizsgálójelek (impulzus, egységugrás, fehérzaj, multiszínusz, bináris álvéletlen jel, stb.) közül melyiket vagy melyeket alkalmazzuk. Nemlineáris rendszereknél a helyzet még bonyolultabb, hiszen nemlineáris rendszereknél az átvitel akár a jelszintnek is a függvénye. Rosszul megválasztott jelszintekkel létrehozott tanító készlet alapján konstruált dinamikus neurális modell más jelszinteknél jelentős hibával írja le a rendszer működését. Ezt illusztrálja a következő példa.

8.3 példa

Legyen a modellezendő rendszer a következő:

$$d(k+1) = 0,3d(k) + 0,6d(k-1) + f[x(k)] \quad (8.79)$$

ahol

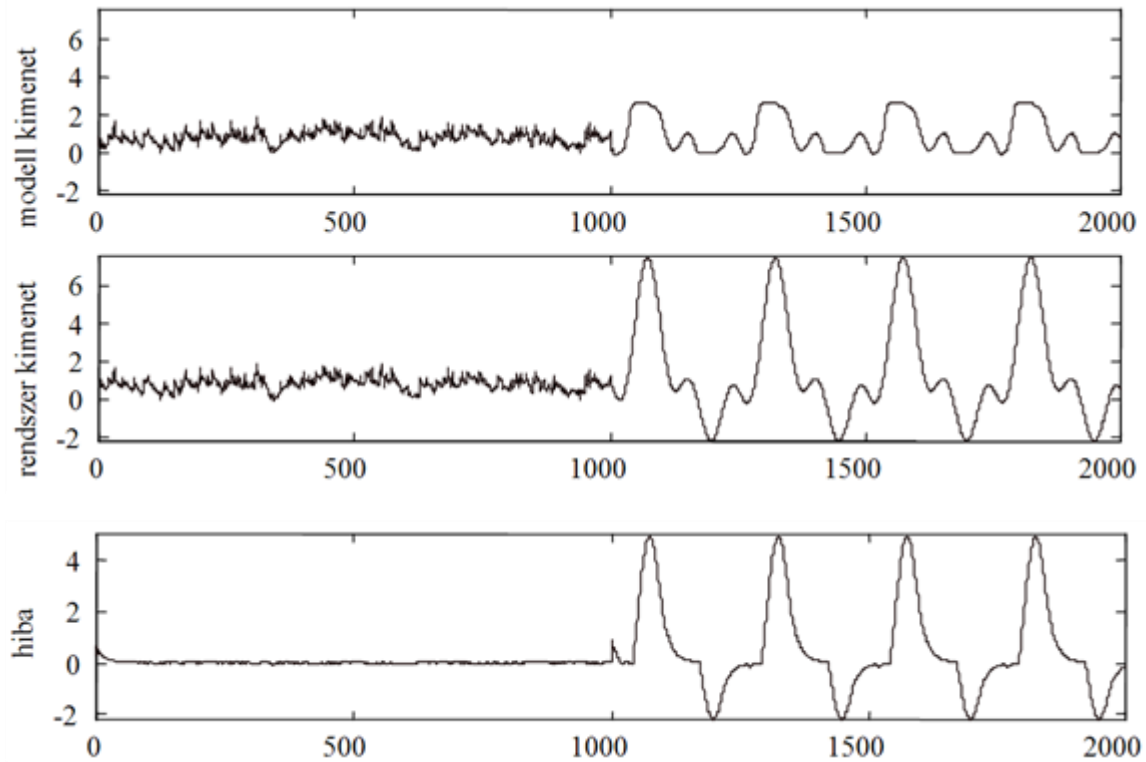
$$f(x) = x^3 + 0,3x^2 - 0,4x \quad (8.80)$$

A feladat NARX vagy NOE modellstruktúrával oldható meg. A tanításnál alkalmazható a NARX struktúra, míg a tesztelésnél már a modell saját kimenetét csatoljuk vissza, vagyis a tesztelés NOE architektúra mellett történik. A neurális modell létrehozásához egyenletes eloszlású véletlen bemenőjelet használva a 8.20 és a 8.21 ábrák a bemenő jelszint helyes megválasztásának fontosságára hívják fel a figyelmet.

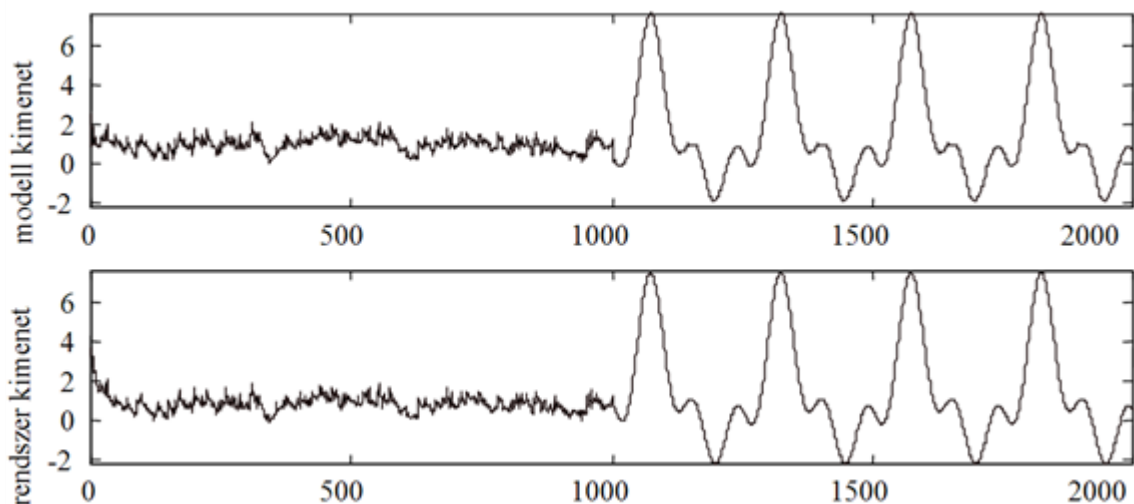
A két ábra-együttes a megtanított háló választát, a kívánt választ (a rendszer kimenete) és a maradék hibát mutatja olyan teszt mintasorozatra, ahol a tesztminta első 1000 pontja kis amplitúdójú véletlen jel, a második

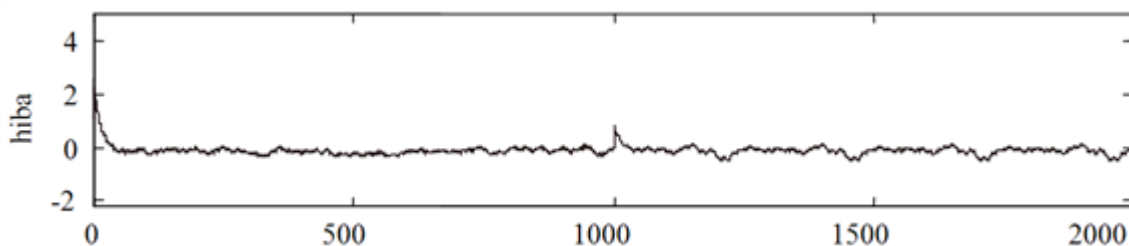
1000 minta nagyobb amplitúdójú szinuszos jel volt. A 8.20 ábra esetében a dinamikus hálót kis amplitúdójú, a $[-1,1]$ intervallumban egyenletes eloszlású véletlen bemenőjel mellett létrehozott tanítómintákkal tanítottuk, míg a 8.21 ábrán a gerjesztő véletlen jel nagyobb amplitúdójú volt (egyenletes eloszlású jel a $[-2,2]$ intervallumban). A teszt válaszokból jól látható, hogy az első esetben a háló csak kis amplitúdójú teszt bemenőjel mellett adott megfelelően jó választ, míg a második esetben a válasz a teszt mindkét részén, mind a kisszintű, mind a nagyobb szintű teszt bemenőjel mellett megfelelő.

8.20. ábra - Kiszintű jellel tanított háló válasza



8.21. ábra - Nagyobb szintű jellel tanított háló válasza





6.1.3. A modellezés további lépései

A megfelelő modellstruktúra és tanítókészlet megválasztását követően kerülhet sor a dinamikus háló paramétereinek meghatározására, a paraméterek becslésére – ami a háló tanítását jelenti –, majd ezt követi az eredmény kiértékelése. Ezenél a lépéseknél egyrészt az egyes dinamikus neuronháló architektúráknál bemutatott tanító eljárások általános és speciális szempontjait, illetve a hálók tesztelésével kapcsolatos általános megállapításokat és a maradék hiba elemzésére szolgáló eljárásokat kell figyelembe venni.

A fentiekből látható, hogy a tanítókészlet nemcsak a tanítási folyamatot befolyásolja, hanem a modellstruktúra kialakításánál is szerepe van, továbbá meghatározza azt is, hogy mi lesz a modell érvényességi tartománya. Ez is azt mutatja, hogy a modellezési folyamat egyes lépései szorosan összefüggnek egymással és a megfelelő modell kialakítása csak visszalépéseket is tartalmazó iteratív folyamat eredménye lehet.

6.2. Szabályozás, modell alapú adaptív szabályozás

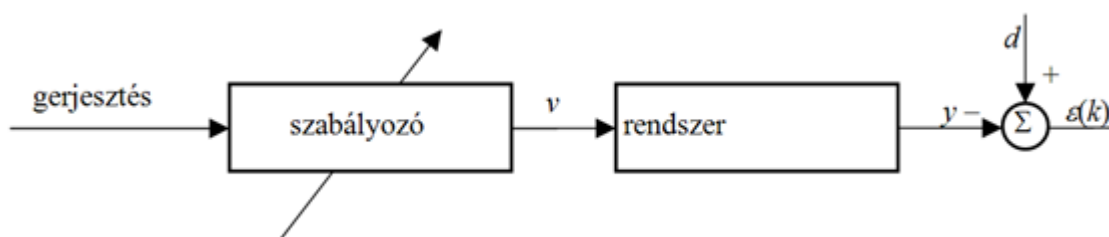
Egyes feladatoknál a rendszeridentifikáció elsődleges célja egy vizsgált rendszer modelljének a létrehozása, más esetekben az identifikáció egy összetettebb feladat részeként jelenik meg. Az első esetre példaként hozható szinte bármely mérési feladat, mivel minden mérés a rendszer valamely modell-paraméterének a meghatározását (becslését) jelenti. Az identifikáció, mint részfeladat leggyakrabban szabályozási feladatokban jelentkezik. Ekkor a rendszer modelljére pl. a szabályozó algoritmus paramétereinek meghatározásához, általánosabban a szabályozó megtervezéséhez van szükség. (A BPTT eljárásnál bemutatott 8.1 példa is illusztrálja ezt.)

A vezérlési alkalmazások célja olyan gerjesztőjel előállítása valamely rendszer meghajtására, hogy annak kimeneti jele valamilyen kritérium alapján a lehető legjobban közelítse a specifikált kimeneti jelet. (Lehet kritérium a jó követés, de lehet pl. a minimális túllövés is.) A szabályozáselmélet irodalma rendkívül sokféle – az alkalmazások nagyon különböző követelményeit kielégítő – módszert ismer.

A hagyományos adaptív szabályozási módszerek blokksémája a 8.22 ábrán látható. Ezek egy állítható paraméterekkel is rendelkező szabályozót, valamint ezen paramétereket állító adaptációs algoritmust tartalmaznak. Az algoritmus a kimeneti hiba, valamint a rendszer be- és kimenete (vagy ezek egy része) alapján állítja be a vezérlő paramétereit.

A gyakorlatban igen gyakran találkozunk azzal az igénnyel, hogy a szabályozandó berendezést – a szakaszt – úgy gerjesszük, hogy a szabályozó-szakasz együttes a zavaró, változó környezeti hatások ellenére is, egy specifikált referenciamodellhez hasonlóan viselkedjen (pl. a repülőgép úgy reagáljon a kormánymozdulatokra, ahogy azt a pilóták "szeretik"). Az ilyen környezetben használható szabályozásokat nevezzük modell alapú adaptív szabályozásnak (*Model Reference Adaptive Control, MRAC*).

8.22. ábra - Az adaptív szabályozás



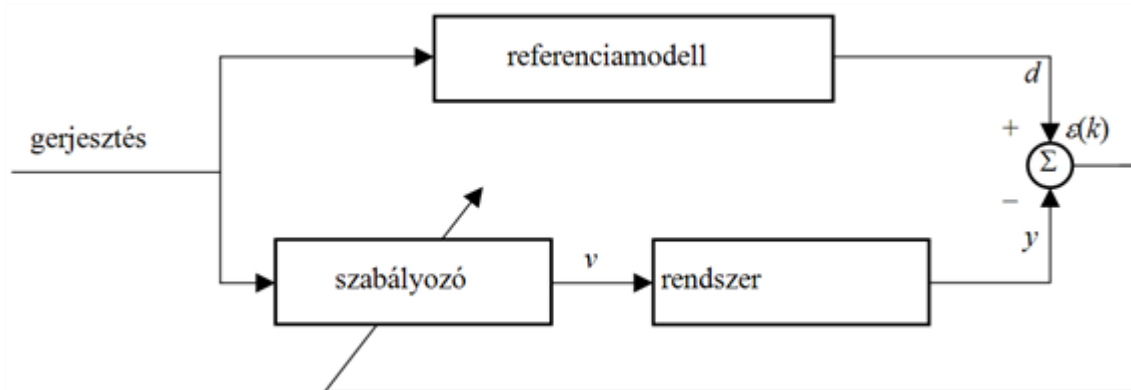
A neurális hálózatok alkalmazása modell alapú adaptív szabályozásra sokmindenben hasonlít az identifikációra. Ha sikerül ugyanis úgy adaptálnunk a vezérlőnek használt neurális hálózat paramétereit, hogy az a szakasszal

együtt úgy viselkedjen, mint a referenciamodell, akkor megoldottuk a feladatot. Ez pedig tulajdonképpen azt jelenti, hogy a vezérlőből és a szakaszból álló modellel identifikáltuk a referenciamodellt. A különbség abban áll, hogy – az identifikációs feladattól eltérően – ebben az esetben nem a neurális hálózat, hanem a szakasz kimenetén képződik az adaptációhoz használható hiba. Ezt a hibát pedig közvetlenül nem lehet felhasználni a hálózat paramétereinek meghatározásához.

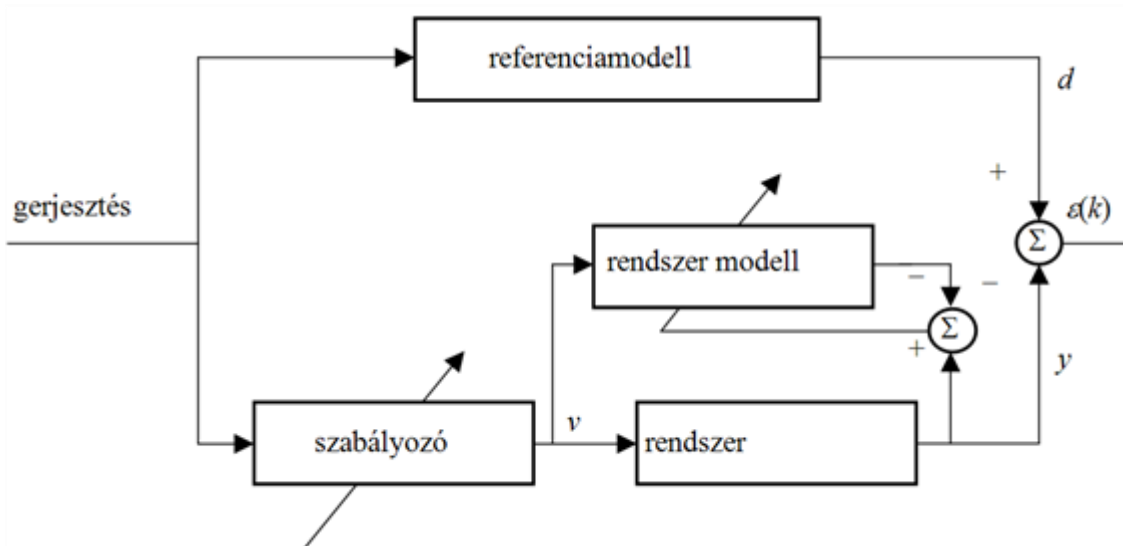
Az MRAC irodalmából az derül ki, hogy a vezérlő paramétereinek közvetlen adaptálására jelenleg még hagyományos szabályozó, sőt lineáris szakasz esetén sincs módszer. (A vezérlő paramétereinek ilyen, azaz közvetlen állítását a szakirodalom közvetlen szabályozás (*direct control*) néven említi. Közvetlen szabályozást azért nem tudunk megvalósítani, mert a valódi, fizikai rendszeren nem tudunk gradienst visszatérjeszteni, ami pl. a hibavisszatérjesztési algoritmus alkalmazásához szükséges lenne.)

Jelenlegi tudásunk alapján a vezérlő paramétereit az ún. közvetett szabályozás (*indirect control*) módszerrel tudjuk állítani. Ennek lényege az, hogy a szakasszal párhuzamosan kapcsolunk egy identifikáló egységet, amely képes a szakaszt megfelelő pontossággal modellezni, ugyanakkor a vezérlő adaptálásához szükséges gradiens visszatérjesztésére is lehetőséget nyújt. Ha a gradiens információ már rendelkezésünkre áll, akkor megfelelő dinamikus tanító algoritmussal a vezérlőnek használt hálózat paramétereit meghatározhatók.

8.23. ábra - A modell alapú adaptív szabályozás (MRAC)



8.24. ábra - A közvetett szabályozás



6.3. Néhány tipikus alkalmazási terület

A következőkben – részben a fentiek illusztrálására – két konkrét alkalmazási területet, illetve néhány mintapéldát említünk meg röviden. A valóságos gyakorlati alkalmazások részletes elemzése meghaladja a könyv lehetőségeit. Egy valóságos „éles” feladat megoldása a megoldandó problémák részletes elemzését is igényli,

így minden feladat valójában egyedi megfontolásokat igényel. A megoldások egyedi jellege ellenére azonban a különböző feladatokból bizonyos általános tapasztalatok is levonhatók. A gyakorlati feladatok megoldásánál felmerülő legfontosabb általános kérdésekkel és tapasztalatokkal a 13. fejezet foglalkozik.

6.3.1. Robotika

Neurális hálózatok alkalmazására a robotika egy igazi „állatorvosi ló”. Alkalmazásuk már csak azért is felvetődik e területen, mert minden robot erősen nemlineáris rendszer. Mivel a felmerülő problémák igen sokfélék, az új módszerek kipróbálására sok lehetőség adódik. Néhány a felvetődő területek közül: pályatervezés, pályakövetés, szenzor információk feldolgozása, autonóm eszközök intelligens irányítása, egyes stabilitási problémák, stb. Az alkalmazások egy részében a neurális hálózat szerepelhet egy korábban már ismert algoritmus, módszer realizációjaként is (pl. sebességi megfontolások miatt). Példaként álljon itt néhány konkrét robotikai alkalmazás.

Isabelle Rivals és munkatársai egy olyan rendszert fejlesztettek ki, mely egy négykerék meghajtású Mercedes terepjáró automatikus irányítását látta el [Riv94]. A neurális megoldás szabályozta a gázadagolást, a fékezést, a kormánybeállító szervó rendszert annak érdekében, hogy az autó egy előírt pályán, előírt sebesség-profillal mozogjon változó környezeti feltételek (pl. széllekedések, csúszós és különböző meredekségű útszakaszok) mellett.

Simon Yang és Max Meng egy neurális robot-mozgás tervező rendszert ismertet [Yan00]. A feladat egy mobilis robot mozgásának irányítása ismeretlen, akadályokat is tartalmazó környezetben.

Frank Lewis neuronhálót alkalmaz robot manipulátor irányítására [Lew96]. A feladatok között egyebek mellett a manipulátor pozíciója és a manipulátor által kifejtett erő szabályozása szerepel.

További sikeres alkalmazási példák a bőséges irodalomban (pl. [Bek93], [Pra96], [Sma06]) találhatók.

6.3.2. Előrejelzés

Számos gyakorlati feladat megoldásánál jelentkezik igény valamiféle előrejelzésre, jóslásra [Che04] (egy nagy haszonnal kecsegtető, bár igen nehéz terület pl. a valutaárfolyamok, tőzsdei árfolyamok előrejelzése). Az előrejelzési feladatok megoldása is alapvetően a neuronhálók modellező képességén alapul, hiszen minden jóslási feladat lényege az, hogy egy rendszer jövőbeli viselkedésének előrejelzése a múltbeli viselkedésének modellezése alapján, a rendszer eddigi "történetének" felhasználásával lehetséges.

A jóslási feladat sok esetben egy idősor alakulásának előrejelzését jelenti. Valójában az idősor előrejelzés is rendszermodellezési feladat; az idősor rendelkezésre álló része alapján próbáljuk annak a rendszernek a viselkedését minél pontosabban leírni, amely az idősor egyes értékeit előállítja. A rendszermodell birtokában az idősor folytatása, az előrejelzés már könnyen megoldható.

Idősor előrejelzési feladatokat „hagyományos” megközelítés alkalmazásával, fejlett statisztikai eljárások felhasználásával is megoldhatunk sikeresen. A neuronháló alapú megoldások eredményei azonban arra engednek következtetni, hogy ilyen jóslási feladatoknál a neurális hálózatok alkalmazása jobb eredményre vezethet, vagy megfelelő eredményt könnyebben, kevesebb erőfeszítéssel nyerhetünk. Tulajdonképpen a jóslási folyamat során felépül egy jelmodell, amely megpróbálja leírni a vizsgált jelsorozatot. Ez a jelmodell neurális hálózatok alkalmazása során „automatikusan”, a hálózaton belül épül fel.

Az alábbiakban két tesztfeladatot mutatunk be.

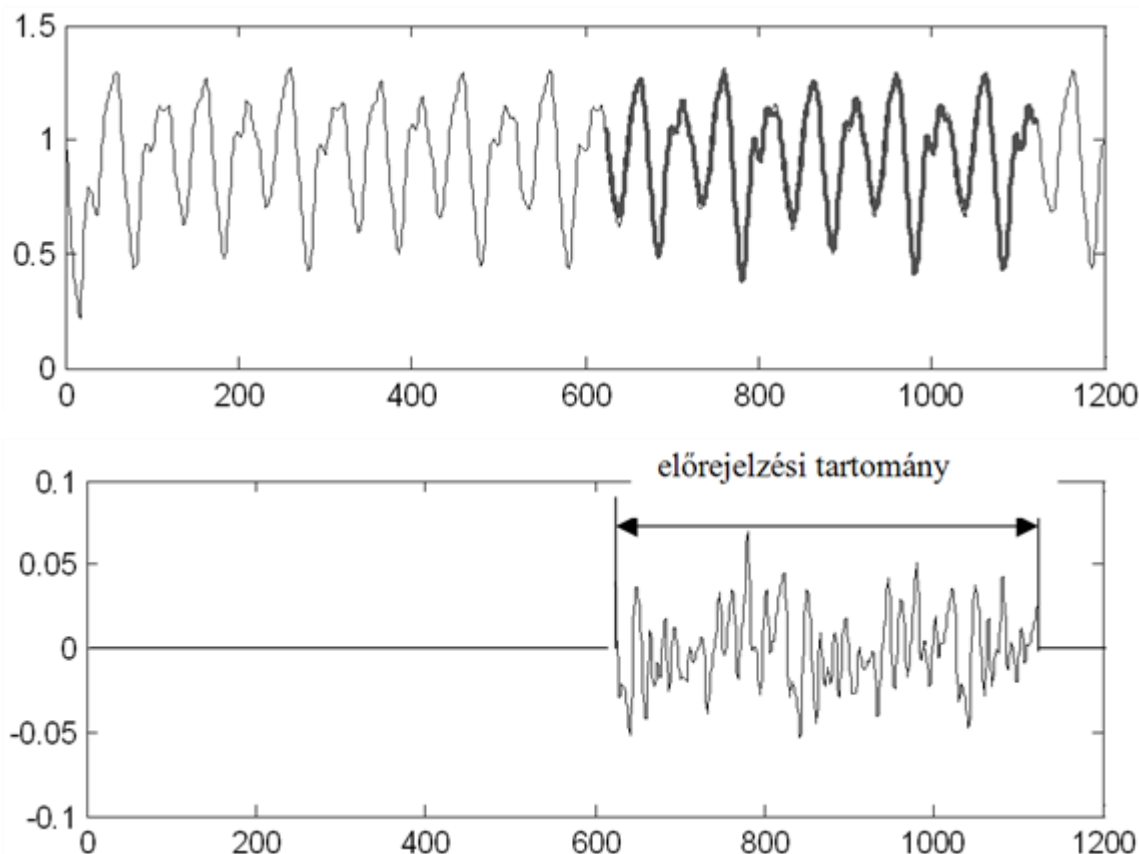
6.3.2.1. Mackey-Glass kaotikus folyamat előrejelzése

A Mackey-Glass kaotikus folyamatot, mint gyakori benchmark problémát a modell-fokszám becslési eljárásoknál említettük (ld. 8.2 példa). A kaotikus folyamatok tesztfeladatként való felhasználását az indokolja, hogy ezen folyamatok jövőbeni alakulásának előrejelzése különösen nehéz, hosszútávú előrejelzésük pedig gyakorlatilag lehetetlen.

Az előrejelzési feladatot itt is úgy fogalmaztuk meg, hogy a folyamatból származtatott diszkrét idősor pillanatnyi és régebbi mintái alapján a következő időponthoz tartozó érték előrejelzésére van szükség, tehát egy lépéses előrejelzési feladattal van dolgunk. A megoldás NARX modell-osztály mellett, a 8.2 példában meghatározott modell-fokszám alkalmazásával, Gauss kernellel dolgozó LS-SVM hálózattal ($\sigma=2,6$, $C=1$ hiperparaméterek mellett) történt. A tanítókészlet 500 mintaszekvenciából állt. Az LS-SVM előrejelzésének

eredménye a 8.25 ábrán látható. Az ábra felső részén az eredeti idősor és az előrejelzés szerepel; a vékonyabb folytonos vonal az eredeti folyamat alakulását, a vastagabb a háló előrejelzését mutatja. Látható, hogy az előrejelzés és a folyamat valódi mintái az ábra szerinti felbontásban szinte egymásra kerülnek. A két görbe különbsége, az előrejelzési hiba látható az ábra alsó részén. Ezzel a megoldással gyakorlatilag ekvivalens eredményt érhetünk el LS^2 -SVM-mel is. Ennek a megoldásnak az előnye ugyanakkor, hogy LS^2 -SVM-mel ritka megoldást érhetünk el lényegében a minőség romlása nélkül: az 500 mintaszekvenciából 55 szupport vektor elegendőnek bizonyult.

8.25. ábra - A Mackey-Glass kaotikus idősor előrejelzése LS -SVM hálózattal. A felső ábrán a vékony vonal a folyamat időfüggvényét, a vastagabb vonal az előrejelzés eredményét mutatja. Az alsó ábrán az előrejelzési tartományban a hiba látható.



6.3.2.2. Santa Fe verseny

1991 augusztusában az amerikai Santa Fe Institute írt ki egy nyilvános versenyt, amely során hat mintasorozatnak kellett megjósolni a folytatását. A szervezők egymástól teljesen eltérő jellegű feladatokat adtak ki, és semmiféle megkötést nem tettek az alkalmazható módszerre vonatkozóan. Maguk a szervezők is meglepődtek azon az eredményen, hogy a neurális hálózatokat alkalmazó megoldások mindegyik jel esetén jobbnak bizonyultak az egyéb – igen széles skálán mozgó – megoldásoknál [Wei94].

A közzétett jelsorozatok az alábbiak voltak:

- egy fizikai kísérlet (egy laboratóriumi NH_3 lézer),
- számítógéppel generált véletlen sorozat,
- svájci frank – amerikai dollár keresztárfolyamának változása,
- asztrofizikai adatok egy változó törpecsillagról,
- fiziológiai adatok,

- mintavett jelsorozat J. S. Bach: "A fuga művészete" c. művének, utolsó, befejezetlen darabjából.

Az adatok – mint látszik – tényleg eléggé eltérőek, még jellegüket tekintve is. Azonban éppen ez a sokszínűség az, amely a neurális hálózatok alkalmazhatóságát látszik igazolni jóslási feladatok során.

A Santa Fe versenyt követően jónéhány hasonló versenyt írtak, ki. Ezek a versenyek az utóbbi években a neuronhálókkal foglalkozó évenként megrendezésre kerülő legrangosabb nemzetközi konferenciához, az *International Joint Conference on Neural Network (IJCNN)*-höz kapcsolódnak.

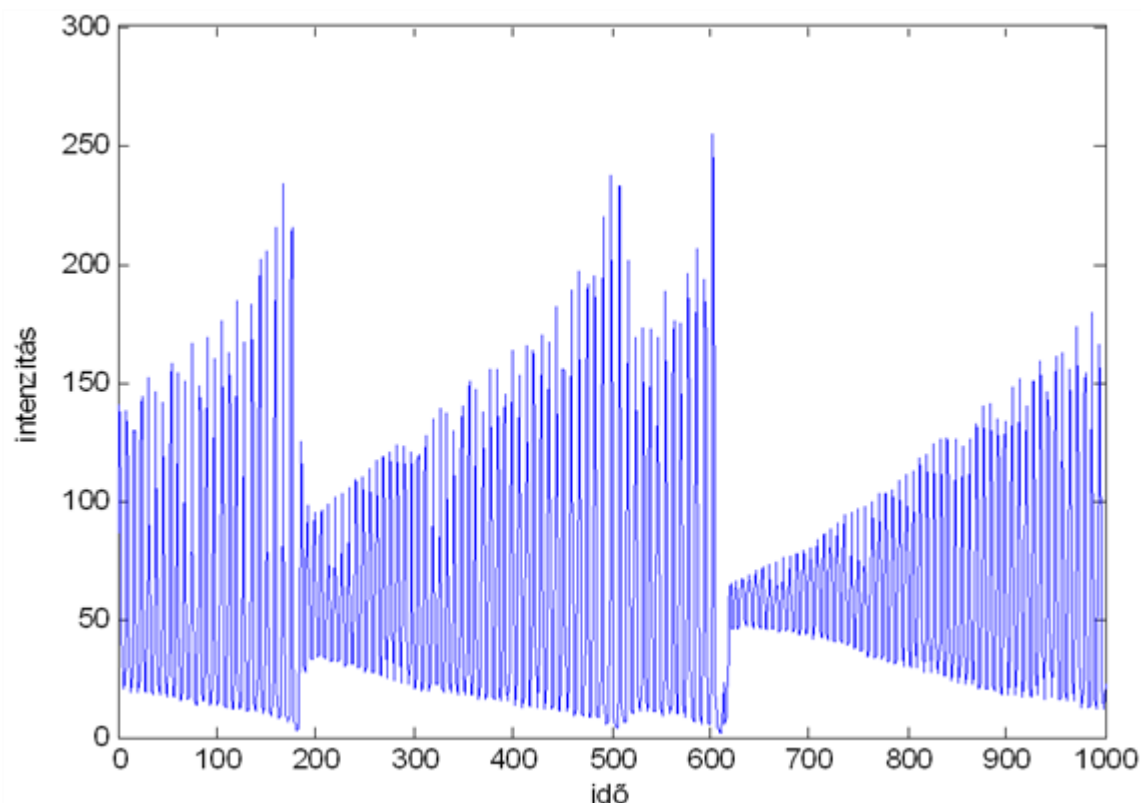
A gyakorlatban előforduló valódi jóslási feladatok megoldása során szinte mindig rendelkezésre áll bizonyos a priori információ a vizsgált jelenséget illetően. Ez az információ beépíthető a modellekbe a megoldás során, jelentősen javítva a jóslás megbízhatóságát. Az a priori információ beépítése neurális hálózatokba (pl. a súlyok kezdeti inicializálása, stb. által) még nem teljesen megoldott, így ebből a szempontból még sok kutatásra van szükség. Mivel a Santa Fe verseny során a rendezők a jelekről semmilyen a priori információt nem tettek közzé, valószínűleg a hagyományos módszerek „nem tudták a legjobb formájukat hozni”.

A következőkben az első jelsorozat, az NH_3 lézer intenzitás előrejelzését megoldó neurális megoldást, mint ennek a feladatnak a győztesét mutatjuk be [Wan93]. A feladat nehézségét az jelenti, hogy a lézerjel intenzitása időben nem állandó, hanem folyamatosan és gyorsan pulzál a szélső értékek között, ráadásul a folyamat kaotikus jellegű. A közreadott kiinduló jelszemszám, mint időfüggvény a 8.26 ábrán látható.

A konkrét feladatban egy 1000 hosszú adatsor állt rendelkezésre, és ebből kellett megjósolni a lézerjel intenzitását a következő 100 időpillanatban. A verseny ideje alatt az idősről semmiféle háttér információt nem tettek közzé. A versenyzőknek a priori ismeretek bevetése nélkül, pusztán számok alapján kellett előrejelzést építeni. (Természetesen ez a tiszta „fekete doboz” megközelítés tényleges gyakorlati feladatok megoldása esetén nem célszerű. Viszont előrejelző algoritmusok összehasonlítására kiválóan alkalmas, mivel így maguk az algoritmusok kerülnek összevetésre, nem pedig az a priori tudás beépítésének módjai.)

A versenyben hibakritériumként a tényleges idősor empirikus szórásnégyzetével normalizált átlagos négyzetes hibát használták. Az alkalmazott megoldás az előző feladat megoldásához hasonlóan egy lépéses predikcióra vezette vissza a középtávú előrejelzési feladatot. Ez azt jelenti, hogy a háló konstrukciójánál NARX modellosztályt alkalmaztak, melyet a tanítás után mint NOE modellt működtettek. Az idősor természete és a megcélzandó hibakritérium alapján nyilvánvaló volt, hogy a nagy intenzitás-visszaesések (lásd a 200-as, az 510-es és a 600-as pontok környékét) megjóslása kulcsfontosságú. Ez érdekes jelleget ad a feladatnak, mivel ilyen mintázatból a rendelkezésre álló idősorban mindössze 3 található.

8.26. ábra - A Santa Fe lézer idősor tanítómintaként felhasználható első 1000 mintája



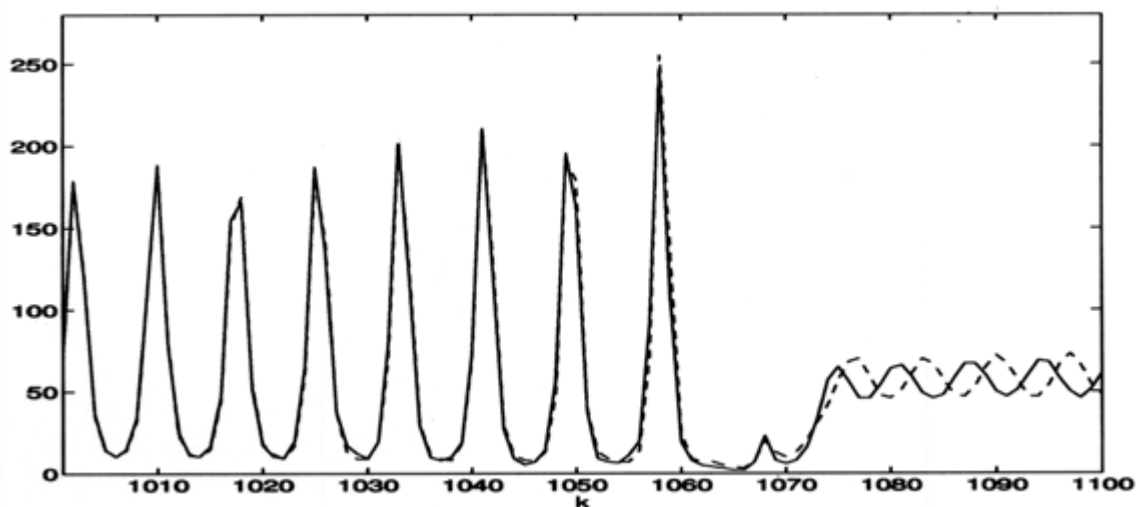
A versenygyőztes FIR-MLP hálózat a bemeneti és a kimeneti neuronon kívül 2 rejtett réteget tartalmazott, mindkettőben 12-12 neuronnal. Az első rejtett réteg bemenetei 25-öd fokú, míg a további két réteg bemenetei 5-ödfokú szűrők voltak. Ezeknek a paramétereknek a beállítása kereszt kiértékeléssel történt. Ilyen felépítés mellett a FIR-MLP hálózatnak összesen 1105 szabad paramétere volt, ami a tanító készlet méretéhez képest igen nagy szám. Erre a szerzők szerint azért volt szükség, mert a kísérletek azt mutatták, hogy kisebb hálózattal a hirtelen nagy visszaesések helyei nem jósolhatók meg elég pontosan. A sok szabad paraméter azt is eredményezi, hogy a távoli időpillanatokra adott becslések varianciája nagy lesz, de ez még mindig kevésbé növeli a hibát, mintha nem sikerült volna eltalálni a hirtelen intenzitás-visszaesés időpontját. Vegyük észre, hogy miközben a modell ténylegesen a NARX modellstruktúrának megfelelően épült fel, az alkalmazott háló architektúra FIR-MLP volt. A FIR-MLP azonban nem valódi bemenőjelet kapott (hiszen ilyen a feladatban nincs is), hanem a kívánt válasz, a lézerjel régebbi értékeit.

A háló kezdeti súlyait úgy állították be, hogy a súlyoknak 0 és 1 közötti véletlen értékeket adtak, majd minden neuron súlyát a bemenetszám négyzetgyökével fordítottan arányosan skálázták. A kimeneten lineáris neuront használtak, hogy a háló kimenete szélesebb értéktartományban mozoghasson. (A kimenet jövőbeli értéktartományát előre nem lehetett megbecsülni, amire normalizált kimenethez szükség lett volna.) Tanulási tényezőként 0.001-et használtak (heurisztikusan állították be így). A rendelkezésre álló idősor első 900 pontját használták tanításra, a maradék 100-on végezték a tesztelést. Egy teljes tanítás során tipikusan kb. 2000-szer haladtak végig a tanító készleten.

Mivel a tesztelő készlet nem tartalmazott egyetlen hirtelen intenzitás-visszaesést sem, ezért ellenőrzésképpen a hálókkal az 550-es ponttól is indítottak 100 lépéses predikciókat. Ez némileg „csalás”, de olyan kevés adat állt rendelkezésre, hogy a szerzők így döntöttek. Természetesen meg kell jegyeznünk, hogy ezeknél az ellenőrzéseknél az 550-es ponton túl a hálózat nem a tanító készlet elemeit kapta bemenetként, hanem a saját korábbi előrejelzéseit (NOE modell). A FIR-MLP-vel kapott előrejelzés eredménye a 8.27 ábrán látható.

Érdekes, hogy a verseny után közzétették a lézer idősor 10 000 pontból álló folytatását. A kísérletek azt mutatták, hogy a versenygyőztes hálózat akkor is jó eredményeket produkált, ha nem az 1001-edik időpillanattól kezdve végezték el a 100 lépéses predikciót, miközben a hosszabbtávú előrejelzésnél a hiba drasztikusan nőtt. Ez azt jelenti, hogy a háló megtanulta a lézerjel dinamikáját, de a NOE architektúra miatt az előrejelzési hibák halmozódása miatt a soklépéses előrejelzés nem lehetséges.

8.27. ábra - A Santa Fe lézer idősor előrejelzése FIR-MLP-vel. A folytonos vonal a kaotikus folyamat valódi alakulását, a szaggatott a neuronhálóval történő előrejelzés eredményét adja meg.



A feladatot megoldották LS-SVM-mel is [Suy02a]. A Gauss kernellel felépített LS-SVM alapú NARX modell 50 régebbi minta alapján adott egy lépéses előrejelzést. Az 50 elemű regresszor következménye, hogy a tanítópontok 50 elemű bemeneti vektorokból és a hozzájuk tartozó skalár kimenetekből álltak. Itt emlékeztetni szeretnénk, hogy SVM-nél a kernel trükk miatt a sokdimenziós bemeneti vektorok nem feltétlenül jelentenek hátrányt. LS-SVM-nél a megoldandó lineáris egyenletrendszer mérete nem függ a bemeneti dimenziótól. A hiperparamétereket, a Gauss kernel szélesség-paraméterét és a C regularizációs együtthatót kereszt kiértékeléssel határozták meg. Az LS-SVM-mel elért megoldás minőségét tekintve teljes mértékben egyenértékű a FIR-MLP eredményével.

Feladatok

8.1 Egy FIR-CMAC hálónál határozza meg a kimeneti réteg szűrő együtthatóinak tanítási összefüggéseit gradiens módszer alkalmazása esetén.

8.2 Konstruáljon FIR-RBF hálót, amelynek rejtett rétege megegyezik az eredeti RBF háló rejtett rétegével, a kimeneti rétegében pedig a súlyok helyett FIR szűrők találhatók. Határozza meg a rejtett réteg paramétereinek tanítási összefüggéseit, ha azokat is ellenőrzött tanítással, gradiens módszerrel kívánjuk meghatározni. (ld. még a 5.3 feladatot).

8.3 A 8.12 (a) ábra szerinti modellnél N egy többrétegű perceptron, a kimeneten lévő lineáris dinamikus rendszer átviteli függvénye pedig a következő:

$$H(z) = 0,1z^{-1} + 1,0z^{-2} + 0,5z^{-3}$$

Határozza meg az MLP tanítási összefüggéseit, ha dinamikus hibavisszaterjesztéses eljárást használ.

8.4 Egy 3-bemenetű, két rejtett elemű, egykimenetű MLP hálózat súlyai helyén két megcsapolásos FIR szűrőket alkalmazunk. Határozza meg a háló virtuális megfelelőjét az időbeli kiterítés módszerét alkalmazva. Hány súly található a kiterített virtuális hálóban?

8.5 Egy személyautót szeretnénk hátramenetben beparkoltatni egy járda mellett két autó közé. Konstruáljon olyan neurális kontrollert, amely a feladatot megoldja. Készítse el az autó neurális modelljét, majd a BPTT megközelítést alkalmazva hozza létre a neurális szabályozót, melynek a kocsikormányzása a feladata. A feladatot két változatban készítse el: (1) a parkoló autók a járdára merőlegesen, (2) a járdával párhuzamosan állnak.

8.6 Az alábbi összefüggéssel megadott Mackey-Glass kaotikus idősor előrejelzésére alakítson ki dinamikus neurális rendszert:

$$x(k+1) = -0,1x(k) + \frac{0,2x(k-\tau)}{1+x^{10}(k-\tau)},$$

ha $\tau=30$ és a kezdeti érték $x(k)=0,9$ ha $0 \leq k \leq \tau$. A feladat során a folyamat 500 pontját felhasználva kell a következő 6-10 pont előrejelzését megoldania. Próbálja megoldani a feladatot LS-SVM-mel. Ehhez határozza meg a modell struktúráját (modellosztály és modell-fokszám), majd konstruálja meg a kernel gépet. A megtanított hálót befagyasztva működtesse a hálót rekurzív módon a következő 100 pont jóslására. Vizsgálja meg a hiba alakulását az eredeti idősor adataihoz viszonyítva. Rajzolja fel mind az eredeti, mind a becsült idősor fázisdiagramját (az összetartozó $x(k)$ és $x(k+1)$ minták kapcsolatát egy kétdimenziós ábrán).

8.7 Az alábbi logisztikai egyenletnek nevezett összefüggéssel adott a Feigenbaum kaotikus idősor:

$$x(k+1) = 4x(k)[1-x(k)],$$

Határozza meg a modell fokszámát a Lipschitz index segítségével. Vizsgálja meg a Lipschitz index alakulását, ha a mintákat additív Gauss zaj terheli.

8.8 Az előző feladat eredményeit felhasználva konstruáljon olyan dinamikus neuronhálót, amely modellezi a kaotikus folyamat viselkedését. Generáljon megfelelő számú mintát egy olyan tanítókészlethez, amelynek segítségével megtanított dinamikus neuronháló az idősor előrejelzését 20 lépésre előre meg tudja adni.

9. fejezet - Moduláris hálók

Az eddigi fejezetekben különböző ellenőrzött tanulású neuronhálókkal foglalkoztunk, melyek felépítésükben, tanításukban, részben a képességeikben és az alkalmazási területeiket tekintve is eltértek egymástól. A különbségek ellenére az eddigi megközelítéseknek volt egy közös vonása: bármely hálót is vizsgáltuk, ezt mint különálló architektúrát tekintettük, mely önmagában lehet képes bizonyos feladatok megoldására.

Bonyolult feladatok megoldásánál azonban – függetlenül attól, hogy a feladatot milyen eszközökkel kívánjuk megoldani – a nagy komplexitás miatt célszerű a feladat dekomponálása részfeladatokra, melyek külön-külön könnyebben megoldhatók, és a részmegoldásokból, azok aggregálása útján nyerhető a teljes megoldás. A részekre bontás eredményeként az egyes részfeladatok számítási komplexitás szempontjából jóval egyszerűbbek lehetnek, mint a teljes feladat, sőt az esetek nagy többségében a részfeladatok együttesen is kisebb számítási kapacitást igényelnek mintha az eredeti feladatot egyben próbálnánk megoldani. Az „oszd meg és uralkodj” elv a mérnöki feladatmegoldás egyik legfontosabb alapelve, és neuronhálós megoldásoknál is sikeresen alkalmazható. A feladatmegoldás általános mérnöki megközelítésén túl – utalva a neuronhálók biológiai eredetére – a moduláris konstrukció alkalmazásának fontosságát a biológiai neurális rendszerek felépítésének tanulmányozása is alátámasztja.

Egy feladat dekompozíciója sokféleképpen elvégezhető. A funkcionális dekompozíció mind a mérnöki gyakorlatban, mind biológiai rendszerekben gyakran alkalmazott (megtalálható) módja a feladat részekre bontásának. További lehetőség, ha a problémateret a tér nagysága miatt bontjuk részekre, és minden egyes részterülethez önálló megoldást készítünk, melyek integrálása biztosítja a teljes feladat megoldását. Az egyes részfeladatokat modulok oldják meg, a teljes megoldást ezen modulok összessége – kiegészítve valamilyen integráló modullal – szolgáltatja.

Erre a megközelítésre láthattunk is már példát az 5. fejezetben, ahol a bázisfüggvényes hálócsaládba tartozó CMAC háló komplexitásának csökkentésére alkalmaztunk moduláris felépítést. A különböző moduláris CMAC hálók sokdimenziós feladatoknál a bemenet komponensei szerint végeztek feladatdekompozíciót (SOP struktúra, HCMAC). Valójában tehát ott is a problémateret szeparálása eredményezett egyszerűbb részfeladatokat. A bemeneti tér részekre bontásán alapuló eljárást a következőkben is fogunk látni. Itt azonban a bemeneti tér felbontása a kívánt válaszok alapján történik.

A moduláris feladatmegoldás a komplex feladatok megoldásának természetes módja. Amennyiben egy összetett feladatot részfeladatokra bontunk, a részfeladatokat megoldó modulok együttműködése, kooperációja biztosítja a teljes feladat megoldását. A részekre bontás nem minden esetben jelenik meg explicit módon a feladatmegoldás során. Olyan megoldás is elképzelhető, amikor több modul is megpróbálja a teljes feladatot megoldani, miközben ezek a modulok valamilyen szempontból eltérő megközelítést alkalmaznak. Az eltérő megközelítés következtében az egyes megoldások kisebb-nagyobb mértékben eltérnek egymástól. Bár mindegyik modul a teljes feladatot oldja meg, az egyes megoldások a teljes feladat más-más részein lehetnek jobbak. A megoldások kombinációja – a teljes feladatra nézve – bármelyik modul megoldásánál is jobb lehet. Az eredményt itt is a modulok kooperációja révén nyerjük, azonban itt nem határolódnak el mereven az egyes modulok által megoldott részfeladatok, a határok inkább elmosódtak.

Olyan megoldási stratégia is lehetséges, hogy egy részfeladatot több modul old meg, és az egyes megoldások versengenek egymással. Itt a moduláris megoldás tipikusan nem együttműködés, hanem versengés eredményeképpen születik.

A kétféle stratégia, az együttműködés és a versengés egy komplex rendszeren belül egyszerre is megjelenhet. Ilyen eset lehetséges, ha a teljes feladatot több modul oldja meg, azonban az egyes megoldások aggregálása különbözőképpen történik. A teljes problémateret egyes tartományain az eredő megoldást egy-egy különálló modul megoldása egymaga is jelentheti, míg más tartományokban az egyes modulok eredményeinek kombinációjaként nyerhető a megoldás. Ha egy modul önmagában szolgáltat eredményt, akkor ennek a modulnak a többivel szemben jobb eredményt kell adnia. A megoldást szolgáltató modul versengés eredményeképpen kerül kiválasztásra. Ha az eredmények kombinációjaként kapjuk a teljes megoldást, akkor az egyes modulok együttműködésére van szükség. Hogy egy adott rendszeren belül a kétféle megközelítés közül melyik érvényesül, az függhet attól, hogy egy konkrét eset a problémateret mely tartományába esik.

A fejezet célja, hogy a moduláris megközelítés általános előnyeit és néhány konkrét moduláris hálóarchitektúrát mutasson be. A bemutatott moduláris neurális architektúrák között található olyan, amely elsősorban osztályozási feladatok megoldására szolgál, és olyan is, amely mind osztályozásra, mind regressziós feladatok

megoldására alkalmazható. Elsőként az ún. feladat dekompozíció (*task decomposition*) megoldást tárgyaljuk, majd a szakértőegyüttesekkel foglalkozunk.

A feladat dekompozíció a teljes feladat diszjunkt részfeladatokra bontását jelenti, ahol az egyes részfeladatok már könnyebben megoldhatók. A feladat dekompozíciója itt statikus és nem is tartozik a háló feladatai közé. Ennél a megoldásnál versengésről semmiképpen nem beszélhetünk, együttműködés is csak annyiban van, hogy szükség van a részmegoldások integrálására. Ezt azonban a megtanított hálók kialakításától függetlenül, a dekompozíció módja határozza meg.

A szakértőegyüttesek valóban együttműködő-versengő modulokból épülnek fel. Itt a feladat részekre bontása is – amennyiben a megoldás alkalmaz dekompozíciót – és a modulok eredményének egyesítése is a komplex moduláris architektúra feladata.

A moduláris neuronhálók kialakítási lehetőségei igen sokfélék. Ez már abból is következik, hogy egy feladat igen sokféleképpen dekomponálható részfeladatokra és a különböző részmegoldások egyesítése is sokféle megközelítés alkalmazásával lehetséges. A moduláris neuronháló-kialakítási lehetőségek kimerítő bemutatására ezért a fejezet nem vállalkozik. Helyette néhány, a gyakorlatban is sikerrel alkalmazható megoldást mutat be.

1. Moduláris háló kialakítása feladat dekompozíció alapján

Egy komplex feladat dekompozíció útján történő megoldására számos lehetőségünk van. Az egyik legkézenfekvőbb eljárás, hogy mielőtt a feladatot valamilyen eszközzel megpróbálnánk megoldani, előbb funkciók szerint részekre bontjuk, és csak a részfeladatok külön-külön történő megoldásával foglalkozunk. Az előzetes részekre bontás a feladat részletes ismerete alapján lehetséges. Ilyenkor a dekompozíció mindig feladatfüggő, ennél fogva csak akkor alkalmazható, ha a problémáról elegendő előzetes ismeret áll a rendelkezésünkre. Jó példa erre az előző fejezetben bemutatott pótkocsis teherautó tolatási feladat egy lehetséges megoldása [Jen93]. E szerint célszerű különválasztani a pótkocsi irányának meghatározását a pótkocsi pozíciójának meghatározásától. A dekompozíció előnye, hogy a részfeladatok az eredeti megoldásnál (ld. 8.1 példa) jóval egyszerűbben megoldhatók – az iránymeghatározáshoz például egyetlen perceptron is elegendő –, továbbá az egyes részfeladatokat egymástól függetlenül, párhuzamosan is meg lehet valósítani.

A dekompozíció egy másik lehetősége, ha a feladat felbontását előre nem végezzük el, az a tanulás során automatikusan alakul ki. Erre a fejezet későbbi részében találunk majd példát.

A következőkben egy olyan eljárást mutatunk be, mely a priori ismeret nélkül is hatékonyan dekomponálja a feladatot, mégpedig úgy, hogy a kapott részfeladatok kisebbek és jóval egyszerűbbek, mint az eredeti feladat. Ráadásul az egész eljárás nagyon egyszerű is. Az egyetlen korlátozás, hogy a dekomponálásnak ez a megoldása csak osztályozási problémáknál alkalmazható.

Osztályozási feladatoknál egy többsztályos feladat természetes dekompozíciója lehet, ha a feladatot kétosztályos feladatok együttesére bontjuk fel. Így pl. egy K -osztályos feladat felbontható K darab olyan kétosztályos feladatra, ahol a K osztály egyikét szeretnénk megkülönböztetni a maradék $K-1$ osztálytól. Itt tehát annyi kétosztályos részfeladatot tudunk definiálni, ahány osztály az eredeti feladatban megkülönböztethető. Az ilyen dekompozíció hátránya, hogy a származtatott kétosztályos feladatok az adatok (tanítóminták) számát tekintve nagyon kiegyensúlyozatlanok lehetnek. Ha ugyanis feltételezzük, hogy az eredeti feladat K osztályának mindegyikébe közel azonos számú tanítópont tartozik, akkor a származtatott kétosztályos feladatoknál a két osztály egyikébe ($K-1$)-szer annyi minta esik, mint a másikba. A kiegyensúlyozatlanság a tanítást nehezítheti, az egész tanítási folyamatot jelentősen lelassíthatja.

A kiegyensúlyozatlanság elkerülhető, ráadásul a dekompozíció eredményeképp kapott osztályozási feladatok még egyszerűbbekké is válnak, ha a K -osztályos feladatot olyan kétosztályos részfeladatokra bontjuk, ahol egy részfeladat az eredeti K osztályból kizárólag két osztály szétválasztására irányul, miközben a maradék $K-2$ osztállyal nem is foglalkozunk [Lu99]. Ha $C_1, C_2, \dots, C_K$ -val jelöljük a megkülönböztetendő K osztályt, akkor az egyszerűbb kétosztályos feladatokra bontás azt jelenti, hogy egy kétosztályos feladat során a C_i és a C_j osztályt szeretnénk megkülönböztetni egymástól, ahol $i, j = 1, \dots, K$, $i \neq j$, és természetesen minden i, j párt csak egyszer veszünk számba. Egy adott kétosztályos részfeladat megoldásánál a maradék $K-2$ osztály tanítópontjait figyelmen kívül hagyjuk. Ha továbbra is feltételezzük, hogy az egyes osztályokba közel azonos számú tanítópont tartozik, akkor a dekompozíció eredményeképp kapott egyszerű kétosztályos feladatoknál mindkét

osztályt hasonló számú tanítópont reprezentál, a tanítókészlet a dekompozíció után is kiegyensúlyozott és az eredeti feladatban szereplő osztályok számától (K -tól) független marad.

Az eljárás részletesebb bemutatásához induljunk ki abból, hogy rendelkezésünkre áll egy

$$Z = \left\{ \mathbf{z}_l \right\}_{l=1}^L = \left\{ \left(\mathbf{x}_l, d_l \right) \right\}_{l=1}^L \quad (9.1)$$

tanítókészlet¹, melyet a K osztálynak megfelelően K diszjunkt részhalmazra bontunk.

$$Z_i = \left\{ \mathbf{z}_l^{(i)} \right\}_{l=1}^{L_i} = \left\{ \left(\mathbf{x}_l^{(i)}, d_l^{(i)} \right) \right\}_{l=1}^{L_i} \quad i = 1, \dots, K \quad (9.2)$$

Az eddigi jelöléseknek megfelelően $\mathbf{x}_l^{(i)}$ egy bemeneti vektor, d_l pedig a hozzá tartozó kívánt válasz, a felső index pedig az osztálybesorolást jelző címke.

A javasolt dekompozíció szerint a feladat $K(K-1)$ egyszerűbb kétosztályos részfeladatra bontható. Ebben még van redundancia, hiszen C_i megkülönböztetése C_j -től megoldja a két osztály fordított irányú megkülönböztetését is. A felesleges ismétlések elkerülésével $K(K-1)/2$ egyszerűbb kétosztályos feladatunk lesz. Kétosztályos feladatnál a kívánt válasz két lehetséges érték valamelyike lesz, tehát pl. $d_l \in \{0,1\}$. A gyakorlatban azonban általában megengedjük, hogy a kívánt válasz: $d_l \in \{\varepsilon, 1-\varepsilon\}$ legyen ahol ε egy kis pozitív szám. Ekkor a C_i , C_j osztálypárhoz tartozó mintakészlet az alábbi módon definiálható:

$$Z_{ij} = \left\{ \left(\mathbf{x}_l^{(i)}, 1-\varepsilon \right) \right\}_{l=1}^{L_i} \cup \left\{ \left(\mathbf{x}_l^{(j)}, \varepsilon \right) \right\}_{l=1}^{L_j} \quad i, j = 1 \dots K \text{ és } i \neq j \quad (9.3)$$

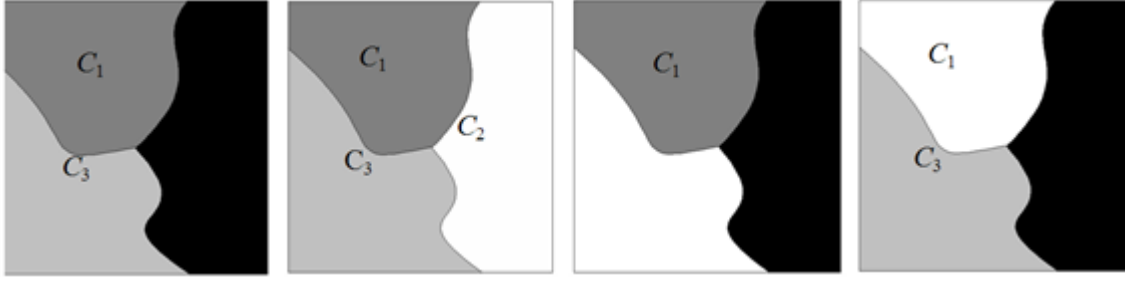
vagyis a C_i osztályba tartozó mintákra a kívánt válasz közel 1, a C_j -be tartozókra pedig közel 0 lesz. (Ez azért előnyös a $\{0,1\}$ kívánt válasz értékhalommal szemben, mert a hálók és tanító eljárások egy részénél a 0, illetve az 1 környezetben nagyon lelassul a tanulási sebesség.) Vegyük észre, hogy Z_{ij} csak a C_i és a C_j osztályba tartozó mintapontokat tartalmazza, a maradék $K-2$ osztály pontjai az elemi kétosztályos feladat megoldásánál nem kerülnek felhasználásra.

Ezt a dekompozíciót $K=3$ mellett illusztrálja a 9.1 ábra. Az eredeti háromosztályos feladatot az ábra szerint három elemi kétosztályos feladatra bontottuk, miközben mindegyik esetben a harmadik osztály pontjait figyelmen kívül hagytuk.

A fenti dekompozíció nem minden esetben elegendő ahhoz, hogy az elemi kétosztályos feladatok kellően egyszerűek legyenek. Bizonyos problémáknál a megkülönböztetendő két osztály elválasztó felülete lehet nagyon bonyolult is. Ilyen esetekben az előzőekben kapott egyszerű kétosztályos feladatokat érdemes még tovább bontani. Hasonló esettel állunk szemben, ha már az eredeti probléma is kétosztályos, de nehéz volt. A tovább-bontás azt jelenti, hogy az egyes osztályokba tartozó tanító halmazokat további részhalmazokra bontjuk, és ezekre a részhalmazokra fogalmazunk meg kétosztályos feladatokat. Ez a felbontás – szemben az előzővel – már nem egyértelmű, hiszen bármely, adott osztályhoz tartozó pontkészlet tetszőleges módon dekomponálható.

9.1. ábra - Egy háromosztályos feladat felbontása 3 egyszerű kétosztályos feladatra

¹ Ebben a fejezetben az eddigiektől eltérően a mintapontok számát nem P -vel, hanem L -lel jelöljük. Ennek oka, hogy a fejezetben tárgyalt egyes moduláris rendszereknél valószínűségi értelmezés, vagy valószínűségek meghatározása is szükséges, és P -t a megfelelő sűrűségfüggvények vagy valószínűségek jelölésére használjuk.



Tételezzük fel, hogy az i -edik osztályhoz tartozó, (9.2) által definiált mintahalmazt tovább bontjuk M_i részhalmazra ($1 \leq M_i$). A C_i osztályhoz tartozó j -edik ilyen részhalmaz az alábbi módon definiálható:

$$Z_i^{(j)} = \left\{ \mathbf{z}_i^{(ij)} \right\}_{l=1}^{L_i^{(j)}} = \left\{ \mathbf{x}_i^{(ij)}, \mathbf{d}_i^{(ij)} \right\}_{l=1}^{L_i^{(j)}} \quad j = 1, \dots, M_i, \quad (9.4)$$

ahol $L_i^{(j)}$ jelöli ennek a részhalmaznak az elemszámát. A részhalmazokra igaz kell legyen, hogy $\bigcup_{j=1}^{M_i} Z_i^{(j)} = Z_i$ és $\bigcap_{j=1}^{M_i} Z_i^{(j)} = \emptyset$. Az így definiált tanító részhalmazok segítségével definiálhatjuk a kétosztályos osztályozási feladatokat. A C_i osztály u -edik részhalmaza és a C_j osztály v -edik részhalmaza által képezett kétosztályos feladat tanító készlete ennek megfelelően:

$$Z_{ij}^{(u,v)} = \left\{ \left(\mathbf{x}_i^{(iu)}, 1 - \varepsilon \right) \right\}_{l=1}^{L_i^{(u)}} \cup \left\{ \left(\mathbf{x}_j^{(jv)}, \varepsilon \right) \right\}_{l=1}^{L_j^{(v)}}$$

$$i, j = 1, \dots, K \text{ és } i \neq j, \quad u = 1, \dots, M_i, \quad v = 1, \dots, M_j \quad (9.5)$$

A dekompozíció akár addig is folytatható, amíg a részhalmazokban csak egy-egy mintapont marad. Ez azt jelenti, hogy az így definiált elemi osztályozási feladatok már biztosan megoldhatók lineáris szeparálásra alkalmas eszközökkel is. Természetesen ilyen mértékű felbontásra nincs minden esetben szükség.

Az elemi osztályozási feladatok megoldása után a következő lépés az eredmények integrálása. Az aggregálásnál két esetet kell megkülönböztetni.

Az első esetben legyenek olyan osztályozó moduljaink, ahol mindegyik feladata az, hogy adott C_i osztályba tartozó mintákat mindig más-más C_j , $j=1,2,\dots,K$, $j \neq i$ osztályba tartozó mintáktól különböztessen meg. Egy ilyen osztályozó modutra az érvényes, hogy a tanításhoz a (C_i, C_j) osztálypárhoz tartozó, a (9.3) összefüggés által definiált mintakészletet használjuk, vagyis az osztályozó tanításánál a kívánt válasz akkor lesz $1-\varepsilon$, ha a tanítóminta a C_i osztályba tartozik és akkor lesz ε , ha a C_j -be.

Az ilyen típusú modulok eredményeinek integrálására a MIN művelet szolgál. A MIN műveletet megvalósító modul feladata, hogy a bemenetére kerülő értékek minimumát adja válaszul:

$$y_{\text{MIN}} = \min(x_1, x_2, \dots, x_N), \quad (9.6)$$

ahol $x_1, x_2, \dots, x_N$ jelölik a MIN modul bemeneteit, y_{MIN} pedig a modul kimenetét. A MIN modullal egyesített osztályozók eredő válasza csak akkor lesz $1-\varepsilon$, ha a MIN modul minden bemenetére $1-\varepsilon$ kerül, vagyis, ha a MIN modult meghajtó minden osztályozó válasza $1-\varepsilon$. Az integrált osztályozó tehát akkor fog egy mintát a C_i osztályba besorolni, ha a MIN modullal összefogott összes osztályozó a C_i osztályba sorolja az adott mintát.

A második esetben olyan moduljaink vannak, melyek feladata, hogy egy adott C_j osztály mintakészletének különböző részhalmazaihoz tartozó mintákat különböztesse meg valamely más osztályba tartozó mintáktól. Ezekre az osztályozó modulokra az érvényes, hogy a tanításukhoz a C_j osztály v -edik és a C_i osztály u -edik részhalmazaihoz tartozó, a (9.5) összefüggés által definiált mintakészleteket használjuk, vagyis mindegyik osztályozó tanításánál a kívánt válasz akkor lesz ε , ha a tanítóminta a C_j osztály valamelyik v részhalmazába tartozik.

Az ilyen típusú modulok eredményének integrálására a MAX művelet szolgál. A MAX műveletet megvalósító modul feladata, hogy a bemenetére kerülő értékek maximumát adja válaszul:

$$y_{\text{MAX}} = \max(x_1, x_2, \dots, x_N), \quad (9.7)$$

ahol a modul bemenetei most is az $x_1, x_2, \dots, x_N$ értékek, kimenete pedig y_{MAX} . A MAX modullal egyesített osztályozók eredő válasza akkor lesz ε , ha a MAX modul egyik bemenetére sem kerül $1-\varepsilon$. Ehhez a MAX modult meghajtó minden osztályozó válasza ε kell legyen, ami egyben azt is jelenti, hogy a MAX modult meghajtó minden osztályozó modulra a C_j osztály valamelyik v részhalmazába tartozó minta kerül.

Ha a MIN és a MAX műveletek elvégzésére alkalmas modulokat kiegészítjük egy INV modullal, akkor az előzőekben bemutatott bármilyen dekompozíció eredményeként kapott elemi kétosztályos osztályozó modulok integrálása felesleges redundancia nélkül lehetséges. Az INV modul egybemenetű, egykimenetű modul, melynek feladata a bemenetére kerülő érték invertálása, vagyis, ha az INV bemenete ε , a kimenete $1-\varepsilon$ kell legyen, és fordítva, $1-\varepsilon$ bemenetre ε választ kell adjon. Ez teszi lehetővé, hogy ha csak az osztályok szerint végezzük a dekompozíciót, egy K osztályos osztályozó helyett elegendő $K(K-1)/2$ kétosztályos osztályozót alkalmazni.

Ha az egyes osztályokat még tovább bontjuk és a (9.5) összefüggés szerinti mintahalmazokkal dolgozunk, akkor az INV modulok használata következtében az elemi kétosztályos osztályozók száma

$$\sum_{i=1}^K \sum_{\substack{j=1 \\ j \neq i}}^K M_i M_j \quad \text{-ről (9.8)}$$

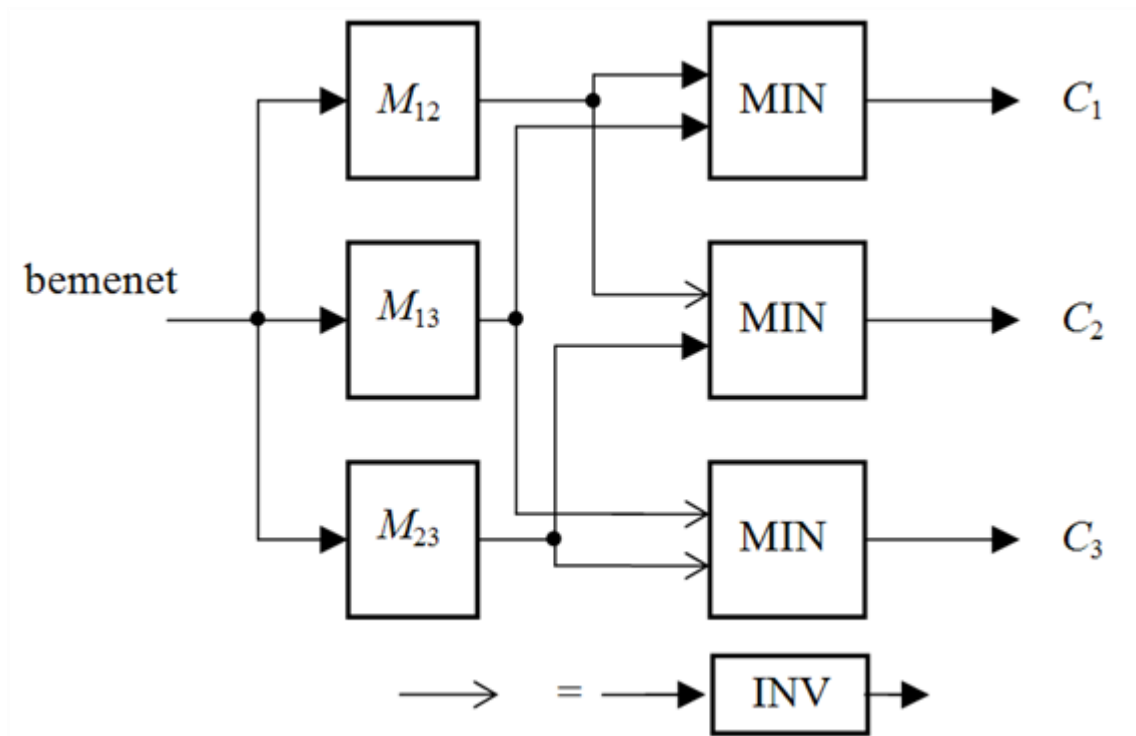
$$\sum_{i=1}^K \sum_{j=i+1}^K M_i M_j \quad \text{-re (9.9)}$$

redukálható, ahol M_k a C_k ($k=1, 2, \dots, K$) osztály részhalmazainak a száma.

Könnyen észrevehető, hogy az integráló modulok feladata hasonló a logikai ÉS, VAGY és NEM műveletekhez. Ha az egyes osztályozó modulok bináris kimenetet szolgáltatnak, akkor az integráló modulok kimeneteit valóban az osztályozók kimeneteinek logikai ÉS, VAGY, illetve NEM kapcsolatait felhasználva kaphatjuk meg.

Az is könnyen belátható, hogy az INV művelet mellett valójában elegendő kizárólag a MIN vagy kizárólag a MAX művelet, bár sok esetben egyszerűbb a modulok összekapcsolása, ha mindkettő rendelkezésre áll. A 9.2 ábrán a 9.1 ábra háromosztályos feladatának moduláris megoldása látható.

9.2. ábra - Az 9.1 ábrán látható háromosztályos feladat moduláris megoldása



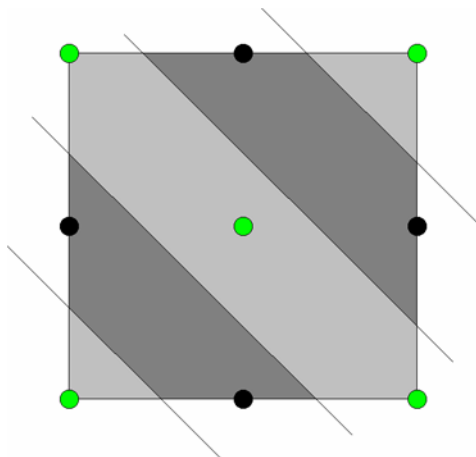
9.1 Példa

Az alábbiakban egy kétsztályos feladat kapcsán mutatjuk be a feladatdekompozíció egy speciális alkalmazási lehetőségét. A feladat a 9.3 ábrán látható. Az ábrából az is nyilvánvaló, hogy a kétsztályos feladat lineárisan nem szeparálható. A feladatdekompozíciós eljárással azonban olyan elemi kétsztályos feladatokra bontható, ahol ezek az elemi feladatok már lineáris szeparálással (pl. egy egyszerű perceptronnal) megoldhatók. Az ábra azt is mutatja, hogy – ideális esetben – milyen tartományok megkülönböztetése szükséges a feladat megoldásához. Az ideális eset itt azt jelenti, hogy az elválasztó egyenesek a mintapontoktól egyforma és a lehető legnagyobb távolságban helyezkednek el. Ilyen megoldást garantáltan egy lineáris SVM-mel érhetünk el, egy egyszerű perceptrontól azonban csak az várható, hogy a két különböző osztályba tartozó pontok között a pontokat hibátlanul elválasztó, de nem feltétlenül ideális elhelyezkedésű egyeneseket határozzon meg.

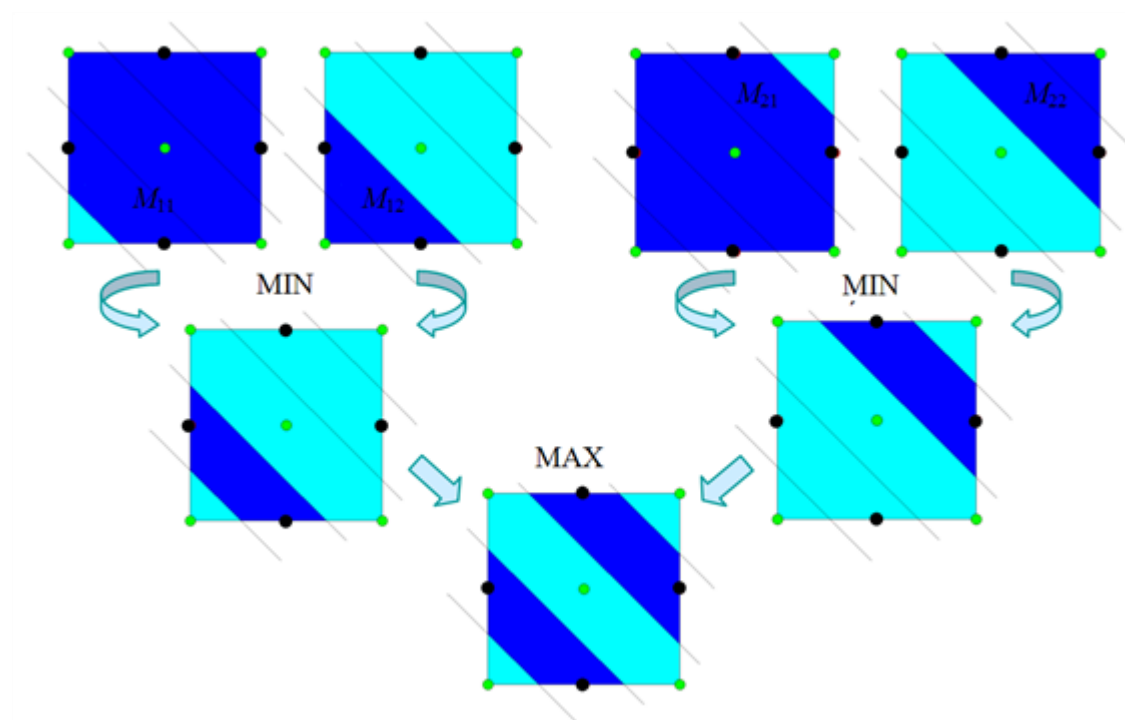
A következő, 9.4 ábrán azokat a lépéseket követhetjük, melyek a moduláris kialakítás eléréséhez vezetnek. Ugyancsak látható, hogy az elemi kétsztályos osztályozók integrálásához milyen modulokra van szükség.

A teljes moduláris osztályozó rendszer felépítését a 9.5 ábra mutatja. A moduláris felépítésnél azt tételeztük föl, hogy az egyes osztályozó modulok és a teljes hálózat is akkor vesz fel magas értéket, ha a fekete pontokkal jelölt bemenetek kerülnek a hálóra.

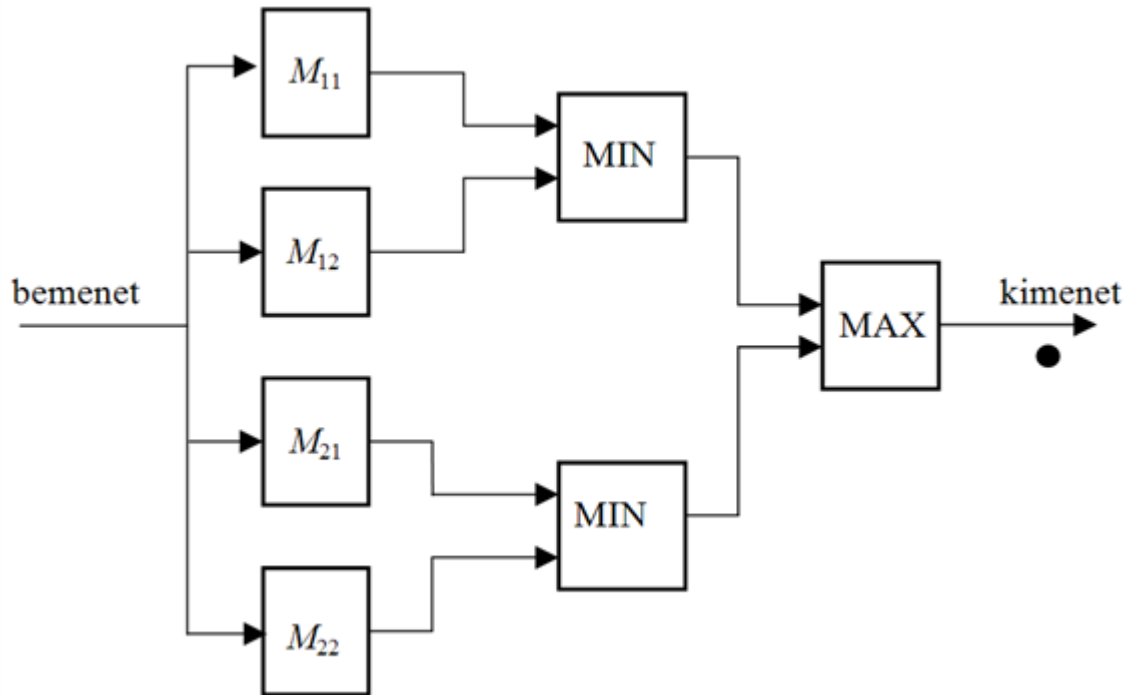
9.3. ábra - Egy egyszerű, lineárisan nem szeparálható kétsztályos feladat (a zöld tanító mintapontok az egyik, a feketék a másik osztályba tartoznak)



9.4. ábra - A moduláris kialakítás lépései (az egyes részmegoldások a sötéttel jelzett területeken adnak közel 1 értéket, a világossal jelölt területeken közel 0 értéket)



9.5. ábra - A 9.3 ábrán látható feladat dekompozíciónak megfelelő moduláris háló felépítése



A feladatdekompozíció sikerrel alkalmazható összetett gyakorlati feladatok megoldására. Egy sikeres példaként említhetjük a kézzel írott számjegyek felismerésére kidolgozott és a 7.1.2 alfejezetben már bemutatott megoldást is, amely ezt a megközelítést alkalmazta, hiszen a 10 számjegy felismerését nem egy 10-osztályos osztályozó hálózattal, hanem $10 \times 9/2 = 45$ egyszerű kétosztályos osztályozóval oldotta meg. Az egyes elemi kétosztályos osztályozók mindig két osztályt (számjegyet) próbáltak egymástól megkülönböztetni, miközben a maradék nyolc további osztály mintapontjaival egyáltalán nem foglalkoztak. Ez a megoldás – mint láttuk – egy nagyon egyszerű háló architektúrát eredményezett, hiszen az elemi kétosztályos osztályozók egyszerű perceptronok voltak. A megfelelő osztályozó modulok eredményeinek integrálása ott ÉS kapukkal volt megoldva, ami megfelel a MIN műveleteknek.

2. Szakértőegysétek

Az előző részben olyan moduláris háló kialakítást láttunk, ahol egy feladatot részfeladatokra bontva és minden egyes részfeladathoz egy önálló hálót rendelve kaptunk moduláris neurális architektúrát. A következőkben olyan együttműködő hálókkal foglalkozunk, amelyek mindegyike a teljes feladatot oldja meg.

Az ún. hálóegysétek általában több háló eredményének lineáris kombinációjaként állítják elő az egyes részeredményekből az aggregált választ. A lineáris kombináció hasznossága intuitíve is alátámasztható, azonban megfelelő feltételek mellett a megoldás indokoltsága formálisan is igazolható.

A lineáris kombináció kialakítása – egyszerűsége ellenére – többféleképpen történhet. A következőkben előbb egy fix súlyozású lineáris kombinációt alkalmazó megoldást mutatunk be, ahol a súlyok optimális értékének meghatározására törekszünk, majd azt mutatjuk meg, hogy megfelelő feltételek mellett az eredő megoldás az egyes megoldások bármelyikénél is jobb lehet. Végül ennek a résznek a befejezéséként egy olyan moduláris architektúrát mutatunk be, ahol a feladat dekomponálása, illetve az egyes modulok eredményeihez rendelhető súlyok meghatározása a teljes háló konstrukció részeként „automatikusan”, a háló tanítása során alakul ki.

2.1. Hálók optimális lineáris kombinációja

A hálók lineáris kombinációjának alapötletét az adta, hogy egy gyakorlati feladat megoldásánál szinte minden esetben több hálót konstruálunk, és ezek közül választjuk ki a legjobbat. Ennek alapvető oka, hogy az „optimális” hálót általában nem tudjuk megtervezni. Egy adott feladat megoldására alkalmas háló konstrukciójának nem minden lépését tudjuk a feladtból egyértelműen származtatni, így azt sem tudjuk általában eldönteni, hogy mit tekinthetünk optimális hálóarchitektúrának. Egy megfelelően jó megoldás eléréséhez ezért valójában több, esetenként meglehetősen sok, kisebb vagy nagyobb mértékben eltérő hálót

hozunk létre. Ezek a hálók különbözhetnek a felépítésükben – akár eltérő hálótípusokkal (pl. MLP, RBF, SVM, stb.) is próbálkozhatunk –, de lehet, hogy csak a háló méretében vagy a tanítás módjában, esetleg a felhasznált tanítókészlet összeállításában van különbség. Ha csak a „legjobb” hálót tartjuk meg és a többit eldobjuk, akkor a „mellékeredményeket” csak részben hasznosítjuk. A mellékeredmények ugyanis nemcsak abból a szempontból tekinthetők fontosnak, hogy kialakításuk a végleges háló konstrukcióját segíti, hanem abból a szempontból is, hogy az egyes megoldások a teljes feladat bizonyos tartományain akár a végleges megoldásnál jobb eredményt is nyújthatnak. Ezt a „képeségüket” is célszerű hasznosítani.

A hálók lineáris kombinációjánál több háló kimenetének súlyozott összegét képezzük [Has97]:

$$\bar{y}(\mathbf{x}, \alpha) = \sum_{j=0}^M \alpha_j y_j(\mathbf{x}) \quad (9.10)$$

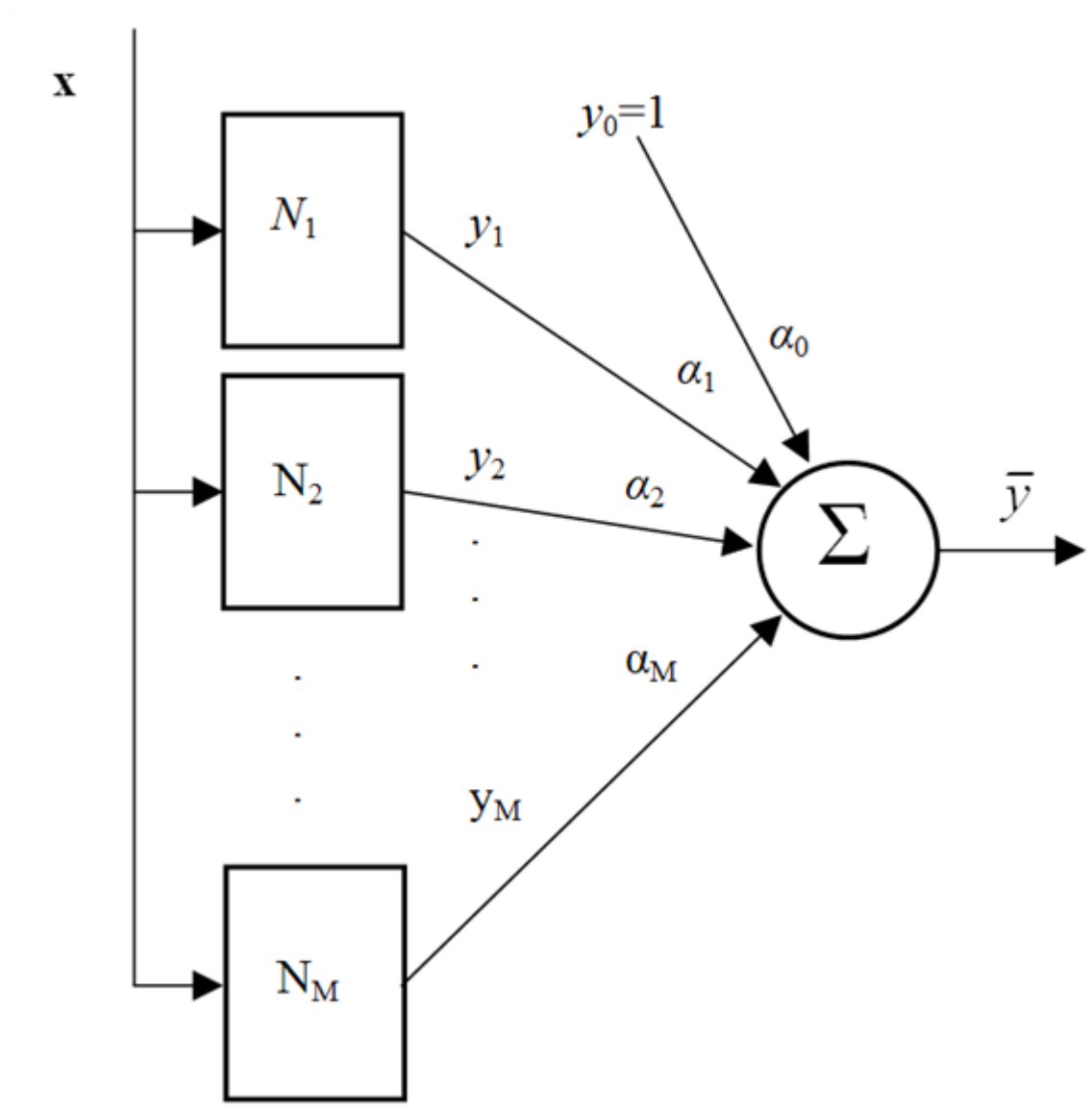
ahol az $\alpha_j, j=0, 1, \dots, M$ együtthatók a lineáris kombináció súlytényezői.

A teljes moduláris háló a 9.6 ábrán látható.

A hálóegyüttesben M modul található, a kimeneti súlyozott összegzésben pedig még egy eltolásérték (*bias*) is szerepel ($y_0(\mathbf{x}) \equiv 1$ és α_0 képviseli az eltolásértéket). Az ábrából látható, hogy a hálók lineáris kombinációja úgy is tekinthető, mint egy olyan többrétegű háló, melynek az utolsó kimeneti rétege lineáris. Ha az egyes modulok MLP-k, akkor az egész moduláris kialakítás egy többrétegű perceptron. A moduláris kialakítás azonban egy általános többrétegű perceptronhoz képest több lényeges eltérést mutat. Egyrészt az egyes modulok kialakítása egymástól függetlenül történik, másrészt – ahogy ezt a továbbiakban látni fogjuk – a kimenő, integráló réteg súlyait nem a hagyományos módon és nem a modulok tanításával együtt tanítjuk.

Ha a hálóegyüttes kialakításánál a fenti gondolatmenetből indulunk ki, akkor hálók lineáris kombinációjánál különállóan megtanított hálókból építjük fel a moduláris hálót. Ez azt jelenti, hogy az egyedi hálók tanításán túl további feladatokat kell megoldani. Meg kell határozni, hogy hány modulból álljon a moduláris rendszer és azt is, hogy a rendelkezésre álló egyedi hálók közül melyek kerüljenek kiválasztásra, majd meg kell határozni a lineáris kombináció súlyait.

9.6. ábra - Hálók lineáris kombinációja több-bemenetű – egykimenetű esetben



2.1.1. Az egyedi hálók kiválasztása

Az egyedi hálók számának meghatározására nincs általános módszer. A hálóegyüttes létrehozásának elsődleges célja a megoldás minőségének javítása, így csak a teljes moduláris háló kiértékelése alapján dönthetjük el, hogy adott számú modulból felépülő hálóegyüttes egy adott feladat megoldása szempontjából megfelelő-e vagy sem. Egyfelől azt mondhatjuk, hogy a modulok számának növelése a teljes rendszer általánosítóképességét várhatóan javítja, másfelől a túl sok modul túl sok erőforrást igényel, ami az alkalmazás korlátja is lehet.

Az általánosítóképesség javulását intuitíve a kimeneti súlyozott összegző átlagoló hatásával magyarázhatjuk, másrészt a következő alfejezetben – legalábbis bizonyos speciális súlyozás esetére – formálisan is alátámasztjuk. A túl sok modul összességében túl sok szabad paramétert jelent, és – amint ezt az előző fejezetekben láttuk – a túl sok szabad paraméter általában túlilleszkedésre vezethet. A túlilleszkedés veszélye a hálóegyüttesnél azonban nem jelentős, hiszen itt olyan modulok összekapcsolásáról van szó, melyeket külön-külön alakítottunk ki, így ha ezek a modulok önmagukban nem mutatnak túlilleszkedési hajlamot, akkor ez a veszély a súlyozott összegükre sem áll fenn. Másfelől hálóegyütteseknél az egyedi hálónál jelentkező túlilleszkedés még előnyös is lehet, ahogy erre a következőkben utalni fogunk.

A modulok számára korlátot inkább az jelent, hogy az egyes modulok mennyiben hoznak új eredményt, illetve milyen kapcsolat van az egyes modulok válaszai között. Ha két vagy több modul kimenetei lineárisan összefüggők, akkor a moduláris kiépítés redundáns, ráadásul az ilyen helyzet az integráló réteg súlyainak meghatározását is nehezíti.

2.1.2. Az integráló réteg súlyainak meghatározása

A kimeneti réteg súlyait leggyakrabban a kimeneti átlagos négyzetes hiba minimalizálása útján határozzuk meg. Ekkor – mivel a modulokat integráló réteg szempontjából az egyes modulok kimenetei lesznek a bemenetek – a súlyozó hálózat nem más, mint egy adaline, melynek optimális súlyait a Wiener-Hopf egyenlet alapján határozhatjuk meg. E szerint:

$$\mathbf{a}^* = \mathbf{R}_{yy}^{-1} \mathbf{p} \quad (9.11)$$

ahol

$$\mathbf{R}_{yy} = E\{\mathbf{y} \mathbf{y}^T\} \quad \text{és} \quad \mathbf{p} = E\{d \mathbf{y}\} \quad (9.12)$$

a modulok kimeneteiből képzett vektornak ($\mathbf{y} = [y_1, y_2, \dots, y_M]^T$) autokorrelációs mátrixa, illetve a modulok kimeneteinek és a kívánt kimenetnek a keresztkorrelációs vektora. A megoldáshoz az $\mathbf{R}_{yy}$ mátrix becslésére és a mátrix inverzének a kiszámítására van szükség.

A súlyok optimális értékének meghatározása után a következőkben arra keressük a választ, hogy az optimális lineáris kombináció eredményeképp kapott hálóegyüttessel milyen eredmény érhető el. Több hálóból álló hálóegyüttes alkalmazásának ugyanis csak akkor van értelme, ha eredőben jobb eredményt érhetünk el, mint amit bármelyik egyedi háló elérne.

2.2. Pontos és különböző szakértők együttese

Hálóegyüttesnél az egyes hálók által szolgáltatott megoldásoknak valamilyen mértékben különbözniük kell egymástól, ugyanis azonos eredmények bármilyen súlyokkal történő súlyozott összegeként is csak ezt az azonos választ kaphatjuk. Eltérő válaszok azonban lehetőséget adnak arra, hogy az egyedi válaszok kombinálásaként az egyes válaszok egymást segítve, egymást kiegészítve, összességében a teljes feladatra nézve jobb megoldást szolgáltatassanak, mint bármelyik egyedi megoldás. A hálóegyüttes alkalmazásának kulcseleme tehát az egyedi megoldások különbözősége. A következőkben ezt a kérdést próbáljuk egy kicsit részletesebben megvizsgálni.

Egy hálóegyüttes vizsgálatánál induljunk ki abból, hogy van M hálónk, melyek mindegyike a teljes feladatot oldja meg. Jelöljük ezeknek a hálónak a kimenetét adott $\mathbf{x}$ bemenet mellett $y_j(\mathbf{x})$ -szel ($j=1,2,\dots,M$). A hálóegyüttes kimenetét az előző részben alkalmazott eljáráshoz hasonlóan az egyes hálók kimeneteinek súlyozott összegeként határozzuk meg. Jelöljük $\bar{y}(\mathbf{x}, \mathbf{a})$ -val az eredő választ, hiszen a súlyozott kimenet nemcsak az $\mathbf{x}$ bemenetnek hanem az $\mathbf{a}_j$ súlyoknak is a függvénye:

$$\bar{y}(\mathbf{x}, \mathbf{a}) = \sum_{j=1}^M \alpha_j y_j(\mathbf{x}) \quad (9.13)$$

A következőkben – az előző részben alkalmazott megoldástól eltérően – kikötjük, hogy a súlyok legyenek pozitívak és összegük legyen egységnyi, vagyis teljesüljenek a

$$\sum_{j=1}^M \alpha_j = 1 \quad \text{és} \quad \alpha_j > 0 \quad j = 1, 2, \dots, M \quad (9.14)$$

feltételek. Ez a kikötés fontos lesz a továbbiakban.

A hálóegyüttes minősítéséhez meg kell határozni az eredő válasz hibáját. Amennyiben most is négyzetes hibakritériumot alkalmazunk, adott mintapontban az eredő válasz hibáját

$$e(\mathbf{x}) = \varepsilon^2(\mathbf{x}) = [d(\mathbf{x}) - \bar{y}(\mathbf{x}, \mathbf{a})]^2 \quad (9.15)$$

adja meg. Az eredő hiba mellett meghatározhatjuk az egyes egyedi megoldások négyzetes hibáit is. A j -edik modul hibája ennek megfelelően:

$$e_j(\mathbf{x}) = \varepsilon_j^2(\mathbf{x}) = [d(\mathbf{x}) - y_j(\mathbf{x})]^2 \quad (9.16)$$

Minthogy az egyes megoldások különbözni fognak egymástól, célszerű, ha meghatározzuk az egyes megoldásoknak az eredő megoldástól való négyzetes eltérését, különbözőségét is. Jelöljük ezt $a(\mathbf{x})$ -szel. (A jelölés az angol *ambiguity* elnevezésre utal, mely elnevezést itt a különbözőség megjelölésére használunk.)

$$a_j(\mathbf{x}) = [y_j(\mathbf{x}) - \bar{y}(\mathbf{x}, \mathbf{a})]^2 \quad (9.17)$$

Az egész hálóegyüttesre értelmezhetünk egy eredő különbözőséget, mint az egyes hálókra számított különbözőségek súlyozott összegét:

$$\bar{a}(\mathbf{x}, \mathbf{a}) = \sum_{j=1}^M \alpha_j a_j(\mathbf{x}) = \sum_{j=1}^M \alpha_j [y_j(\mathbf{x}) - \bar{y}(\mathbf{x}, \mathbf{a})]^2 \quad (9.18)$$

Látható, hogy az eredő különbözőség egy súlyozott variancia jellegű mennyiség.

Ha az egyedi hibák súlyozott átlagát az alábbi összefüggés szerint definiáljuk

$$\bar{e}(\mathbf{x}, \mathbf{a}) = \sum_{j=1}^M \alpha_j e_j(\mathbf{x}) \quad , \quad (9.19)$$

a hálóegyüttes eredő válaszáinak négyzetes hibája a következő formában is felírható:

$$e(\mathbf{x}) = [d(\mathbf{x}) - \bar{y}(\mathbf{x}, \mathbf{a})]^2 = \bar{e}(\mathbf{x}, \mathbf{a}) - \bar{a}(\mathbf{x}, \mathbf{a}) \quad (9.20)$$

Az eddigi összefüggések által meghatározott mennyiségek adott $\mathbf{x}$ bemenet mellett jellemezték az egyedi hálózatokat illetve a hálóegyüttest. Az összes lehetséges bementre integrálva a fenti összefüggéseket meghatározhatjuk a teljes feladatra is az eredő hibát, az egyedi hibák súlyozott átlagát és a különbözőségek súlyozott átlagát.

$$E = \int_{\mathbf{x}} e(\mathbf{x}, \mathbf{a}) p(\mathbf{x}) d\mathbf{x} \quad (9.21)$$

$$\bar{E} = \int_{\mathbf{x}} \bar{e}(\mathbf{x}, \mathbf{a}) p(\mathbf{x}) d\mathbf{x} \quad (9.22)$$

$$\bar{A} = \int_{\mathbf{x}} \bar{a}(\mathbf{x}, \mathbf{a}) p(\mathbf{x}) d\mathbf{x} \quad (9.23)$$

Ennek alapján a hálóegyüttes általánosítási hibájára is érvényes az egy mintapontra meghatározott (9.20) összefüggés megfelelője.

$$E = \bar{E} - \bar{A} \quad (9.24)$$

A (9.24) összefüggés fontos jellemzője, hogy ebben szétválasztva szerepel az egyes hálók általánosítási hibáitól függő tag és a hálók válaszaik közötti korrelációtól függő tag. Az első tag ugyanis az egyes hálók hibáitól függ, tehát ebben a tagban szerepet kapnak a hálóknak a kívánt választól való eltérései, míg a második tag anélkül is meghatározható, hogy ismernénk a kívánt válaszokat. A (9.24) összefüggés a torzítás-variancia dilemma (ld. 2. fejezet) egyfajta megfogalmazásának is tekinthető. Ugyanis, ha a hálóegyüttes válasza torzított, ez arra utal, hogy a különálló válaszok meglehetősen hasonlóak (és torzítottak is) még azokra a bemenetekre is, melyeket nem használtunk fel a tanításnál. Ilyenkor az eredő megoldás általánosítási hibája alapvetően meg fog egyezni az egyes megoldások általánosítási hibáinak súlyozott összegével. Ha ugyanakkor a variancia nagy, vagyis az

egyes megoldások jelentős mértékben eltérnek egymástól, az eredő megoldás általánosítási hibája kisebb lehet, mint az általánosítási hibák súlyozott összege. Általánosságban is igaz, hogy

$$E \leq \frac{1}{M} \sum_{j=1}^M E_j \quad (9.25)$$

ahol a j -edik háló teljes hibája

$$E_j = \int_{\mathbf{x}} e_j(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \quad (9.26)$$

Az előbbiekből közvetlenül következik, hogy az eredő hiba úgy redukálható, ha az egyes hálók válaszai minél inkább különböznek, vagyis a (9.24) egyenletben a második tag egyre nagyobb, feltéve, ha közben biztosítani tudjuk, hogy az első tag ne növekedjen. Az eredő válasz négyzetes hibájának felírásából látható, hogy a négyzetes kifejezés miatt az semmiképp nem fordulhat elő, hogy (9.24) negatív eredményt adjon, bármekkora is a különбözőség az egyes különálló válaszok között.

A (9.24) összefüggés az ún. **pontos és különböző** (*accurate and diverse*) elv megfogalmazásának az alapja [Kro95], [Sol96]. Ennek megfelelően egy hálóegyüttes kialakításánál arra kell törekedni, hogy az egyes megoldások külön-külön minél pontosabbak és egymástól minél inkább eltérőek legyenek.

A különböző, de pontos eredmények elérésének számos módja van. Általában különböző hálótípusokat alkalmazhatunk, vagy ha az összes háló azonos típusú, akkor a hálók méretében vagy a tanítókészlet összeállításában, esetleg a tanítás módjában lehet különbség. Különböző megoldást kaphatunk akkor is, ha eltérő kezdeti értékeket alkalmazunk.

A hálóegyütteseknél a túlilleszkedés – amit az egyedi hálók esetében hátrányként említettünk – még előnyös is lehet. Ez azt jelenti, hogy a regularizáció szerepe, mely a túltanulás veszélyét hivatott mérsékelni itt kevésbé fontos.

Bár az eddigiekben mindig hálóegyüttesről beszéltünk, egy moduláris rendszer létrehozásánál az egyes megoldások akár eltérő megközelítés alkalmazásával is születhetnek. Például neuronhálós megoldások mellett fuzzy, szabályalapú, vagy más alapokon nyugvó megoldások együttesét is kialakíthatjuk. Ilyenkor hálóegyüttes helyett inkább szakértőegyüttesről beszélhetünk.

2.3. MOE (Mixture Of Experts) architektúra

A MOE (Mixture Of Experts) [Jac91], [Jor94] architektúra egy olyan speciális hálóegyüttes (szakértőegyüttes), ahol az előbbiekből bemutatott hálóegyütteshez hasonlóan szintén az egyes hálók válaszainak súlyozott összegeként kapjuk az eredő választ, azonban itt a súlyokat nem rögzítjük, hanem egy ún. kapuzóhálózat kimenetén nyerjük. Ellentétben ugyanis az eddigiekben vizsgált hálóegyüttesektől, amelyekben a mintahalmazra adott válaszok alapján egy általánosan minden bemenet esetén használt (az adott feladatra konstans) súlykészletet határozzunk meg, itt a súlyok közvetlenül függenek a bemenettől is. A kapuzóhálózat valójában maga is egy nemlineáris több-bemenetű-többkimenetű tanítható rendszer, mely bemenetként szintén a háló bemeneteit kapja, kimenetként pedig a súlyozó együtthatók értékeit, a kapuzóértékeket szolgáltatja.

A MOE specialitását részben épp ez a kapuzóhálózat adja, hiszen ennek kialakítása biztosítja, hogy az előzőekben tárgyalt hálóegyüttestől eltérően itt a modulok kimeneteit nem fix súlyozással fogjuk össze, hanem bemenetfüggő kapuzóértékekkel. A bemenetfüggő súlyozás indítéka nyilvánvaló: feltételezhető, hogy az egyes szakértők a bemeneti tér különböző tartományain különböző minőségben oldják meg a feladatot, így, ha az egyes szakértőknek az eredő kimenethez való hozzájárulását a teljes feladatra azonos módon rögzítenénk, biztosan nem kapnánk optimális megoldást. Célszerűbb, ha megengedjük, hogy az egyes szakértők szerepe bemenetfüggő legyen, vagyis az eredő válaszhoz való hozzájárulásuk a problémátér egyes tartományaiban kisebb, más tartományaiban pedig nagyobb legyen. A bemenetfüggő súlyozás a részeredmények rugalmas aggregálását teszi lehetővé, hiszen ebbe a keretbe belefér az is, ha a választ a szakértők válaszáának egyszerű átlagaként kapjuk, és az is, ha egyes tartományokban az eredő választ csak egyetlen háló határozza meg.

A kapuzóértékek maguk is függvényei a bemenetnek, sőt a kapcsolat a bemenet és a kapuzóértékek között nemlineáris, ezért a MOE architektúra akkor is nemlineáris leképezést valósít meg, ha az egyes szakértők önmagukban lineárisak. A MOE architektúránál általában meg is engedjük, hogy az egyes szakértők egyszerű

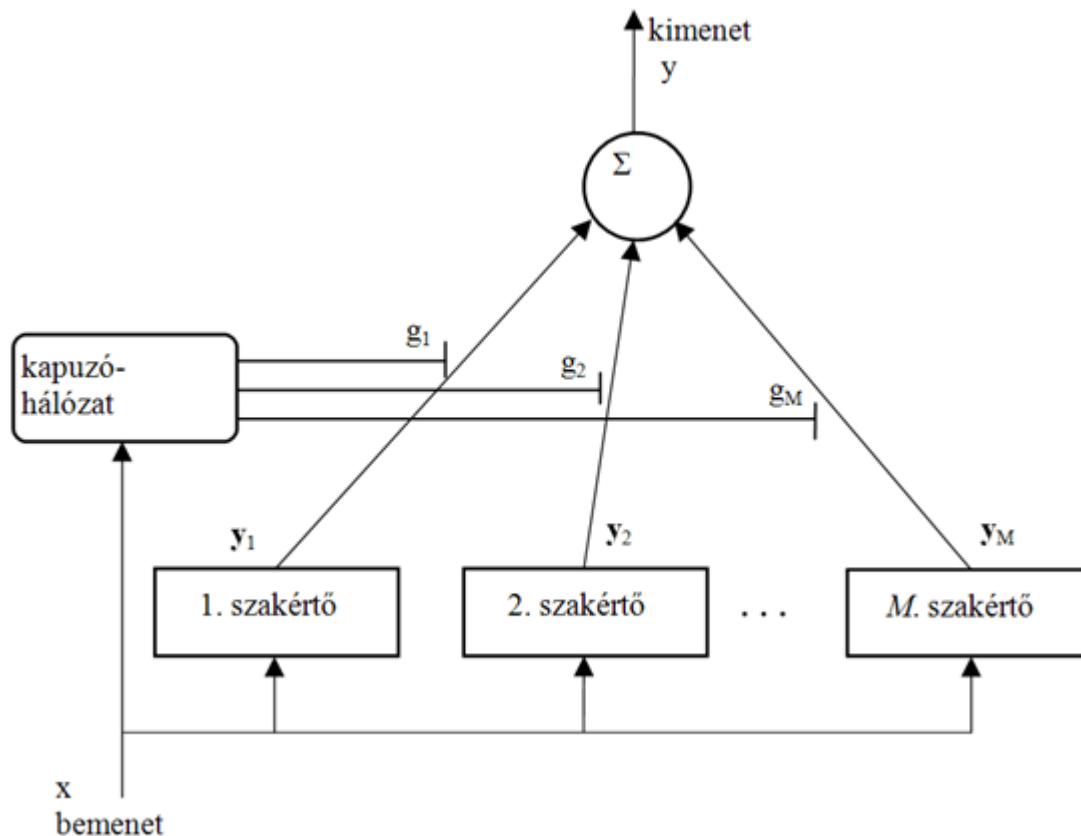
felépítésűek, pl. lineáris hálózatok legyenek. A továbbiakban azonban a bemenetek és a kimenetek között az általánosabb

$$\mathbf{y}_i = f(\mathbf{x}, \Theta_i) \quad i=1,2,\dots,M, \quad (9.27)$$

kapcsolatot tételezünk fel, ami valamilyen statikus, lineáris vagy nemlineáris leképezést jelent, de dinamikus hálóból is képezhetünk szakértőegyüttest (ld. pl. [Zee97]). A (9.27) összefüggésben Θ_i az i -edik szakértő paramétereit összefogó mátrixot jelöli.

A MOE hálózatnál minden szakértő a teljes feladatot oldja meg, így az eredmény az egyes modulok közötti versengés és együttműködés eredményeként születik meg.

9.7. ábra - A MOE (Mixture of Experts) architektúra



A hálózat felépítését a 9.7 ábra mutatja. Az $\mathbf{x}$ bemenetet mindegyik szakértő (egyedi háló) és a kapuzó hálózat is megkapja. Az i -edik háló kimenetét $\mathbf{y}_i$ jelöli, míg $\mathbf{y}$ lesz a teljes hálóegyüttes kimenete. A súlyozott összeg képzéséhez a g_i súlyokat a kapuzóhálózat állítja elő.

A teljes szakértőegyüttes kimenete ennek megfelelően:

$$\mathbf{y} = \sum_{i=1}^M g_i \mathbf{y}_i \quad (9.28)$$

A többdimenziós kimenet (9.28) összefüggése azt jelzi, hogy a kimenet egyes komponenseinek azonos a súlyozása, tehát a kapuzóhálózatot nem komponensenként, hanem a teljes kimeneti vektorra együtt alkalmazzuk. A MOE architektúránál is kikötjük, hogy a g_i kapuzóértékekre teljesüljenek az alábbi feltételek:

$$\sum_{i=1}^M g_i = 1 \quad \text{és} \quad 0 \leq g_i \leq 1 \quad i = 1, 2, \dots, M \quad (9.29)$$

Ezeknek a feltételeknek a háló működésének valószínűségi értelmezésénél lesz szerepük.

A (9.29) feltételeknek eleget tevő kapuzóértékeket sokféleképpen elő lehet állítani. A MOE hálózatnál leggyakrabban az ún. „softmax” függvényt alkalmazzuk:

$$g_i = \frac{e^{\xi_i}}{\sum_{j=1}^M e^{\xi_j}}, \quad (9.30)$$

ahol a ξ_i közbelső változót az $\mathbf{x}$ bemenővektor és egy $\mathbf{v}_i$ paramétervektor skalár szorzataként kapjuk: $\xi_i = \mathbf{v}_i^T \mathbf{x}$.

A softmax függvény hatása valójában az lesz, hogy a MOE a bemeneti tartományt „lágyan” particionálja, olyan tartományokat határoz meg, melyekhez eltérő kapuzóértékek tartoznak, de amely tartományok határai általában elmosódottak. (A g_i értékek az i -edik szakértő esetén tipikusan egy tartományon dominánsak, közel 1 értékűek, de nem változnak meredeken sem $0 \rightarrow 1$, sem $1 \rightarrow 0$ irányban a tartomány határain.) A MOE architektúra ezért úgy is felfogható, mintha a feladatot a bemeneti tér particionálása alapján dekomponáltuk volna, ahol azonban a particionálás nem éles határokkal történik.

A lágy particionálás könnyen interpretálható egy olyan architektúra esetén, ahol csak két szakértőnk van. Az ezekhez tartozó két kapuzóérték g_1 és g_2 . Írjuk fel g_1 kifejezését (nyilvánvalóan $g_2 = 1 - g_1$):

$$g_1 = \frac{e^{\xi_1}}{e^{\xi_1} + e^{\xi_2}} = \frac{1}{1 + e^{-(\mathbf{v}_1 - \mathbf{v}_2)^T \mathbf{x}}} \quad (9.31)$$

ami láthatóan egy olyan logisztikus gerincfüggvény (*logistic ridge function*), melynek orientációját a $\mathbf{v}_1 - \mathbf{v}_2$ vektor iránya szabja meg. Mindazon $\mathbf{x}$ -ekre, melyek $(\mathbf{v}_1 - \mathbf{v}_2)$ -re merőlegesek a két kapuzóérték megegyezik: $g_1 = g_2 = 1/2$, vagyis ezekre a bemenetekre, melyeknél a logisztikus függvény gerince található, a MOE válasza a két szakértő válaszáinak átlaga. A gerinctől távolodva az egyik irányban az első szakértő, a másik irányban a második szakértő szerepe kezd nőni, mindig a másik szakértő rovására. A gerinc meredeksége $\mathbf{v}_2 - \mathbf{v}_1$ nagyságától függ. Ha $\mathbf{v}_2 - \mathbf{v}_1$ nagy, a gerinc meredek, a két szakértő szerepe élesen elhatárolódik egymástól, míg kis értéknél az elválasztás lágy, a szakértők „kompetencia-területei” erősen átlapolódnak.

A MOE architektúra többszintű, hierarchikus hálóegyüttessé is kiterjeszthető, amikor a szakértőkből több csoportot alkotunk és ezeket a csoportokat külön-külön egy-egy MOE architektúrába fogjuk össze, majd a MOE kimenetekből egy újabb kapuzóhálózattal állítjuk elő a hierarchikus rendszer kimenetét. A hierarchikus MOE (HMOE) tehát nem a szakértőkből képez hierarchikus rendszert, hanem a kapuzóhálózatokból. A HMOE egy fa struktúrájú moduláris háló, melynek csak a leveleinél vannak szakértő modulok.

2.3.1. A MOE tanítása

A MOE architektúrának két, tanítással meghatározandó paraméter-készlete van. A Θ_i paramétermátrixok a szakértők szabad paramétereiből állnak, míg a $\mathbf{v}_i$ vektorok a softmax függvényen keresztül határozzák meg a kapuzóhálózat leképezését. A tanítást most is egy $Z = \{\mathbf{z}_i\}_{i=1}^L = \{(\mathbf{x}_i, \mathbf{d}_i)\}_{i=1}^L$ tanítópontkészlet alapján tudjuk elvégezni.

A háló tanítása többféle megközelítésben is lehetséges. Amennyiben az egyes szakértők neuronhálók, mind a Θ_i szakértő-paraméterek, mind a $\mathbf{v}_i$ kapuzóhálózat-paraméterek a speciális MOE architektúrára alkalmazott hibavisszaterjesztési algoritmussal taníthatók, hiszen a kimenet mind a Θ_i , mind a $\mathbf{v}_i$ paraméterek folytonos függvénye. Ha a tanításnak ezt a megközelítését alkalmazzuk, akkor a MOE valójában nem több, mint egy speciális felépítésű komplex hálóarchitektúra. Ebben a megközelítésben a kapuzóértékekre megfogalmazott (9.29) feltételeknek nincs is különösebb jelentőségük, a feltételektől akár el is tekinthetünk. Ennél érdekesebb azonban, ha az architektúrához valószínűségi értelmezést rendelünk. A valószínűségi értelmezés a háló működését új megvilágításba helyezi, sőt a háló tanítása is más értelmezést nyer. A valószínűségi értelmezés mellett a háló tanítása maximum likelihood becslési problémaként fogalmazható meg. A következőkben ezt az interpretációt és az ennek megfelelő tanítási eljárást foglalkoztatjuk össze.

2.3.2. Tanítás valószínűségi értelmezés alapján

A valószínűségi értelmezésben mind a kapuzóhálózatok, mind a szakértők szerepe megváltozik. A kapuzóhálózat úgy is felfogható, mint egy olyan osztályozó, ami az $\mathbf{x}$ bemenővektort egy multinomiális valószínűségi változóra képezi le. A kapuzóértékekre megfogalmazott (9.29) feltételek biztosítják, hogy a ξ_i értékek valószínűségekként is értelmezhetők legyenek. g_i értelmezhető, mint az a valószínűség, hogy a kimenet az i -edik szakértőtől származik:

$$g_i = P(i | \mathbf{x}, \mathbf{v}_i) \quad (9.32)$$

A kapuzóhálózat szerepe ekkor azt eldönteni, hogy melyik szakértő kimenete kerüljön ki a teljes moduláris rendszer kimenetére. A HMOE esetében több döntési szintet különböztethetünk meg, tehát egy hierarchikus döntési szekvencia eredménye az, hogy a kimenet melyik szakértőtől származik. Amennyiben ez a döntés vagy döntés-sorozat megszületett, feltételezhetjük, hogy a $\mathbf{d}$ kimenetet egy statisztikai modell generálta. Ez azt jelenti, hogy $\mathbf{d}$ valószínűségi változónak tekinthető, melyet valamilyen P valószínűség-sűrűségfüggvénnyel jellemezhetünk. Ennek megfelelően az egyes szakértőkhöz is valószínűségi értelmezés rendelhető és egy szakértő kimenete úgy interpretálható, mint a $\mathbf{d}$ valószínűségi változó feltételes várható értéke.

Ha a kimenet származtatását e szerint a valószínűségi értelmezés szerint nézzük, akkor először a szakértők választ kell tekintenünk. Egy szakértő a valószínűségi modell alapján az

$$\mathbf{y}_i = E(\mathbf{d} | \mathbf{x}, \Theta_i) \quad (9.33)$$

feltételes várhatóértéknek megfelelően állít elő egy kimenetet, ahol a várhatóérték-képzést egy $P(\mathbf{d} | \mathbf{x}, \Theta_i)$ feltételes sűrűségfüggvény szerint képezzük. A következő lépésben egy szakértőkimenet a kapuzóértéknek megfelelő valószínűséggel kerül a teljes háló kimenetére (vagy HMOE esetében a következő hierarchiaszintre).

Ebből a valószínűségi modellből kiindulva annak valószínűsége, hogy adott $\mathbf{x}_l$ bemenetre $\mathbf{d}_l$ választ kapunk a teljes valószínűségek tétele alapján az alábbi formában írható fel:

$$\begin{aligned} P(\mathbf{d}_l | \mathbf{x}_l, \Theta^*) &= \sum_i g_i(\mathbf{x}_l, \mathbf{v}_i^*) P(\mathbf{d}_l | \mathbf{x}_l, \Theta_i^*) \\ &= \sum_i P(i | \mathbf{x}_l, \mathbf{v}_i^*) P(\mathbf{d}_l | \mathbf{x}_l, \Theta_i^*) \end{aligned} \quad (9.34)$$

A (9.34) összefüggésben a paraméterek melletti * azt jelzi, hogy itt a paraméterek „igazi”, de számunkra ismeretlen értékei szerepelnek. A Θ^* az összes szakértő Θ_i^* paramétereit összefogó eredő paramétermátrix. Amennyiben az igazi értékek helyett valamilyen Θ_i és $\mathbf{v}_i$ paramétereink vannak, az l -edik tanítóponthoz tartozó bemenet és kimenet közötti valószínűségi kapcsolatot a

$$P(\mathbf{d}_l | \mathbf{x}_l, \Theta) = \sum_i P(i | \mathbf{x}_l, \mathbf{v}_i) P(\mathbf{d}_l | \mathbf{x}_l, \Theta_i) \quad (9.35)$$

feltételes sűrűségfüggvény adja meg. A teljes tanítókészletre is felírhatjuk a valószínűségi kapcsolatot:

$$P(\mathbf{d} | \mathbf{x}, \Theta) = \prod_{l=1}^L P(\mathbf{d}_l | \mathbf{x}_l, \Theta) = \prod_{l=1}^L \left[\sum_{i=1}^M P(i | \mathbf{x}_l, \mathbf{v}_i) P(\mathbf{d}_l | \mathbf{x}_l, \Theta_i) \right] \quad (9.36)$$

A (9.36) összefüggés annak valószínűségét adja meg, hogy az adott $\mathbf{x}_l$ bemenet mellett a megfelelő $\mathbf{d}_l$ kívánt választ kapjuk ($l=1,2,\dots,L$), feltéve, ha a szakértőink paramétermátrixai a Θ_i mátrixok, a kapuzóhálózatunk paramétervektorai pedig a $\mathbf{v}_i$ vektorok ($i=1,2,\dots,M$).

A (9.36) összefüggés képezi a paraméterek meghatározásának, a MOE tanításának az alapját. Keressük ugyanis azokat a Θ_i mátrixokat és $\mathbf{v}_i$ vektorokat, melyek mellett a (9.36) valószínűség-sűrűségfüggvény épp az adott

tanítóponthalmaz esetén veszi fel a maximumát. A feladatot tehát visszavezettük egy maximum likelihood becslési problémára, ahol a likelihood függvényt épp (9.36) adja meg. A tanító összefüggések meghatározásának további lépéseiben a likelihood függvény helyett célszerű annak logaritmusával, a log-likelihood függvénnyel dolgozni.

$$\mathcal{L}(\mathbf{x}, \Theta) = \log P(\mathbf{d} | \mathbf{x}, \Theta) = \sum_{l=1}^L \log \left[\sum_{i=1}^M P(i | \mathbf{x}_l, \mathbf{v}_l) P(\mathbf{d}_l | \mathbf{x}_l, \Theta_i) \right] \quad (9.37)$$

A maximum likelihood becslés eredményét a (log-)likelihood függvény maximumát biztosító paramétervektorok szolgáltatják. A szélsőérték-keresést itt is gradiens módszerrel végezve meg kell határoznunk a

$$\frac{\partial \mathcal{L}(\mathbf{x}, \Theta)}{\partial \Theta_i} = \mathbf{0} \quad \text{és a} \quad \frac{\partial \mathcal{L}(\mathbf{x}, \Theta)}{\partial \mathbf{v}_i} = \mathbf{0} \quad (9.38)$$

likelihood egyenletek megoldásait. (Itt a Θ_i mátrixokat formálisan vektorra alakítva használjuk.) A gradiens-számításnál alkalmazva a láncszabályt a Θ_i szerinti gradienst $\mathbf{y}_i$ -n keresztül, a $\mathbf{v}_i$ szerinti gradienst pedig ξ_i -n keresztül határozhatjuk meg:

$$\frac{\partial \mathcal{L}(\mathbf{x}, \Theta)}{\partial \Theta_i} = \frac{\partial \mathcal{L}(\mathbf{x}, \Theta)}{\partial \mathbf{y}_i} \frac{\partial \mathbf{y}_i}{\partial \Theta_i} \quad \text{és} \quad \frac{\partial \mathcal{L}(\mathbf{x}, \Theta)}{\partial \mathbf{v}_i} = \frac{\partial \mathcal{L}(\mathbf{x}, \Theta)}{\partial \xi_i} \frac{\partial \xi_i}{\partial \mathbf{v}_i} \quad (9.39)$$

A részletektől eltekintve – feltételezve, hogy a valószínűségi interpretációban Gauss eloszlásunk van – a gradiens eljárás eredményeképp a következő tanító összefüggéseket nyerjük:

$$\Theta_i(k+1) = \Theta_i(k) + \eta \sum_{l=1}^L h_{l,i} (\mathbf{d}_l - \mathbf{y}_{l,i}) \frac{\partial \mathbf{y}_{l,i}}{\partial \Theta_i} \quad (9.40)$$

és

$$\mathbf{v}_i(k+1) = \mathbf{v}_i(k) + \eta \sum_{l=1}^L (h_{l,i} - \xi_{l,i}) \mathbf{x}_l \quad (9.41)$$

ahol η jelöli a tanulási tényezőt, $\mathbf{y}_{i,l}$ jelöli az i -edik szakértő $\mathbf{x}_l$ -re adott választát, $g_{i,l}$ az i -edik kapuzóértéket az l -edik tanítópontnál, míg $h_{i,l}$ -t az alábbi összefüggés definiálja:

$$h_{i,l} = \frac{P(i | \mathbf{x}_l, \mathbf{v}_l) P(\mathbf{d}_l | \mathbf{x}_l, \Theta_i)}{\sum_j P(j | \mathbf{x}_l, \mathbf{v}_j) P(\mathbf{d}_l | \mathbf{x}_l, \Theta_j)} \quad (9.42)$$

A szakértőparaméter tanítására vonatkozó (9.40) összefüggés csak akkor alakítható tovább, ha ismerjük a szakértő felépítését. Lineáris szakértőt feltételezve, ahol a szakértő kimenetét

$$\mathbf{y}_{i,l} = \Theta_i \mathbf{x}_l \quad (9.43)$$

adja meg, a súlymódosító összefüggésre a következőt kapjuk:

$$\Theta_i(k+1) = \Theta_i(k) + \eta \sum_{l=1}^L h_{l,i} (\mathbf{d}_l - \mathbf{y}_{l,i}) \mathbf{x}_l^T \quad (9.44)$$

A (9.44) összefüggést szemügyre véve látható, hogy a szakértő paramétereinek módosítása a szakértő hibájától függ, hiszen az i -edik szakértő kimeneti hibáját az l -edik tanítópontban épp $(\mathbf{d}_l - \mathbf{y}_{l,i})$ adja meg. Hasonló

megállapítás tehető a kapuzóhálózat paramétereinek tanító összefüggésére is, ugyanis a (9.41) összefüggés szerint a súlymódosítás $h_{i,l}$ és $g_{i,l}$ eltérésétől függ, ahol ez az eltérés is hibaként értelmezhető.

A MOE mindkét paraméterképletének súlymódosító összefüggései függenek a (9.42) által definiált $h_{i,l}$ értékektől. Vegyük észre, hogy amennyiben a $g_{i,l} = P(i | \mathbf{x}_l, \mathbf{v}_l)$ valószínűségeket a priori valószínűségeknek tekintjük, a $h_{i,l}$ értékek az a posteriori valószínűségek. Ez (9.42)-ből a Bayes szabály alkalmazásával közvetlenül adódik. A $g_{i,l}$ kiinduló kapuzóérték ugyanis annak a valószínűségét adja meg, hogy adott $\mathbf{x}_l$ bemenet mellett az eredő kimenetet az i -edik szakértő válasza határozza meg. $g_{i,l}$ értékét úgy határozzuk meg (vesszük fel), hogy az $\mathbf{x}_l$ -re adandó kívánt választ, $\mathbf{d}_l$ -t nem ismerjük. Ezzel szemben $h_{i,l}$ azt az állapotot tükrözi, amikor már $\mathbf{d}_l$ -t is felhasználunk, tehát $h_{i,l}$ az l -edik tanító mintapár ismerete utáni állapotra vonatkozik. Ennek megfelelően

$$h_{i,l} = P(i | \mathbf{x}_l, \mathbf{d}_l, \mathbf{v}_l). \quad (9.45)$$

Az (9.41) és az (9.44) összefüggések köteget (batch) tanításra vonatkoznak, vagyis amikor az összes tanítópont felhasználását követően történik a súlymódosítás. Ugyanakkor a MOE tanításánál is alkalmazhatunk mintánkénti súlymódosítást; ekkor a fenti összefüggésekben az l mintapont-index szerinti összegzés elmarad.

A MOE valószínűségi értelmezése lehetővé teszi, hogy a paraméterek meghatározást maximum likelihood paraméterbecslési eljárásra vezessük vissza. A maximum likelihood becslés az előbbiekben bemutatott gradiens alapú eljárásnál hatékonyabban is meghatározható. A MOE architektúrára származtatták az EM (expectation-maximization)² alapú eljárást, mely a számítások egyszerűsítését eredményezi. Ennek részleteivel azonban itt nem foglalkozunk, csak a megfelelő irodalomra ([Jor94]) utalunk.

2.3.3. A MOE konstrukciója

A MOE, illetve a HMOE architektúrák konstrukciójánál is meg kell határozni a moduláris rendszer struktúráját. Ez itt a szakértők számának, illetve hierarchikus MOE esetében a hierarchia szintek számának a meghatározását is jelenti. A MOE/HMOE struktúrájának meghatározása a modellválasztás problémakörbe tartozik, így a modellválasztás eddig említett megközelítéseit alkalmazhatjuk. Leginkább az adatokból kiindulva, az eredmények folyamatos tesztelése útján, a kereszti kiértékelés alkalmazásával határozhatjuk meg a megfelelő MOE architektúrát.

HMOE konstrukciójánál gyakran alkalmaznak olyan eljárást, amely bináris fa struktúrájú hierarchikus moduláris rendszert eredményez. Erre ad lehetőséget például a CART (classification and regression tree) [Bre84] alapú eljárás, ahol a feladatot a kívánt válaszok alapján dekomponáljuk egyre kisebb részfeladatokra úgy, hogy az egyes lépésekben a problémateret mindig két részre hasítjuk. A dekomponálás alapja, hogy a problémateret különválasztott tartományaihoz tartozó mintapontoknál a kívánt válaszok szórása minél kisebb legyen. A hierarchikus dekompozíció egy HMOE architektúrát eredményez, annak ellenére, hogy az egyes szakértők tulajdonképpen továbbra is a teljes feladatot oldják meg.

A MOE architektúráját nagymértékben befolyásolja, hogy az egyes szakértők milyen felépítésűek. Tipikus választás az egyszerű lineáris szakértő. Ebben az esetben az eredő feladat valójában szakaszonként lineáris működést valósít meg, ahol azonban az átmenet az egyes szakaszok között a szoftmax kapuzó függvények következtében folytonos és „legömbölyített”.

Bonyolult, komplex leképezésre képes szakértő modulok alkalmazása már csak azért sem célszerű, mert ekkor nagy a veszélye annak, hogy akár egyetlen szakértő is meg tudja oldani a feladatot. Ha ilyen eset áll elő, akkor ennek az egyetlen szakértőnek megfelelő kapuzóérték a teljes problémateret közel 1 értéket vesz fel, ami ennek a szakértőnek a hasznosítását és a többi indokolatlan alkalmazását jelenti. Az egyszerűbb képességű szakértők „garantálják”, hogy külön-külön egyikük sem képes a feladat megoldására, együttműködésükre feltétlenül szükség van. A MOE előnyeik akkor érvényesülnek, ha a feladat valóban megoszlik az egyes szakértők között. Az előnyök között elsősorban a gyorsabb tanulást kell megemlíteni, de az egyszerű szakértők miatt általában az egész hálózat komplexitása is jóval kisebb lehet, mint egy hasonló képességű egyedi hálózaté.

3. Moduláris háló kialakítása a tanító mintakészlet módosításával

² Az EM-algoritmus rövid összefoglalását a 13. fejezetben adjuk meg.

Az eddigi moduláris háló kialakításoknál akár dekomponáltuk a feladatot, akár a teljes feladatot oldottuk meg különböző hálókkal, minden esetben az eredeti tanítókészlettel vagy annak a dekompozícióhoz tartozó adott részhalmazával dolgoztunk. A következőkben olyan moduláris háló konstrukciókkal foglalkozunk, ahol az egyes modulokat tudatosan eltérő eloszlású adatkészletekkel tanítjuk, miközben továbbra is minden tanítópontunk az eredeti megoldandó problémából származik, bármelyik adatkészletről van is szó.

A tanító mintakészlet módosításával létrehozott moduláris hálónál több hálót tanítunk és minden hálóhoz más-más eloszlású tanítókészletet használunk. A tanítókészlet módosításának legkézenfekvőbb módja az ún. bagging³ eljárás. A bagging eljárás olyan hálókat hoz létre, melyeket az eredeti tanítókészletből véletlenszerű válogatással létrehozott mintakészletekkel tanítunk. Minden egyes újabb hálóhoz létre kell hoznunk egy új tanítókészletet, melynek elemszáma megegyezik az eredeti mintakészlet elemszámával és amelyet az eredeti mintakészletből visszatevése mintavételezéssel hozunk létre. Az így létrehozott mintakészleteket az eredeti készlet bootstrap másolatainak, az egész technikát pedig bootstrap aggregálásnak nevezzük. A bootstrap másolatokra jellemző, hogy az eredeti mintakészletből egyes minták többször is előfordulhatnak benne, míg más minták teljes mértékben kimaradnak. A bootstrap másolatok átlagosan az eredeti mintakészlet 63,2%-át tartalmazzák [Bre96]. A bagging így az egyes hálók tanításához az eredetitől eltérő eloszlású mintakészleteket használ fel.

Az eredeti mintakészletből eltérő eloszlású mintakészletek előállításának további fontos lehetőségét jelentik a boosting eljárások [Scha02]. A boosting valójában fokozást jelent, ami itt arra utal, hogy a boosting eljárás eredményeképpen kapott moduláris háló jobb teljesítményre képes, mint ha az eredeti tanítókészlettel egyetlen hálót hoztunk volna létre. A boosting eljárások tehát a teljesítőképesség javítása érdekében módosítják a tanítókészletek eloszlását. A következőkben a legfontosabb boosting eljárásokat foglaljuk össze, és kitérünk annak rövid bemutatására is, hogy milyen mechanizmus áll a boosting eljárások teljesítőképesség-javító hatása mögött.

3.1. Boosting

Az előzőek alapján látható, hogy a boosting eljárások lényege, hogy a rendelkezésre álló tanítókészlet mintáinak a felhasználásával eltérő eloszlású mintakészleteket hozunk létre azzal a céllal, hogy az így kialakított mintakészletekkel különálló hálókat konstruáljunk. Az eredő megoldás itt is az egyes hálók eredményeinek valamilyen egyesítése útján nyerhető. A boosting eljárások tehát szintén moduláris hálóarchitektúrát eredményeznek, azonban az eddigi moduláris háló-konstrukcióktól lényegesen eltérő módon. A boosting eljárások az egyes modulok tanításához szükséges különböző eloszlású mintakészleteket szűréssel vagy az egyes minták tanítókészleten belüli szerepének módosításával állítják össze. A boosting eljárások osztályozós feladatok megoldására alkalmas moduláris architektúrákat eredményeznek.

3.1.1. Boosting szűréssel

A szűréssel történő boosting eljárás [Scha90] eredményeképpen egy három hálóból álló moduláris architektúrát kapunk, ahol a három háló tanítása három eltérő, de azonos (L) számú mintából álló tanítókészlettel történik. Az eljárás feltételezi, hogy nagyszámú tanítópont áll a rendelkezésünkre, mely mintakészletből az alábbi eljárás során hozzuk létre a három tanítókészletet:

1. A rendelkezésre álló tanítómintákból válasszunk ki véletlenszerűen L mintapontot, és ezzel a tanítókészlettel tanítsuk meg az első hálót.
 2. A megtanított első háló felhasználásával szűrjük ki a rendelkezésre álló és az első készletben fel nem használt mintakészletből újabb L pontot a következő eljárással:
- Generáljunk azonos valószínűséggel 0-t vagy 1-et, vagyis dobjunk fel egy érmét.
 - Ha az eredmény fej (0), vegyünk elő egymás után újabb mintákat, és dolgozzuk fel ezeket a mintákat az első hálóval. Ha egy mintát az első háló helyesen osztályoz, akkor ezt a mintát dobjuk el, sőt dobjuk el a sorban elővett mintákat mindaddig, amíg egy hibásan osztályozott mintához nem érünk. Ezt a tévesen osztályozott mintát adjuk a második háló tanítókészletéhez.

³ A bagging elnevezés a bootstrap aggregation elnevezésből született, ahol a bootstrap önerőből történő képességjavítást jelent. A bagging ezért olyan integrált megoldás, ahol különböző, külső segítség nélkül javított megoldásokból, azok egyesítése útján kapjuk az eredő megoldást.

- Ha az eredmény írás (1), az előzővel ellentétesen kell cselekednünk. Vegyünk elő ismét egymás után újabb mintákat, és nézzük meg ezekre az első háló válaszát. Ha a válasz hibás, a mintát dobjuk el, sőt dobjuk el a további egymást követő mintákat is mindaddig, amíg egy helyesen osztályozott mintát nem kapunk. Most ezt a helyesen osztályozott mintát adjuk a második tanítókészlethez.
- Folytassuk az eljárást, amíg L számú tanítópontot össze nem gyűjtünk. Az így létrehozott készlet képezi a második háló tanítóhalmazát.

Ez a mintapontválogató eljárás biztosítja, hogy a második mintakészletet az első háló 50%-os hibával osztályozza, tehát a második tanítóhalmaz eloszlása biztosan más lesz, mint az első háló tanításához használt mintakészleté.

A második háló megtanítása után alakítsunk ki egy harmadik tanítókészletet. Ehhez új pontokat vegyünk elő a kiinduló mintakészletből és nézzük meg ezekre a pontokra az első két háló válaszát.

- Ha a két háló válasza megegyezik, a mintapontot dobjuk el. Ha a válaszok különböznek, a mintát tartsuk meg és adjuk a harmadik tanítókészlethez.
- Folytassuk a válogató eljárást, amíg L pontot össze nem gyűjtöttünk.

Tanítsuk meg a harmadik hálót az így kapott készlettel.

3. A három hálóból képezzünk egy moduláris hálóegyüttest, ahol az eredő választ egyszerű szavazással nyerhetjük. Az eredmények aggregálásának más módját is javasolták [Dru93]. E szerint a javaslat szerint adjuk össze az egyes hálók kimeneteit (vegyük figyelembe, hogy a hálók kimenetei nem feltétlenül binárisak, hanem a $[0,1]$ zárt intervallumban tetszőleges értéket felvehetnek) és az összeg alapján hozzuk meg az eredő osztálybasorolást. Ez utóbbi változat előnyét gyakorlati vizsgálatok támasztják alá.

A szűréssel dolgozó boosting eljárás hátránya, hogy igen sok tanítópontot igényel. Jelölje rendre L_1 , L_2 és L_3 az első, a második és a harmadik háló tanítókészletének kiválogatásához felhasznált tanítópontok számát. Az első hálónál ki kell választanunk véletlenszerűen L tanítópontot, tehát $L_1=L$. A második és a harmadik tanítókészlethez viszont L -nél sokkal több pontra van szükségünk, hiszen a szűrési szabály szerint bizonyos pontokat el kell dobnunk. Ebből adódóan általában mind L_2 , mind L_3 jóval nagyobb lesz, mint L , vagyis a három L -elemű tanítókészlethez L_{total} -elemű mintakészletből kell kiindulnunk, ahol általában $L_{\text{total}} \gg 3L$. Ha a kiinduló mintapontok (ismert válaszu pontok) száma korlátozott, akkor a megfelelő mintaszámú három tanítókészlet kiválogatása nehézségekbe ütközhet.

A nagy kiinduló mintapontszámra elsősorban azért van szükség, mert a tanítókészletek nem tartalmaznak közös mintapontokat. A következőkben bemutatandó AdaBoost eljárás ezt a hátrányt kiküszöböli, sokkal hatékonyabban használja föl a kiinduló mintakészletet, így összességében jóval kevesebb mintával is tud dolgozni.

3.1.2. AdaBoost

Az AdaBoost eljárás [Scha99] a mintapontok újtramintavételezésével biztosítja a tanítópontkészlet eloszlásának módosítását. A legfontosabb különbség az előzőekben bemutatott mintaszűréssel dolgozó boosting eljáráshoz képest, hogy megengedi a mintapontok újra felhasználását. Valójában az egyes tanítópontoknak az egyes hálók tanításánál betöltött szerepét változtatjuk.

Az eljárás bemutatásához induljunk ki abból, hogy rendelkezésünkre áll egy $Z = \{\mathbf{z}_l\}_{l=1}^L = \{(\mathbf{x}_l, d_l)\}_{l=1}^L$ tanítókészlet, ahol feltesszük, hogy a d_l kívánt válaszok a $\{-1, +1\}$ értékkészletből vehetnek fel értékeket, vagyis kétféle osztályozási problémával állunk szemben. A mintakészletben a tanítópontok valamilyen eloszlásnak megfelelően fordulnak elő. Az AdaBoost eljárás a tanítópontok eloszlását – az eredeti tanítópontoknak az adott mintahalmazban való előfordulási gyakoriságát – változtatja adaptív módon. Innen ered az eljárás elnevezése is. Az eljárás iteratív módon működik: minden iterációban egy új tanuló rendszert (neuronhálót) hozunk létre egy módosított eloszlású tanítókészlettel. (Megjegyezzük, hogy az AdaBoost eljárás és az egyéb boosting eljárások általános mintaeloszlás-módosító eljárások, tehát nemcsak neuronhálóknál alkalmazhatók. Ennek ellenére most feltételezzük, hogy az eljárást moduláris háló kialakításához alkalmazzuk.)

Jelölje a t -edik iterációban az l -edik mintának a tanítókészleten belüli előfordulási gyakoriságát $D_t(l)$. Kiinduláskor ez minden tanítópontra azonos, vagyis $D_t(l)=1/L$, minden $l=1,2,\dots,L$ -re. (Tehát minden egyes minta egyszer fordul elő az L elemű halmazban.) Az egymást követő iterációkban az egyes mintapontok előfordulási gyakoriságát módosítjuk: a helytelenül osztályozott minták gyakoriságát növeljük, a helyesen osztályozottakét csökkentjük. Ez azt eredményezi, hogy arra kényszerítjük a következő iterációban az újabb osztályozót, hogy a nehéz esetekkel többet foglalkozzon.

Az algoritmus lépései a $t=1,\dots,T$ iteráció során a következők:

1. Tanítsunk meg egy hálót a D_t eloszlású mintakészlettel.
2. Minősítsük a megtanított hálót az alábbi hibadefiníció alapján:

$$\varepsilon_t = P\{y_t(\mathbf{x}_l) \neq d_l\} \quad (9.46)$$

ahol $y_t(\mathbf{x}_l)$ a t -edik iterációban alkalmazott háló válasza $\mathbf{x}_l$ bemenetre. A hiba fenti kifejezése annak a valószínűségét adja meg, hogy a t -edik háló az l -edik mintát hibásan osztályozza. A hibaválosztás a mintapontok előfordulási gyakoriságával is kifejezhető:

$$\varepsilon_t = \sum_{l: y_t(\mathbf{x}_l) \neq d_l} D_t(l) \quad (9.47)$$

3. Válasszunk egy α_t együtthatót a következőképpen

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right) \quad (9.48)$$

4. Frissítsük a mintapontok eloszlását (módosítsuk a mintapontok előfordulási gyakoriságát) az alábbiak szerint:

$$\begin{aligned} D_{t+1}(l) &= \frac{D_t(l)}{S_t} \times \begin{cases} e^{-\alpha_t} & \text{ha } y_t(\mathbf{x}_l) = d_l \\ e^{\alpha_t} & \text{ha } y_t(\mathbf{x}_l) \neq d_l \end{cases} \\ &= \frac{D_t(l) \exp(-\alpha_t d_l y_t(\mathbf{x}_l))}{S_t} \end{aligned} \quad (9.49)$$

ahol S_t egy normalizáló tényező, melyet úgy kell megválasztani, hogy D_{t+1} egy diszkrét valószínűség-sűrűségfüggvény legyen, tehát $\sum_l D_{t+1}(l) = 1$.

5. Amennyiben nem értük még el az iteráció leállási feltételét (kielégítő pontosságot, adott iterációs számot vagy más feltételt), akkor növeljük t -t eggyel és lépünk az 1. lépésre. Ha az iteráció végére értünk (T lépés után), akkor hozzuk létre az eredő osztályozót:

6. Az eredő választ a következőképpen határozzuk meg:

$$y(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T \alpha_t y_t(\mathbf{x}) \right) \quad (9.50)$$

Láthatóan az algoritmus során egymást követően több hálót konstruálunk, mindegyiket egy valamilyen mértékben módosított tanítókészlettel. Mind a tanítókészlet módosításához, mind a részeredmények aggregálásához felhasználunk egy α_t együtthatót, ami a t -edik háló fontosságának mértékeként is tekinthető.

Vegyük észre, hogy $\alpha_t \geq 0$, ha $\varepsilon_t \leq 1/2$, és α_t egyre nagyobb lesz, ha ε_t csökken. Az $\varepsilon_t \leq 1/2$ nem tekinthető megszorításnak, hiszen bármilyen osztályozó konstruálásánál nyilvánvaló követelmény, hogy az osztályozási hiba nem haladhatja meg az 50%-ot.

A mintapontok eloszlásának módosítását megadó (9.49) összefüggés szerint hibás osztályozás esetén növelni kell az adott mintapont szerepét a tanítókészletben és csökkenteni, ha az osztályozás helyes. Így a később létrehozott hálók egyre inkább a „nehéz” mintákra koncentrálnak. Az eredő választ az egyes hálók válaszainak súlyozott összegeként kapjuk, ahol a súlyozás szintén az α_i együtthatókkal történik.

A fentiek alapján az AdaBoost eljárással konstruált moduláris háló beleillik a hálóegyüttesek sorába. Itt is különálló hálók lineáris kombinációjával nyerjük az eredményt, és a teljesítőképesség javulásának az esélyét adja, hogy az egyes hálók különböző minőségben oldják meg a feladatot. Az eltérő minőségű megoldások alapját a tudatosan konstruált eltérő eloszlású tanítókészletek adják.

3.1.3. A boosting eljárások minősítése

A boosting eljárások teljesítőképesség-növelő hatása általánosan is vizsgálható. Ez a megközelítés a valószínűleg közelítőleg helyes (VKH) (*probably approximately correct, PAC*) tanulási modellen alapul. A valószínűleg közelítőleg helyes tanulási modell [Val84], [Rus05] a tanulás számítási elméletéhez kapcsolódik és bináris osztályozási feladatoknál a tanulás alapkérdésére keresi a választ: miért működik egyáltalán a mintákból történő tanulás?

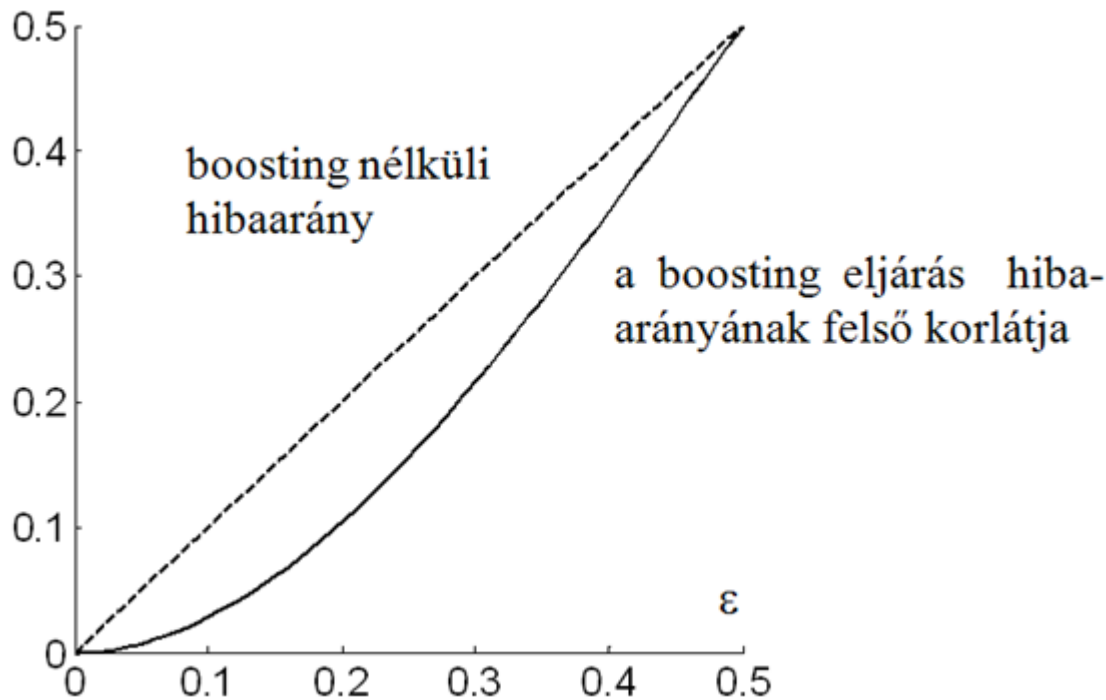
A boosting eljárásokkal kapcsolatban a kérdés úgy merül fel, hogy egy olyan tanuló algoritmus, amelynek a teljesítőképessége az 50%-os találati aránynál (a véletlen választásnál) épp hogy csak jobb, feljavítható-e tetszőlegesen pontos tanuló eljárássá. Az olyan tanulást, amelynek eredménye az 50%-os találati arányt épp hogy meghaladja gyenge tanulásnak (*weak learning*), míg a tetszőlegesen kis hibát eredményező eljárást erős tanulásnak (*strong learning*) nevezzük. Kérdés tehát, hogy a gyenge tanulás feljavítható-e erős tanulássá. A válasz általánosan is pozitív, amit úgy fogalmazhatunk meg, hogy az erős és a gyenge tanulás ekvivalens fogalmak [Scha90], [Fre95], [Scha99].

A szűréssel történő boosting eljárás hibacsökkentő hatását Schapire vizsgálta [Scha90]. E szerint, ha az eredeti hibaarány (a válogatás nélkül kapott mintakészlettel tanított első háló hibája) ε , a hálóegyüttes eredő hibájára a következő felső korlát érvényes:

$$g(\varepsilon) = 3\varepsilon^2 - 2\varepsilon^3 \quad (9.51)$$

Látható, hogy $g(\varepsilon)$ jóval kisebb is lehet, mint ε . Ezt mutatja a 9.8 ábra, ahol ε függvényében a folytonos vonal jelöli a fenti felső korlátot. Az ábra alapján az is nyilvánvaló, hogy ha rekurzív módon alkalmazzuk a mintaszűrő eljárást, az eredő hiba tetszőlegesen kicsivé tehető. A rekurzív alkalmazás azonban általában inkább csak elvi lehetőség, hiszen ez az amúgy is nagy kiinduló mintakészlet további növelését igényli.

9.8. ábra - A szűréssel történő boosting eljárás teljesítőképesség-javító hatása



Az AdaBoost eljárás teljesítőképesség-növelő hatásának elemzését Yoav Freund és Robert Schapire adták meg [Fre97]. E szerint ha a t -edik iterációban a (9.47) összefüggés szerinti hiba $\varepsilon_t = 1/2 - \gamma_t$, ($\gamma_t > \gamma$, ahol γ valamilyen kis pozitív konstans) akkor az eredő megoldás hibája a tanítópontokra legfeljebb:

$$\prod_t \sqrt{\varepsilon_t(1-\varepsilon_t)} = \prod_t \sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_t \gamma_t^2\right) \quad (9.52)$$

Vagyis ha a gyenge tanulás bármilyen kismértékben, de jobb, mint a véletlen választás, az AdaBoost eljárással kapott eredő hiba exponenciálisan gyorsan csökken. A (9.52) összefüggés ugyanakkor csak a tanítópontokban meghatározott hibára vonatkozik és nem az általánosítási hibára.

4. A moduláris eljárások összefoglaló értékelése

Ebben a fejezetben a moduláris hálóarchitektúrák kialakításának néhány fontosabb lehetőségét mutattuk be. A moduláris felépítés alapjában két fő célt szolgálhat: egyrészt biztosíthatja, hogy egy feladatot könnyebben, eredőben kisebb komplexitású hálóval, gyorsabb tanulási eljárással oldjunk meg, másrészt lehetővé teheti, hogy moduláris kialakítással jobb eredményt érjünk el. A fejezetben mindkét lehetőségre mutattunk néhány példát.

A moduláris hálók témaköre az itt összefoglalt területnél jóval bővebb és szerteágazóbb. A különböző további megoldások szintén az itt alkalmazott megközelítéseket használják: vagy egymással versengő vagy együttműködő, esetleg versengő és együttműködő modulok együtteséből épülnek fel. A különbség az itt bemutatott eljárásokhoz képest a feladat dekomponálásában, illetve a részeredmények integrálásánál található.

A fejezetben a moduláris kialakítások három fő területével foglalkoztunk. Az első terület a feladatdekompozíció, amely komplex osztályozási feladatok megoldásánál ad segítséget.

A második terület a kooperatív hálóegyüttesekkel foglalkozik. A hálóegyüttesek különösen sikeresen alkalmazhatók az alábbi körülmények között:

- túl kevés mintapont áll a rendelkezésünkre,
- nagy a lokális minimumba kerülés esélye,
- a megoldandó feladat az adott eszköz modellező képességének határán, esetleg azon túl található.

A túl kevés mintapont azzal a következménnyel járhat, hogy a mintapontokhoz könnyen igazodó megoldás nyerhető, miközben az egyes különálló megoldások általánosítási képesség tekintetében eltérőek. Itt az átlagolás vagy súlyozott átlagolás jelentősen javíthatja a moduláris kialakítás eredő pontosságát. A pontos és különböző megoldások együttese lényegében ezt az esetet képviseli.

A lokális minimumba ragadás hatásának mérséklését a moduláris kialakítás úgy biztosíthatja, hogy a több különálló modult eltérő feltételek mellett konstruáljuk (pl. eltérő kezdeti értékekkel indítjuk az egyes hálók tanítását). Így még ha mindegyik tanítás lokális minimumhoz vezetne is, az integrált megoldás bármelyik egyedi megoldásnál jobb eredményt biztosíthat. Jó példa erre az esetre a hálók optimális lineáris kombinációja.

A korlátozott modellező képesség reprezentációs problémát takar. Ez azt jelenti, hogy külön-külön egyik modul sem képes egy adott feladat megoldására, mert a feladat az adott modulok reprezentációs, modellezési körén kívül helyezkedik el. Az egyes modulok által nyújtott megoldások megfelelő kombinálásával azonban az eredő modellezési tartomány kiterjeszthető. Tipikus példa erre a lineáris szakértőkből összeállított MOE architektúra.

A harmadik fő terület a mintapontok eloszlásának változtatása útján segít hozzá a jobb eredmény eléréséhez. A boosting eljárások a gyenge és az erős tanulás kapcsolatának bemutatása miatt elvi fontosságúak, miközben a gyakorlat számára is könnyen alkalmazható megközelítést adnak.

Feladatok

9.1 Oldja meg a 6.5 ábrán bemutatott kettős spirál problémát feladatdekompozíciós megközelítéssel. A problémát dekomponálja több különböző módon: véletlenszerűen, az x_1 bemeneti változó értéke szerint, illetve mind az x_1 , mind az x_2 bemeneti változó szerint. Alakítson ki 3, 9 (3×3), illetve 36 (6×6) részfeladatot és az egyes részfeladatokat lineáris osztályozóval oldja meg. Elemezze a megoldásokat a háló komplexitása, az általánosítási hiba és a tanítási idő szempontjából. Hasonlítsa össze a megoldásokat egy egyetlen MLP-t alkalmazó megoldással. A kettős spirál mintakészletet az 5. fejezetben (5.1 példa) megadott MATLAB kód felhasználásával generálja.

9.2 Határozza meg az optimális súlyokat egy súlyozott lineáris kombinációt alkalmazó moduláris hálóegyüttesnél, ha a modulokat összefogó lineáris súlyozott összegző (*) tartalmaz eltolásértéket (*bias*), (**) nem tartalmaz eltolásértéket. Vizsgálja meg a kétféle megoldást általánosan és egy konkrét feladat megoldása kapcsán is, ha az egyes modulok hagyományos MLP-k.

9.3 Származtassa a MOE kapuzóhálózat $\mathbf{v}_i$ paramétervektorainak tanító összefüggését ((9.41) egyenlet).

9.4 Származtassa a HMOE tanításának alapösszefüggéseit, ha a HMOE két kapuzóhálózat-szintet tartalmaz.

9.5 Oldja meg a $\sin(x)/x$ függvényapproximációs feladatot lineáris hálókba felépített MOE architektúra segítségével. Vizsgálja meg a lineáris modulok száma és az approximációs hiba kapcsolatát. Elemezze a kapuzóértékek alakulását, illetve azt, hogy milyen módon dekomponálja a feladatot a moduláris megoldás.

9.6 Oldja meg az N -dimenziós ($N \geq 3$) $\sin(r)/r$ függvényapproximációs feladatot SVM alapú MOE architektúrával, ahol $r = \sqrt{x_1^2 + x_2^2 + \dots + x_N^2}$. Az egyes SVM modulokat eltérő σ szélességparaméterű Gauss kernelfüggvényekkel konstruálja.

9.7 Illusztrálja a szűrővel dolgozó boosting eljárás teljesítőképesség-növelő hatását a 9.1 feladatban megfogalmazott kettős spirál problémán. Elemezze az egyes tanítókészletek eloszlását. Az egyes modulok tanításához minimum 30 mintapontot használjon. Nézze meg, hogy hány mintapontra van szüksége ahhoz, hogy a három tanítókészlet létre tudja hozni. (Elegendő-e a 9.1 feladatban meghatározott mintakészlet a három tanítóhalmaz előállításához, ha mindháromban 30-30 mintapontot szeretnénk összegyűjteni?)

9.8 Oldja meg az előző feladatot AdaBoost eljárás segítségével. Hasonlítsa össze az eredményt a megoldás pontossága és a felhasznált tanítópontok száma szempontjából.

9.9 Oldja meg a Mackey-Glass kaotikus idősor-előrejelzési feladatot olyan moduláris CMAC háló architektúrával, mely egybementű CMAC hálóból épül fel. A mintapontokat a 8.2 példában definiált módon állítsa elő.

10. fejezet - Nemellenőrzött tanulási hálózatok

A tanulással foglalkozó 2. fejezetben láttuk, hogy a nemellenőrzött tanulás (vagy felügyelet nélküli, tanító nélküli tanulás) legfontosabb jellemzője, hogy nem áll rendelkezésünkre kívánt válasz, a tanítás során nincs tanító. A kívánt válasz hiánya miatt a nemellenőrzött tanítási hálózatok konstrukciója és alkalmazási területei jelentősen különböznek az ellenőrzött tanítási hálónál látottaktól. Mivel nincs kívánt válaszunk, meghatározott be-kimeneti leképezés megvalósítása nem lehet cél. További lényeges különbség, hogy a nemellenőrzött tanítási hálók jelentős része lineáris háló, tehát az alkalmazott processzáló elemek egyszerű súlyozott összegzéste végeznek.

A nemellenőrzött tanítási hálók főbb alkalmazási területei:

- a bemenetre kerülő minták közötti hasonlóság megállapítása,
- a bemeneti mintatérben csoportok, klaszterek kialakítása,
- adattömörítés, főkomponens analízis,
- független komponensek meghatározása.

A hasonlóság megállapítása, ill. csoportok kialakítása egymáshoz szorosan kapcsolódó képességek. Ezen képességek teszik lehetővé, hogy a nemellenőrzött tanítási hálók is – legalább közvetve – alkalmazhatók osztályozási, felismerési feladatok megoldására. Szemben azonban az ellenőrzött tanítási hálókkal, itt nem ismerjük előre a megkülönböztetendő osztályok számát, tehát nemcsak a bemeneti minták adott osztályba sorolása a feladat, hanem maguknak az osztályoknak a kialakítása is. Valójában tehát nem is osztályozási, hanem klaszterezési feladatot oldunk meg. Ezzel kapcsolatban két alapvető feltételezést kell tennünk; az első, hogy egy klasztert tágabb értelemben úgy definiálunk, mint azon minták összességét, amelyek hasonló jellemzőkkel rendelkeznek, a másik, hogy a hálózat képes kell legyen a teljes bemeneti mintakészlet hasonló jellemzőinek azonosítására. A klaszterezésre alkalmas nemellenőrzött tanítási hálók tipikus példája a Kohonen háló.

Az adattömörítés során az adatok olyan reprezentációját keressük, amely lehetővé teszi, hogy adatainkat kevesebb paraméterrel írjuk le, vagyis az adattömörítés azt jelenti, hogy a többdimenziós adatok egy (esetenként sokkal) kisebb dimenziós térben is reprezentálhatók. Ennek során azt is megengedjük, hogy az új reprezentáció csak közelítőleg írja le az adatokat, tehát az ábrázolás során valamekkora hibát is megengedünk. Az új reprezentációra való áttérés az adatok valamilyen transzformációját igényli. A főkomponens analízis (*principal component analysis, PCA*) hálózatok olyan – általában lineáris – transzformációt végeznek, amely lehetővé teszi, hogy közelítő ábrázolásnál a tömörített leírás hibája átlagos négyzetes értelemben minimális legyen.

Adattömörítést azonban nemcsak a PCA hálózatok végeznek, hiszen a klaszterek kialakítása szintén felfogható adattömörítésnek. Ekkor ugyanis egy adott klaszterbe tartozó összes adat a klaszter azonosításával megadható. Minthogy egy klaszter általában több adatot tartalmaz, az ezek közötti megkülönböztetésre a klaszter-azonosító nem képes, ez a reprezentáció is információvesztéssel, hibával jár.

A független komponens analízis (*independent component analysis, ICA*) komplex jelek összetevőkre történő bontására szolgál. A független komponens analízis során olyan összetevőket keresünk, melyek egy adott feladat szempontjából önálló jelentéssel rendelkezhetnek, de amelyeket közvetlenül nem tudunk megfigyelni.

A felügyelet nélküli tanulás a tanulás egyik leginkább elemi formája; olyan adatelemzés, amikor azt próbáljuk felderíteni, hogy az adataink egyáltalán hordoznak-e valamilyen információt. Ha igen, akkor azt valamilyen módon meg akarjuk „ragadni”, az adatokból ki akarjuk nyerni. Ezért a nemellenőrzött tanulást egyfajta adatbányászatnak is tekinthetjük.

A fejezet célja, hogy összefoglalja a felügyelet nélküli tanulási eljárások főbb jellemzőit, valamint azokat a hálóarchitektúrákat, melyek fontos gyakorlati feladatok megoldását teszik lehetővé. A nemellenőrzött tanulási eljárások többsége a Hebb szabályon alapul. A Hebb szabály alapú tanulás egyes hálózatoknál kiegészül a versengő tanulással, ezért elsőként e két tanulási szabállyal foglalkozunk.

1. Hebb tanulás

A Hebb szabály biológiai eredetű eljárás [Heb49]. Legegyszerűbb formájában a következőképpen fogalmazható meg: két processzáló elem közötti kapcsolat erőssége (a processzáló elemek közötti súlytényező értéke) a processzáló elemek aktivitásának szorzatával arányosan változik.

$$w_{ij}(k+1) = w_{ij}(k) + \mu y_i(k) y_j(k), \quad (10.1)$$

ahol w_{ij} az i -edik és a j -edik processzáló elem közötti súly, y_i ill. y_j a két processzáló elem kimenetének értéke, μ pedig az eddigiekhez hasonlóan a tanulási tényező. Ha a súly egy bemenet és egy processzáló elem között található, ahol a bemenet értéke x_i , a processzáló elem kimenetének értéke pedig y_j , akkor a súlymódosítás értelemszerűen:

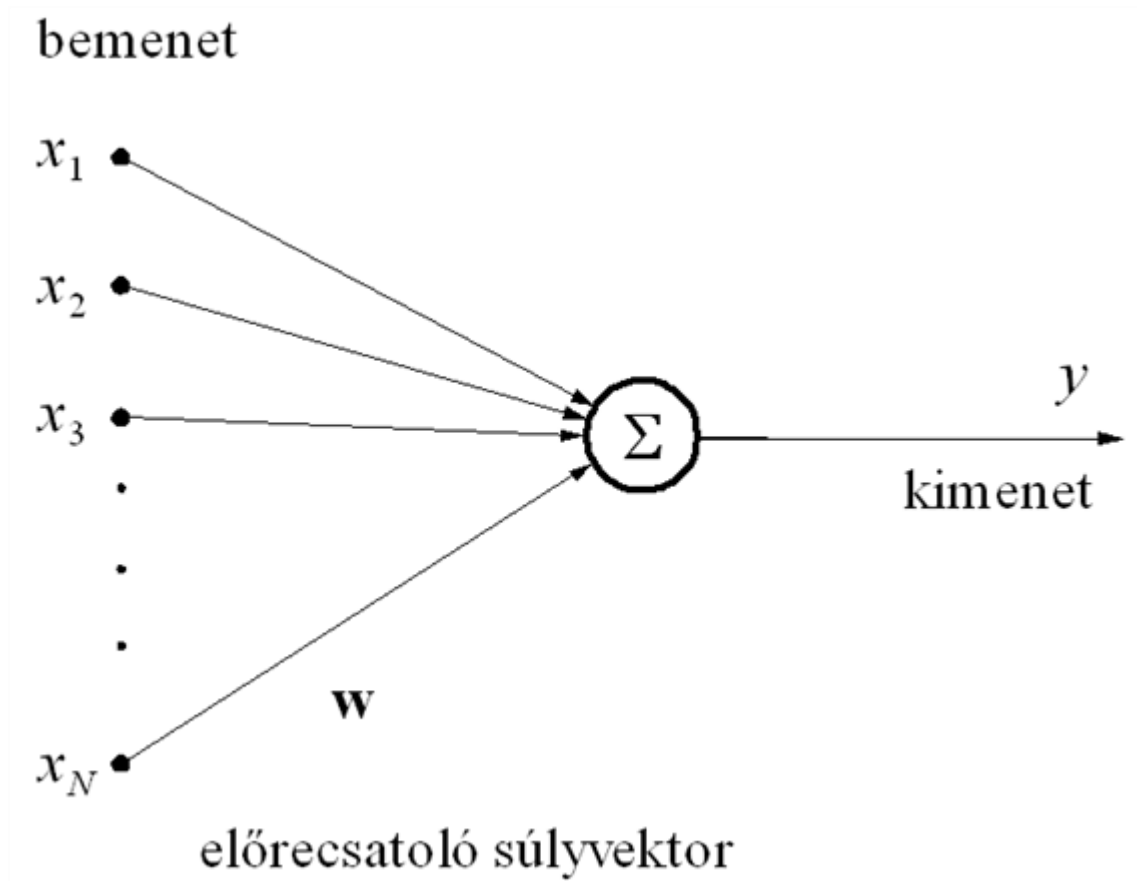
$$w_{ij}(k+1) = w_{ij}(k) + \mu x_i(k) y_j(k) \quad (10.2)$$

Az eredeti Hebb tanulási szabálynak különböző változatait dolgozták ki. Ezen változatok legfontosabb jellemzője, hogy a szabályba beépítettek egy normalizáló eljárást is, ugyanis az eredeti Hebb szabály mellett a súlyok minden határon túl növekedhetnek. A normalizálásnak több módjával találkozhatunk, melyeket az egyes nemellenőrzött tanítású hálózatok bemutatásánál fogunk tárgyalni.

A nemellenőrzött tanítású hálók többnyire lineáris neuronokból épülnek fel. Egy ilyen lineáris neuront mutat a 10.1 ábra, ahol a neuron kimenete a bemenetek súlyozott összegeként áll elő:

$$y = \mathbf{w}^T \mathbf{x} = \sum_{i=1}^N w_i x_i \quad (10.3)$$

10.1. ábra - Egy lineáris neuron



Megmutatható, hogy egy lineáris processzálo elem súlyainak Hebb szabály alapján történő módosítása azt eredményezi, hogy a processzálo elem kimenetének varianciája, vagyis

$$E\left\{\left(y - E\{y\}\right)^2\right\} \quad (10.4)$$

maximumot vesz fel azzal a feltétellel, hogy

$$\sum_i w_i = \text{konstans} \quad (10.5)$$

Ez utóbbi feltétel vagy valamilyen normalizáló eljárással, vagy a súlyokra vonatkozó telítési feltétel beépítésével biztosítható.

Bizonyos hálózatoknál alkalmazzák az ún. *anti-Hebb* tanulási szabályt is, amelynél – hasonlóan a Hebb szabályhoz – a súlymódosítás most is a bemenet és a kimenet aktivitásának szorzatával arányos, csak negatív előjellel:

$$w_{ij}(k+1) = w_{ij}(k) - \mu x_i(k) y_j(k). \quad (10.6)$$

Az anti-Hebb szabály – szemben a Hebb szabály kimeneti variancia-maximumra törekvő hatásával – a kimeneti variancia minimumát igyekszik biztosítani. A súlymódosítás mindkét esetben a két aktivitás közötti korrelációval arányos, csak a változtatás iránya eltérő. A Hebb- és az anti-Hebb szabályok együttes alkalmazásának fontos szerepe van az ún. főkomponens meghatározó hálózatoknál; szerepüket részletesebben ezen hálózatok ismertetésénél mutatjuk be.

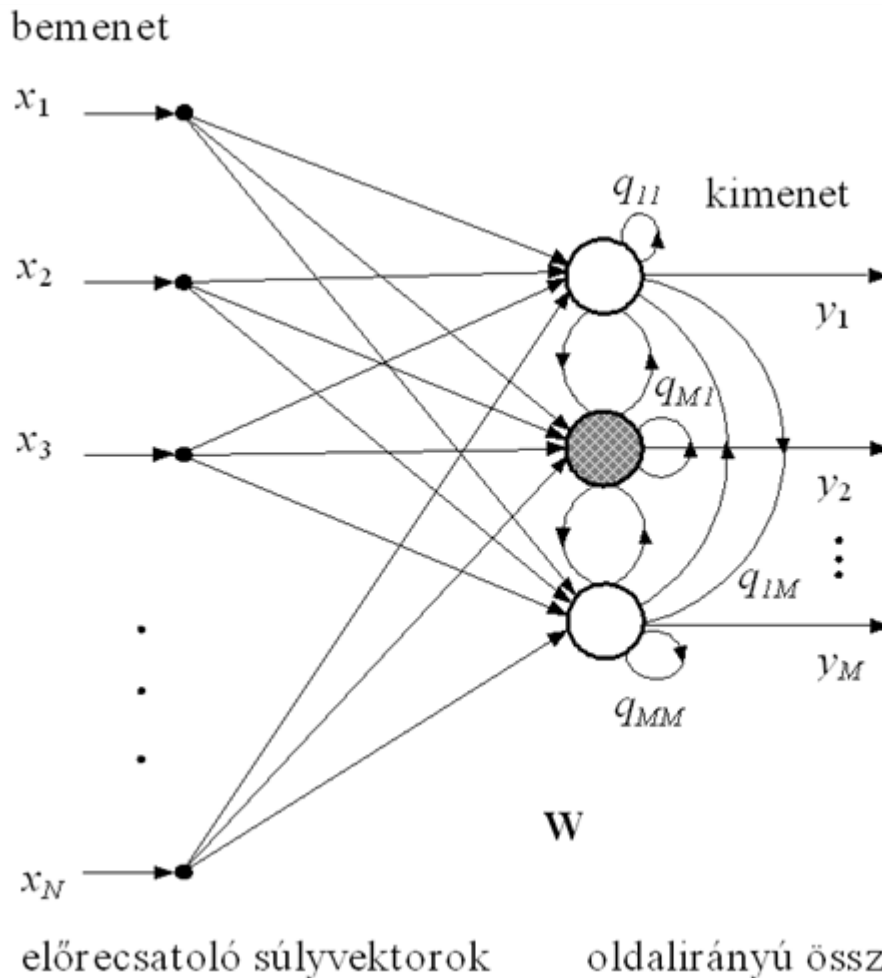
2. Versengő tanulás

A versengő tanulás a nemellenőrzött tanulások csoportjába tartozó másik eljárás, amelynek a célja, hogy egy neuronhálónál a neuronok közül egy győztest válasszunk ki. A győztes neuron kimenete aktív lesz, (bináris kimenet mellett a kimenet értéke 1 lesz), miközben az összes többi neuron passzív marad (bináris kimenet mellett mindegyik 0 értéket vesz fel). A versengő tanulás célja általában a bemeneti mintatér olyan tartományokra osztása, szegmentálása, hogy egy adott tartományba tartozó bemenet hatására egy és csakis egy processzálo elem aktivizálódjon. Az eljárás tehát egyfajta klaszterkialakítást eredményez.

A versengő tanulás tulajdonképpen két lépésből áll. Az első lépés során a neuronokból álló elrendezés minden processzálo elemének kimenetét meghatározzuk az aktuális súlyvektorok felhasználásával. Ezt követi a győztes kiválasztása, amely a "győztes mindent visz" (*winner-takes-all*) elv alapján működik. A tényleges tanulás, vagyis a súlyvektor módosítása csak a győztes kiválasztása után lehetséges, ugyanis a versengő tanulásnál csupán a győztes processzálo elem súlyvektorát (vagy a győztes valamely környezetében található processzálo elemek súlyvektorait) módosítjuk. Meg kell jegyeznünk, hogy a súlymódosítás a versengő tanulásnál is általában a Hebb szabállyal történik. A versengő tanulásnál a versengés során tehát valójában tanulás nincs is. A versengés arra szolgál, hogy kiválaszthassuk azt a neuront, melynek a súlyait valamilyen súlymódosító eljárással majd módosítani fogjuk.

A győztes elem kiválasztása többféleképpen történhet. Triviális megoldás, ha kiszámítjuk egy adott bemenetre az összes processzálo elem kimenetét, és a legnagyobb kimenettel rendelkező PE, a győztes értékét 1-re állítjuk, az összes többiét pedig nullázzuk. A győztes kiválasztása azonban történhet "automatikusan" is, amikor a processzálo elemek között q_{ij} oldallirányú kapcsolatok vannak (ld. 10.2 ábra), és ezen kapcsolatokat megvalósító súlyok megfelelő kialakítása (és a kimeneten megfelelő nemlineáris aktivációs függvény alkalmazása) biztosítja, hogy a győztes (az ábrán az árnyékolt elem) kimeneti értéke 1, az összes többi pedig nulla lesz.

10.2. ábra - Versengő tanulással tanított hálózat (Kohonen-háló)



Az oldalirányú kapcsolatok súlytényezői rögzítettek: a közeli processzáló elemek között gerjesztő kapcsolat, a távolabbiak között gátló kapcsolat van.

3. Kohonen háló, kompetitív hálózatok

A Kohonen háló – hasonlóan számos, eddig bemutatott hálózathoz – biológiai eredetű [Koh82]. A jelenlegi ismeretek alapján egyértelműen megállapítható, hogy az agy komplex adatstruktúrák belső modellezésénél térbeli leképezést használ [Koh89]. Kohonen ezt az ötletet használta fel, amikor a róla elnevezett hálózatot kialakította. A térbeli leképezés biztosítja azt is, hogy a hálózat adatokat tároljon olyan módon, hogy a tanító adatok térbeli és topológiai kapcsolatait megtartsuk és valamilyen jelentéssel bíró módon reprezentáljuk.

A Kohonen háló alapeleme egy lineáris összegző funkciót megvalósító processzáló elem; a processzáló elemek egyetlen rétegben, általában egy sík rácspontjaiban vannak elhelyezve. Minden bemenet a hálózat összes csomópontjához (processzáló eleméhez) kapcsolódik. A hálózatban nemcsak előrecsatolást megvalósító súlyok, hanem visszacsatolás is található, ez azonban a közvetlen szomszédos csomópontokkal való oldalsó összeköttetésekre korlátozódik. A háló további jellemzője, hogy nincs különálló kimeneti réteg – a rács minden egyes csomópontja egyben kimeneti csomópont is. A 10.2 ábrán bemutatott elrendezés valójában egy N -bemenetű – M -kimenetű Kohonen hálót mutat. A háló egyes kimenetei a bemeneti komponensek súlyozott összegei, vagyis a háló a bemenetek és a kimenetek között lineáris kapcsolatot valósít meg:

$$y_i = \sum_{j=1}^N w_{ij} x_j \quad (10.7)$$

Az oldalirányú kapcsolatok rögzítettek; jellegük gerjesztő, ha egy processzáló elem önmagára való visszacsatolásáról van szó és gátló a többi esetben. Kialakításukról és szerepükről a későbbiekben szólunk.

A Kohonen hálózatoknál az adaptív, önszerveződő tanulás két legfontosabb kérdése a súly-adaptációs eljárás és a csomópontok topológiai környezetének fogalma. Mindkét fogalom jelentős mértékben eltér az eddig tárgyalt neurális hálózatoktól, ezért a Kohonen hálózat működésének bemutatását erre a két kulcsfogalomra építjük.

3.1. A Kohonen háló tanítása

A Kohonen háló súlymódosítása a versengés felhasználásával a Hebb szabályon alapul. Ez azt jelenti, hogy a súlymódosítás a Hebb szabály, illetve a Hebb szabály normalizált változata szerint történik; de nem módosítjuk a hálózat összes súlyát, hanem csak a győztes processzáló elemét.

Egy adott bemenet mellett győztes az a processzáló elem, amelyik a legnagyobb kimenetet produkálja. Ha tehát

$$\mathbf{w}_{i*}^T \mathbf{x} \geq \mathbf{w}_i^T \mathbf{x} \text{ az összes } i\text{-re}, \quad (10.8)$$

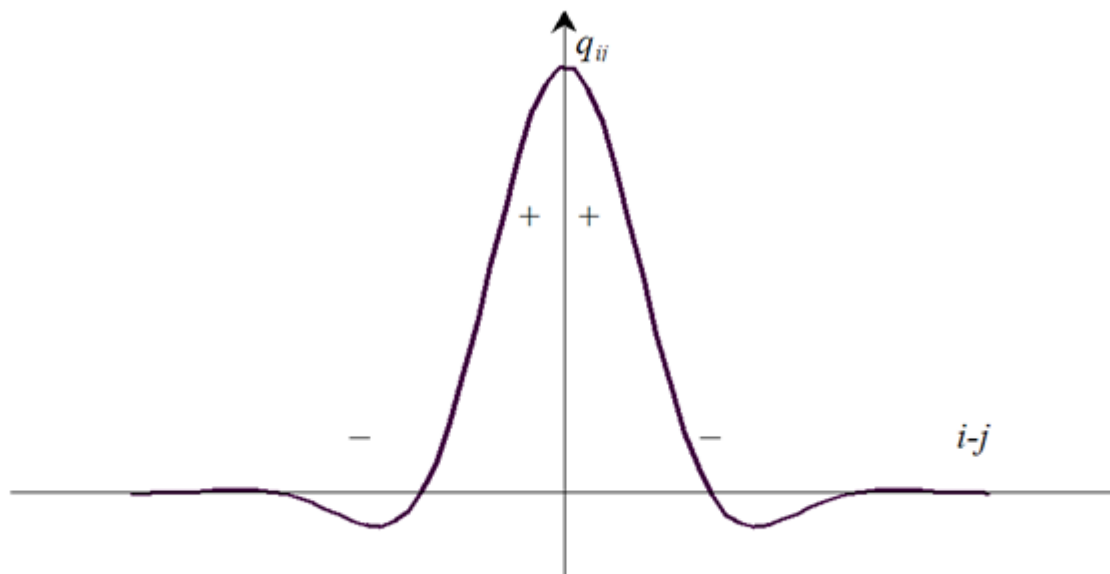
akkor a győztes processzáló elem indexe i^* . A tanítás során csak $\mathbf{w}_{i^*}$ -ot módosítjuk:

$$\Delta \mathbf{w}_{i*} = \mu y_{i*} \mathbf{x} \quad (10.9)$$

Látható, hogy a súlymódosítás a Hebb szabály szerint történik, ami – ha figyelembe vesszük, hogy a győztes elem kimenete 1 lesz – az alábbi formában is felírható:

$$\Delta \mathbf{w}_{i*} = \mu \mathbf{x} \quad (10.10)$$

10.3. ábra - Az oldalirányú kapcsolatokat meghatározó mexikói kalap függvény



Ez azt mutatja, hogy a módosítás során a súlyvektort azon bemeneti vektorok irányába forgatjuk, amelyek mellett az adott elem a győztes. Egy adott neuron súlyvektora ennek megfelelően – konvergencia esetén – olyan értékre áll be, amely azon bemenő vektorok átlagaként áll elő, melyek hatására az adott neuron győztesé válik. A győztes elem vagy az egyes lineáris kimenetek összehasonlításával választható ki, vagy automatikusan az oldalirányú kapcsolatok eredményeképpen alakul ki. Az oldalirányú (gerjesztő és gátló) kapcsolatok ugyanis azt eredményezik, hogy annak a neuronnak a kimenete, amelyiknek kezdetben legnagyobb volt az értéke egyre nő, míg a többi csökken, addig, amíg a győztes kimenete 1, a többi 0 nem lesz. (Megjegyezzük, hogy a bináris kimenetet a legegyszerűbben úgy biztosíthatjuk, ha a lineáris összegzőt itt is egy nemlinearitás – a 0 és 1 közötti telítési lineáris függvény – követi.)

Az oldalirányú kapcsolatok súlyait a processzáló elemek egymástól való, valamilyen értelemben vett távolsága függvényében mutatja a 10.3 ábra, amelyet *mexikói kalap* függvénynek nevezünk. Az ábrán a (+) jel a gerjesztő kapcsolatot, a (–) jel a gátló kapcsolatot jelöli. Az oldalirányú kapcsolatok ilyen kialakítása azt is biztosítja, hogy a háló ún. topológikus leképezést valósít meg, ami azt jelenti, hogy a tanítás során a bemenetre kerülő

minták közötti topológiai kapcsolatokat a háló megtartja. A topológikus leképezésről és a most bemutatott hálóarchitektúráról bővebben a következőkben szólnunk.

Megjegyezzük, hogy az oldalirányú kapcsolatok ilyen kialakítása biológiai eredetű. Mind anatómiai, mind fiziológiai bizonyíték van arra, hogy az emlősök agyában megtalálhatók a szomszédos neuronok között a gerjesztő és gátló kapcsolatokat megvalósító oldalirányú összeköttetések [Koh89]

A súlyok (10.9) ill. (10.10) szerinti módosítása önmagában azonban nem alkalmazható, hiszen azt eredményezi, hogy a súlyok minden határon túl nőhetnek. A korrekt versengéshez pedig az kell, hogy a súlyoknak ne az abszolút értéke, hanem az iránya határozza meg a győztest. A megoldást a súlyok normalizálása jelenti. A normalizálásra több lehetőség is van. Leggyakrabban az ún. standard versengő tanulást szokták alkalmazni, ami a következő:

$$\Delta \mathbf{w}_{i*} = \mu (\mathbf{x} - \mathbf{w}_{i*}). \quad (10.11)$$

Ez az eljárás biztosítja a súlyvektorok folyamatos normalizálását feltéve, hogy a bemeneti vektorok is normalizáltak.

A Kohonen háló megfelelő működéséhez is szükség van a súlyok kezdeti beállítására és a tanulási tényező megválasztására.

A súlyvektorok kezdeti értékének beállításánál némi óvatosságra van szükség. Az általános javaslat most is az, hogy indulásnál a hálózat súlyait véletlenszerű, normalizált értékekre állítsuk be. Azonban, ha a súlyvektorok valóban véletlen értékeket vesznek fel, lehetséges, hogy a tanítás nem lesz konvergens vagy a háló nagyon lassan fog tanulni. Ennek oka jórészt intuitív módon mutatható meg. Tipikus esetekben a bemeneti tanító vektorok a mintatér korlátozott tartományában valamilyen klasztert fognak képezni. Ha a hálózat súlyvektorai véletlenszerűen szóródnak, akkor könnyen előállhat az az eset, amikor sok súlyvektor iránya különbözik jelentősen a tanító minták többségének irányától. Az ilyen súlyvektorokkal rendelkező processzáló elemek sohasem fognak győzni (ún. halott processzáló elemek), és így nem is vesznek részt a tanulási folyamatban. Amennyiben túl sok halott neuronunk van, a háló tényleges mérete és képessége nem lesz elegendő a kitűzött feladat megoldására, pl. megfelelő számú klaszter kialakítására. Mindenképpen olyan megoldásra van szükség, amely minél több processzáló elemet bevon a tanulásba. Ennek érdekében a következő lehetőségekkel élhetünk:

- a súlyvektorok kezdeti értékeit magukból a tanítópontokból választjuk,
- nemcsak a győztes, hanem a többi neuron súlyvektorait is módosítjuk, de kisebb μ -vel; így a vesztes neuronok súlyvektorai is módosulnak és ezért esélyük lesz arra, hogy győztesse váljanak (ezt a tanulási változatot szivárgó tanulásnak (*leaky learning*) nevezik),
- geometriai alakzatban elrendezett neuronok esetében a győztes adott környezetében lévőket is módosítjuk a győztesselel megegyező vagy ahhoz hasonló módon (ez a változat az ún. Kohonen térképet eredményezi, amit önszervező térképnek is szokás nevezni, ld. később),
- a bemeneti vektorokat egy közös kezdeti beállítási irányból ($\mathbf{v}$) kiindulva fokozatosan adjuk a hálózatra; a győztes neuron súlymódosítása $\alpha \mathbf{x} + (1 - \alpha) \mathbf{v}$, ahol α kezdetben 0, majd lassan növekszik 1-re,
- a hálózat lineáris kimenetéből egy változó küszöbértéket vonunk ki; a gyakran győztesnél a küszöböt növeljük, így nehezebben lesznek ismét győztesek, a ritkán győzőknél pedig a küszöb csökkentésével a győzelmi esélyt növeljük,
- a bemeneti vektorokhoz zajt adunk, amelynek elég nagy a szórása ahhoz, hogy "mindenfajta" $\mathbf{x}$ érték előforduljon, így minden neuront be tudunk vonni a módosításokba.

A Kohonen hálót Kohonen térképnek, vagy önszervező térképnek (*Kohonen map*, *self-organizing map*, *SOM*) hívjuk, ha a bemeneti minták topológiai kapcsolatait is megőrzi: hasonló bemeneti mintákra egymáshoz geometriailag közelálló neuronok válaszolnak győztesként. (Ehhez az kell, hogy a Kohonen háló processzáló elemei között is értelmezni lehessen szomszédosági viszonyokat. A Kohonen térkép esetén a neuronok általában vagy egy vonal mentén vagy egy síkban vannak elrendezve, ahol a szomszédosság definiálható.) Kohonen a topológikus kapcsolatok modellezésére javasolta az ún. környezetek definiálását és ehhez kapcsolódóan egy módosított tanulási szabály alkalmazását. Környezetnek nevezzük a győztes processzáló elem valamekkora –

általában dinamikusan változó méretű – szomszédságában lévő processzáló elemek összességét. A módosított tanulási szabály a következő:

- nemcsak a győztes neuron súlyait módosítjuk a tanulás során, hanem a győztes adott méretű környezetében lévőket is,
- a tanulás során a környezetet csökkentjük, mintegy rázsugorítjuk a győztes neuronra, a környezet összes neuronjának súlyait a győztesrel azonos tanulási szabály alapján módosítjuk:

$$\Delta \mathbf{w}_i = \mu \Phi(\mathbf{r}_i, \mathbf{r}_{i*}) (\mathbf{x} - \mathbf{w}_i) \quad i = 1, 2, \dots, M \quad (10.12)$$

Itt $\mathbf{r}_i$ és $\mathbf{r}_{i*}$ az i -edik, illetve a győztes processzáló elem geometriai helyét adja meg, $\Phi(\mathbf{r}_i, \mathbf{r}_{i*})$ pedig az ún. szomszédsági függvény. A szomszédsági függvény megfelelő megválasztása alapvető fontosságú a topológiai tulajdonságok megtartása szempontjából. A rendszerint szimmetrikus függvény $i=i^*$ -nál ($\mathbf{r}_i = \mathbf{r}_{i*}$) 1 értéket, az i^* -hoz közeli i értékekre 1-höz közeli értéket vesz fel, és a győztes neurontól való euklideszi távolság függvényében monoton csökken. Egy lehetséges választás:

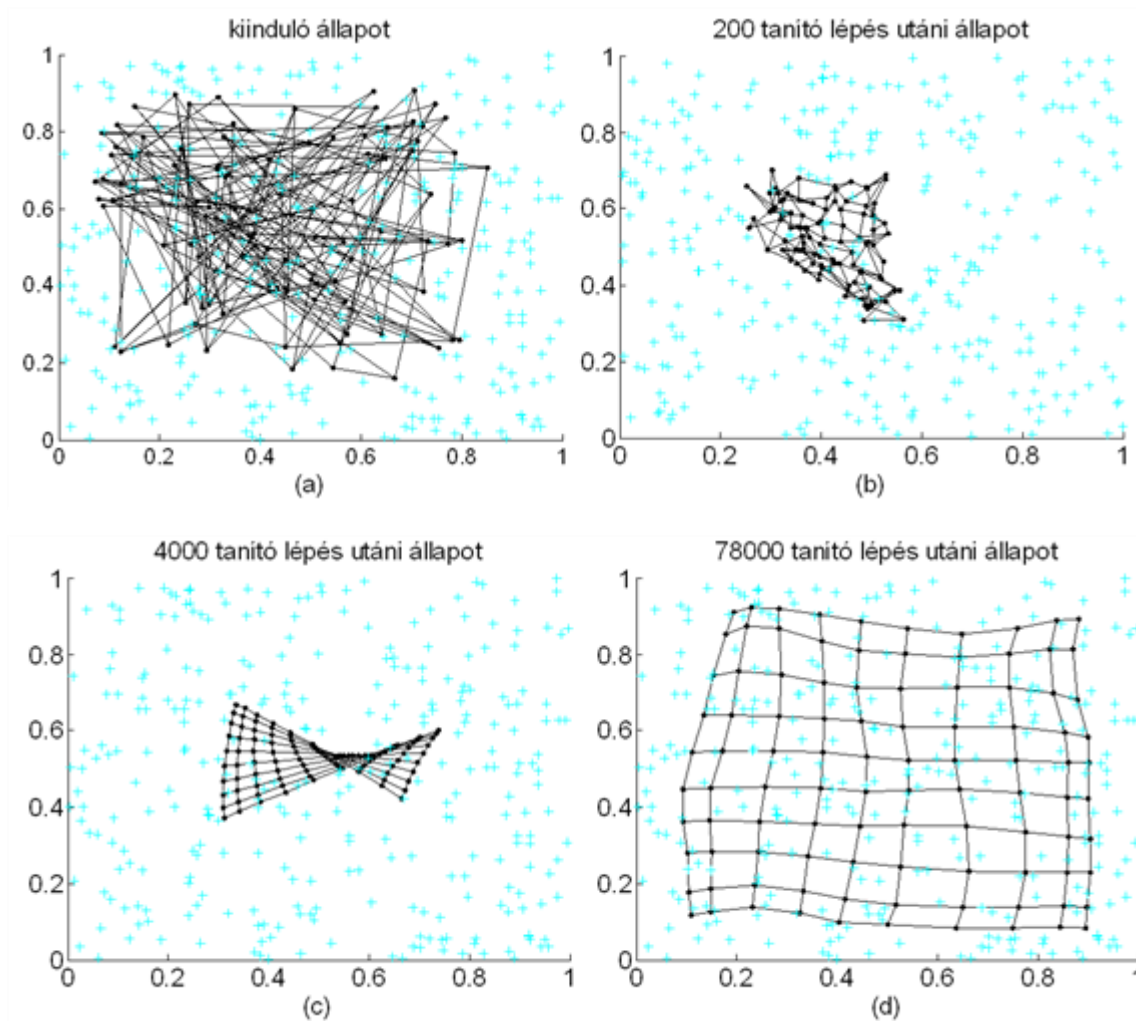
$$\Phi(\mathbf{r}_i, \mathbf{r}_{i*}) = e^{-\frac{\|\mathbf{r}_i - \mathbf{r}_{i*}\|^2}{2\sigma^2}}, \quad (10.13)$$

ahol σ^2 a szomszédság szélességét befolyásoló, a tanulás előrehaladtával csökkenő értékű paraméter.

A súlyok alakulása a Kohonen térkép lépésenkénti felrajzolásával követhető. A Kohonen térkép egy olyan összekötött pontelrendezés a súlyvektor térben, ahol az egymással szomszédos neuronok súlyvektorainak végpontjai vannak összekötve. A Kohonen térkép alakulását mutatja be a tanulás folyamán a Kohonen mozi, melynek néhány képkockáját mutatja a 10.4 ábra.

Az ábrán olyan esetet mutatunk be, ahol a bemeneti mintapontok egy kétdimenziós egyenletes eloszlású valószínűségi változó egyes realizációi, a háló processzáló elemeit pedig egy egyenletes négyzetrács rácspontjaiban helyezzük el. Ebben az esetben, ha a Kohonen háló topológiai leképezést valósít meg, akkor egy adott geometriai pozícióban lévő neuron a bemeneti adatok azon részhalmazába tartozó minták hatására aktivizálódik, melyek geometriai elhelyezkedése a bemeneti mintatérben a processzáló elem geometriai elhelyezkedéséhez közeli. Látható, hogy a kiinduló helyzetben (a) az egyes processzáló elemek súlyvektorai véletlen elhelyezkedésűek, és ahogy a tanítással előrehaladunk – 200 iteráció után a (b), 4000 után a (c) ábrák –, a súlyvektorok kezdenek beállni a bemeneti minták eloszlásának megfelelően. A háló tanításának előrehaladásával – a 10.4 (d) ábra 78000 iterációs lépés utáni állapotot mutat – a súlyvektorok egyre inkább egy egyenletes rács pontjaira mutatnak, vagyis a súlyvektorok irányai megfelelnek a megfelelő neuronok topológiai elrendezésének; a Kohonen térkép tükrözi a hálózat bemeneti adatainak eloszlását. Az ábrákon a színes „+” jelek a tanítópontokat jelzik.

10.4. ábra - A Kohonen térkép



A topológiai térkép kialakulása általában két tanulási szakaszban valósul meg. Az első szakaszban egy durva leképezés jön létre, ekkor a térkép pontjai nagymértékben változnak, a tanulás viszonylag nagy tanulási együtthatóval történik (tipikusan $\mu > 0,5$). A második szakaszban következik a súlyok finomhangolása. Ekkor a súlyokon kisebb változásokat kell eszközölni, ezért a tanulási együttható kisebb, és az eljárás során fokozatosan még tovább is kell csökkentenünk. A finomhangolási szakasz tipikusan mintegy 100-1000-szer annyi lépést igényel, mint a durva leképezés kialakulása. A tanulás során előfordulhat, hogy egyes neuronok súlyvektorai a pozíciójuknak megfelelő súlyvektorok tükörképének felelnek meg. Ilyen esetre mutat példát a 10.4 (c) ábra, ahol a súlyvektorokat reprezentáló háló megcsavarodva látszik. A megcsavart állapotból általában különösen lassan alakul ki a mintapontok eloszlását tükröző súlyvektor-eloszlás. A 10.4 (d) ábra mutatja is ezt, hiszen a sok tízezer tanítólépés még mindig nem eredményezte a súlyvektorok egyenletes eloszlását.

Az önszervező térkép (SOM) számos alkalmazása közül ki kell emelni a szövegbányászati alkalmazásokat, melyet az utóbbi években a SOM kiterjesztéseként kifejlesztett WEBSOM felhasználásával oldhatunk meg sikeresen [Koh00], [Lag02], [Lag04]. A WEBSOM módszer az önszervező térképet (SOM) használja szöveges dokumentumok kétdimenziós térképre való leképezésére. A térképen a hasonló dokumentumok azonos vagy egymáshoz közeli térkép-elem(ek)en jelennek meg és minden egyes térkép-elemhez egy mutató is tartozik, ami a dokumentum adatbázisra mutat. Ezáltal egy keresésnél, miközben azon dokumentumokat megtaláljuk, melyek legjobban illeszkednek a kereső kifejezésre, további releváns eredményeket is találunk, melyek a megtalált dokumentumokat jelképező térképelemmel azonos vagy ahhoz közeli térképelemre voltak leképezve, függetlenül attól, hogy a keresési kifejezésnek megfeleltek-e vagy sem. A WEBSOMot kimondottan nagy szövegyűjteményekben való keresésre dolgozták ki.

3.2. Felügyelt tanulás alkalmazása Kohonen hálózatonál: tanuló vektorkvantálás (LVQ).

Annak ellenére, hogy a Kohonen hálózat felügyelet nélküli tanulást használ, Kohonen kialakított egy ellenőrzött tanuláson alapuló változatot is, amit tanuló vektorkvantálásnak nevezett el (*learning vector quantization, LVQ*) [Koh86], [Koh90].

Vektorkvantálásnak azt az eljárást nevezzük, amikor adattömörítési céllal a bemeneti vektorok egy tartományát az egész tartományt jellemző egyetlen vektorral, az ún. prototípus vektorral ill. a vektor sorszámaival reprezentálunk. Vektorkvantálásnál tehát a bemeneti teret diszjunkt tartományokra bontjuk. Egy adott tartományt reprezentáló prototípus vektor (vagy a vektor sorszáma) úgy tekinthető, mint az adott tartományba eső bemenő vektorok kódja, ahol a lehetséges kódok száma – a diszjunkt tartományok száma – általában jóval kisebb, mint a lehetséges bemenő vektorok száma. A diszjunkt tartományokat reprezentáló kódok együttesét a vektorkvantálásnál kódkönyvnek nevezzük. Ha a kódkönyv elemeit, illetve a tartományokat úgy alakítjuk ki, hogy a tartományba eső bemenő vektoroknak a tartomány prototípusától vett átlagos euklideszi távolsága minimális legyen, akkor a vektorkvantálót Voronoi kvantálónak, a tartományokat pedig Voronoi tartományoknak vagy Voronoi régióknak nevezzük. A Voronoi tartományokat reprezentáló vektorok a Voronoi vektorok.

A tanuló vektorkvantálás valójában a Voronoi tartományok kialakításánál egy kétlépcsős eljárás második lépése: a Kohonen önszervező térkép által kialakított felosztás finomítását ellenőrzött tanulással biztosítja. Az ellenőrzött tanulás során a Voronoi vektorok és így a Voronoi régiók kismértékű módosítása következik be, javítva, finomítva a bemeneti mintatérben a klaszterek kialakítását és a Kohonen osztályozó minőségét.

A hálózat tanítása a következőképpen történik: a bemeneti tér egy vektorát a hálózatra adjuk, majd a választól függően végezzük a súlymódosítást:

- ha a győztes neuron (PE_i) a kívánt neuronnal egyezik meg, akkor ennek súlyait a Kohonen szabály szerint kell módosítani:

$$\Delta w_{i*,j} = \mu y_i (x_j - w_{i*,j}), \quad (10.14)$$

vagyis a Voronoi vektort az öt győztesként kiváltó bemenő vektor irányához közelítjük,

- ha győztes nem a megfelelő PE, akkor a győztes súlyvektorát a Kohonen szabály (-1)-szeresével módosítjuk:

$$\Delta w_{i*,j} = -\mu y_i (x_j - w_{i*,j}), \quad (10.15)$$

vagyis itt épp ellenkezőleg, az adott bemenő vektor irányától távolítjuk.

A fenti LVQ1-nek nevezett eljárást Kohonen 1986-ban javasolta [Koh86]. Az eljárásnak később módosított változatai is születtek (LVQ2, LVQ3), melyek tovább finomítják a klaszterek kialakítását. Az alábbiakban csak az LVQ2 szabály lépéseit foglaljuk össze, az LVQ3 eljárással kapcsolatban csak az irodalomra utalunk [Koh90].

Az LVQ2 tanulási szabálya:

- ha a győztes neuron (PE_i) a megfelelő PE, akkor az előző szerint módosítjuk a súlyait:

$$\Delta w_{i*,j} = \mu y_i (x_j - w_{i*,j}), \quad (10.16)$$

- ha azonban a győztes nem a megfelelő PE, és i^o lenne a megfelelő neuron indexe, továbbá, ha a bemenő vektor ($\mathbf{x}$) az i^* és az i^o indexű tartományok határvonalához közel helyezkedik el az i^* indexű oldalon, akkor a súlymódosítás:

$$\Delta w_{i*,j} = -\mu y_i (x_j - w_{i*,j}), \quad (10.17)$$

és

$$\Delta w_{i^o,j} = \mu y_i (x_j - w_{i^o,j}), \quad (10.18)$$

- egyébként az előzőek (LVQ1) szerint történik a súlymódosítás.

A tanuló vektorkvantálásnak számos alkalmazását ismerjük. Ezek közül az egyik első az ún. fonetikus írógép (*phonetic typewriter*), melyet Kohonen a finn beszéd felismerésére hozott létre [Koh88]. A fonetikus írógép egy olyan összetett rendszer, amely hagyományos digitális jelfeldolgozó eljárásokat (spektrumanalízis), neurális hálózatot (Kohonen térkép) és szakértői rendszert együttesen alkalmazva oldja meg a beszédfelismerési feladatot.

4. Adattömörítés Hebb tanuláson alapuló hálózatokkal, PCA, KLT

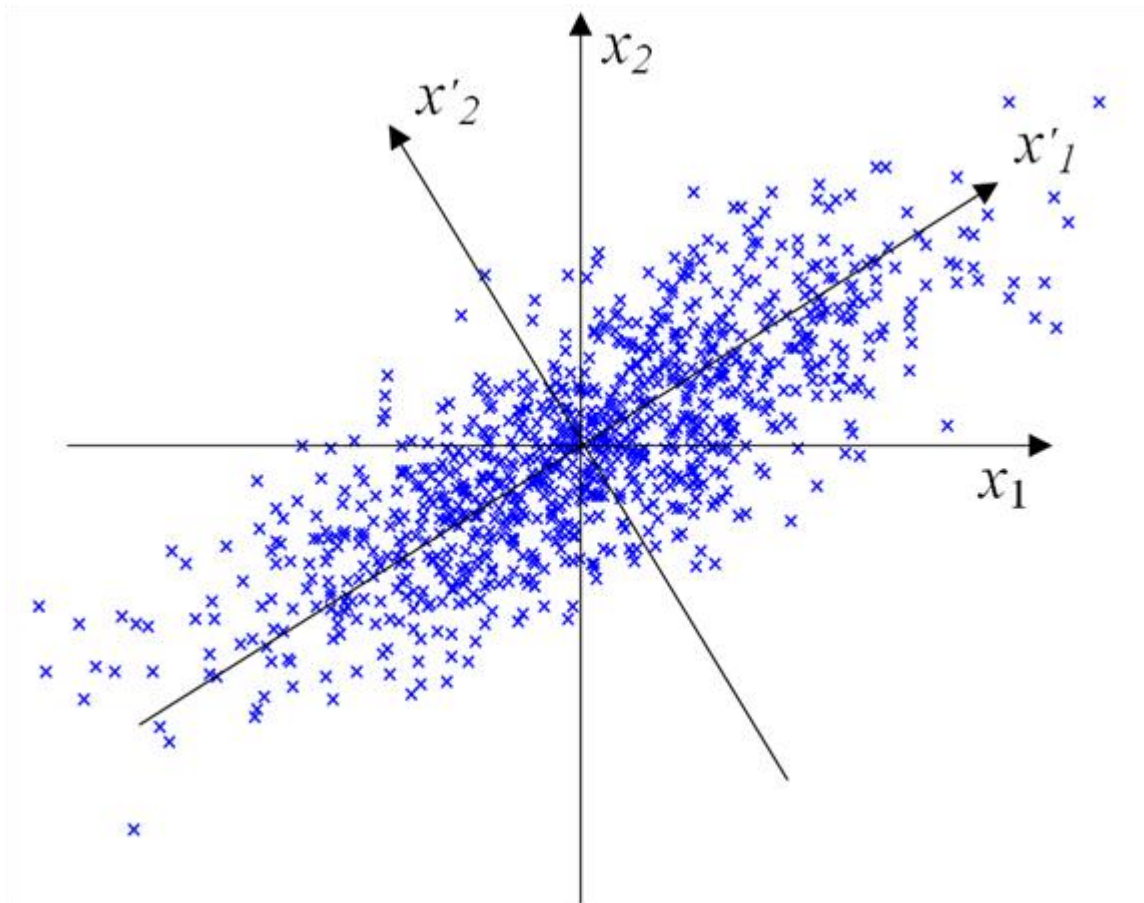
A neurális hálózatok egyik fontos alkalmazási területe az adattömörítés. Az adattömörítést általában az adatok hatékonyabb reprezentációja céljából végezzük, de alkalmazásával az is lehet a célunk, hogy az eredeti adatokból a későbbi feldolgozás szempontjából lényeges információt kiemeljük, és a lényegtelen elhagyjuk. Az adattömörítést ekkor lényegkiemelés érdekében végezzük. Az adattömörítés – többek között – elvégezhető olyan – lineáris vagy nemlineáris – transzformációk segítségével, amelyek egy adott, többdimenziós térbeli vektort kisebb dimenziós altérbe transzformálnak (vetítenek). Az ilyen típusú feladatoknál két nehézséggel találkozunk magunkat szemben. Először is meg kell találnunk azt a kisebb dimenziós alteret, amelyben az eredeti vektor hatékonyan ábrázolható. Másodszor, ha a megfelelő alteret megtaláltuk el kell végezzük a bemeneti vektorok transzformációját.

A hatékony ábrázolás az alkalmazási körtől függően különbözőképpen definiálható. Adattömörítésnél törekedhetünk arra, hogy az altérbe való vetítés során a vektor reprezentációjánál a közelítő ábrázolásból adódó hiba minél kisebb legyen. Ebben a megközelítésben definiálni kell valamilyen hibakritériumot, pl. átlagos négyzetes hibát, majd egy olyan, az eredeti dimenziószámánál kisebb dimenziós altér megtalálása a feladat, amelybe vetítve a kiinduló vektort a kritérium szerint értelmezett reprezentációs hiba a lehető legkisebb lesz. Más feladatnál, pl. felismerési vagy osztályozási feladatot megelőző lényegkiemelésnél a reprezentáció akkor tekinthető hatékonynak, ha az altér dimenziója minél kisebb, miközben a közelítő reprezentációban mindazon információ megmarad, amely a felismeréshez, osztályozáshoz elegendő. Ebben az esetben tehát nem követelmény a kiinduló vektor minél kisebb hibájú reprezentálása, csupán arra van szükség, hogy olyan kisebb dimenziós ábrázolást kapjunk, amely a feladat szempontjából szükséges lényeges információkat megtartja.

E feladatok megoldására univerzális eljárás nem létezik. Általában a megfelelő altér feladat- és adatfüggő, tehát a tényleges feladattól függetlenül előre nem meghatározható, és mind az altér meghatározása, mind a transzformáció elvégzése meglehetősen számításgépes. Éppen ezért jelentős eredmény, hogy mind az ellenőrzött, mind a nemellenőrzött tanítási hálózatok körében találunk olyan hálókat, melyek hatékony eszköznek bizonyultak az adattömörítési feladatok megoldására.

A lineáris adattömörítő eljárások között kitüntetett szerepe van a Karhunen-Loève (KL) transzformációnak (KLT), amely az eredeti jeltér olyan ortogonális bázisrendszerét és az eredeti vektorok ezen bázisrendszer szerinti transzformáltját határozza meg, amelyben az egyes bázisvektorok (bázisfüggvények) fontossága különböző. Egy N -dimenziós térből kiindulva az új bázisvektorok közül kiválasztható a legfontosabb $M < N$ bázisvektor, amelyek egy M -dimenziós alteret határoznak meg. Egy vektornak ezen altérbe eső vetülete az eredeti vektorok közelítő reprezentációját jelenti, ahol a közelítés hibája átlagos négyzetes értelemben a legkisebb, vagyis a KLT a közelítő reprezentáció szempontjából a lineáris transzformációk között optimális bázisrendszert határoz meg.

10.5. ábra - A Karhunen-Loève transzformáció



A KL transzformáció működését illusztrálja a 10.5 ábra. A transzformáció feladata az eredeti x_1 , x_2 koordináta-rendszerben ábrázolt adatokból kiindulva az x'_1 , x'_2 koordináta-rendszer megtalálása, majd az adatoknak ebben az új koordináta-rendszerben való megadása. Látható, hogy míg az eredeti koordináta-rendszerben a két komponens fontossága hasonló, addig az új koordináta-rendszerben a két komponens szerepe jelentősen eltér: x'_1 mentén jóval nagyobb tartományban szóródnak a mintapontok, mint x'_2 mentén, tehát az egyes mintapontok közötti különbséget az x'_1 koordináták jobban tükrözik. Amennyiben az adatok egydimenziós, közelítő reprezentációját kívánjuk előállítani célszerűen x'_1 -t kell meghagynunk és x'_2 -t eldobnunk; így lesz a közelítés hibája minimális. A KL transzformáció szokásos elnevezése a matematikai statisztikában faktoranalízis vagy főkomponens analízis (*principal component analysis, PCA*). Egy $\mathbf{x}$ vektor x'_1 és x'_2 irányú vetületeit főkomponenseknek is szokás nevezni. Közelítő reprezentációnál a főkomponensek közül csak a legfontosabbakat tartjuk meg, a többit eldobjuk. Az ábrán látható esetben ez azt jelenti, hogy egy $\mathbf{x}$ vektor legfontosabb főkomponense az x'_1 tengely irányú vetülete.

Az alábbiakban előbb bemutatjuk a KLT származtatását, majd olyan neurális hálózatok felépítésével, működésével és tanításával foglalkozunk, melyek a KLT megvalósítására alkalmasak.

4.1. A KL transzformáció és optimalitása

A KL transzformáció alapfeladata a következő: keressük meg azt az ortogonális (ortonormált) bázisrendszert, amely átlagos négyzetes értelemben optimális reprezentációt ad, majd e bázisrendszer segítségével végezzük el a transzformációt. Az eddigiekhez hasonlóan diszkrét reprezentációval dolgozunk, tehát a bemeneti jelet az $\mathbf{x}$ N -dimenziós vektorok képviselik, a transzformációt pedig egy $\mathbf{T}$ mátrisszal adhatjuk meg.

E szerint a transzformált jel ($\mathbf{y}$) előállítására:

$$\mathbf{y} = \mathbf{T}\mathbf{x}, \quad (10.19)$$

ahol transzformációs mátrix a Φ_i bázisvektorokból épül fel:

$$\mathbf{T} = [\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2, \dots, \boldsymbol{\varphi}_M]^T. \quad (10.20)$$

Mivel bázisrendszerünk ortonormált, ezért:

$$\boldsymbol{\varphi}_i^T \boldsymbol{\varphi}_j = \delta_{ij}, \text{ ebből adódóan } \mathbf{T}^T \mathbf{T} = \mathbf{I}, \text{ vagyis } \mathbf{T}^T = \mathbf{T}^{-1}. \quad (10.21)$$

Feladatunk legyen a következő: $\mathbf{x}$ közelítő reprezentációját ($\hat{\mathbf{x}}$) akarjuk előállítani úgy, hogy a közelítés átlagos négyzetes hibája minimális legyen. Mivel $\mathbf{x}$ előállítható, mint a $\boldsymbol{\varphi}_i$ bázisvektorok lineáris kombinációja

$$\mathbf{x} = \sum_{i=1}^N y_i \boldsymbol{\varphi}_i \quad (10.22)$$

ahol y_i a $\boldsymbol{\varphi}_i$ irányú komponens nagysága, és mivel a közelítő reprezentáció

$$\hat{\mathbf{x}} = \sum_{i=1}^M y_i \boldsymbol{\varphi}_i \quad M \leq N, \quad (10.23)$$

az átlagos négyzetes hiba felírható az alábbi formában:

$$\varepsilon^2 = E \left\{ \|\mathbf{x} - \hat{\mathbf{x}}\|^2 \right\} = E \left\{ \left\| \sum_{i=1}^N y_i \boldsymbol{\varphi}_i - \sum_{i=1}^M y_i \boldsymbol{\varphi}_i \right\|^2 \right\} = \sum_{i=M+1}^N E \left\{ (y_i)^2 \right\} \quad (10.24)$$

Továbbá, mivel

$$y_i = \boldsymbol{\varphi}_i^T \mathbf{x} \quad (10.25)$$

a következő összefüggés is az átlagos négyzetes hibát adja meg:

$$\varepsilon^2 = \sum_{i=M+1}^N E \left\{ \left(\boldsymbol{\varphi}_i^T \mathbf{x} \right) \left(\mathbf{x}^T \boldsymbol{\varphi}_i \right) \right\} = \sum_{i=M+1}^N \boldsymbol{\varphi}_i^T E \left\{ \mathbf{x} \mathbf{x}^T \right\} \boldsymbol{\varphi}_i = \sum_{i=M+1}^N \boldsymbol{\varphi}_i^T \mathbf{R}_{\mathbf{x}\mathbf{x}} \boldsymbol{\varphi}_i, \quad (10.26)$$

ahol $\mathbf{R}_{\mathbf{x}\mathbf{x}}$ az $\mathbf{x}$ bemenet autokorrelációs mátrixa. A továbbiakban feltételezzük, hogy $E\{\mathbf{x}\} = \mathbf{0}$, ekkor $\mathbf{R}_{\mathbf{x}\mathbf{x}}$ helyett $\mathbf{C}_{\mathbf{x}\mathbf{x}}$, vagyis $\mathbf{x}$ kovarianciamátrixa szerepel az átlagos négyzetes hiba kifejezésében.

Ezekután keressük meg azt a $\boldsymbol{\varphi}_i$ bázist, amely mellett ε^2 minimális lesz. Mivel a $\boldsymbol{\varphi}_i^T \boldsymbol{\varphi}_i = 1$ feltételt be kell tartanunk, feltételes szélsőértéket kell keresnünk. A Lagrange multiplikátoros módszert alkalmazva

$$\hat{\varepsilon} = \varepsilon^2 - \sum_{i=M+1}^N \lambda_i \left(\boldsymbol{\varphi}_i^T \boldsymbol{\varphi}_i - 1 \right) = \sum_{i=M+1}^N \left[\boldsymbol{\varphi}_i^T \mathbf{C}_{\mathbf{x}\mathbf{x}} \boldsymbol{\varphi}_i - \lambda_i \left(\boldsymbol{\varphi}_i^T \boldsymbol{\varphi}_i - 1 \right) \right], \quad (10.27)$$

ahol λ_i -k, ($i=M+1, \dots, N$) a Lagrange multiplikátorok. A (10.27) összefüggés $\boldsymbol{\varphi}_i$ szerinti szélsőértékét keressük, vagyis a

$$\frac{\partial \hat{\varepsilon}}{\partial \boldsymbol{\varphi}_i} = \sum_{i=M+1}^N \left[2 \mathbf{C}_{\mathbf{x}\mathbf{x}} \boldsymbol{\varphi}_i - 2 \lambda_i \boldsymbol{\varphi}_i \right] = \mathbf{0} \quad (10.28)$$

feltételt kell kielégítenünk. Ehhez az szükséges, hogy teljesüljön a

$$\mathbf{C}_{\mathbf{x}\mathbf{x}} \boldsymbol{\varphi}_i = \lambda_i \boldsymbol{\varphi}_i, \quad (10.29)$$

összefüggés, vagyis a KLT bázisrendszerét alkotó Φ_i vektorok a bemeneti jel autokorrelációs (autokovariancia) mátrixának sajátvektorai legyenek. A közelítő, M -dimenziós reprezentáció esetén elkövetett hiba ilyenkor

$$\varepsilon^2 = \sum_{i=M+1}^N \Phi_i^T \mathbf{R}_{\mathbf{xx}} \Phi_i = \sum_{i=M+1}^N \Phi_i^T \lambda_i \Phi_i = \sum_{i=M+1}^N \lambda_i \quad (10.30)$$

ahol a λ_i értékek az autokovariancia mátrix sajátértékei.

Minimális hibát nyilvánvalóan akkor fogunk elkövetni, ha a (10.30) összefüggésben a λ_i sajátértékek ($i=M+1, \dots, N$) a mátrix legkisebb sajátértékei, vagyis a közelítő, M -dimenziós reprezentációnál az autokovariancia mátrix első M legnagyobb sajátértékéhez tartozó sajátvektort, mint M -dimenziós bázist használjuk fel. A bemeneti jel ezen vektorok irányába eső vetületei lesznek a főkomponensek (innen ered a főkomponens analízis elnevezés). Megjegyezzük, hogy a KL transzformáció korrelálatlan komponenseket eredményez, vagyis a transzformált jel autokovariancia mátrixa diagonál mátrix, melynek főátlójában a λ_i sajátértékek vannak.

A KLT tehát egy kétlépéses eljárás: először a bemeneti jel autokorrelációs (autokovariancia) mátrixát, és ennek sajátvektorait és sajátértékeit kell meghatározni, majd ki kell választani a legnagyobb M sajátértéknek megfelelő sajátvektort, amelyek a megfelelő altér bázisait képezik. A bázisrendszer ismeretében lehet elvégezni második lépésként a jel transzformációját. Minthogy ez az eljárás meglehetősen összetett (mind a sajátértékek és sajátvektorok meghatározása, mind a későbbi transzformáció elvégzése számításigényes feladat), fontos eredmény, hogy léteznek olyan neurális hálózatok, amelyek e feladatok megoldására alkalmasak. A KLT-t megvalósító neurális hálózatokat az irodalomban PCA hálózatoknak nevezik. E hálózatok a következőkben bemutatásra kerülő Oja szabályon alapulnak.

4.2. Az Oja szabály, a legnagyobb sajátértéknek megfelelő sajátvektor meghatározása

A bementi jel legfontosabb főkomponensének (az autokovariancia mátrix legnagyobb sajátértékéhez tartozó sajátvektor irányába eső jel-vetületnek) meghatározására Erkki Oja javasolt egy, a Hebb tanuláson alapuló hálót [Oja82].

A háló egy egyszerű, lineáris, előreecsatolt hálózat, amely legegyszerűbb formájában tulajdonképpen egyetlen, lineáris kombinációt megvalósító processzáló elem. Az Oja háló felépítésében tehát megegyezik a 10.1 ábrán bemutatott egyszerű lineáris neuronnal. Az Oja háló a bemeneti N -dimenziós vektort vetíti a kimeneti egydimenziós térbe. E háló specialitását nem is a felépítése, hanem a tanulási eljárás adja. A súlyok meghatározására a Hebb tanulást, ill. annak módosított változatát alkalmazzuk.

Vizsgáljuk meg, hogy mit eredményez a Hebb tanulás a hálózaton. Bemenetként N -dimenziós véletlen vektorokat használunk. Megmutatható [Oja82], hogy a hálózat a tanulási szabály alkalmazásával valamilyen egyensúlyi helyzet elérésére törekszik. Ez az egyensúlyi helyzet akkor áll be, ha a súlyvektor a bemeneti vektorok autokorrelációs mátrixának egy sajátvektora lesz. Stabilitás azonban csak akkor érhető el, ha ez a sajátvektor a legnagyobb sajátértékhez tartozó sajátvektor. Vagyis a hálózat kimenetén a bemenet legfontosabb főkomponensét kapjuk, ami a bementi vektorok legfontosabb sajátvektor irányú vetülete.

A Hebb tanulás önmagában azt eredményezi, hogy a súlyvektorok a tanulás során minden határon túl növekedhetnek. A növekedésnek határt kell szabni, ami normalizálás útján érhető el. A normalizálás biztosítható, ha a Hebb szabályt az alábbiak szerint módosítjuk: legyen a tanulási szabály a következő:

$$\Delta w_i = \mu y (x_i - w_i), \quad (10.31)$$

vagyis egészítsük ki a Hebb tanulási szabályt egy taggal. Az Oja szabály egy olyan $\mathbf{w}$ súlyvektorhoz konvergál, amelynek a tulajdonságai az alábbiak:

- a $\mathbf{w}$ súlyvektor az $\mathbf{R}$ autokorrelációs mátrix legnagyobb sajátértékének megfelelő sajátvektor irányú vektor (a továbbiakban az egyszerűség kedvéért a bemenet autokorrelációs mátrixának jelölésére $\mathbf{R}_{\mathbf{xx}}$ helyett $\mathbf{R}$ -et használunk).

- a súlyvektor normalizált: $\|\mathbf{w}\| = 1$,
- a $\mathbf{w}$ irány olyan, hogy a bemenetnek ebbe az irányba eső vetületének lesz a legnagyobb a varianciája, vagyis a kimenet (y) varianciája ($E\{y^2\}$) akkor lesz a legnagyobb, ha y a bemenet $\mathbf{w}$ irányú vetületének a hossza. A variancia:

$$E\{y^2\} = \mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\} \mathbf{w} = \mathbf{w}^T \mathbf{R} \mathbf{w} \quad (10.32)$$

Egyensúlyi helyzetben

$$E\{\Delta w_i\} = E\{\mu y(x_i - y w_i)\} = 0 \quad (10.33)$$

vagy vektorosan felírva:

$$E\{\Delta \mathbf{w}\} = \mathbf{R} \mathbf{w} - (\mathbf{w}^T \mathbf{R} \mathbf{w}) \mathbf{w} = \mathbf{0} \quad (10.34)$$

Tehát egyensúlyi helyzetben a súlyvektornak ki kell elégítenie az

$$\mathbf{R} \mathbf{w} = \lambda \mathbf{w} \quad (10.35)$$

összefüggést, ahol

$$\lambda = \mathbf{w}^T \mathbf{R} \mathbf{w} = \mathbf{w}^T \lambda \mathbf{w} = \lambda \|\mathbf{w}\|^2 \quad (10.36)$$

A (10.35) összefüggés mutatja, hogy egyensúlyi állapotban $\mathbf{w}$ az $\mathbf{R}$ sajátvektora kell legyen, továbbá, ha $\|\mathbf{w}\|=1$, akkor λ a megfelelő sajátérték.

Hogy az Oja szabály valóban a Hebb szabály normalizált változata, az alábbiak szerint látható be. Az eredeti Hebb szabály szerint a súlymódosítás:

$$\tilde{\mathbf{w}}(k+1) = \mathbf{w}(k) + \mu \mathbf{x}(k) y(k) \quad (10.37)$$

ahol most $\tilde{\mathbf{w}}(k+1)$ nem normalizált. Normalizáljuk a módosított súlyvektort:

$$\mathbf{w}(k+1) = \frac{\tilde{\mathbf{w}}(k+1)}{\|\tilde{\mathbf{w}}(k+1)\|} = \tilde{\mathbf{w}}(k+1) \|\tilde{\mathbf{w}}(k+1)\|^{-1} \quad (10.38)$$

$\|\tilde{\mathbf{w}}(k+1)\|^{-1}$ meghatározásához írjuk fel $\|\tilde{\mathbf{w}}(k+1)\|^2$ -et, majd ennek $(-1/2)$ -edik hatványát fejtsük Taylor sorba $\mu=0$ környezetében.

$$\begin{aligned} \|\tilde{\mathbf{w}}(k+1)\|^2 &= \|\mathbf{w}(k)\|^2 + 2\mu \tilde{\mathbf{w}}(k)^T \mathbf{x}(k) y(k) + O(\mu^2) \\ &= \|\mathbf{w}(k)\|^2 + 2\mu [y(k)]^2 + O(\mu^2) \end{aligned} \quad (10.39)$$

Figyelembe véve, hogy $\|\mathbf{w}(k)\|^2 = 1$, vagyis hogy az előző lépésben a súlyvektor már normalizált, a következőt kapjuk:

$$\|\tilde{\mathbf{w}}(k+1)\|^{-1} = 1 - \mu y^2(k) + O(\mu^2) \quad (10.40)$$

Felhasználva a (10.37), (10.38) és (10.40) összefüggéseket és a μ -ben magasabbrendű tagokat elhanyagolva:

$$\begin{aligned} \mathbf{w}(k+1) &= [\mathbf{w}(k) + \mu \mathbf{x}(k) y(k)] \left[1 - \mu y^2(k) + O(\mu^2) \right] \\ &\cong \mathbf{w}(k) + \mu y(k) [\mathbf{x}(k) - \mathbf{w}(k) y(k)] \end{aligned} \quad , (10.41)$$

ami megfelel (10.31)-nek.

Fentiekben megmutattuk, hogy az Oja szabály alapján végzett tanulás eredményeképp – feltéve, hogy a tanuló eljárás konvergens (a konvergencia tényét azonban nem bizonyítottuk) – a súlyvektor egy λ sajátértékhez tartozó sajátvektorhoz tart. Azt azonban még meg kell mutatni, hogy λ egyben a legnagyobb sajátérték, vagyis $\lambda_{\max}$.

Ehhez tételezzük fel, hogy $\mathbf{w}$ az $\mathbf{R}$ valamelyik sajátvektorához, Φ_i -höz áll közel. Vizsgáljuk meg, hogyan alakul a tanulás során a súlyvektor változása, tekintsük a súlyvektorváltozások várható értékét $E\{\Delta \mathbf{w}\}$ -t. Minthogy a súlyvektor az egyik sajátvektorhoz áll közel: $\mathbf{w} = \Phi_i + \mathbf{e}$ és feltéve, hogy normalizált sajátvektorokkal dolgozunk, vagyis $\|\Phi_i\| = 1$,

$$E\{\Delta \mathbf{w}\} = E\{\mathbf{e}\} = \mathbf{R}\mathbf{e} - 2\lambda_i \left(\mathbf{e}^T \Phi_i \right) \Phi_i - \lambda_i \mathbf{e} + O(\mathbf{e}^2) \quad , (10.42)$$

ahol λ_i a Φ_i sajátvektorhoz tartozó sajátérték.

Annak eldöntésére, hogy a súlyvektor melyik sajátértékhez tartozó sajátvektorhoz tart, vegyük a súlyvektorváltozás várhatóértékének egy másik normalizált sajátvektorra, Φ_j -re vett vetületét és hanyagoljuk el az $O(\mathbf{e}^2)$ tagot.

$$\left(\Phi_j \right)^T E\{\mathbf{e}\} = \lambda_j \left(\Phi_j \right)^T \mathbf{e} - 2\lambda_i \left(\mathbf{e}^T \Phi_i \right) \delta_{ij} - \lambda_i \left(\Phi_j \right)^T \mathbf{e} = \left(\lambda_j - \lambda_i - 2\lambda_i \delta_{ij} \right) \left(\Phi_j \right)^T \mathbf{e} \quad (10.43)$$

Látható, hogy ez a vetület csökken, ha λ_i a nagyobb, és növekszik, ha λ_j a nagyobb sajátérték. Tehát ha $\lambda_j > \lambda_i$, akkor a súlyvektor a nagyobb sajátértéknek megfelelő sajátvektor irányába fordul be.

4.2.1. Az Oja algoritmus, mint szélsőérték-kereső eljárás

Az Oja algoritmus a fentiekől eltérően szélsőérték-kereső eljárás eredményeként is származtatható, minthogy a bemenő sztochasztikus vektorfolyamat mintáinak a legnagyobb sajátvektor irányában vett vetülete várható

értékben maximumot kell adjon. Keressük tehát $f(\mathbf{w}) = E\{y^2\}$ maximumát $\mathbf{w}$ függvényében.

$$f(\mathbf{w}) = \frac{E\{y^2\}}{\mathbf{w}^T \mathbf{w}} = \frac{\mathbf{w}^T \mathbf{R} \mathbf{w}}{\mathbf{w}^T \mathbf{w}} \quad (10.44)$$

Felírva a deriváltat és figyelembe véve, hogy $\mathbf{w}^T \mathbf{w} = 1$ adódik, hogy

$$\nabla f(\mathbf{w}) = 2\mathbf{R}\mathbf{w} - \left(\mathbf{w}^T \mathbf{R} \mathbf{w} \right) 2\mathbf{w} \quad . (10.45)$$

Ha figyelembe vesszük, hogy $\mathbf{R} = E\{\mathbf{x}\mathbf{x}^T\}$

$$\begin{aligned}\nabla f(\mathbf{w}) &= 2E\{\mathbf{xx}^T\}\mathbf{w} - 2E\{\mathbf{w}^T\mathbf{xx}^T\mathbf{w}\}\mathbf{w} \\ &= 2E\{\mathbf{xy}\} - 2E\{y^2\}\mathbf{w}\end{aligned}, \quad (10.46)$$

továbbá, ha az LMS algoritmushoz hasonlóan a pillanatnyi gradiens alapján számolunk, vagyis a várhatóérték-képzést nem végezzük el, akkor:

$$\nabla f(\mathbf{w}) = 2\mathbf{xy} - 2y^2\mathbf{w}, \quad (10.47)$$

ami az Oja szabályt adja. Az Oja szabály (10.47) összefüggése egy sztochasztikus gradiens algoritmus, mivel a gradiens számításnál a várhatóérték-képzést elhagyjuk.

Megjegyezzük, hogy az (10.44) összefüggést Rayleigh hányadosnak is nevezik [Gol96b], mely, ha $\mathbf{w}$ az $\mathbf{R}$ mátrix egy sajátvektora, a hányados a megfelelő sajátértéket adja. Ebből is következik, hogy (10.44) $\mathbf{w}$ szerinti maximuma a legnagyobb, minimuma a legkisebb sajátértéket eredményezi.

Összefoglalva, megállapítható, hogy az Oja hálózat a módosított Hebb tanulással, a legnagyobb sajátértéknek megfelelő sajátvektort alakítja ki súlyvektorként, amennyiben a tanulási szabály konvergens. A konvergencia bizonyítható, azonban a bizonyítás menete bonyolult és az eljárás lényegének megértéséhez nem járul hozzá, ezért nem mutatjuk be. A bizonyítás a megfelelő irodalomban [Oja82] megtalálható.

4.3. Főkomponens- és altér hálózatok

A különböző alkalmazásokban a legfontosabb sajátvektornak és az ebbe az irányba eső főkomponensnek a meghatározása általában nem elegendő. Olyan hálózatot szeretnénk kapni, amely N -dimenziós bemenetből kiindulva az M legfontosabb sajátvektor ($M \leq N$) meghatározására képes.

Az Oja hálózatot, illetve az Oja szabályt többféleképpen módosították, melynek eredményeképpen létrejött hálózatok alkalmasak a teljeskörű főkomponens analízisre, vagyis képesek a bemeneti jel KL transzformáltjának meghatározására. Ezeket a hálózatokat **főkomponens hálózatoknak** (*principal component networks*) nevezzük.

Az Oja hálózaton alapuló hálózatok egy másik csoportja, amely csoport tagjai – bár nem a tényleges főkomponenseket, vagyis a legfontosabb sajátvektorok irányába eső vetületeket határozzák meg – a főkomponens analízissel rokon eredményre vezetnek. Adattömörítésnél ugyanis nincs feltétlenül szükség magukra a főirányokra, tehát a legfontosabb sajátvektorokra, sokszor elegendő, ha csak azt az alteret és ebbe az alterbe eső vetületet határozzuk meg, amelyet az első M legfontosabb sajátvektor feszít ki. Az alteret nemcsak a sajátvektorok határozzák meg, hanem bármely bázisa. Azokat a hálózatokat, amelyek az alteret és a bemeneti vektorok alterbe eső vetületeit meghatározzák, de a sajátvektorokat nem, **altér hálózatoknak** (*subspace networks*) nevezzük. Az alábbiakban előbb egy altér hálózatot ismertetünk, majd az eredeti Oja hálózat olyan módosításait vizsgáljuk, amelyek a tényleges főkomponensek meghatározását eredményező hálózatokra vezetnek.

4.3.1. Oja altér hálózat

Az Oja altér hálózat az Oja algoritmussal tanított elemi neuron kiterjesztése olyan többkimenetű hálózattá, amely nem a legfontosabb főkomponens meghatározását végzi, hanem az első M sajátvektor által kifeszített alterbe vetíti. A hálózat tehát átlagos négyzetes értelemben minimális hibájú közelítést eredményez. Minimális átlagos négyzetes eltérést biztosító megoldást kapunk, ha az eredeti Oja szabályt egy M -dimenziós $\mathbf{y}$ kimeneti vektorra alkalmazzuk. Az eredmény az Oja általánosított szabály [Oja83]:

$$\Delta \mathbf{W} = \mu \left[\mathbf{W} \mathbf{x} \mathbf{x}^T - (\mathbf{W} \mathbf{x} \mathbf{x}^T \mathbf{W}^T) \mathbf{W} \right], \quad (10.48)$$

ahol $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_M]^T$ az M -kimenetű háló súlyvektoraiból, mint sorvektorokból képezett mátrix.

Az Oja altér hálózat egy N -bemenetű- M -kimenetű hálózat, amely egyetlen, lineáris neuronokból felépülő rétegből áll. Mivel az Oja altér háló súlyvektorai nem a sajátvektorokhoz konvergálnak, hanem a sajátvektorok által kifeszített tér egy bázisához, az általánosított Oja szabályt Oja altér szabálynak is szokás nevezni.

Az Oja alter hálózaton kívül egyéb, kisebb-nagyobb mértékben eltérő felépítésű alter hálózatot publikáltak (pl. [Föl89]). A Földiák háló az előreccsatoló súlyokon kívül a neuronok között szimmetrikus oldalirányú súlyozott kapcsolatokat is használ. Az előreccsatoló súlyokat a Hebb szabállyal, az oldalirányú kapcsolatok súlyait pedig az anti-Hebb szabállyal tanítja. Bizonyítható, hogy az előreccsatoló súlyvektorok itt is olyan vektorokhoz konvergálnak, melyek a megfelelő számú legfontosabb sajátvektor által kifeszített tér egy bázisát képezik. Ez, illetve egyéb alter háló változatok lényegi újdonságot – különösen a műszaki alkalmazási lehetőségeket tekintve – nem hoznak, ezért a részletesebb bemutatásukra nem térünk ki.

4.3.2. Főkomponens hálók

Az alábbiakban a valódi főkomponensek meghatározására alkalmas két hálózatot mutatunk be. E hálózatok a processzáló elemek számának megfelelő számú sajátvektort és a bemeneti vektorok ezen sajátvektor irányokba eső vetületeit határozzák meg olyan módon, hogy M processzáló elem esetén az első M legnagyobb sajátértéknek megfelelő sajátvektort és a hozzájuk tartozó vetületeket szolgáltatják. A két eljárás – bár kissé eltérő architektúrával rendelkeznek és a tanulási szabályaik sem azonosak – hasonló elveken alapul: mindkettő az eredeti Oja szabályt kombinálja a Gram-Schmidt ortogonalizálási eljárással.

4.3.2.1. A Sanger algoritmus (Generalized Hebbian Algorithm, GHA)

A Sanger hálózat egy N -bemenetű, $M \leq N$ kimenetű lineáris előreccsatolt hálózat, amelynek első kimenete a legfontosabb főkomponenst az Oja szabály szerint határozza meg. Az első neuronnak a súlyvektora a legnagyobb sajátértékhez tartozó sajátvektor. A további neuronok szintén az Oja szabályt alkalmazzák, azonban az eredeti bemenőjeltől eltérő bemenetekre. Az eltérő bemeneteket Gram-Schmidt ortogonalizálással állítjuk elő. Az eredeti bemenőjeltől kivonjuk a már előállított főkomponenseket és az így kapott jellel alkalmazzuk az Oja szabályt.

Nyilvánvaló, hogy ha a bemenőjeltől kivonjuk az első főkomponenst, a módosított bemenőjel legfontosabb főkomponense (és a módosított bemenőjelhez rendelt autokorrelációs mátrix első sajátvektora) az eredeti jel második legfontosabb főkomponense (az eredeti autokorrelációs mátrix második sajátvektor irányára vetített komponens) lesz. A hálózat második neuronja ezért a második legfontosabb sajátvektort és az ebbe az irányba eső jelkomponenst állítja elő. A hálózatot további processzáló elemekkel bővítve és az eljárást tovább folytatva az összes sajátvektor, illetve főkomponens meghatározható.

Az eljárás bemutatásához vezessük be az alábbi jelölést: legyen $\mathbf{x}^{(i)}$ az i -edik neuronra kerülő módosított bemenet. Ez azt jelenti, hogy $\mathbf{x}^{(1)} = \mathbf{x}$ a hálózat eredeti bemenőjele.

Az Oja szabály szerint az első neuron súlyvektorának tanító összefüggése:

$$\Delta \mathbf{w}_1 = \mu \left(y_1 \mathbf{x}^{(1)} - y_1^2 \mathbf{w}_1 \right). \quad (10.49)$$

Képezzük a módosított bemenetet, vagyis vonjuk ki az eredeti bemenetből a bemenetnek az első sajátvektorra eső komponensét:

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \left(\mathbf{w}_1^T \mathbf{x}^{(1)} \right) \mathbf{w}_1 = \mathbf{x}^{(1)} - y_1 \mathbf{w}_1, \quad (10.50)$$

és erre alkalmazzuk ismét az Oja szabályt:

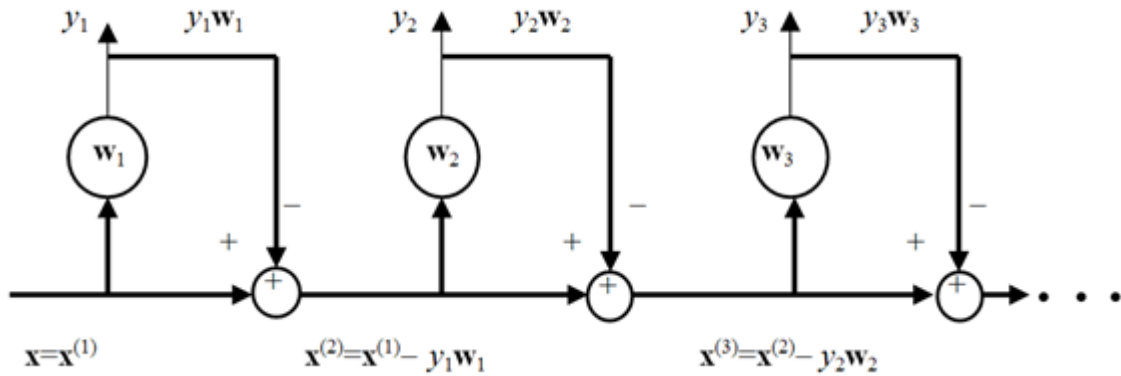
$$\Delta \mathbf{w}_2 = \mu \left(y_2 \mathbf{x}^{(2)} - y_2^2 \mathbf{w}_2 \right) = \mu \left(y_2 \mathbf{x}^{(1)} - y_1 y_2 \mathbf{w}_1 - y_2^2 \mathbf{w}_2 \right) \quad (10.51)$$

A módosított bemenet előállításánál figyelembe vettük, hogy a súlyvektor normalizált, vagyis $\|\mathbf{w}_1\| = 1$.

Hasonlóan tovább folytatva az eljárást és figyelembe véve, hogy a további súlyvektorok hossza is egységyi, az i -edik súlyvektor módosító összefüggésére a következőt kapjuk:

$$\begin{aligned}
 \Delta \mathbf{w}_i &= \mu \left(y_i \mathbf{x}^{(i)} - y_i^2 \mathbf{w}_i \right) \\
 &= \mu y_i \left(\mathbf{x}^{(1)} - y_1 \mathbf{w}_1 - \dots - y_i \mathbf{w}_i \right) \\
 &= \mu \left(y_i \mathbf{x}^{(1)} - \sum_{j=1}^{i-1} y_i y_j \mathbf{w}_j - y_i^2 \mathbf{w}_i \right) \quad (10.52)
 \end{aligned}$$

10.6. ábra - A GHA működési elve



A 10.6 ábra azt mutatja, hogy hogyan számítjuk ki az egymást követő főkomponenseket. A háló összes súlyvektorának módosító összefüggését egy közös egyenletbe is összefoghatjuk. A fenti algoritmust vektorosan felírva kapjuk a Sanger tanulási szabályt, amit általánosított Hebb algoritmusnak (*Generalized Hebbian Algorithm, GHA*) is szoktak nevezni [San89]:

$$\Delta \mathbf{W} = \mu \left[\mathbf{y} \mathbf{x}^T - \text{LT}(\mathbf{y} \mathbf{y}^T) \mathbf{W} \right], \quad (10.53)$$

ahol $\mathbf{W}$ hasonlóan az Oja altér háló összefüggéséhez most is a háló súlyvektoraiból, mint sorvektorokból képezett mátrix.

A Sanger szabály (10.53) szerinti összefüggése nagyon hasonló az általánosított Oja szabály (10.48) összefüggéséhez. A különbség mindössze annyi, hogy a zárójelen belüli kifejezés második tagját képező mátrixnak a Sanger szabálynál csak az alsó háromszög mátrixával (LT) dolgozunk. Ez valójában azt fejezi ki, hogy itt egy hierarchikus számítási modelltől van szó.

4.3.3. Az APEX háló

Az APEX háló szintén a főkomponensek meghatározására alkalmas. Felépítése hasonló az Oja altér háló felépítéséhez azzal kiegészítve, hogy itt a neuronok között egyirányú laterális kapcsolatok is vannak. Valójában ez a háló is az Oja szabály és a Gram-Schmidt eljárás kombinálásával dolgozik, azonban ezt módosított architektúrával, oldalirányú kapcsolatok bevezetésével éri el. A hálózat a főkomponenseket egyenként, egymás után határozza meg, és oldalirányú kapcsolatok csak a már meghatározott kimenetek és az éppen meghatározás alatt álló kimenet között vannak (10.7 ábra).

A hálózat tanításának összefüggéseit arra az esetre adjuk meg, amikor feltételezzük, hogy az első $j-1$ processzáló elem súlyai már beálltak, és most a j -edik neuron tanítása folyik. A hálózat j -edik kimenete:

$$y_j(k) = \mathbf{w}_j^T(k) \mathbf{x}(k) - \mathbf{q}_j^T \mathbf{y}_{j-1}(k), \quad (10.54)$$

ahol $\mathbf{y}_{j-1}(k) = [y_1(k), y_2(k), \dots, y_{j-1}(k)]^T$ a hálózat első $j-1$ kimenetéből képezett vektor, $\mathbf{w}_j$ a j -edik kimenet előrecsatoló súlyvektora, $\mathbf{q}_j$ pedig az első $j-1$ kimenetről a j -edik kimenethez csatoló oldalirányú kapcsolatok súlyvektora.

A súlymódosítás összefüggései a j -edik processzáló elem számára az alábbiak:

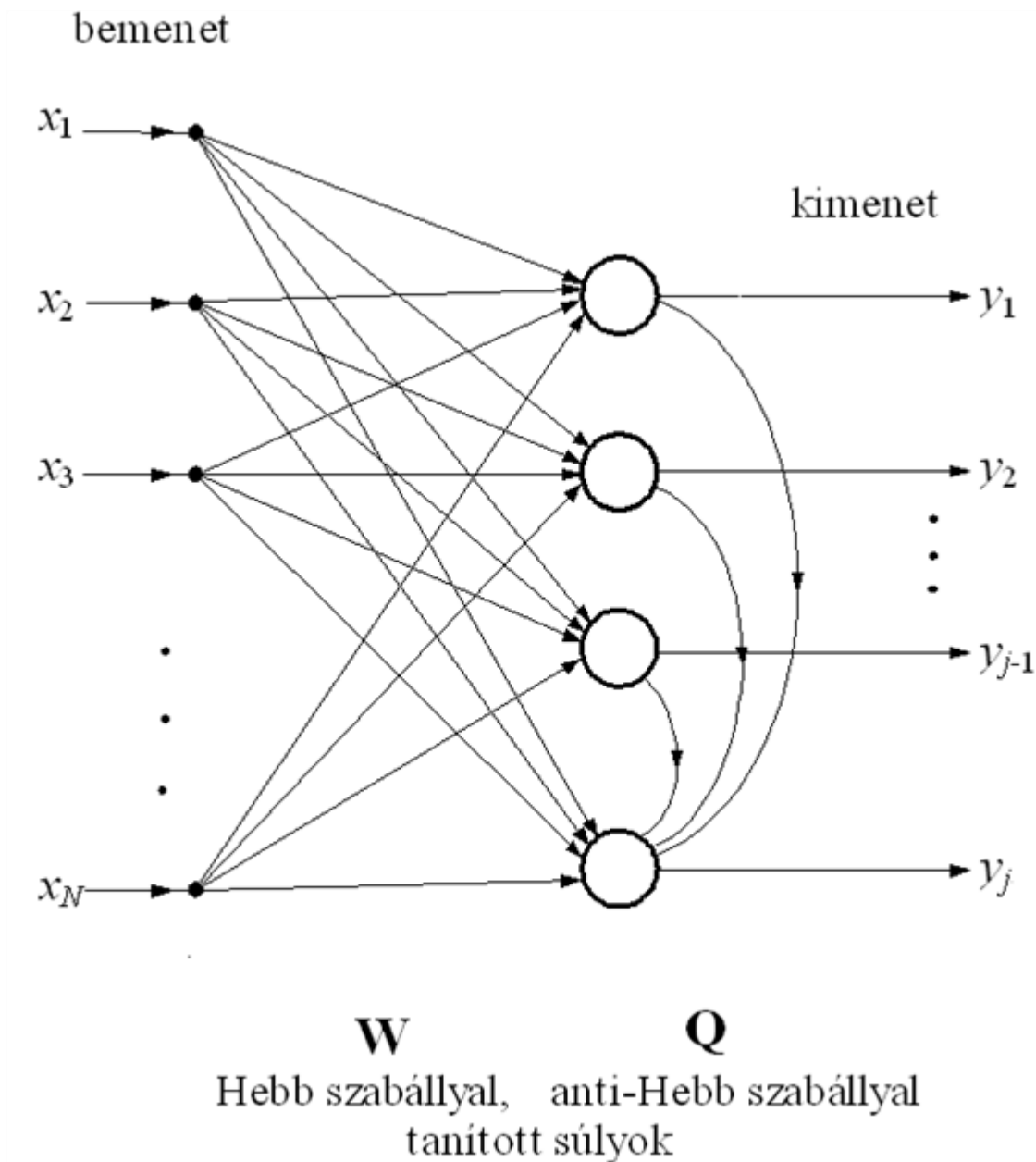
$$\Delta \mathbf{w}_j = \mu \left[y_j \mathbf{x} - y_j^2 \mathbf{w}_j \right], \quad (10.55)$$

és

$$\Delta \mathbf{q}_j = -\mu \left[y_j \mathbf{y}_{j-1} + y_j^2 \mathbf{q}_j \right]. \quad (10.56)$$

Látható, hogy az előrecsatoló súlyok tanítása az Oja szabállyal, a laterális kapcsolatok tanítása a módosított anti-Hebb szabállyal (tulajdonképpen az anti-Oja szabállyal) történik. Bizonyítható [Dia96], hogy konvergencia esetén az előrecsatoló súlyvektor a j -edik legnagyobb sajátértékhez tartozó sajátvektorhoz, az oldalirányú csatolás súlyvektora pedig $\mathbf{0}$ -hoz tart. Megmutatható az is [San93], hogy a GHA és az APEX megoldás lényegében ekvivalensek. Mindkettő valójában egymás után, egyenként, csökkenő „fontossági” sorrendben (csökkenő sajátértékek szerint) határozza meg a sajátvektorokat és a bemeneti jel főkomponenseit. Egyik háló sem állítja elő ugyanakkor a megfelelő sajátértékeket, bár ezek a kimeneti értékek átlagos négyzetes értékeivel becsülhetők. Ugyancsak közös mindkét hálónál, hogy legpontosabban a legfontosabb sajátvektort határozzák meg, az egyre csökkenő fontosságúak hibái – minthogy meghatározásukban szerepet játszanak a rangsorban előttük állók – egyre növekednek.

10.7. ábra - Az APEX háló



Fontos kérdés még a hálózatok megfelelő működéséhez a μ tanulási tényező megválasztása. Egyrészt magát a konvergenciát is befolyásolja, hogy milyen tanulási tényezőt választunk, másrészt, ha konvergens is az eljárás, a konvergenciasebesség is nagymértékben függ μ értékétől. A PCA hálózatok konvergenciájának bizonyítása a sztochasztikus approximáció eredményein alapul, ezért itt is bizonyos számú tanító lépésenként csökkenő μ alkalmazása (ld. (2.136) összefüggés) célszerű, bár nagyon sok alkalmazásban konstans μ felhasználását javasolják. Ez utóbbi esetben a konvergenciát úgy biztosíthatjuk, ha megfelelően kicsire választjuk μ -t. Az APEX hálónál az is megmutatható, hogy létezik optimális, lépésenként változó tanulási tényező is, amely ráadásul neuronfüggő. A j -edik PE súlyainak tanításához a k -adik lépésben az optimális tanulási faktor:

$$\mu_{j,opt}(k) = \frac{1}{\sigma_j^2(k)}, \quad (10.57)$$

ahol $\sigma_j^2(k)$ a hálózat j -edik kimenetének átlagos négyzetes értéke (varianciája). Az optimális érték helyett a gyakorlatban könnyebben alkalmazható az alábbi összefüggés szerint választott együttható [Kun90]:

$$\mu_j = \frac{1}{\lambda_{j-1}}, \quad (10.58)$$

Ezzel a választással a biztonságos, de kissé lassúbb konvergencia irányában módosítjuk μ -t, hiszen σ_j^2 a j -edik sajátérték, λ_j becslése ($\sigma_j^2(k) \rightarrow \lambda_j$ ha $k \rightarrow \infty$), és $\lambda_j < \lambda_{j-1}$.

A PCA hálózatok egyik legfőbb előnye az egyéb KLT eljárásokhoz képest, hogy a transzformációt nem két, hanem egyetlen lépésben végzik. Nincs szükség tehát előbb a bemeneti jel autokorrelációs (autokovariancia) mátrixának meghatározására és ennek alapján a sajátvektorok kiszámítására, hanem közvetlenül a bemeneti adatokból dolgozhatunk. Ennek ellenére nem állíthatjuk, hogy a neurális módszer a legjobb a KLT meghatározására. Számos olyan kérdés merül föl (konvergenciasebesség, pontosság, beleértve a numerikus pontosságot is), melyek részletes elemzése még nem vagy csak részben történt meg.

A PCA hálózatok aszimptotikusan a KL transzformációt eredményezik, így az optimális lineáris transzformáció megvalósításának eszközei. Számos gyakorlati feladatnál szokás azonban egyéb transzformációkat alkalmazni, melyek az optimális transzformációt csak közelítik, de melyek kiszámítása lényegesen egyszerűbb (pl. képtömörítésnél a KL transzformáció helyett a diszkrét koszinusz transzformációt (DCT) alkalmazzák [Wal91]).

Az eddigi adattömörítő hálózatokra jellemző, hogy lineáris egyrétegű felépítéssel rendelkeznek. A következőkben röviden bemutatjuk, hogy adattömörítést többretegű hálózatokkal is lehetséges.

4.4. Lineáris többretegű perceptron, mint adattömörítő hálózat

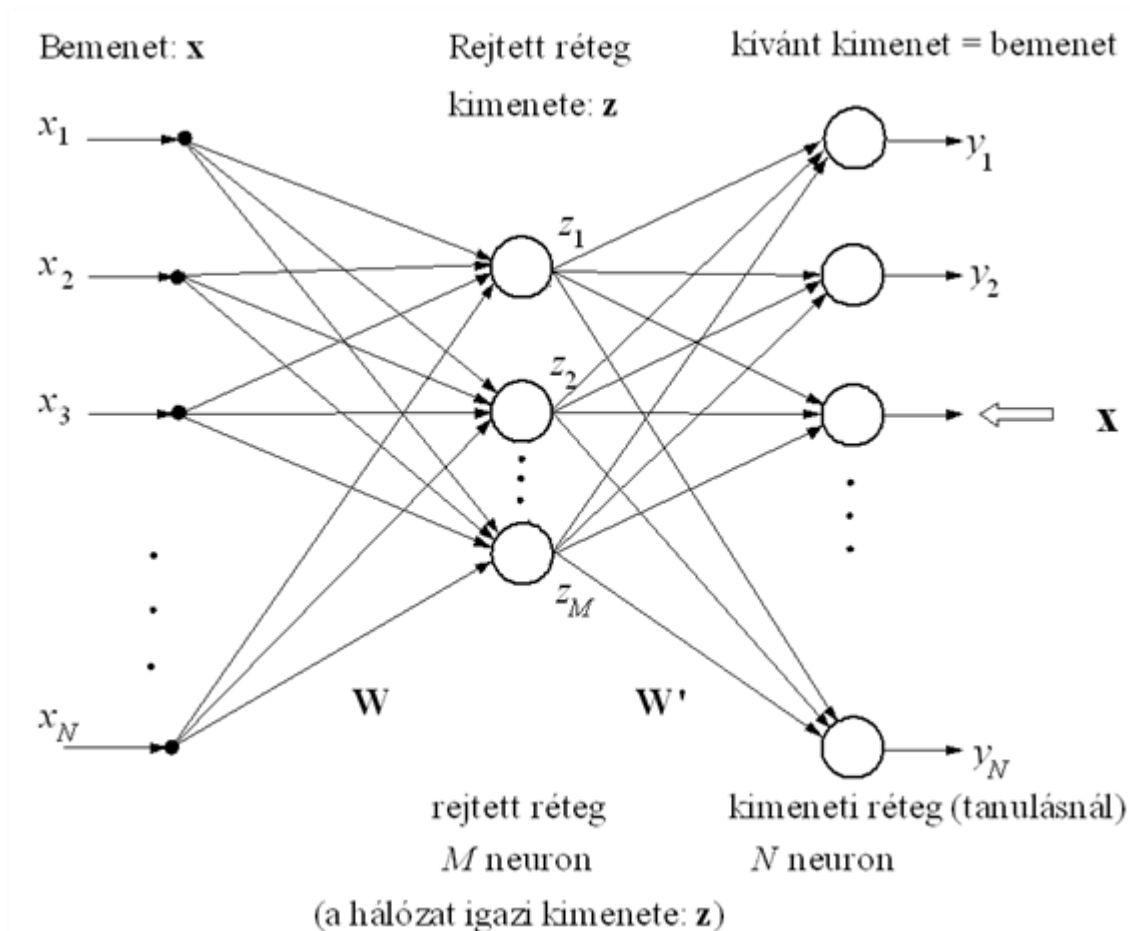
Képzeljünk el egy olyan két aktív rétegű perceptront, amely lineáris neuronokból épül fel és autoasszociatív módon működik, vagyis adott bemenetre válaszként magát a bemenetet várjuk. Az autoasszociatív hálónál a kívánt kimenet megegyezik a bemenettel. Amennyiben a rejtett rétegbeli neuronok száma (M) kisebb, mint a bemenetek (és ennek megfelelően a kimenetek) száma (N), akkor a rejtett rétegbeli neuronok kimenő értékei a bemenet tömörített (közelítő) reprezentációját adják (ld. 10.8 ábra).

A rejtett réteg képezi a háló "szűk keresztmetszetét". Ha a hálót a szokásos hibavisszaterjesztéses algoritmussal tanítjuk, a háló által előállított kimenet ($\mathbf{y}$) átlagos négyzetes értelemben közelíti a háló bemeneti jelét ($\mathbf{x}$). A háló kimeneti rétege a rejtett rétegbeli M -dimenziós reprezentációból állítja vissza az N -dimenziós kimenetet, tehát a rejtett réteg kimenetén a bemenőjel kisebb dimenziós altérbe vett vetületét kapjuk meg, olyan módon, hogy e közelítő ábrázolásból az eredeti jel a legkisebb átlagos négyzetes hibával állítható vissza. Az altér lineáris neuronok mellett bizonyítottan [Bal89] a megfelelő KLT alteret jelenti, de az altérben a bázisvektorok nem feltétlenül lesznek a sajátvektorok.

Megmutatható, hogy az autoasszociatív hálózat hibája nem függ attól, hogy a rejtett rétegbeli neuronok lineáris vagy nemlineáris kimenettel rendelkeznek. A háló mindkét esetben a bemenetnek a főkomponensek alterébe eső vetületét adja.

Az adattömörítő többretegű perceptron olyan hálózatra példa, ahol annak ellenére, hogy lineáris processzáló elemekkel dolgozunk, a több réteg alkalmazásának értelme van, ugyanis épp a közbenső, kisebb dimenziós rejtett réteg szolgáltatja a bemenet tömörített változatát.

10.8. ábra - Lineáris többretegű perceptron, mint adattömörítő autoasszociatív háló



5. Nemlineáris PCA és altér hálók

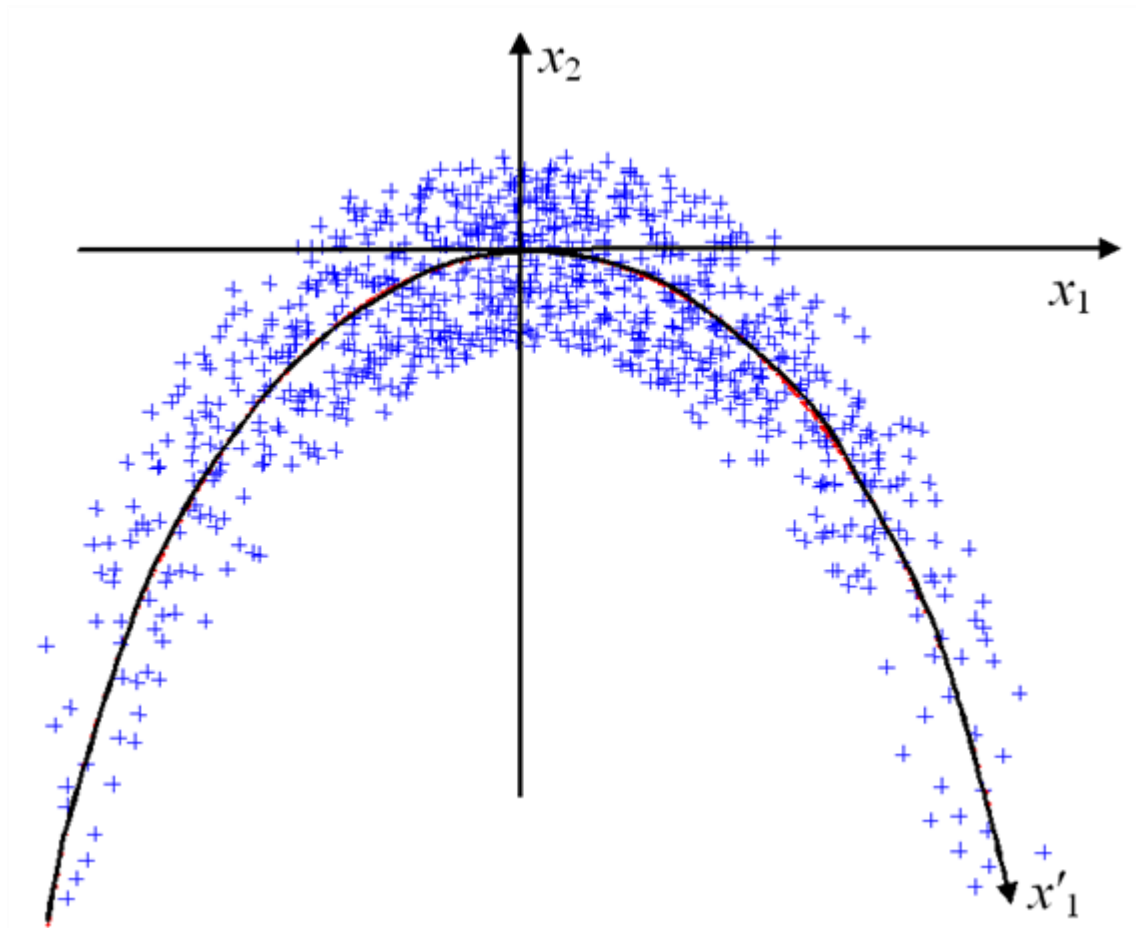
Az utóbbi időben több olyan eredmény is született, melyek nemlineáris kiterjesztései a PCA illetve az altér hálóknak. A nemlineáris hálók működésének elve nagymértékben hasonlít az eddig bemutatott PCA hálókéhoz azzal a különbséggel, hogy a nemlineáris hálónál nem közvetlenül a bemeneti adatokat dolgozzuk fel, hanem előbb egy nemlineáris transzformációt alkalmazunk – amely általában dimenziónövelő transzformáció is egyben –, majd a transzformált adatokra alkalmazzuk a már ismert eljárásokat. A nemlineáris transzformáció miatt itt esetenként a lineáris hálókénál is nagyobb mértékű tömörítés, ill. azonos mértékű tömörítés mellett kisebb hiba is elérhető.

A nemlineáris tömörítésre mutat példát a 10.9 ábra. Az ábrán látható adatok esetén nem találunk olyan lineáris transzformációt, melynek alkalmazását követően a kétdimenziós adatok olyan egydimenziós reprezentációját nyernénk, hogy a közelítés átlagos négyzetes értelemben vett hibája ne lenne jelentős. Nemlineáris transzformációt is megengedve – tehát ha az ábrán az x_1, x_2 koordináta-rendszer helyett az x'_1 , és az x'_1 -re minden pontban merőleges x'_2 koordináta-rendszerben adjuk meg az adatokat – látható, hogy az x'_2 koordináta elhagyása átlagos négyzetes értelemben az adatok reprezentációjában csak kis hibát eredményez: az így kapott egydimenziós reprezentáció hibája jóval kisebb, mintha az eredeti komponensek bármelyikét hagytuk volna el.

A főkomponens analízis célja az adattömörítésen kívül más is lehet. Sok esetben nem az adatok dimenzióját akarjuk redukálni, hanem a főkomponensek meghatározásával az adatok belső struktúráját szeretnénk meghatározni. A 10.9 ábrán bemutatott példa esetében sem feltétlenül az a cél, hogy a kétdimenziós reprezentáció helyett egydimenziós közelítő reprezentációt kapjunk. Önmagában annak a felismerése is hasznos lehet, hogy az x'_1 -menti és egy erre merőleges komponens az adatok struktúráját jobban kifejezi, mintha az eredeti x_1 és x_2 komponenseket használnánk. A főkomponenseket ezért nemcsak a bemeneti térben, az adatok eredeti reprezentációja mellett, hanem valamilyen nemlineáris transzformáció útján kapott jellemzőtérben is érdemes keresni. A nemlineáris transzformáció közbeiktatása miatt az adatok komponensei között magasabbrendű korreláció, az adatokban rejtve meglévő valamilyen nemlineáris struktúra felderítése is lehetséges.

A nemlineáris főkomponens analízis eljárásoknál is – hasonlóan a lineáris eljárásokhoz – olyan új koordináta-rendszert keresünk, melynek egyes koordinátái jelentős mértékben eltérő fontosságúak az adatok előállításában. A kétféle eljárás közötti alapvető különbség, hogy itt a megfelelő transzformáció keresését nem korlátozzuk a lineáris transzformációk körére. A jellemzőtérben történő főkomponens analízis elvégzéséhez előbb meg kell határozni a jellemzőtérre való leképezés nemlineáris transzformációját, majd ebben – a bemeneti térnél sokszor nagyobb dimenziós – térben kell elvégezni a főkomponensek meghatározását. A módszer nehézségét már lineáris eljárásnál is az okozta, hogy a transzformáció bázisa a kiinduló jel függvénye. Nemlineáris esetben a megfelelő transzformáció megtalálása és hatékony megvalósítása még nehezebb feladat.

10.9. ábra - Nemlineáris dimenzió redukció



A következőkben két nemlineáris eljárást mutatunk be. Az első eljárásnál alkalmazott nemlineáris transzformáció általában a bemeneti térnél sokkal nagyobb dimenziós jellemzőteret eredményez, azonban a jellemzőtérbeli főkomponens analízist nem ebben a térben, hanem az ebből származtatott kernel térben tudjuk megoldani. Itt tehát nincs szükség a jellemzőtérbeli transzformáció explicit definiálására és a jellemzőtérbeli reprezentáció meghatározására. A kernel gépeknél bemutatott kernel trükk segítségével ugyanis a jellemzőtérbeli főkomponens analízis elvégezhető a kernel térben is. Az ún. kernel PCA célja az adatokban meglévő rejtett (nemlineáris) struktúra meghatározása. A kernel PCA tehát elsődlegesen nem adattömörítésre szolgál.

A második bemutatott megoldás nemlineáris adattömörítést végez, mégpedig többrétegű perceptronnal. A megoldás tehát az előzőekben bemutatott perceptron-alapú altér háló nemlineáris megfelelője.

5.1. Kernel PCA

A PCA során a bemeneti térben keresünk főkomponenseket úgy, hogy a bemenetek megfelelő lineáris transzformációját végezzük. A kernel PCA ezzel szemben nem a bemeneti térben keres főkomponenseket, hanem előbb a bemeneti vektorokat nemlineáris transzformációval egy ún. jellemzőtérbe transzformálja, és itt keres főkomponenseket.

Az eljárás bemutatásához a következő jelölésekből induljunk ki. Jelöljük a bemeneti tér $\mathbb{R}^N$ -t a jellemzőtérbe való nemlineáris transzformációt Φ -vel. A bemeneti tér lehet pl. a valós szám N -esek tere, ekkor a nemlineáris transzformáció Φ -ből egy F jellemzőtérbe képez le:

$$\Phi: \mathbb{R}^N \rightarrow F, \quad \mathbf{x} \mapsto \mathbf{X} = \Phi(\mathbf{x}) \quad (10.59)$$

Az F jellemzőtér tetszőlegesen sokdimenziós, akár végtelen dimenziós tér is lehet. Tételezzük fel, hogy az F térben is fennáll, hogy $\sum_{k=1}^P \Phi(\mathbf{x}_k) = \mathbf{0}$, ahol P a bemeneti vektorok száma. Becsüljük a jellemzőtérbeli kovarianciamátrixot a véges számú mintapont (jellemzőtérbeli vektor) alapján:

$$\bar{\mathbf{C}} = \frac{1}{P} \sum_{j=1}^P \Phi(\mathbf{x}_j) \Phi(\mathbf{x}_j)^T \quad (10.60)$$

A jellemzőtérbeli főkomponensek meghatározásához először most is meg kell határoznunk a kovarianciamátrix nemnulla sajátértékeit és a megfelelő sajátvektorokat, melyek kielégítik a szokásos sajátvektor-sajátérték egyenletet:

$$\lambda \mathbf{V} = \bar{\mathbf{C}} \mathbf{V}, \quad (10.61)$$

majd a jellemzőtérbeli főkomponenseket a $\Phi(\mathbf{x})$ jellemzőtérbeli vektorok és az egységnyi hosszúságúra normált $\mathbf{V}$ sajátvektorok skalár szorzataként kapjuk.

A sajátértékek és a sajátvektorok meghatározásához hasznos, ha felhasználjuk, hogy a $\bar{\mathbf{C}}$ kovarianciamátrix sajátvektorai a jellemzőtérbeli vektorok által kifeszített térben vannak:

$$\mathbf{V} = \sum_{i=1}^P \alpha_i \Phi(\mathbf{x}_i) \quad (10.62)$$

tehát léteznek olyan α_i ($i=1, \dots, P$) együtthatók, melyek segítségével a sajátvektorok előállíthatók a bemeneteket a jellemzőtérben reprezentáló vektorok súlyozott összegeiként. A (10.62) összefüggés felhasználásával azonban meg tudjuk mutatni, hogy a jellemzőtérbeli főkomponensek anélkül is meghatározhatók, hogy a bemeneti vektorok jellemzőtérbeli reprezentációját meghatároznánk.

Ennek érdekében tekintsük a következő egyenletet:

$$\lambda \Phi^T(\mathbf{x}_k) \mathbf{V} = \Phi^T(\mathbf{x}_k) \bar{\mathbf{C}} \mathbf{V}, \quad k=1, \dots, P \quad (10.63)$$

Helyettesítsük ebbe az egyenletbe $\bar{\mathbf{C}}$ (10.60) és $\mathbf{V}$ (10.62) összefüggését. Ekkor minden $k=1, \dots, P$ -re a következőt kapjuk:

$$\lambda \sum_{i=1}^P \alpha_i \Phi^T(\mathbf{x}_k) \Phi(\mathbf{x}_i) = \frac{1}{P} \sum_{i=1}^P \alpha_i \Phi^T(\mathbf{x}_k) \sum_{j=1}^P \Phi(\mathbf{x}_j) \Phi^T(\mathbf{x}_j) \Phi(\mathbf{x}_i) \quad (10.64)$$

Vegyük észre, hogy ebben az összefüggésben a jellemzőtérbeli $\Phi(\mathbf{x})$ vektorok mindig csak skalár szorzat formájában szerepelnek.

Definiáljunk egy $P \times P$ méretű $\mathbf{K}$ kernel mátrixot, melynek (i,j) -edik eleme:

$$K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i) \Phi(\mathbf{x}_j). \quad (10.65)$$

Ezzel a (10.64) összefüggés az alábbi tömör formában is felírható:

$$P \lambda \mathbf{K} \boldsymbol{\alpha} = \mathbf{K}^2 \boldsymbol{\alpha}, \quad (10.66)$$

ahol az $\mathbf{a}$ oszlopvektor az α_i $i=1, \dots, P$ együtthatókból áll. $\mathbf{K}$ szimmetrikus mátrix, és ha megoldjuk a következő sajátvektor-sajátérték problémát:

$$P\lambda \mathbf{a} = \mathbf{K}\mathbf{a}, \quad (10.67)$$

ahol az $\mathbf{a}$ vektorok $\mathbf{K}$ sajátvektorai és a $P\lambda$ értékek a sajátértékek, a megoldás kielégíti a (10.66) egyenletet is. Jelöljük $\mathbf{K}$ nemnulla sajátértékeit nagyság szerint sorbarendezve $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_P$ -vel, a hozzájuk tartozó sajátvektorokat pedig $\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(P)}$ -vel, és legyen λ_r az első (legkisebb) nemnulla sajátérték. (Ha feltételezzük, hogy $\Phi(\mathbf{x})$ nem azonosan $\mathbf{0}$, akkor mindig léteznie kell egy ilyen λ_r -nek.) Normalizáljuk az $\mathbf{a}^{(r)}, \dots, \mathbf{a}^{(P)}$ sajátvektorokat, hogy az F térben a következő egyenlőség teljesüljön $k=r, \dots, P$ -re :

$$\mathbf{V}^{(k)T} \mathbf{V}^{(k)} = 1 \quad (10.68)$$

Ez a következő normalizálási feltételt szabja az $\mathbf{a}$ sajátvektorokra:

$$\begin{aligned} 1 &= \sum_{i,j=1}^P \alpha_i^{(k)} \alpha_j^{(k)} \Phi^T(\mathbf{x}_i) \Phi(\mathbf{x}_j) \\ &= \sum_{i,j=1}^P \alpha_i^{(k)} \alpha_j^{(k)} K_{ij} = \mathbf{a}^{(k)T} \mathbf{K} \mathbf{a}^{(k)} \\ &= \lambda_k \mathbf{a}^{(k)T} \mathbf{a}^{(k)} \end{aligned} \quad (10.69)$$

A főkomponensek meghatározása után szükségünk van még a jellemzőtérbeli vektorok sajátvektorok szerinti vetítésére. Legyen $\mathbf{x}$ egy tesztpont, $\Phi(\mathbf{x})$ képpel F -ben, ekkor

$$\mathbf{V}^{(k)T} \Phi(\mathbf{x}) = \sum_{i=1}^P \alpha_i^{(k)} \Phi^T(\mathbf{x}_i) \Phi(\mathbf{x}) = \sum_{i=1}^P \alpha_i^{(k)} K(\mathbf{x}_i, \mathbf{x}) \quad (10.70)$$

A jellemzőtérbeli főkomponens tehát a közvetlenül a kernel értékek függvényében kifejezhető, anélkül, hogy a $\Phi(\mathbf{x})$ nemlineáris leképezéseket meg kéne határozni. Tehát itt is a kernel trükköt alkalmazhatjuk, ha a nemlineáris PCA számítását nem a $\Phi(\mathbf{x})$ nemlineáris leképezések rögzítésével, hanem a $\mathbf{K}$ mátrix (a kernel függvény) megválasztásával végezzük. A kernel PCA-nál tehát nem a $\Phi(\mathbf{x})$ nemlineáris leképezésekből, hanem a kernel függvényből indulunk ki. A kernel függvény implicit módon definiálja a jellemzőtérbeli leképezést.

Összefoglalva a következő teendőink vannak a főkomponensek meghatározása során. Először meg kell választanunk a kernel függvényt, majd meg kell határoznunk a $\mathbf{K}$ mátrixot. Ennek a mátrixnak kell kiszámítanunk az $\mathbf{a}^{(k)}$ sajátvektorait. A sajátvektorok normalizálását követően határozhatjuk meg a bemeneti vektorok jellemzőtérbeli főkomponenseit a (10.70) összefüggés felhasználásával.

Az eljárás fő előnye abban rejlik, hogy a $\Phi(\mathbf{x})$ függvény ismeretére nincs szükségünk, továbbá, hogy míg az eredeti PCA során a kovarianciamátrix mérete a bemeneti dimenziótól függ, addig itt a $\mathbf{K}$ mátrix méretét a tanítópontok száma határozza meg. Lineáris PCA-nál legfeljebb N sajátvektort és így N főkomponenst találunk, ahol N a bemeneti vektorok dimenziója. Kernel PCA-nál maximum P nemnulla sajátértéket kaphatunk, ahol P a mintapontok száma.

5.1.1. A nulla várhatóérték biztosítása a jellemzőtérben

A korábbiakban tények $\Phi(\mathbf{x})$ zeltük, hogy az F térben igaz a $\sum_{k=1}^P \Phi(\mathbf{x}_k) - \frac{1}{P} \sum_{k=1}^P \Phi(\mathbf{x}_k)$ egállapítás. Ez nyilvánvalóan nem lehet igaz minden függvényre, így szükségünk van arra, hogy a jellemzőtérbeli vektorokat is $\mathbf{0}$ átlagértékűvé transzformáljuk. Ez megoldható, ha a vektorokból kivonjuk az átlagukat:

$$\tilde{\Phi}(\mathbf{x}_i) = \Phi(\mathbf{x}_i) - \frac{1}{P} \sum_{k=1}^P \Phi(\mathbf{x}_k) \quad (10.71)$$

Az eddigi megállapítások szerint most ez alapján kell meghatározni a kovarianciamátrixot, illetve a

$$\tilde{K}_{ij} = \tilde{\Phi}^T(\mathbf{x}_i) \cdot \tilde{\Phi}(\mathbf{x}_j) \quad (10.72)$$

mátrixot az F térben. Az így kapott $\tilde{\mathbf{K}}$ mátrix sajátérték-sajátvektor rendszerét kell meghatároznunk:

$$\tilde{\lambda} \tilde{\mathbf{u}} = \tilde{\mathbf{K}} \tilde{\mathbf{u}} \quad (10.73)$$

ahol $\tilde{\mathbf{u}}$ a sajátvektorok együtthatóit tartalmazza a következő formában:

$$\tilde{\mathbf{V}} = \sum_{i=1}^P \tilde{\alpha}_i \tilde{\Phi}(\mathbf{x}_i) \quad (10.74)$$

A $\tilde{\mathbf{K}}$ mátrix kiszámítása a definíciós összefüggés szerint azonban nem lehetséges a módosított jellemzőtérbeli vektorok ismerete nélkül. Lehetőségünk van viszont arra, hogy a $\tilde{\mathbf{K}}$ mátrixot $\mathbf{K}$ -val kifejezzük.

Használjuk a következő jelöléseket: $K_{ij} = \left(\Phi^T(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \right)$, $1_{ij} = 1$ minden i, j -re és $(1_P)_{ij} = 1/P$. Ezek után $\tilde{\mathbf{K}}$ számítása:

$$\begin{aligned} \tilde{K}_{ij} &= \left(\left(\Phi(\mathbf{x}_i) - \frac{1}{P} \sum_{p=1}^P \Phi(\mathbf{x}_p) \right) \cdot \left(\Phi(\mathbf{x}_j) - \frac{1}{P} \sum_{k=1}^P \Phi(\mathbf{x}_k) \right) \right) = \\ &= K_{ij} - \frac{1}{P} \sum_{p=1}^P 1_{ip} K_{pj} - \frac{1}{P} \sum_{k=1}^P K_{ik} 1_{kj} + \frac{1}{P^2} \sum_{p,k=1}^P 1_{ip} K_{pk} 1_{kj} = \\ &= (\mathbf{K} - \mathbf{1}_P \mathbf{K} + \mathbf{K} \mathbf{1}_P + \mathbf{1}_P \mathbf{K} \mathbf{1}_P)_{ij} \end{aligned} \quad (10.75)$$

Ezután számíthatók ki a sajátértékek és a sajátvektorok, a főkomponensek számítása pedig ugyanaz, mint a nem központosított adatok esetében.

5.1.2. Jelviszaállítás

Láttuk, hogy a kernel PCA a jellemzőtérben határoz meg főkomponenseket, ezért a főkomponensekből szintén a jel jellemzőtérbeli reprezentációját tudnánk előállítani. Azt is láttuk ugyanakkor, hogy a kernel trükk miatt valójában nem is dolgozunk a jellemzőtérben, hiszen a jellemzőtérbeli vetületeket is meg tudjuk határozni a kerneltérbeli reprezentáció segítségével. Ha azt szeretnénk tudni, hogy mi a jellemzőtérbeli közelítő reprezentáció hatása a bemeneti térben, akkor a jellemzőtérbeli főkomponensekből vissza kell állítanunk a jelet a bemeneti térben. Ez a feladat egyáltalán nem triviális, sőt nem is feltétlenül egyértelmű. A jelviszaállításhoz Sebastian Mika [Mik99] és munkatársai javasoltak közvetett eljárást. E szerint a bemeneti térben keresünk olyan vektort, amelynek jellemzőtérbeli főkomponensei minél inkább hasonlók a visszaállítandó jel főkomponenseihez.

Jelöljük az eredeti jel m főkomponens alapján kapott jellemzőtérbeli közelítő reprezentációját $\hat{\mathbf{x}}_m$ -mel. Ekkor

$$\hat{\mathbf{X}}_m = \sum_{k=1}^m \beta_k \mathbf{V}^{(k)}, \quad (10.76)$$

vagyis a közelítő reprezentáció a jellemzőtérbeli sajátvektorok lineáris kombinációjaként állítható elő. A jelviszaállításhoz olyan bemenetet keresünk, melynek a jellemzőtérbeli képe minél kisebb mértékben tér el $\hat{\mathbf{X}}_m$ -től. E mögött az a feltevés áll, hogy ha két jel jellemzőtérbeli reprezentációja között az eltérés kicsi, akkor a bemeneti térben is kicsi a köztük lévő eltérés. Négyzetes hibakritériumot alkalmazva ez azt jelenti, hogy keressük azt az $\hat{\mathbf{x}}$ bemeneti vektort, melyre

$$C(\hat{\mathbf{x}}) = \|\Phi(\hat{\mathbf{x}}) - \hat{\mathbf{X}}_m\|^2 \quad (10.77)$$

minimális. Behelyettesítve (10.77)-be (10.76)-ot és $\mathbf{V}^{(k)}$ (10.62) összefüggését, az eltérésre a következőt kapjuk:

$$C(\hat{\mathbf{x}}) = K(\hat{\mathbf{x}}, \hat{\mathbf{x}}) - 2 \sum_{k=1}^m \beta_k \sum_{i=1}^P \alpha_i^{(k)} K(\hat{\mathbf{x}}, \mathbf{x}_i) + \Omega \quad (10.78)$$

ahol Ω függtelen $\hat{\mathbf{x}}$ -től. A (10.78) kritérium minimumát biztosító $\hat{\mathbf{x}}$ gradiens eljárással megkereshető, ha rögzítettük a kernel függvényt.

10.1 példa

A kernel PCA működését egy egyszerű példán illusztráljuk [Sch96c]. Egy kétdimenziós adathalmaz főkomponenseit keressük, ahol az adatok generálása a következő módon történt: egy mintapont x_1 komponense a $[-1,1]$ intervallumba egyenletes eloszlású véletlen szám, x_2 komponensét pedig az alábbi kapcsolat alapján

határozhatjuk meg $x_2 = x_1^2 + \xi$ ahol ξ egy nulla várható értékű $\sigma=0,2$ szórású normális eloszlású véletlen zaj.

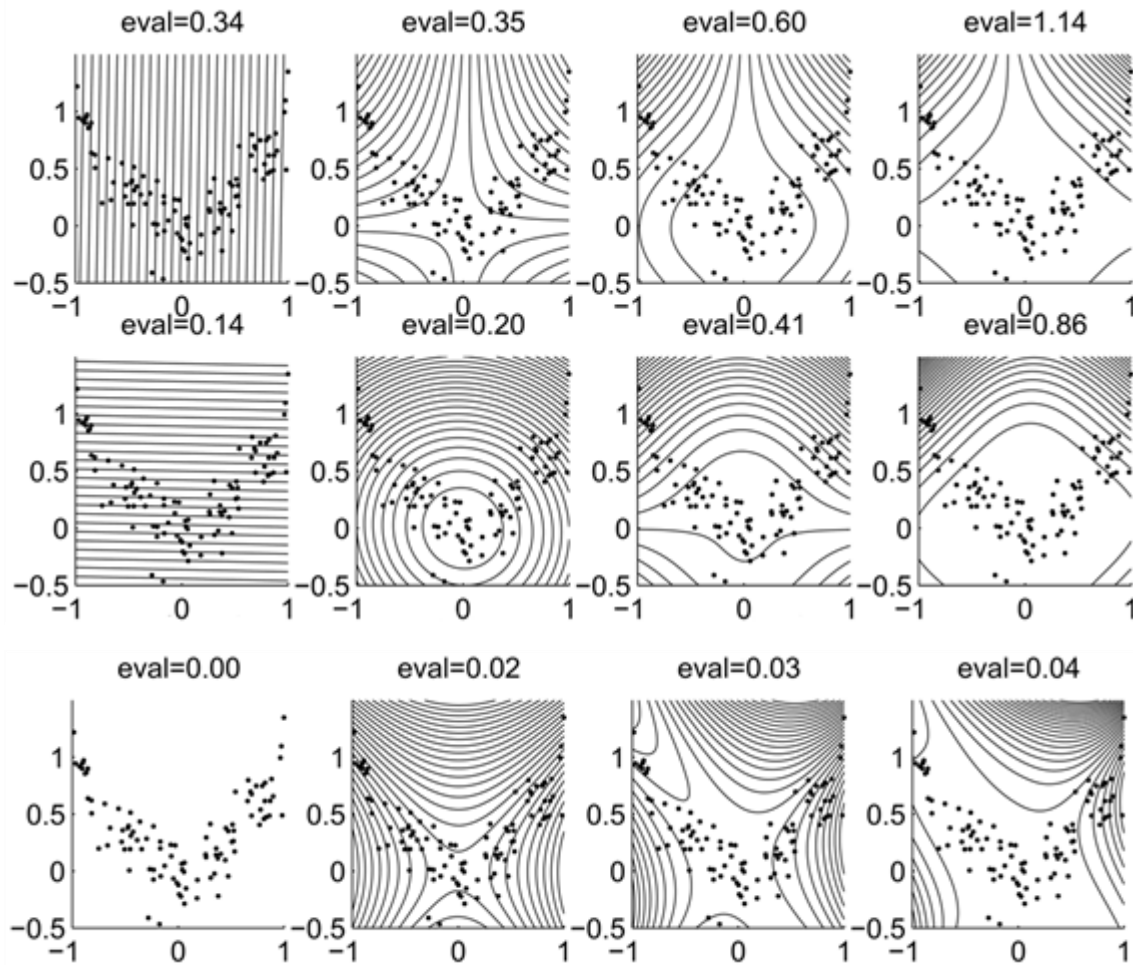
Válasszunk a feladat megoldásához polinomiális kernelt:

$$K(\mathbf{x}_i, \mathbf{x}) = \left(\mathbf{x}_i^T \mathbf{x} \right)^q$$

A 10.10 ábra különböző q értékekre mutatja a főkomponens analízis eredményét. Balról jobbra az egyes oszlopok rendre $q=1, 2, 3$ és 4 érték melletti polinomiális kernellel született eredményeket mutatnak olyan módon, hogy az egyes oszlopokban a három legnagyobb sajátértéknek megfelelő eredmény látható. Az ábrán a vonalak a konstans főkomponensek kontúr vonalai.

A bal szélső oszlop a $q=1$ esethez tartozik, ami valójában a lineáris PCA. Látható, hogy itt csak két sajátérték lesz nullától különböző, ami triviális is kétdimenziós bemenetek esetében. Lineáris PCA-nál a sajátvektorok az ábra vonalaira merőleges irányúak és nyilvánvalóan egymásra is merőlegesek. Az első oszlop ábráiból az is látható, hogy az adott példában a lineáris PCA főkomponensei nem tükrözik az adatok struktúráját.

10.10. ábra - Egy egyszerű példa a kernel PCA működésének illusztrálására [Sch96c]. (dr. Klaus-Robert Müller hozzájárulásával).



A magasabb fokszámú esetekre a főkomponensek már visszaadnak valamit az adatok struktúrájából. Ez jól látszik a három utolsó oszlop legfelső, a legfontosabb jellemzőtérbeli sajátértékekhez tartozó ábráiból. A magasabb fokszámú esetekben a jellemzőtér dimenziója nagyobb, mint a bemeneti tér dimenziója, így itt a várakozásnak megfelelően kettőnél több nemnulla sajátérték lehet. A második oszlopban ($q=2$) a harmadik főkomponens – ami a legkisebb sajátértékhez tartozik – az additív Gauss zaj szerepét mutatja.

Ez utóbbi megfigyelés arra is utal, hogy a kisebb sajátértékekhez tartozó főkomponensek sok esetben az adatokat terhelő zajhoz köthetők. Ilyenkor ezeknek a főkomponenseknek az elhagyása elsősorban nem a dimenzióredukciót, hanem a zajszűrést, az adatok „zajtalanítását” (*denoising*) szolgálja. A PCA eljárások mind lineáris mind nemlineáris esetben eredményezhetnek zajtalanítást. Ezt illusztrálja a következő példa.

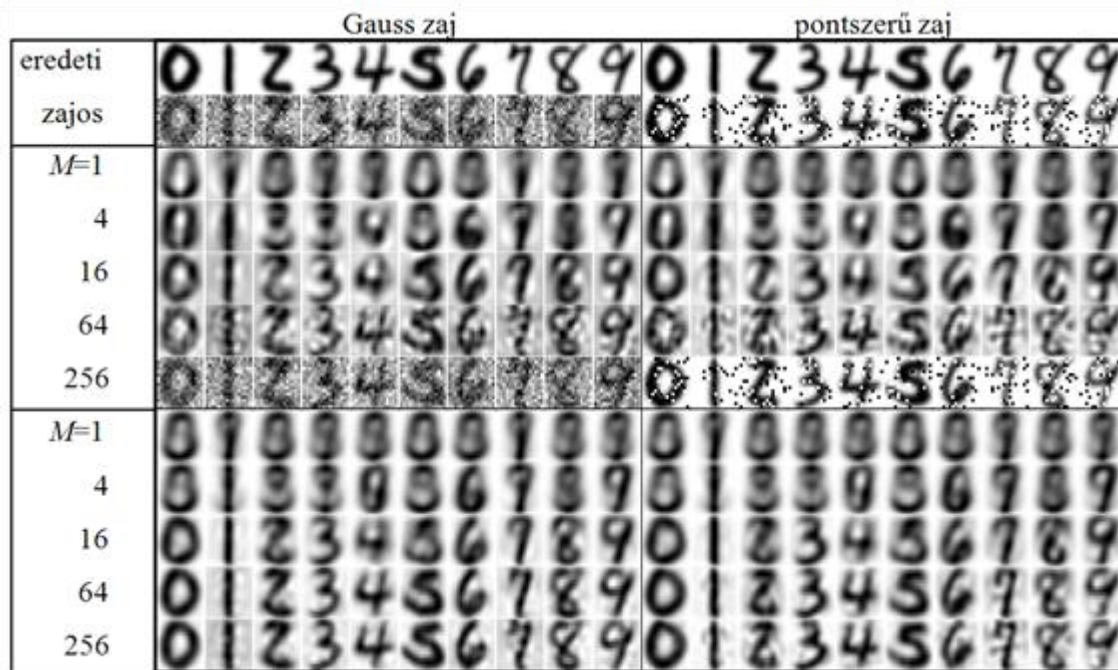
10.2 példa

A kernel PCA zajtalanító hatását egy éles feladatra való alkalmazáson mutatjuk be. A feladat kézzel írott számjegyek felismeréséhez kapcsolódik [Mik99]. A kiinduló minták 16×16 pixeles szürkeárnyaltos képek, melyeket különböző tulajdonságú additív zaj terhel. A feladatban a kernel PCA alkalmazása azt illusztrálja, hogy a kernel térben a legfontosabb főkomponensek meghagyása és a kisebb sajátértékekhez tartozó főkomponensek elhagyása alkalmas lehet a képek zajszűrésére. A példa ugyanakkor azt is illusztrálja, hogy az adott feladatban a lineáris PCA a zajszűrést megfelelő minőségben nem tudja megoldani.

A 10.11 ábra mind a kiinduló, mind a kétfajta zajjal (nulla várható értékű, $\sigma=0,5$ szórású Gauss zaj, illetve pontszerű zaj, ahol egy pixel 0,2 valószínűséggel változik fekete vagy fehér pixellé) terhelt karakterek képeit mutatja (az ábra első két sora), valamint bemutatja a lineáris PCA és a kernel PCA adott számú főkomponenséből visszaállított karakterképeket is. Minthogy az eredeti képek 256 pixelből állnak, lineáris PCA esetében maximum 256 főkomponens lehetséges. Kernel PCA mellett azonban a jellemzőtér dimenziója a választott kernel függvényről függően ennél jóval nagyobb is lehet, a sajátvektorok számát pedig a mintapontok száma határozza meg. A példában Gauss kernel függvényt és 300 tanítómintát használtak.

Az ábrából látható, hogy lineáris PCA mellett, ha nem az összes főkomponenst használjuk, az additív zaj mérsékelhető ugyan, de a képek is jelentősen torzulnak, ami a későbbi felismerést nehezíti. A kernel PCA-nál a többdimenziós jellemzőtér biztosítja, hogy a zajt úgy mérsékeljük, hogy közben a képek minősége alig változik.

10.11. ábra - A kernel PCA zajtalanító hatása [Mik99]. (dr. Klaus-Robert Müller hozzájárulásával)



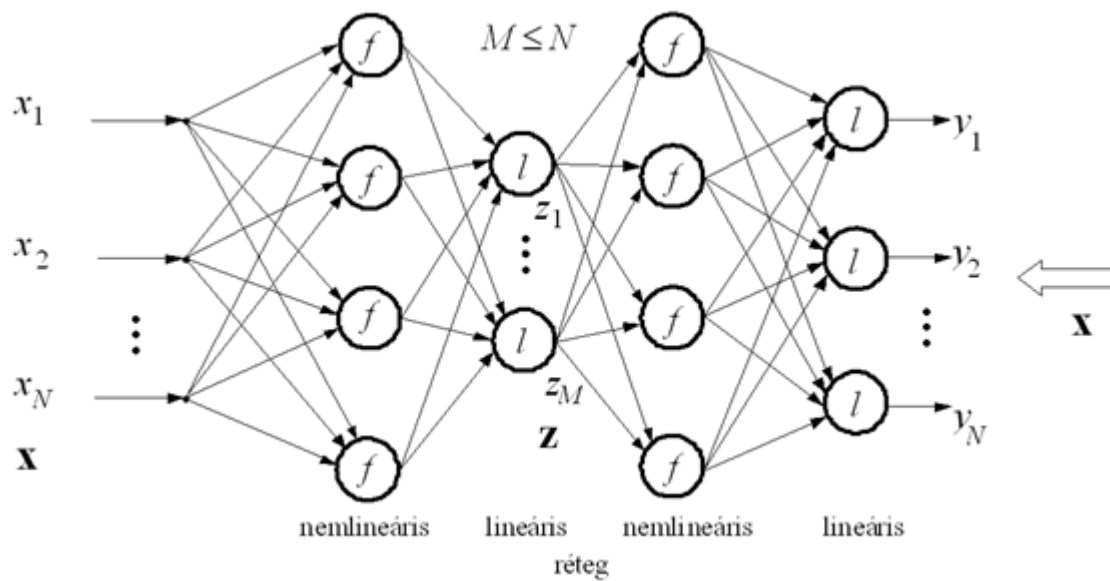
5.2. Nemlineáris többrétegű perceptron, mint adattömörítő hálózat

Természetesen adódik a gondolat, hogy használjuk ki a megfelelő neurális hálózatok azon képességét, hogy lényegében tetszőleges folytonos nemlineáris leképezés közelítésére megtaníthatók. Ugyancsak fontos tulajdonság, hogy a tanítás a kívánt választól való átlagos négyzetes eltérés minimalizálását végzi. Ha tehát megfelelő többrétegű nemlineáris leképezésre alkalmas hálót a lineáris adattömörítő MLP-hez hasonlóan autoasszociatív módon tanítunk, várható, hogy a háló nemlineáris adattömörítésre képes lesz [Mal96].

Nemlineáris adattömörítést úgy valósíthatunk meg, hogy olyan 5 rétegű (4 aktív rétegű) hálót alkalmazunk (10.12 ábra), amelynek első és harmadik rejtett rétege nemlineáris – ezt f jelöli az ábrán –, második rejtett rétege és kimeneti rétege pedig l -vel jelölt lineáris neuronokból épül fel. A hálót most is a szokásos hibavisszaterjesztéses eljárással tanítjuk, így a megtanított háló kimenete átlagos négyzetes értelemben a legkisebb hibájú közelítése a bemenetnek. Mint utaltunk rá, egy rejtett rétegű hálózat akár lineáris, akár nemlineáris rejtett neuronokat tartalmaz csak lineáris adattömörítésre alkalmas. Nemlineáris adattömörítés csak legalább három rejtett rétegű hálózattal lehetséges. Bár ez a háló nem tartozik sem a PCA hálócsaládba, sem a nemellenőrzött tanítású hálók közé, mégis itt említjük, mivel a háló által megoldott feladat rokon az előbb említettekkel.

A nemlineáris rejtett rétegbeli neuronok számát alapvetően a szükséges leképezés "nemlinearitásának mértéke" határozza meg. A középső, lineáris rejtett réteg – amelyben a bemeneti komponensek számánál kevesebb neuron található – kimenetén nyerjük a tömörített reprezentációt. A nemlineáris adattömörítő MLP tehát abban különbözik a lineáris tömörítést végző, a 10.8 ábrán bemutatott változattól, hogy kiegészül két nemlineáris rejtett réteggel, melyek alapvetően felelősek a nemlineáris leképezésért, és a tömörítést, illetve a visszaállítást hivatottak biztosítani. Az l -vel jelölt neuronoknál is alkalmazhatunk nemlineáris aktivációs függvényeket, bár a megfelelő működéshez erre valószínűleg nincs szükség. A bemutatott nemlineáris hálózat – hasonlóan lineáris megfelelőjéhez – nem feltétlenül a nemlineáris főkomponenseket, hanem az M nemlineáris főkomponens által meghatározott alterbe eső vetületet határozza meg.

10.12. ábra - Nemlineáris adattömörítő MLP hálózat



6. Független komponens analízis

A független komponens analízis (*independent component analysis, ICA*) olyan statisztikai módszer, amely adatokban (véletlen változóban, mérési eredményekben vagy akár időfüggvényekben) rejtve meglévő komponensek meghatározására szolgál. A független komponens analízis a megfigyelt többváltozós adatok generálásának egy modelljét definiálja. E modell szerint a rendelkezésre álló többdimenziós adatok egyes komponensei valamilyen ismeretlen változók lineáris vagy nemlineáris „keverékei”, súlyozott összegei, ahol a keverő rendszer szintén ismeretlen. A független komponens analízis feladata ezen rejtett változók meghatározása, feltéve, hogy a rejtett változók statisztikailag függetlenek egymástól.

A független komponens analízis a főkomponens analízis (PCA) kiterjesztésének is tekinthető, amennyiben a megfigyelt adatokban, jelekben itt is valamilyen kitüntetett komponenseket keresünk. A független komponens analízis számos gyakorlati feladat kapcsán felmerül. Ha például időfüggvények független komponensekre való szétválasztása a feladat, szokás vak forrás szeparálásról (*blind source separation, BSS*) beszélni. Ilyenkor a rendelkezésünkre álló megfigyelt időfüggvények valamilyen források időfüggvényeinek súlyozott összegeiként állnak elő és a megfigyelésekből az eredeti időfüggvényeket szeretnénk visszaállítani. Bár nem időfüggvényekre történő szétbontásról van szó, mégis hasonló feladról beszélhetünk, ha pl. képek (vagy tetszőleges többdimenziós jelek) keveréke áll rendelkezésünkre, melyekből az eredeti képeket (komponenseket) szeretnénk visszakapni.

Egy komplex jel független komponensekre bontása későbbi feldolgozó lépések (pl. felismerési, osztályozási feladatok) elvégzését is megkönnyítheti. Ilyenkor a független komponens analízis eljárásokat lényegkiemelő eljárásoknak (*feature extraction*) tekinthetjük. További fontos gyakorlati alkalmazás az ún. vak dekonvolúció (*blind deconvolution*). Ebben az esetben általában csak egyetlen megfigyelt és egyetlen forrás jelünk van, melyek tipikusan időfüggvények. A megfigyelt jel a forrás jel valamilyen szűrt változata. Lineáris szűrőt feltételezve a megfigyelt jel a forrás jel és a szűrő súlyfüggvényének konvolúciójaként nyerhető, tehát a megfigyelt jel adott időpillanatbeli értéke a forrás jel régebbi értékeinek súlyozott összegeként áll elő. A megfigyelt (szűrt) jelből a forrásjel visszaállítása dekonvolúciós feladat. Amennyiben a szűrő együtthatóit nem ismerjük a jelvisszaállítást vakon kell elvégeznünk. Megfelelő feltételek fennállta esetén a vak dekonvolúció is értelmezhető független komponens analízis feladatként.

A független komponens analízis feladatot egy hétköznapi vak forrás szeparálási feladat kapcsán, az ún. koktélparti probléma kapcsán fogjuk pontosabban definiálni. A koktélparti probléma során beszédjelek keverékéből szeretnénk az egyes beszédjeleket szétválasztani.

Képzeld el, hogy egy olyan teremben vagyunk, ahol többen beszélnek egyszerre. Amit hallunk az a sok beszéd keveréke. Ebből a beszédkeverékből szeretnénk az egyes különálló beszédjeleket elkülöníteni, hogy megértsük az egyes beszélőket.

Általánosabban a problémát a következőképpen fogalmazhatjuk meg. $s_1(t), s_2(t), \dots, s_N(t)$ ú, nulla várható értékű, statisztikailag kölcsönösen független jelünk (időfüggvényünk): , ahol t jelöli az időt. Ezek a jelek lehetnek $s_i(t)$ beszéd hullámforma időfüggvények, ahol minden egyes időfüggvény $x_1(t), x_2(t)$ bes: $x_L(t)$ származik. Az jeleket közvetlenül nem tudjuk megfigyelni, hanem helyette egy ,..., jelkész $x_j(t)$ gyelhetünk meg, pl. L különböző helyen elhelyezett mikrofon felhasználásával. Minden egyes megfigyelt jel (mikrofonjel) az egyes beszédjelek súlyozott összegeként áll elő:

$$x_i(t) = \sum_{j=1}^N a_{ij} s_j(t), \quad i = 1, 2, \dots, L \quad (10.79)$$

ahol az a_{ij} együtthatók a súlyozó értékek, melyek pl. a beszélők és a mikrofonok közötti távolságok függvényei. A kevert mikrofonjelekből szeretnénk az eredeti beszédjeleket visszaállítani, vagy legalább becsülni anélkül, hogy az a_{ij} együtthatókat ismernénk.

Amennyiben ezen jelek mintavételezett értékeivel dolgozunk, az $x_j(k)$, $j=1, 2, \dots, L$ megfigyelési sorozatokból szeretnénk az $s_i(k)$, $i=1, 2, \dots, N$ független jelsorozatokot visszaállítani, ahol k a diszkrét időindex. Ekkor a jelmodell vektoros formában is felírható:

$$\mathbf{x}(k) = \mathbf{A} \mathbf{s}(k) + \mathbf{n}(k) = \sum_{j=1}^N \mathbf{a}_j s_j(k) + \mathbf{n}(k) \quad (10.80)$$

ahol $\mathbf{A}$ a „keverő” mátrix, $\mathbf{a}_j = [a_{1j}, a_{2j}, \dots, a_{Lj}]^T$ a mátrix j -edik oszlopa., $\mathbf{s}(k) = [s_1(k), s_2(k), \dots, s_N(k)]^T$ pedig a keresett független jelforrások jeleiből álló vektor. A (10.80) összefüggéssel megadott feladat annyiban általánosabb, mint a (10.79) összefüggés, hogy itt feltételeztük, hogy a megfigyeléseinket még egy $\mathbf{n}(k)$ additív zaj is terheli. Általában feltételezhető, hogy $L \geq N$, tehát legalább annyi megfigyelt jelünk van, mint amennyi a független komponensek száma. Bár az általános feladatnál az additív zaj jelenlétével is kell számolni, a továbbiakban zajmentes esetet tételezünk fel.

A fenti alkalmazási példák mindegyike mögött ott húzódik a következő alaprobléma. Adottak az $x_1, x_2, \dots, x_N$ valószínűségi változók, melyek az $s_1, s_2, \dots, s_N$ valószínűségi változók lineáris kombinációiként modellezhetők:

$$x_i = a_{i1}s_1 + a_{i2}s_2 + \dots + a_{iN}s_N, \quad i=1, 2, \dots, N, \quad (10.81)$$

ahol az a_{ij} együtthatók valamilyen valós értékű konstansok és ahol feltételezzük, hogy az $s_1, s_2, \dots, s_N$ valószínűségi változók statisztikailag kölcsönösen függetlenek egymástól. A független komponens analízis feladatoknál mindig feltételezzük, hogy mind az eredeti s_i valószínűségi változók, mind az x_i kevert (megfigyelt) valószínűségi változók nulla várható értékűek. A független komponensek és a megfigyelések közötti kapcsolatot vektorosan felírva:

$$\mathbf{x} = \mathbf{A} \mathbf{s}. \quad (10.82)$$

ahol $\mathbf{A}$ a keverési mátrix, $\mathbf{s} = [s_1, s_2, \dots, s_N]^T$ a független komponensek vektora és $\mathbf{x} = [x_1, x_2, \dots, x_N]^T$ a megfigyelések vektora. Az alaproblémánál tehát feltételezzük, hogy a megfigyeléseink zajmentesek, továbbá azt is, hogy pontosan annyi megfigyelésünk van, mint ahány független komponensünk ($L=N$). Ez utóbbiból következik, hogy $\mathbf{A}$ kvadratikusan mátrix. A továbbiakban még azt is feltételezzük, hogy $\mathbf{A}$ invertálható. Ezekkel az egyszerűsítő feltételezésekkel a független komponensek meghatározása valójában $\mathbf{A}^{-1}$ meghatározását igényli. Ha ismernénk $\mathbf{A}$ -t, kiszámíthatnánk az inverzét, amivel a független komponensek közvetlenül meghatározhatók lennének:

$$\mathbf{s} = \mathbf{A}^{-1} \mathbf{x}. \quad (10.83)$$

A probléma legegyszerűbb változatánál a megfigyelések a független komponensekből lineáris transzformáció útján származnak és a jelvisz $\mathbf{A}^{-1}$ írás is egy lineáris transzformáció. A független komponens analízis tehát a jelvisz állító transzformáció mátrixának a meghatározását igényli.

A független komponens feladat megoldásánál tehát olyan lineáris transzformációt keresünk, mely minél jobban közelíti az $\mathbf{A}^{-1}$ mátrixot. A nehézséget az okozza, hogy $\mathbf{A}$ -t nem ismerjük. Számos valós feladatnál további nehézségeket okoz, hogy a fentiekben megfogalmazott egyszerűsítő feltételezések sem érvényesek.

A fejezetben csak az alapesetre térünk ki, a problémakör általános tárgyalására itt nincs mód. Az általánosabb esetekkel, illetve a nagyon speciális esetekkel a bőszes irodalom pl. [Jut88], [Car94], [Com94], [Kar94], [Bel95], [Ama95], [Gir96], [Kar97], [Kar98], [Hyv00], [Hyv01], [Hyv06] foglalkozik.

A fenti feltételeken túl egy további általános feltételnek is eleget kell tenni: a független komponensek nem lehetnek Gauss eloszlású valószínűségi változók. Ennek a feltételnek az a magyarázata, hogy a független komponens analízis eljárások olyan kritériumok alapján dolgoznak, melyek csak nem Gauss eloszlású valószínűségi változókra alkalmazhatók. Itt érdemes utalni a főkomponens analízis problémára. Míg a főkomponens analízis egy többdimenziós jelet korrelálatlan komponensekre bontja szét, addig a független komponens analízis összetett jelek kölcsönösen független komponenseit próbálja megtalálni, vagy legalább olyan komponenseket keres, amelyek a lehető legnagyobb mértékben függetlenek egymástól. A korrelálatlan komponensekre bontáshoz a bemenet első és másodrendű statisztikai jellemzőinek felhasználása elegendő, az ICA a magasabbrendű statisztikákat is felhasználja.

Gauss eloszlás jellemzésére a magasabbrendű statisztikák alkalmazása nem szükséges, hiszen az első és másodrendű statisztikai jellemzők a Gauss valószínűségi változók teljes statisztikai leírását megadják. Továbbá egy $\mathbf{x}$ Gauss eloszlású valószínűségi vektorváltozóra az is igaz, hogy $\mathbf{x}$ lineáris transzformációja, (pl. az (10.83) jelvisz állításnak megfelelő transzformációval) szintén Gauss eloszlású lesz, melyet várhatóérték vektorával és kovariancia mátrixával teljes mértékben tudunk jellemezni. Ennek következménye, hogy a standard független komponens analízis eljárások képtelenek a keverő, illetve a visszaállító mátrixot identifikálni. Ugyanakkor meg kell jegyezni, hogy ha a független komponensek között csak egyetlen Gauss eloszlású található, a feladat megoldható. Azt is mondhatjuk, hogy a független komponens analízis feladat úgy is megfogalmazható, hogy az $\mathbf{x}$ valószínűségi vektorváltozók olyan lineáris transzformációját, olyan vetületeit keressük, melyek a lehető legkevésbé Gauss eloszlásúak. A független komponens analízis eljárásoknál ezért a statisztikai függetlenség és a „nemgauss-ság” felcserélhető fogalmak.

A fentiek indokolják a független komponens analízis eljárások formális megfogalmazásait. Olyan eljárásokat keresünk, melyek eredményeképpen kapott komponensek kielégítik a statisztikai függetlenség valamelyik definícióját, vagy melyek eredményeképpen kapott komponensek a lehető legkevésbé Gauss eloszlásúak.

Mindkét megközelítés alapján számos eljárást dolgoztak ki. A statisztikai függetlenség mérésére a függetlenség definíciójából indulhatunk ki. A statisztikai függetlenség a valószínűség sűrűségfüggvények vagy eloszlásfüggvények segítségével definiálható. Legyen pl. y_1 és y_2 két skalár értékű valószínűségi változó és jelölje $p(y_1)$ illetve $p(y_2)$ a megfelelő valószínűség sűrűségfüggvényeket. A két valószínűségi változó független, ha a $p(y_1, y_2)$ együttes valószínűségi sűrűségfüggvényük a két sűrűségfüggvény szorzataként felírható:

$$p(y_1, y_2) = p(y_1)p(y_2) \quad (10.84)$$

Ennek felhasználásával könnyen belátható, hogy független valószínűségi változókra fennáll a következő összefüggés is:

$$E\{h_1(y_1), h_2(y_2)\} = E\{h_1(y_1)\}E\{h_2(y_2)\} \quad (10.85)$$

ahol h_1 és h_2 tetszőleges abszolút integrálható függvény lehet.

A függetlenség megállapítása a fentiekben túl a valószínűségi változók statisztikai jellemzőinek (pl. momentumainak), illetve számos az információelmélet alapján származtatott mennyiségnek az alapján is lehetséges [Hyv01]. Ilyen mennyiségek pl. az entrópia és a kölcsönös információ vagy a Kullback-Leibler divergencia. A nemgauss-ságon alapuló eljárások egyrészt a magasabbrendű momentumokat, elsősorban a

kurtózist alkalmazzák, de a nemgauss-ság mérésére is használhatók egyes információelméleti fogalmak, mint pl. a negentrópia. A különböző függetlenség vagy nemgauss-ság mértékek eltérő eljárásokat eredményeznek. Ezen eljárások közül itt csak néhányat mutatunk be. Olyan eljárásokat, melyek neuronháló architektúrával, mintapontokból való iteratív tanulással oldják meg a független komponens analízis feladatát.

A következőkben előbb egy magasabbrendű momentumokon alapuló eljárást, majd az egyik legsikeresebb független komponens meghatározó algoritmust, az ún. FastICA eljárást mutatjuk be. Végül olyan megoldásra is mutatunk példát, ahol a neurális PCA hálózatok nemlineáris változatával becsülhetők a független komponensek. A bemutatott eljárásoknál mindig feltételezzük, hogy mind a kiinduló adatok, mind az eredményként kapott adatok nulla várható értékűek.

6.1. Független komponens analízis nemgauss-ság alapján

Annak eldöntése, hogy egy valószínűségi változó Gauss eloszlású-e vagy sem bizonyos magasabbrendű momentumok segítségével is lehetséges. Ilyen magasabbrendű momentum a kurtózis, melyet nulla várható értékű valószínűségi változó esetére a következőképpen definiálhatunk:

$$\text{kurt}(y) = E\{y^4\} - 3\left(E\{y^2\}\right)^2. \quad (10.86)$$

Gauss valószínűségi változóknál a kurtózis nulla, míg olyan eloszlásoknál, melyek a nulla körül csúcsosodnak, de elnyúló farokrésszel rendelkeznek pozitív, lapos sűrűségfüggvény estében pedig negatív a kurtózis. A nemgauss-ság mérésére ezért a kurtózis abszolút értéke vagy négyzete használható. A pozitív kurtózisú eloszlásokat szuper-Gauss, a negatív kurtózisúakat pedig szub-Gauss eloszlásoknak nevezzük. A Laplace eloszlás (kétoldalú exponenciális eloszlás) jó példa a szuper-Gauss eloszlásra, egy egyenletes eloszlás valószínűségi változó pedig szub-Gauss változó. Sok gyakorlati feladatnál a jelforrások szub-Gauss vagy szuper-Gauss jelet szolgáltatnak. A beszédjelek pl. tipikusan szuper-Gauss jelek.

A kurtózis alkalmazásának a független komponensek meghatározásánál a legfőbb előnye, hogy könnyű becsülni. Ha ugyanis a véletlen változó varianciáját egységnyire tudjuk választani, a kurtózis meghatározása lényegében csak a negyedik momentum számítását igényli. Hátrányos ugyanakkor, hogy a kurtózis nagyon érzékeny a kilógó adatokra, ami viszonylag kevés mintapont alapján történő becslésnél jelentős hibát eredményezhet. A kurtózis nem robusztus mértéke a nemgauss-ságnak.

A független komponenseket megadó szeparáló mátrix meghatározása könnyebb, ha előzetesen elvégezzük az adatok ún. fehérítését. A fehérítés a megfigyelések olyan lineáris transzformációja, amelynél a transzformált (fehérített) adatok komponensei kölcsönösen korrelálatlanok és varianciájuk egységnyi.

Ha a fehérített jel $\mathbf{v} = \mathbf{V}\mathbf{x}$, ahol $\mathbf{V}$ a fehérítő transzformáció mátrixa, akkor $\mathbf{v}$ autokorrelációs mátrixa egységmátrix lesz: $E\{\mathbf{v}\mathbf{v}^T\} = \mathbf{I}$. A fehérített jel és az eredeti független komponensek közötti kapcsolat ennek megfelelően:

$$\mathbf{v} = \mathbf{V}\mathbf{x} = \mathbf{V}\mathbf{A}\mathbf{s} = \mathbf{B}\mathbf{s}. \quad (10.87)$$

Ha feltesszük, hogy a független komponensek varianciája egységnyi, fehérített jel esetén a $\mathbf{B}$ keverő mátrix ortogonális lesz: $\mathbf{B}\mathbf{B}^T = \mathbf{I}$.

A fehérített megfigyelésekből kell a visszaállítást elvégeznünk:

$$\mathbf{s} = \mathbf{B}^{-1}\mathbf{v} = \mathbf{B}^T\mathbf{v}. \quad (10.88)$$

Így egy teljes rangú, de tetszőleges mátrix meghatározása helyett egy egyszerűbb feladatot, egy ortogonális mátrix meghatározását kell megoldanunk. Ha $\mathbf{B}$ i -edik oszlopvektorát $\mathbf{b}_i$ jelöli, akkor az i -edik független komponens a fehérített megfigyelésekből $s_i = \mathbf{b}_i^T \mathbf{v}$ formában határozható meg.

Minthogy a fehérítés következtében $\mathbf{v}$ komponensei korrelálatlanok és egységnyi varianciájúak lesznek, a fehérítés PCA felhasználásával is végezhető. A PCA azonban csak a korrelálatlanságot biztosítja, az egységnyi varianciát nem. A PCA eredményeképpen kapott jel autokorrelációs mátrixa ugyan diagonális lesz, de általában

nem egységmátrix, ezért még egy normalizáló lépésre is szükség van. A fehéritő transzformáció mátrixa tehát: $\mathbf{V} = \mathbf{\Lambda}^{-1/2} \mathbf{\Phi}$, ahol $\mathbf{\Phi}$ a fehéritendő jel ($\mathbf{x}$) autokorrelációs mátrixának ($\mathbf{R}$) sajátvektoraiból képzett mátrix, $\mathbf{\Lambda}$ pedig $\mathbf{R}$ sajátértékeiből álló diagonálmátrix.

A továbbiakban feltételezzük, hogy a független komponens analízis fehéritett adatokból dolgozik. Egy független komponens meghatározásánál ezeknek az adatoknak olyan $\mathbf{w}^T \mathbf{v}$ lineáris kombinációját keressük, hogy a kurtózis maximális vagy minimális értéket vegyen fel. Természetesen a szélsőérték-keresés csak akkor alkalmazható, ha a súlyvektor hossza konstans, ezért feltesszük, hogy $\|\mathbf{w}\| = 1$.

A kurtózis maximalizálás (minimalizálás) többféle úton is elérhető, most azonban csak olyan algoritmussal foglalkozunk, ahol a szélsőérték-keresés pillanatnyi gradiens felhasználásával történik. Ekkor olyan iteratív megoldást kapunk, mely megfelelő neurális háló architektúrával megvalósítható. A neurális megoldásnál valójában egy elemi neuront használunk melynek súlyvektora lesz a keresett $\mathbf{w}$.

A fehéritett adatok miatt a kritériumfüggvény, melynek a szélsőértékét keressük a következő lesz:

$$C(\mathbf{w}) = \text{kurt}(\mathbf{w}^T \mathbf{v}) = E \left\{ \left(\mathbf{w}^T \mathbf{v} \right)^4 \right\} - 3 \left(E \left\{ \left(\mathbf{w}^T \mathbf{v} \right)^2 \right\} \right)^2 = E \left\{ \left(\mathbf{w}^T \mathbf{v} \right)^4 \right\} - 3 \|\mathbf{w}\|^4. \quad (10.89)$$

Ha figyelembe vesszük még a $\|\mathbf{w}\| = 1$ feltételt, a végleges kritériumfüggvényre a következőt kapjuk:

$$C(\mathbf{w}) = E \left\{ \left(\mathbf{w}^T \mathbf{v} \right)^4 \right\} - 3 \|\mathbf{w}\|^4 + F(\|\mathbf{w}\|^2). \quad (10.90)$$

ahol F egy büntető tag a súlyvektor hosszára vonatkozó feltétel biztosítására. A büntető tag megválasztására különböző javaslatok születtek. Az eljárás mostani bemutatásánál F konkrét megválasztásával nem foglalkozunk. Meghatározva a kritériumfüggvény pillanatnyi gradiensét – vagyis a kritériumfüggvényből a várhatóérték-képzést elhagyjuk – a súlymódosításra a következő összefüggést kapjuk:

$$\mathbf{w}(k+1) = \mathbf{w}(k) \pm \mu(k) \left[\mathbf{v}(k) \left(\mathbf{w}(k)^T \mathbf{v}(k) \right)^3 - 3 \|\mathbf{w}(k)\|^2 \mathbf{w}(k) + f(\|\mathbf{w}(k)\|^2) \mathbf{w}(k) \right], \quad (10.91)$$

ahol $\mathbf{v}(k)$ az éppen felhasznált fehéritett megfigyelés, $\mu(k)$ a lépésfüggő tanulási tényező, f pedig a büntető tagból származó komponens, $F/2$ deriváltja. A büntető tagból származó gradiens komponensnél feltételeztük, hogy $F(\|\mathbf{w}\|^2)$ valóban csak $\|\mathbf{w}\|^2$ függvénye, és így a gradiense skalár $\times \mathbf{w}$ formában írható. A módosító tag előjele maximumkeresésnél pozitív, minimumkeresésnél negatív.

Az eljárás konvergenciája a sztochasztikus approximáció alapján bizonyítható [Hyv97]. Bár az iteratív eljárás konvergenciája nem gyors, a pontonkénti tanítás biztosítja a hasonló egyéb pillanatnyi gradiens alapú tanuló eljárásoknál is meglévő tulajdonságot, a változó környezethez való folyamatos adaptálódás képességét. A konvergenciasebességet – ugyancsak hasonlóan az egyéb pillanatnyi gradiens felhasználásával dolgozó eljárásokhoz – $\mu(k)$ megválasztása jelentősen befolyásolja.

A konvergenciasebesség drasztikusan növelhető, ha fix-pont iterációt alkalmazunk. A fix-pont iterációk származtatásához abból indulunk ki, hogy egy gradiens eljárás stabil pontja csak olyan pont lehet, ahol a gradiens az aktuális súlyvektor irányába mutat, vagyis a gradiens a súlyvektor skalár konstansszorosával egyenlő. Csak ilyen esetben lehetséges, hogy egy gradiens irányú módosító vektort adunk a súlyvektorhoz és annak az iránya mégsem változik [Aga01], [Hea02], [Gra03].

6.2. FastICA: egy kurtózis alapú fix-pont ICA algoritmus

A (10.91) súlymódosító összefüggés fix pontjait megkapjuk, ha a súlyváltozás várható értékét vesszük és ezt nullává tesszük. Ha ugyanis az algoritmus konvergens, akkor kialakul egy egyensúlyi állapot, amikor a súlymódosítások várható értéke nulla. A diszkrét időindexet elhagyva a súlymódosítás várható értéke:

$$E\{\Delta(\mathbf{w})\} = E\left\{\mathbf{v}\left(\mathbf{w}^T \mathbf{v}\right)^3\right\} - 3\|\mathbf{w}\|^2 \mathbf{w} + f\left(\|\mathbf{w}\|^2\right) \mathbf{w} = \mathbf{0} \quad (10.92)$$

Az ilyen egyenletek megoldására számos determinisztikus iteratív eljárás létezik. Nagyon gyors iterációt kapunk, ha az (10.92) összefüggést a következő formában írjuk fel:

$$\mathbf{w} = \text{skalár} \times \left(E\left\{\mathbf{v}\left(\mathbf{w}^T \mathbf{v}\right)^3\right\} - 3\|\mathbf{w}\|^2 \mathbf{w} \right) \quad (10.93)$$

Most csak $\mathbf{w}$ iránya érdekes számunkra, a vektor normája lényegtelen. Ezért az összefüggésben szereplő skalár konstans hatása egy explicit normalizálással kiváltható.

Az alábbiakban az algoritmus fő lépéseit mutatjuk be, előbb egyetlen, majd több független komponens meghatározására.

Tételezzük fel, hogy a megfigyelhető mintapontok független komponensek lineáris kombinációiként jöttek létre és a mintapontokon a fehérítést elvégeztük. Ekkor egy független komponens az alábbi lépések alkalmazásával határozható meg:

1. Válasszunk véletlenszerűen egy $\mathbf{w}(0)$ kezdeti súlyvektort és legyen a diszkrét lépésindex $k=1$.

2. Legyen

$$\mathbf{w}(k) = E\left\{\mathbf{v}\left(\mathbf{w}(k-1)^T \mathbf{v}\right)^3\right\} - 3\mathbf{w}(k-1)$$

ahol a várhatóértéket kellően nagyszámú (pl. 1000) $\mathbf{v}$ mintapont alapján becsüljük.

3. Normalizáljuk $\mathbf{w}(k)$ -t úgy, hogy osszuk el $\mathbf{w}(k)$ -t a normájával.

4. Ha $\left|\mathbf{w}(k)^T \mathbf{w}(k-1)\right|$ közel 1, $\mathbf{w}(k)$ -t tekintsük eredménynek, egyébként legyen $k=k+1$ és menjünk vissza a 2. lépéshez.

A kapott $\mathbf{w}(k)$ vektor az ortogonális keverő mátrix egyik oszlopvektora lesz. Ez azt jelenti, hogy $\mathbf{w}(k)^T \mathbf{v}$ megadja az egyik független komponens becslését. Az eljárás igen kedvező tulajdonsága, hogy kevés (tipikusan 5-10) iterációs lépés elegendő, hogy a mintapontok által biztosított maximális pontosságú becslést kapjunk.

A fenti algoritmus egyetlen független komponens meghatározására alkalmas. A független komponens analízis feladatoknál azonban többször az összes vagy legalábbis több független komponens meghatározására van szükség. Több független komponens meghatározása az előbbi algoritmus segítségével úgy lehetséges, ha az algoritmust egymás után többször lefuttatjuk. A többszöri lefuttatásnál azonban biztosítani kell, hogy ne ugyanazt a független komponenset határozzuk meg többször. Ehhez egy ortogonalizáló vetítést kell beiktatnunk az algoritmus lépései közé. Itt emlékeztetni szeretnénk arra, hogy a fehérítés következtében a keverő mátrix ortogonális. Ezért a független komponenseket egyenként becsülhetjük, ha az aktuális $\mathbf{w}(k)$ megoldást egy olyan térre vetítjük, amely ortogonális a $\mathbf{B}$ keverőmátrix eddig megtalált oszlopvektoraira.

Legyen $\bar{\mathbf{B}}$ az eddig megtalált oszlopvektorokból felépített mátrix. A fenti algoritmusban a 3. lépést cseréljük ki a következőre:

3. Legyen $\mathbf{w}(k) = \mathbf{w}(k) - \bar{\mathbf{B}}\bar{\mathbf{B}}^T \mathbf{w}(k)$, majd normalizáljuk $\mathbf{w}(k)$ -t.

Ezen a módosításon kívül egy hasonló vetítésre szükség van a kezdeti súlyvektor felvételénél is.

Hogy elkerüljük az előző vektorok (tehát $\bar{\mathbf{B}}$) becslési hibájának hatását a későbbi vektorok becslésénél, a fenti vetítést csak az algoritmus első néhány iterációjában alkalmazzuk. Ha az algoritmus már közelítőleg megtalálta

a keverőmátrix egy oszlopát, akkor már egy fix pont vonzáskörzetében vagyunk, ahonnan a megfelelő pontba történő konvergencia már biztosított.

Az algoritmus bizonyítottan konvergens, sőt a konvergenciasebességre is lehet állítást megfogalmazni. A részletek [Hyv97]-ben és [Hyv01]-ben megtalálhatók.

Megjegyezzük, hogy a bemutatott gyors ICA eljárás csak egy a hasonló fix pont algoritmusok közül. Hasonló eredményeket a nemgauss-ság más megfogalmazása alapján is származtathatunk. Így pl. a negentrópia alapján is származtatható FastICA algoritmus, melynek részletes bemutatása [Hyv01]-ben található.

6.3. Független komponens analízis nemlineáris PCA alkalmazásával

Az eddigiek alapján látható, hogy a független komponensekre bontás a (fehérített) megfigyelések lineáris transzformációjával lehetséges, ahol a transzformáló mátrix meghatározása nemlineáris feldolgozó lépéseket is igényel. Ez veti fel azt a gondolatot, hogy érdemes a PCA algoritmus nemlineáris változatával próbálkoznunk.

A (10.88) összefüggés szerint egy olyan $\mathbf{B}$ mátrixot keresünk, amely a fehérített megfigyelésekből visszaállítja az eredeti független komponenseket. Konstruáljunk ezért egy olyan egyrétegű neurális hálót, melynek súlyvektorait a PCA hálózatoknál alkalmazott Oja altér szabály nemlineáris változatával tanítjuk. A súlymódosító összefüggésünk ennek megfelelően legyen a következő:

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \mu(k) [\mathbf{v}(k) - \mathbf{W}(k) \mathbf{g}(\mathbf{y}(k))] \mathbf{g}(\mathbf{y}^T(k)) \quad (10.94)$$

Itt $\mathbf{W}$ a többkimenetű háló összes súlyvektorából, mint sorvektorból felépített súlymátrix, $\mathbf{v}$ a fehérített megfigyelésvektor, $\mathbf{y}$ a háló kimeneti vektora, $\mathbf{g}(\mathbf{y})$ pedig egy olyan vektort jelöl, melynek i -edik komponense $g(y_i)$, ahol $g(\cdot)$ egy megfelelő nemlineáris függvény. Megmutatható, hogy a (10.94) összefüggés egy olyan

sztochasztikus gradiens algoritmus, amely a $\sum_{i=1}^M E\{f(y_i)\}$ kritérium szélső értékét keresi, azzal a feltétellel, hogy $\mathbf{W}$ ortonormált (M a kimenetek száma). Ha $f(t) = t^4/4$, a kritérium a negyedik momentumok összegét jelenti, és minthogy a sztochasztikus gradiens eljárás miatt $g(t)$ az $f(t)$ függvény deriváltja, $g(t) = t^3$. A nemlineáris függvény ilyen megválasztása mellett (10.94) valóban a negyedik momentumok összegének szélsőértékét biztosító, ortonormált $\mathbf{W}$ -t eredményezne, de az algoritmus nem stabil. Stabil algoritmust nyerünk, ha pl. a $g(t) = \tanh(t)$ függvényt választjuk, amelynek Taylor-soros közelítésében a t^3 tag domináns lehet.

Fehérített jelek és ortogonális szeparáló mátrix mellett igaz, hogy $E\{y_i^2\} = 1$, ezért, ha a negyedik momentumok összegét, $\sum_{i=1}^M E\{y_i^4\}$ -t minimalizáljuk a szub-Gauss jelekre, ill. maximalizáljuk a szuper-Gauss jelekre, akkor a

$$C(\mathbf{y}) = \sum_{i=1}^M \left| E\{y_i^4\} - 3E^2\{y_i^2\} \right| \quad (10.95)$$

kritériumfüggvény maximumát biztosító megoldást kapunk, vagyis most is a kurtózis alapján dolgozó algoritmust nyertünk.

A nemlineáris PCA háló tehát egy Oja altér háléhoz hasonló felépítésű azzal az eltéréssel, hogy nemlineáris neuronokból épül fel, ahol a nemlinearitást célszerűen a szokásos tangens hiperbolikus függvényre kell választani. A háló azonban csak a tanulási fázisban működik nemlineáris hálóként, hiszen a független komponenseket a fehérített adatok lineáris transzformációjával nyerhetjük. A visszaállító mátrix meghatározása azonban nemlineáris művelet, ezért a tanulási fázisban szükség van a nemlineáris neuronokra.

A nemlineáris PCA hálózatoknak is több változatát dolgozták ki. Ezek között említést érdemel az ún. bigradiens algoritmus, mely egyszerűsítést jelent az Oja nemlineáris PCA szabályhoz képest. A bigradiens algoritmus súlymódosító összefüggése a következő [Wan96b]:

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \mu(k) \mathbf{g}(\mathbf{W}(k) \mathbf{v}(k)) \mathbf{v}(k)^T + \alpha (\mathbf{I} - \mathbf{W}(k) \mathbf{W}(k)^T) \mathbf{W}(k) \quad (10.96)$$

ahol α egy konstans együttható, melynek értékét a $[0,5, 1]$ intervallumból kell választani, a $\mathbf{g}$ függvénykapcsolat pedig azt jelöli, hogy egy g függvényt a $\mathbf{W}\mathbf{v}$ vektor komponenseire külön-külön alkalmazzuk.

A bigradiens algoritmus előnye, hogy a (10.94) összefüggésben szereplő $\mathbf{W}(k) \mathbf{g}(\mathbf{y}(k))$ számítását nem igényli.

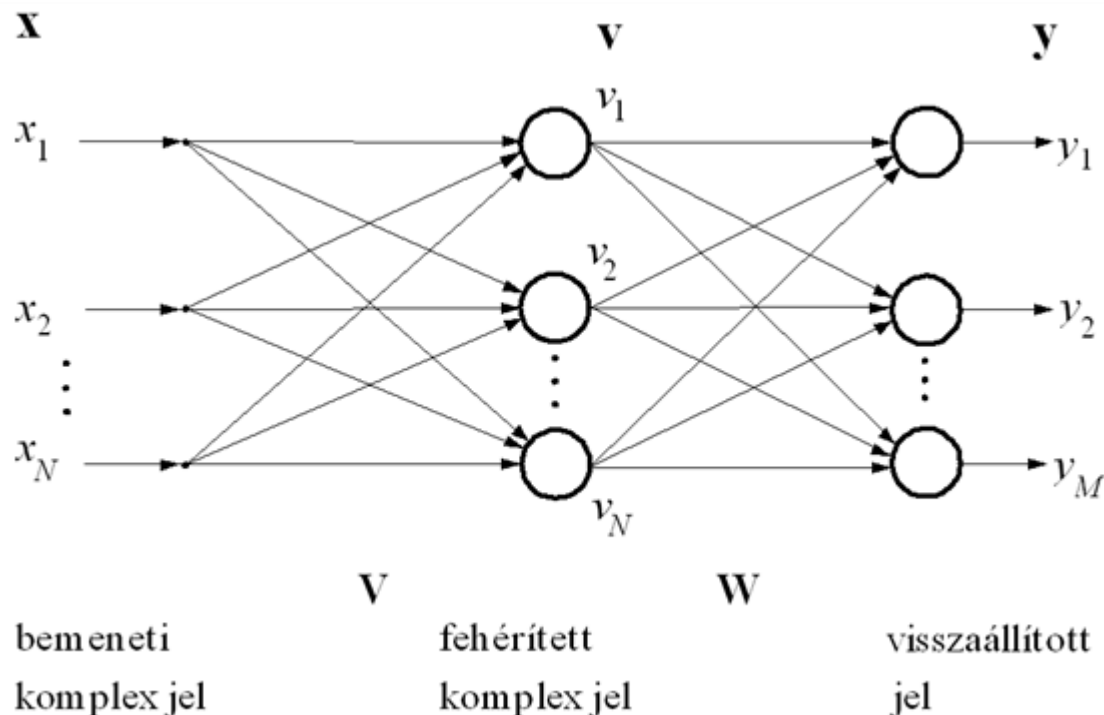
A nemlineáris PCA algoritmuson alapuló megoldás, mint kétlépes szeparáló algoritmus neurális architektúrával is megvalósítható. A háló két rétegből áll (10.13 ábra). Az első, lineáris réteg a fehéritést végzi, ennek a rétegnek a súlyait a

$$\mathbf{V}(k+1) = \mathbf{V}(k) - \mu(k) (\mathbf{V}(k) \mathbf{x}(k) \mathbf{x}(k)^T \mathbf{V}(k)^T - \mathbf{I}) \mathbf{V}(k) \quad (10.97)$$

összefüggés szerinti tanítással határozhatjuk meg [Plu93], ahol $\mathbf{x}$ a tényleges bemeneti adat. A második réteg, melynek súlyait a (10.94) vagy a (10.97) összefüggés szerinti nemlineáris Hebb szabállyal tanítjuk, a fehéritett $\mathbf{v}$ vektorokon dolgozik.

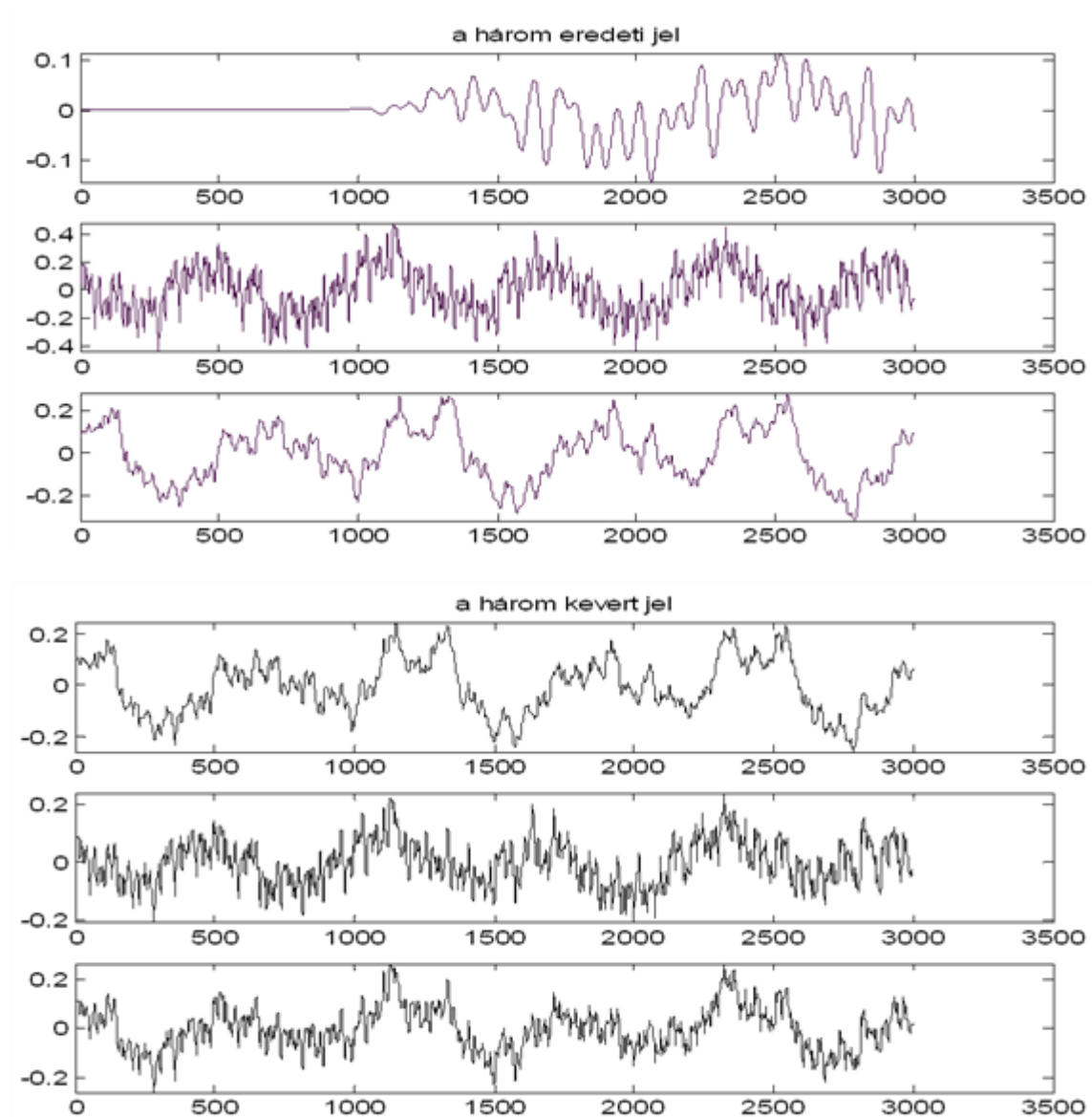
A független komponens analízis működését két példán illusztráljuk. Az első példa (10.14 ábra) három időfüggvény lineárisan súlyozott keverékeiből indul ki. Az ábrán a három kiinduló időfüggvényt, ezen időfüggvények keverékeit, valamint a három visszaállított jelet láthatjuk. A 10.15 ábra kétdimenziós jelek (képek) független komponenseinek meghatározására mutat példát.

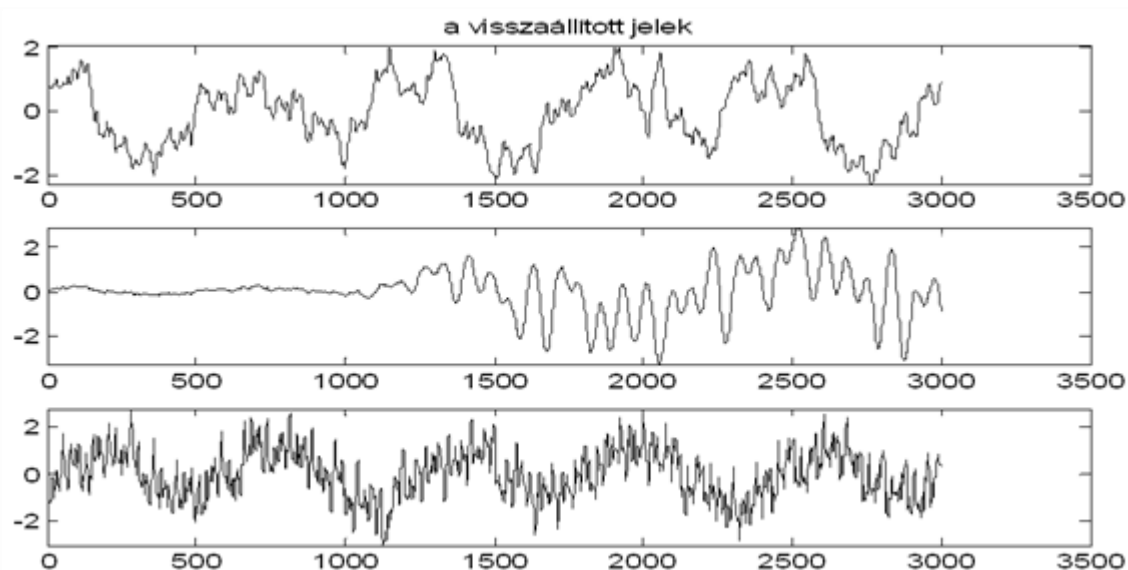
10.13. ábra - Független komponens analízis hálózat



Mindkét példa jól illusztrálja a független komponens analízis eljárások két „hiányosságát”. Egyrészt nem tudjuk megmondani, hogy az eljárások milyen sorrendben állítják elő a független komponenseket. Másrészt az eredmények csak egy skalár szorzótényezőtől eltekintve lesznek egyértelműek. A független komponensként egy eredeti jel valamilyen skalárszorását kaphatjuk. Ha rögzítjük a jelek varianciáját például egységre, még akkor is visszakaphatjuk az egyes eredeti komponensek (-1) -szeresét. Ezt figyelhetjük meg a 10.14 ábrán a második időfüggvényénél, valamint a képvisszaállításnál a fatörzs esetében, ahol az eredeti kép helyett annak negatívját kaptuk eredményül.

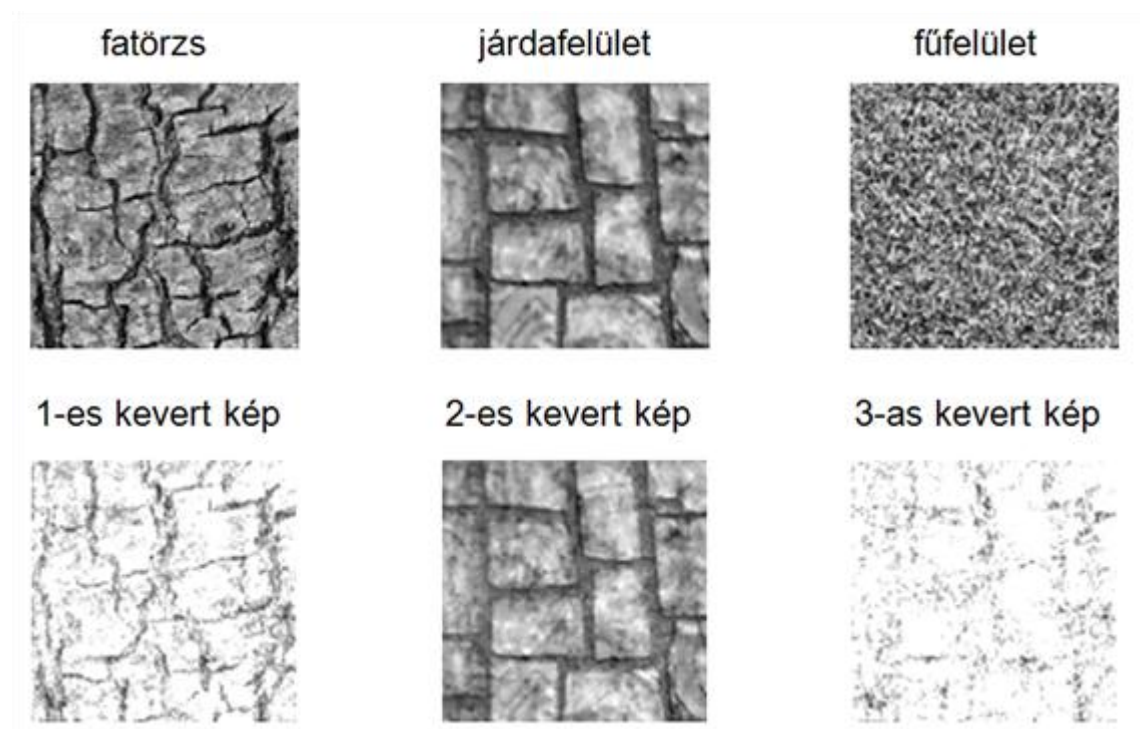
10.14. ábra - A független komponens analízis alkalmazása időfüggvényekre (beszédjelekre)





A független komponens analízis feladat megoldására a fentiekben röviden bemutatott eljárásokon túl számos egyéb javaslat született. Ezek az eljárások általában a feladat más megfogalmazásából származtathatók. További olyan változatokat is kidolgoztak, melyek az itt bemutatott alapeladaton túl általánosabb eseteket próbálnak megoldani. Ilyen eset lehet, ha a megfigyeléseinket zaj terheli, továbbá, ha a keverőmátrix nem konstans. A független komponens analízis eljárásoknak kidolgozták olyan változatait is, melyek valamilyen mértékben kezelni tudják azon eseteket is, amikor kevesebb megfigyelésünk van, mint ahány független komponens szeretnénk meghatározni.

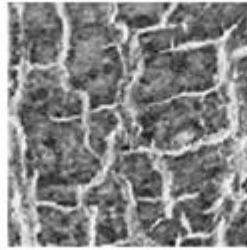
10.15. ábra - Független komponens analízis alkalmazása képekre



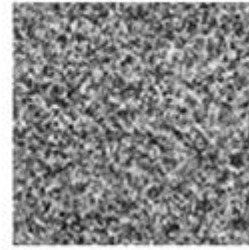
1-es visszaállított kép



2-es visszaállított kép



3-as visszaállított kép



Feladatok

10.1 Mutassa meg, hogy a Kohonen hálónál alkalmazott tanítási összefüggés (10.5 egyenlet) a normalizált Hebb szabály.

10.2 A $[0,5 \ 3,5] \times [0,5 \ 3,5]$ tartományon egyenletes eloszlású, kétdimenziós adathalmaz feldolgozására topologikus leképezést biztosító Kohonen hálót alkalmazunk. A háló – kétdimenziós rácsban elhelyezve – 9 neuront tartalmaz az $[i, j]$ pozíciókban, ahol $i, j=0,1,2$. Milyen súlyvektorok kialakulása várható a háló tanítása után?

10.3 Milyen súlyvektorokat kapunk a tanítás végén, ha az előző feladat adathalmazát egy 9 neuront lineáris elrendezésben tartalmazó Kohonen hálónak tanítjuk. A neuronok pozíciói: 1, 2, 3, . . . , 9.

10.4 Mutassa meg, hogy az Oja szabállyal tanított háló súlyvektora egyensúlyi helyzetben a bemenet autokovariancia mátrixának sajátvektora.

10.5 A GHA (Sanger) háló csak előrecsatoló összeköttetéseket használ. Ezzel szemben az APEX algoritmus mind előrecsatoló, mind oldalirányú kapcsolatokkal rendelkezik. Mutassa meg, hogy a két eltérő hálózat-architektúra a megfelelő tanulási algoritmussal azonos súlyvektorokhoz ($\mathbf{w}_i$) konvergál.

10.6 Miért van szükség minimum 5 rétegű hálózatra, ha nemlineáris adattömörítést szeretnénk végezni?

10.7 Kernel PCA esetére, Gauss kernel mellett határozza meg a jellemzőtérbeli tömörített reprezentáció bemeneti térbeli közelítő megfelelőjét. Adja meg a gradiens eljárással dolgozó jelvisszaállítás algoritmusát.

10.8 Mutassa meg, hogy két független komponens súlyozott összegeiből kapott kevert adatokból kiindulva a kurtózis maximalizálása útján valóban megkapjuk a független komponenseket, ha szuper-Gauss komponensekből indulunk ki és kikötjük, hogy az eljárás eredményeképpen kapott adatok varianciája legyen egységsi. A feladatot, mint feltételes szélsőérték-kereső problémát oldja meg.

10.9 Adja meg, hogy milyen feltételek mellett tekinthető a nemgauss-ság és a függetlenség ekvivalens fogalmaknak.

11. fejezet - Analitikus tanítású hálózatok

1. A Hopfield hálózat

A Hopfield hálózat a legegyszerűbb neurális hálózat, amely asszociatív memóriát valósít meg. Az egyrétegű visszacsatolt struktúrát John Hopfield amerikai biológus publikálta először [Hop82]. A hálózatot leggyakrabban autoasszociatív feladatok megoldására használjuk. Ilyenkor a memóriában mintákat, gyakran digitalizált képeket tárolunk [Egm02]. A neurális hálózattól azt várjuk, hogy a tárolt információ zajos, torzított, esetleg hiányos változatának megmutatásakor az eredeti mintára asszociáljon.

A Hopfield hálózat legegyszerűbb megvalósítását mutatja a 11.1 ábra. Az egyrétegű modellben a neuronok kimenetei minden neuron bemenetére súlyozott összeköttetésekön keresztül vannak visszacsatolva. A hálózat bemenetének a neuronok kezdeti állapotai tekinthetők, míg a kimenetet ugyanezen neuronok állapotai jelentik a működés során.

Az alapmodell esetében a Hopfield hálózat neuronjai kétértékűek. Az egyszerűbb matematikai leírás kedvéért a továbbiakban az aktív egységek értéke $x_i=+1$, míg az inaktívaké $x_i=-1$ legyen. (A bemutatásra kerülő igazolásokban az $x_i=2u_i-1$ konverziót használva vissza lehet kapni a jobban megszokott, az itt használttal teljesen egyenértékű 0–1 bináris kódolást.) Ezek alapján a hálózat dinamikáját a következőképpen írhatjuk fel:

$$x_i := \operatorname{sgn} \left(\sum_j w_{ij} x_j - \Theta_i \right), \quad (11.1)$$

ahol

$$\operatorname{sgn}(x) = \begin{cases} +1 & \text{ha } x \geq 0 \\ -1 & \text{ha } x < 0 \end{cases} \quad (11.2)$$

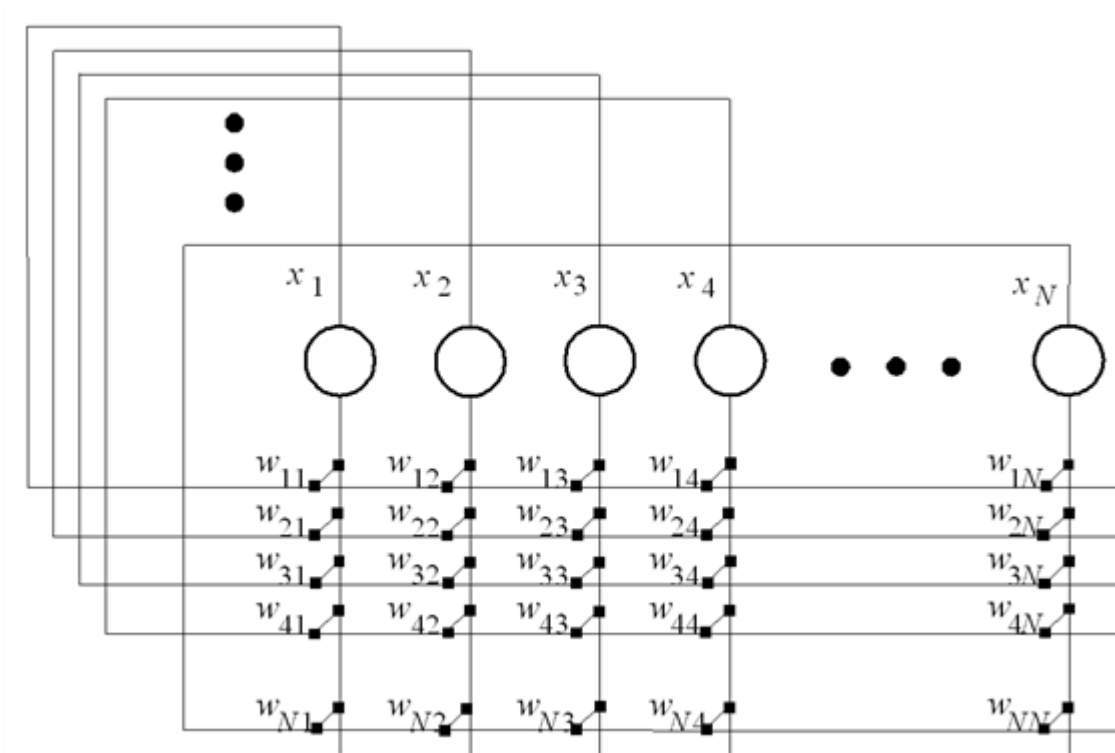
(A következőkben bemutatásra kerülő, véletlen mintákat feltételező vizsgálatoknál a Θ küszöbérték használata nem szükséges, így $\Theta=0$ -t feltételezve elhagyjuk.)

A hálózat neuronjainak egy adott állapotát egy konfigurációnak nevezzük, az összes lehetséges állapotot, pedig konfigurációs térnek. Ebben a térben a stabil állapotokhoz tartozó pontokat vonzási pontoknak nevezzük. A megfelelően beállított súlyokkal rendelkező hálózattól azt várjuk, hogy az adott kezdeti konfigurációból kiindulva az ehhez legközelebbi vonzási pontba stabilizálódik. Az egy-egy vonzási ponthoz tartozó vonzási körzet sematikus rajzát mutatja a 11.2 ábra.

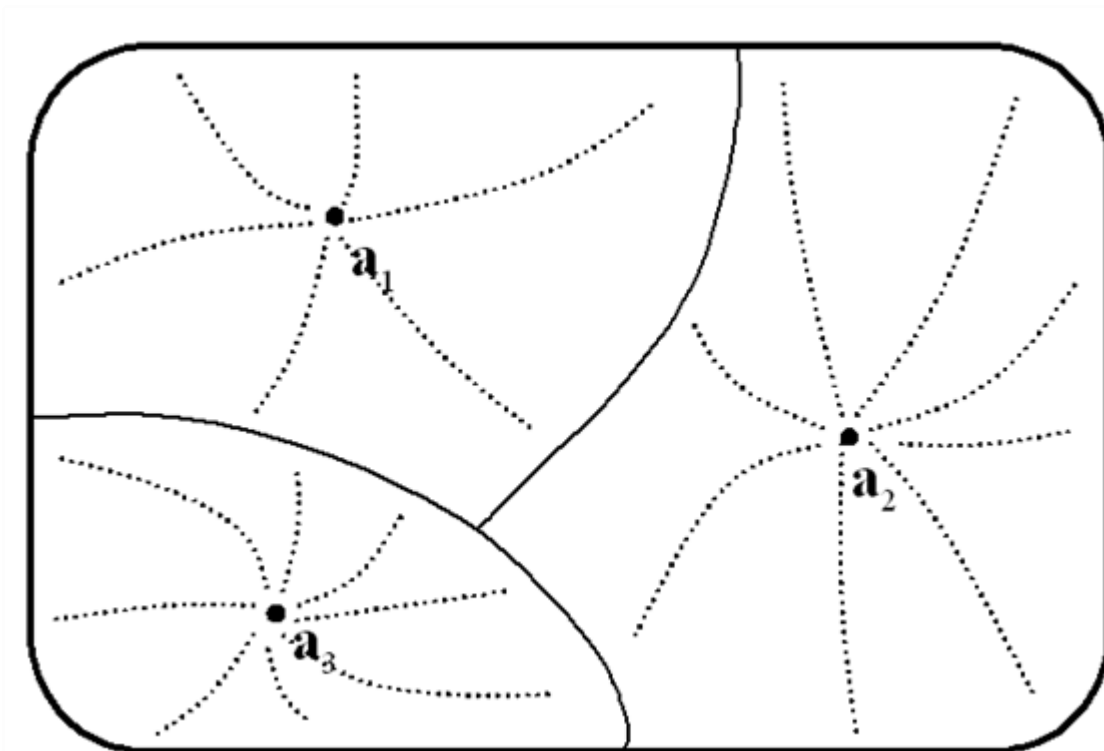
A hálózat működtetésének (akár hardver megvalósításról, akár szoftver szimulációról van szó) alapvetően kétféle módja van: szinkron vagy aszinkron ütemezés.

Szinkron rendszer esetén az összes neuront minden időlépésnél egyszerre módosítjuk. Aszinkron működésnél viszont egy pillanatban csak egy egység válthat értéket. Ide szokták sorolni azt az esetet is, amikor az egységek egymástól függetlenül, véletlenszerűen választják meg állapotuk módosításának időpontját, mivel ilyenkor elhanyagolhatóan kicsi annak a valószínűsége, hogy egyszerre több neuron is értéket váltson. Mind a biológiai, mind a mesterséges neurális hálózatok esetében természetesebb az aszinkron működés feltételezése, így a továbbiakban mi is ezt vizsgáljuk. (Meg kell jegyezni, hogy az aszinkron és szinkron rendszerek dinamikus tulajdonságai különböznek. Az első esetben a hálózat szomszédos konfigurációkon keresztül jut el a vonzási pontba, míg az utóbbi módszert használva a rendszer bármely állapotváltozás során a konfigurációs tér bármely pontjába kerülhet.)

11.1. ábra - A Hopfield háló



11.2. ábra - A konfigurációs tér



A hálózatot egy adott kiindulási konfigurációból addig működtetjük, amíg stabil állapotba nem kerül. A rendszer válaszában a stabil állapothoz tartozó konfigurációt tekintjük.

1.1. Minták tárolása

A hálózat tanítása, a megjegyzendő minták tárolása, itt is a súlyok megfelelő beállításával történik. Az elsőként bemutatásra kerülő módszernél, a tanítás a módosított Hebb szabállyal történik, amit itt felügyelt tanítási eljárásaként alkalmazunk, mivel a bemeneti minták egyben a kívánt kimenetek is. A tanítás során minden egyes tárolandó mintát egyszer adunk a hálózat bemenetére és ennek alapján határozzuk meg a súlyokat. Az eljárás során, az eddig bemutatott iteratív módszerektől eltérően, minden mintát csak egyszer használunk fel a súlyok meghatározásakor. A Hebb-szabály alapgondolatát a 11. fejezet már ismertette. Itt most azt fogjuk bemutatni, hogy hogyan valósítható meg a tanulás a Hopfield hálózatokon ezzel az algoritmussal.

1.1.1. Egy minta tárolása

Az egyszerűség kedvéért nézzük meg először, hogy hogyan képes a hálózat egyetlen mintát megjegyezni. A tanulás eredményeképpen a megtanulandó mintákhoz tartozó konfigurációnak stabilnak kell lennie és minél nagyobb vonzási körzetnek kell hozzá tartoznia. Egy minta stabil, ha $\forall i$ -re (legyen $\mathbf{a}$ vektor a tanítandó minta):

$$\text{sgn}\left(\sum_j w_{ij} a_j\right) = a_i, \quad (11.3)$$

mivel ekkor a dinamikus szabály nem okozza a neuron értékének váltását. Válasszuk a súlyokat a következőképpen:

$$w_{ij} \equiv a_i a_j \quad (11.4)$$

Könnyen belátható, hogy a tanított minta így stabil, hiszen $a_i^2=1$. Később bemutatásra kerülő szempontok miatt az arányossági tényezőt válasszuk $1/N$ -re, ahol N a neuronok száma:

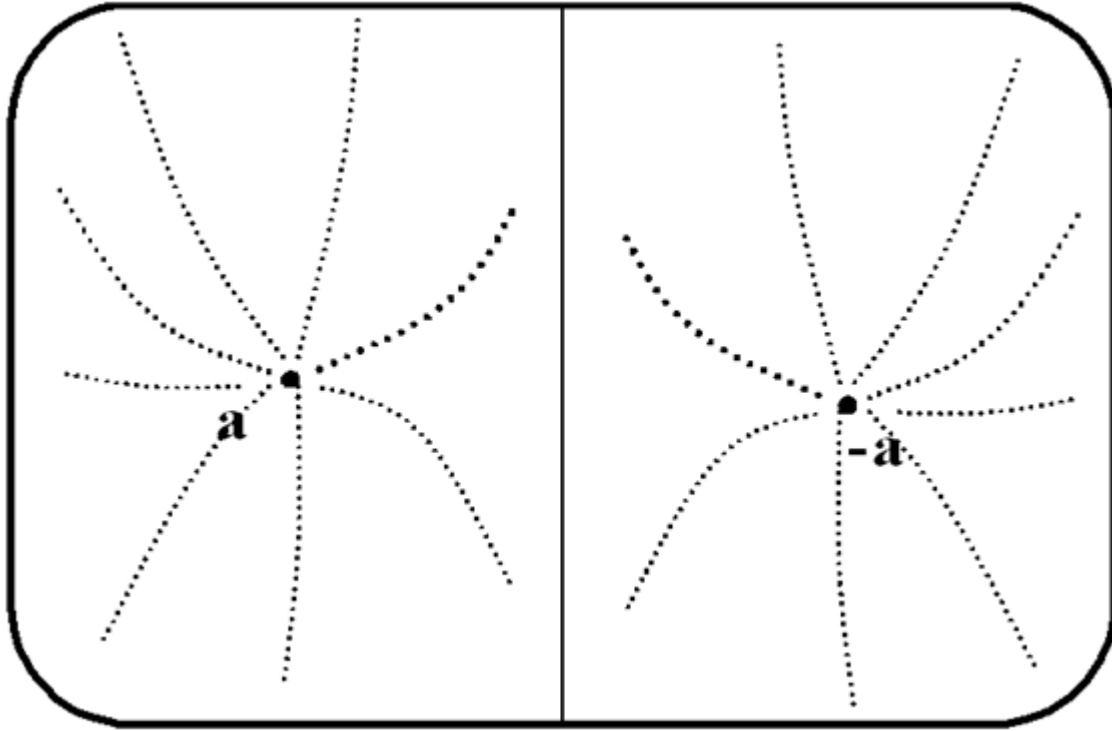
$$w_{ij} = \frac{1}{N} a_i a_j \quad (11.5)$$

Az így meghatározott súlyokkal a hálózatban a minta nemcsak stabil, de jelentős vonzási körzete van, pontosan a konfigurációs tér fele. Egy neuron akkor vált a kívánt értékre, ha a bemenetén keletkező összeg s_i előjele megegyezik a minta megfelelő bitjének előjelével:

$$s_i = \sum_j w_{ij} x_j = \sum_j a_i a_j x_j = a_i \sum_j a_j x_j \quad (11.6)$$

Tehát az előbb említett feltétel akkor teljesül, ha a felismerendő minta bitjeinek több mint a fele helyes, azaz a tanítottéval megegyező. Az is látható, hogy ennek a hálózatnak két stabil állapota van: a tárolt minta és annak inverze (11.3 ábra).

11.3. ábra - A konfigurációs tér egy minta tárolása után



1.1.2. Több minta tárolása

Természetesen a hálózat gyakorlati alkalmazás szempontjából csak akkor lehet érdekes, ha számos minta "megjegyzésére" képes. Az előző szakaszban használt ötletet próbáljuk meg itt is alkalmazni, és a súlyok kiszámításakor vegyük az egy minta tárolására használt tagok szuperpozícióját:

$$w_{ij} = \frac{1}{N} \sum_{\alpha=1}^l a_i^{(\alpha)} a_j^{(\alpha)}, \quad (11.7)$$

ahol l a tanítandó minták száma (az eddig használt nagy P betűt a valószínűség jelölésére alkalmazzuk ebben a fejezetben), és α ezek indexe.

Vizsgáljuk meg, hogy az $\mathbf{a}^{(\alpha)}$ minták stabil állapotai lesznek-e a rendszernek. A stabilitási feltétel most a β indexű mintára:

$$\text{sgn}(s_i^{(\beta)}) = a_i^{(\beta)}, \quad (11.8)$$

ahol az $s_i^{(\beta)}$ összeg:

$$s_i^{(\beta)} = \sum_j w_{ij} a_j^{(\beta)} = \frac{1}{N} \sum_{j=1}^N \sum_{\alpha} a_i^{(\alpha)} a_j^{(\alpha)} a_j^{(\beta)} \quad (11.9)$$

Az összegből az $\alpha=\beta$ -hoz tartozó tagokat:

$$s_i^{(\beta)} = a_i^{(\beta)} + \frac{1}{N} \sum_j \sum_{\alpha \neq \beta} a_i^{(\alpha)} a_j^{(\alpha)} a_j^{(\beta)} \quad (11.10)$$

Látható, hogyha a második, keresztkapcsolat tag $a_i^{(\beta)}$ -vel megegyező előjelű, vagy ellentétes előjelű és 1-nél kisebb abszolút értékű, akkor a tárolt mintának ez a bite stabil. Mint majd látni fogjuk ez a feltétel megfelelően kis számú minta tanításakor nagy valószínűséggel igaz. Ilyenkor a tárolt mintákhoz stabil állapotok tartoznak. A tárolt mintáktól valamilyen kis távolságban lévő konfigurációkból indítva a hálózatot a tárolt mintához tartozó

állapotban stabilizálódik a rendszer, azaz képes hibás, zajos, hiányos minták helyreállítására, felismerésére. A vonzási körzet mérete természetesen függ a tanított minták számától, egymáshoz viszonyított elhelyezkedésüktől.

Most nézzük meg hány minta tárolható egy adott méretű hálózatban. Mivel ez függ a konkrét tanítandó példaktól, így azért, hogy legalább becslést tudjunk tenni, véletlenszerűen választott mintákat feltételezünk. Először vizsgáljuk meg a stabilitást egy véletlenszerű minta egy bitjére. A keresztkapcsolat tag hatásának egyszerűbb elemezhetősége céljából képezzük a tagot $-a_i^{(\beta)}$ -vel szorozva a következő kifejezést:

$$\begin{aligned} C_i^{(\beta)} &= -a_i^{(\beta)} \frac{1}{N} \sum_j \sum_{\alpha \neq \beta} a_j^{(\alpha)} a_j^{(\alpha)} a_i^{(\beta)} = \\ &= -\frac{1}{N} \sum_j \sum_{\alpha \neq \beta} a_j^{(\alpha)} a_j^{(\alpha)} a_i^{(\beta)} \end{aligned} \quad (11.11)$$

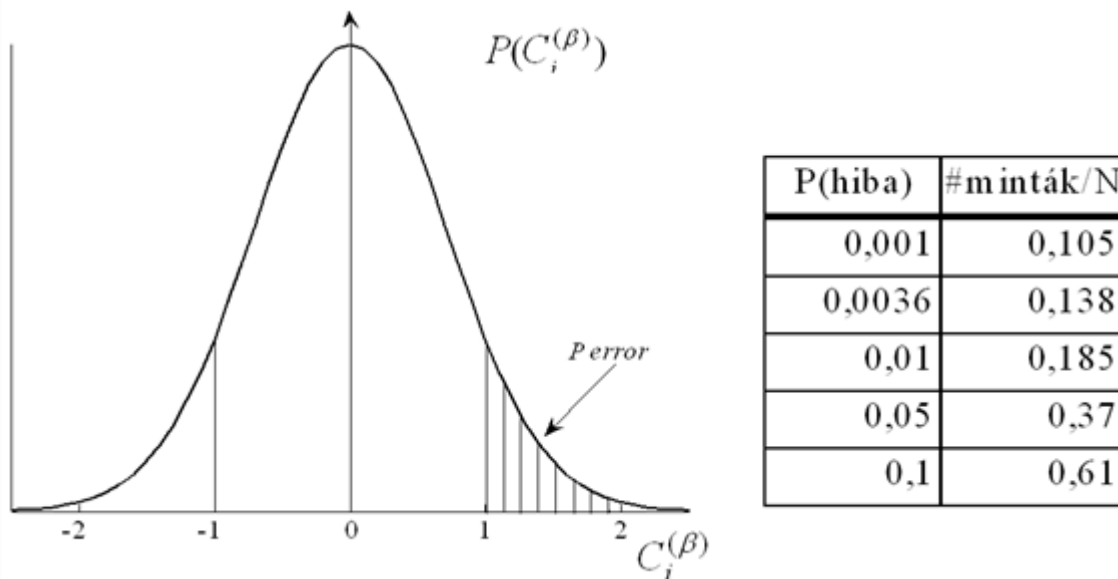
A fent elmondottak szerint, ha $C_i^{(\beta)}$ kisebb, mint +1, akkor a β -edik minta i -edik bitje stabil. Látható, hogy $C_i^{(\beta)}$ kizárólag a tárolt mintáktól függ. Ezek alapján az instabilitás valószínűségét becsüljük: $P_{\text{error}} = P(C_i^{(\beta)} > 1)$.

A $C_i^{(\beta)}$ összeg $N(l-1)$ véletlenszerű előjelű, $1/N$ abszolút értékű tagból áll, és nagy l és N értékeket feltételezve, $C_i^{(\beta)}$ közelíthető egy 0 várható értékű és $\sigma = \sqrt{l/N}$ szórású Gauss eloszlással:

$$P_{\text{error}} = \frac{1}{\sqrt{2\pi}\sigma} \int_1^{\infty} e^{-x^2/2\sigma^2} dx \quad (11.12)$$

Az eloszlást és néhány hibaértékhez tartozó l_{max}/N arányt mutat be a 11.4 ábra. Ez például azt mutatja, hogyha azt kívánjuk, hogy egy minta egy bitje 0.999 valószínűséggel legyen stabil, akkor egy 1000 neuront tartalmazó hálózatban 105 mintát tárolhatunk. Hasonló módon lehet az összes minta összes bitjére valószínűségi állítást megfogalmazni. Szintén véletlenszerű példákat tekintve erre az esetre jó közelítésnek vehető a $l_{\text{max}} = N/\log N$ arány. Mivel ez az eredmény a leírt feltételek esetén igaz, így célszerű az alkalmazásoknál ennél kisebb l_{max}/N arányt választani.

11.4. ábra - Véletlenszerűen választott minták stabilitása



A Hopfield hálózatok az eddig bemutatott autoasszociatív jellegű problémákon kívül használhatók heteroasszociatív feladatok megoldására is, ahol a tanítás során a bemenetekhez önmaguktól eltérő kimeneteket rendelünk (pl. osztályozási feladatok). Ilyenkor a neuronok egy része a minták bemeneti elemeinek felelnek

meg, míg a másik, kisebb részét (tipikusan 10-20%-át) kimenetnek tekintjük. Így a tanítás során a tárolandó minták a bemeneti vektorokból és a hozzájuk tartozó kívánt kimenetekből állnak.

Teszteléskor és a már betanított hálózat alkalmazásakor a bemenetekhez tartozó neuronok kezdeti állapotát a bemeneti minta szerint állítjuk, míg a kimenetet reprezentáló neuronokat tetszőleges értékűekre inicializálhatjuk. Ha a hálózat súlyait korábban sikerült megfelelően beállítani, akkor az autonóm működés során a hálózat egy olyan állapotban stabilizálódik, amelyben a bemeneti neuronok kezdeti értéküket veszik fel, míg a kimeneti neuronok arra az értékre állnak be, amelyet a tanítás során az adott bemenethez kívánt kimenetként megadtunk.

1.2. Az energiafüggvény

A Hopfield modell egyik előnyös tulajdonsága, amit később majd alkalmazásoknál ki is használunk, hogy egyszerűen rendelhető hozzá energiafüggvény. A rendszer minden konfigurációjához egy "energia" értéket lehet meghatározni, és ezzel meghatározunk egy a konfigurációs tér fölött értelmezett energia felületet.

Az energiafüggvény (amely Ljapunov függvényként használható) alapvető tulajdonsága, hogy az autonóm rendszer működése során értéke mindig csökken vagy változatlan marad. Tehát egy tetszőleges konfigurációból elindított hálózat az energiafüggvény valamely lokális minimumpontjában fog stabilizálódni, azaz a vonzási pontok az energiafelület lokális minimumai [Bru90]. Ha a súlyokat sikerül úgy megválasztani, hogy a tárolandó minták a minimumpontokba essenek, akkor a rendszer vizsgálható az energiafüggvény alapján.

A hálózatra felírható energiafüggvény a következő:

$$E = -\frac{1}{2} \sum_i \sum_j w_{ij} x_i x_j - \sum_i \Theta_i x_i \quad (11.13)$$

Ha a felírt képlet energiafüggvénye a hálózatnak, akkor ennek definíciójából következő tulajdonsága, hogy autonóm működés során értéke csökken vagy nem változik. Az energiafüggvény megfelelőségét igazolandó vizsgáljuk meg a rendszer energiaváltozását egy állapotváltozás során:

$$\begin{aligned} \Delta_k E &= E(k+1) - E(k) \\ &= -\frac{1}{2} \sum_i \sum_j w_{ij} x_i(k+1) x_j(k+1) - \sum_i \Theta_i x_i(k+1) \\ &\quad + \frac{1}{2} \sum_i \sum_j w_{ij} x_i(k) x_j(k) + \sum_i \Theta_i x_i(k) \\ &= -\sum_i (x_i(k+1) - x_i(k)) \left(\sum_j w_{ij} x_j(k) + \Theta_i \right) \\ &\quad - \frac{1}{2} \sum_i (x_i(k+1) - x_i(k)) \left(\sum_j w_{ij} (x_j(k+1) - x_j(k)) \right) \\ &= -\mathbf{s}^T(k+1) \Delta \mathbf{x}(k+1) - \frac{1}{2} \Delta \mathbf{x}^T(k+1) \mathbf{W} \Delta \mathbf{x}(k+1) \end{aligned} \quad (11.14)$$

ahol

$$\mathbf{s}(k+1) = \mathbf{W} \mathbf{x}(k) + \Theta \quad (11.15)$$

Aszinkron működésű hálózat minden állapotváltozásakor pontosan egy neuron vált értéket. Ha a súlyok mátrixa szimmetrikus ($w_{ij} = w_{ji}$), akkor a r -edik ($1 \leq r \leq N$, ahol N a neuronok száma) neuron értékvtását vizsgálva a következőt kapjuk:

$$\Delta_k E = -s_r(k+1) \Delta x_r(k+1) - \frac{1}{2} w_{rr} \quad (11.16)$$

Mivel a (11.1) képlet szerint s_r határozza meg a neuron értékvtását, így s_r és Δx_r előjele megegyezik, azaz az első tag értéke nem lehet pozitív. Így, ha a w_{ii} visszacsatoló súlyok nem negatívak, akkor az energia változás

értéke nem lehet pozitív, azaz a (11.13) összefüggéssel definiált függvény valóban Ljapunov függvénye a rendszernek.

Néhány megjegyzést érdemes tenni a fenti eredménnyel kapcsolatban. Bár a súlymátrix szimmetrikusságának megkötelese erős feltétel, az alkalmazások jelentős részénél ez nem okoz problémát. Ilyen súlymátrixot eredményez az említett Hebb szabály alkalmazása és a gráfelméleti alapokra visszavezethető optimalizációs problémák jelentős részénél is természetesen adódik a szimmetrikus mátrix megválasztása. Az önvisszacsatoló kapcsolatok súlyára az energiafüggvény hozzárendelés szempontjából csak a nem negatív érték a feltétel, de megmutatható, hogy a hálózat dinamikáját előnytelenül befolyásolja, ha értékeit 0-nál nagyobbra választjuk. A pozitív önvisszacsatoló súlyok hatásának lényege, hogy nem kívánt stabil állapotokat hozhatnak létre a rendszerben, sőt egy a többi paramétertől függő érték felett akár egy neuronnak mindkét állapota is stabil lehet a többi neuron értékétől függetlenül.

Az önvisszacsatoló súlyok 0-ra választásának van egy további előnye is. Ilyenkor a (11.16) egyenlet értelmében nemcsak az igaz, hogy az autonóm hálózat állapotváltása egyértelműen meghatározza az energiafüggvény értékváltozásának előjelét, de a kapcsolat fordított irányban is közvetlen. Tehát ebben az esetben az aszinkron hálózat akkor és csakis akkor vált értéket, ha az energiafüggvény értéke csökken. Ez a gyakorlat számára azt jelenti, hogy a neurális hálózat szimulációja és az energiafüggvény szomszédos állapotokon keresztül történő minimalizálása ekvivalens eljárások, így választhatjuk közvetlenül ez utóbbit is egy probléma szimulációval történő megoldására.

A párhuzamos, szinkron ütemezésű Hopfield modell dinamikus tulajdonságai lényegesen eltérnek az aszinkron modellétől. A stabilitás vizsgálatát itt is a (11.13) szerint definiált energiafüggvény változásával végezhetjük el:

$$\begin{aligned}\Delta_k E &= E(k+1) - E(k) \\ &= -\mathbf{s}^T(k+1)\Delta\mathbf{x}(k+1) - \frac{1}{2}\Delta\mathbf{x}^T(k+1)\mathbf{W}\Delta\mathbf{x}(k+1)\end{aligned}\quad (11.17)$$

Az első tag értéke az aszinkron modellnél leírt indoklás miatt itt sem pozitív. A második tagra ugyanez a feltétel – a benne szereplő szorzat miatt – akkor igaz, ha a $\mathbf{W}$ súlymátrix pozitív szemidefinit. A szinkron modell esetében tehát elégséges feltételként a súlymátrix pozitív szemidefinit tulajdonsága szerepel, ami azt jelenti, hogy a főátlóban lévő értékeket általában nem választhatjuk 0-ra, és az sem igaz, hogy a rendszer mindig állapotot vált, ha ezzel az energiafüggvény csökkenne. Ha a súlymátrixot az aszinkron modellnél alkalmazotthoz hasonlóan, itt is szimmetrikusra választjuk 0 értékű főátlóbeli elemekkel, akkor a szinkron modell vagy egy stabil állapotba vagy egy 2 hosszúságú határciklusba konvergál. Energiafüggvény létezésére a fentieknél általánosabb feltételek is ismertek, amely szerint ún. kvázi szimmetrikus súlymátrix is stabil rendszert eredményez, de ezek az eredmények közvetlenül nem használhatóak fel a hálózatok alkalmazásakor.

1.3. Az energiafüggvény felhasználása

Először azt mutatjuk be, hogy az előző fejezetben bevezetett energiafüggvény és a korábban ismertetett Hebb-szabály alkalmazása azonos súlyok meghatározásához vezet. Induljunk ki most az energiafüggvényből. Próbáljuk ezt úgy megválasztani, hogy minimumpontjai a tárolandó minták legyenek. Egyetlen tanítandó minta esetében ez egyszerűen biztosítható. Az energiafüggvény minimuma legyen ott, ahol a minta és a konfiguráció átfedése (meggegyező értékű bitek száma) maximális:

$$E = -\frac{1}{N} \left(\sum_i x_i a_i \right)^2, \quad (11.18)$$

ahol az $1/N$ -es szorzó csak a korábbi képletekkel való hasonlóság kedvéért szerepel. Több minta esetében is a tárolt minták az energiafüggvény minimumai kell, hogy legyenek. Az itt alkalmazott módszer szerint határozzuk meg az energiafüggvényt úgy, hogy az egyes mintákhoz választható optimális függvények szuperpozícióját vesszük:

$$E = -\frac{1}{N} \sum_{\alpha=1}^l \left(\sum_i x_i a_i^{(\alpha)} \right)^2 \quad (11.19)$$

A tagokat kibontva és átrendezve a következőket kapjuk:

$$E = -\frac{1}{N} \sum_{\alpha=1}^l \left(\sum_i x_i a_i^{(\alpha)} \right) \left(\sum_j x_j a_j^{(\alpha)} \right) = -\sum_i \sum_j \left(\frac{1}{N} \sum_{\alpha=1}^l a_i^{(\alpha)} a_j^{(\alpha)} \right) x_i x_j \quad (11.20)$$

Látható, hogy a Hopfield hálózathoz definiált eredeti energiafüggvényt kaptuk vissza, feltételezve, hogy a súlyokat a Hebb-szabály szerint választjuk. Ez az eredmény a gyakorlatban is jól alkalmazható. Keressünk egy olyan függvényt, amely energiafüggvénye a rendszernek és minimumpontjai az adott feladat megoldásaihoz, a tanított mintákhoz tartozó konfigurációknak felelnek meg. Ha sikerül ilyet találni, akkor a tagok szétválasztásával a hálózat súlyai meghatározhatók, azaz létre tudunk hozni egy, a problémát megoldani képes hálózatot.

Tisztázni kell még, hogy milyen alakú tagokat tartalmazhat a választandó energiafüggvény. Bebizonyítottuk, hogy a $w_{ij}x_i x_j$ alakú tagok alkalmazhatók. Nyilván nem jelent gondot, ha a függvény konstans is tartalmaz. Ezenkívül megfelelnek a feltételeknek a $w_i x_i$ alakú tagok is, mivel ezek felfoghatók egy konstans 1 értékű x_0 neuron és az x_i közötti kapcsolathoz tartozó tagnak. Azonban a $w_{ijk}x_i x_j x_k$ alakú, illetve magasabb-rendű kifejezések már nem értelmezhetők a bemutatott struktúrájú Hopfield hálózat energiafüggvényében.

Meg kell még jegyezni, hogy szoftver megvalósítás esetében nem feltétlenül kell a hálózat működését szimulálni. Ha a neuronok önvisszacsatoló súlyait 0-ra választjuk ($w_{ii}=0$), akkor (11.16) szerint nem csak az lesz igaz, hogy a hálózat autonóm működése során az energiafüggvény csökken, de az is, hogyha az energiafüggvény csökkenne egy neuron értékváltásakor, akkor a neuron értéket is vált. Ilyenkor a hálózat szimulációja és az energiafüggvény minimalizálása ekvivalens eljárások, így választhatjuk közvetlenül ez utóbbit a probléma megoldására. Ezt a módszert néhány alkalmazási példával illusztráljuk a 11.2. fejezetben.

Szinkron működésű hálózat esetén is igaz, hogy (11.13) energiafüggvénye a rendszernek, ha a súlyokat a Hebb-szabály (11.7) alapján határozzuk meg. Ennél általánosabb elégséges feltétel a hálózat stabilitására a súlymátrix pozitív szemidefinitre választása. Hangsúlyozni kell, hogy ilyenkor általában az önvisszacsatoló súlyok nem választhatók önkényesen 0-ra (nem feltétlenül maradna a súlymátrix pozitív szemidefinit), és így az sem igaz, hogy a rendszer mindig állapotot vált, ha ezzel az energiafüggvény csökkenne ismétlés. Bár az energiafüggvény lokális/globális minimumai itt is stabil állapotai a rendszernek, a szinkron Hopfield hálózatnak (ellentétben az aszinkronnal) nem minden stabil állapota lokális/globális minimum. Ez azt jelenti, hogy a rendszerben "nemkívánatos" stabil állapotok jelennek meg. Előnye viszont az aszinkron hálózathoz képest, hogy könnyebben kerül ki a rendszer a lokális minimumokból, mivel nemcsak szomszédos állapotokon keresztül jut el egy stabil állapotba.

2. A Boltzmann gépek

2.1. Problémák a Hopfield hálózat alkalmazásakor

Az előző fejezetben bemutattuk, hogy akár a Hebb-szabály alkalmazásával, akár egy alkalmas energiafüggvény megválasztásával a súlyok általában meghatározhatók úgy, hogy a tárolandó minták vonzási pontok legyenek a konfigurációs térben, azaz lokális minimumai legyenek a rendszerhez tartozó energiafüggvénynek. A hálózat egy (Hamming távolság értelemben) megfelelően közeli konfigurációból, a vonzási körzet valamely pontjából indítva, a dinamikus működés során a kívánt állapotban fog stabilizálódni.

Az ideális az volna, ha a teljes konfigurációs teret kitöltenék a tárolt minták és ezek vonzási körzetei, mivel így tetszőleges állapotból indítva a hálózatot, az a legjobban "hasznosító" tárolt mintára asszociálna. A helyzet természetesen nem ilyen szép. A tanított konfigurációkon kívül számos más vonzási pont is létezik. Ezek típusait igazolás nélkül ismertetjük:

- az összes tárolt minta inverzéhez $(-x_i^{(\alpha)})$ is lokális minimum tartozik,
- olyan ún. kevert állapotok is vonzási pontok, amelyek páratlan számú tárolt minta lineáris kombinációiból állíthatók elő. Például ilyenek az alábbi állapotok: $a_i^{\text{kevert}} = \text{sgn}(\pm a_i^{(\alpha_1)} \pm a_i^{(\alpha_2)} \pm a_i^{(\alpha_3)})$,
- nagyméretű hálózatokban viszonylagosan nagyszámú tanított minta esetén bizonyítható, hogy kialakulnak olyan lokális minimumok, amelyek nem korreláltak a minták semmilyen részalmazával sem.

Bár az utóbbi két osztályba sorolt vonzási pontokhoz magasabb energiaértékek tartoznak és általában kisebb a vonzási körzetük is, a rendszer ezekben a hamis lokális minimumokban is stabilizálódhat. Célunk mindenképpen az, hogy elkerüljük a "nem kívánt" lokális minimumokba történő beragadást. A helyes, optimális megoldáshoz tartozó állapot megtalálása egy adott hálózat esetén a kiindulási konfigurációtól függ. A kezdeti állapot az alkalmazások többségében vagy adott, vagy nincs is információnk ennek megfelelő megválasztásáról, így a legtöbb alkalmazás esetében célszerű a Hopfield hálózat működési algoritmusának módosítása.

Az említett hátrányos tulajdonságok kiküszöbölésére többféle ötlet is adódik:

- Ha a kiindulási konfiguráció nem adott, akkor a rendszert különböző kezdeti állapotokból indítva kereshetünk minél mélyebb energiájú megoldást.
- A működési algoritmust úgy módosítjuk, hogy egyszerre (szinkron) több neuron is értéket válthasson, így a konfigurációs tér nagyobb részében végezzük el a keresést.
- Hasonlóképpen megnöveli egy alacsony energiaszintű minimum megtalálásának esélyét, ha a neuronok működését sztochasztikussá tesszük. Ennek egy módja lehet, hogy korlátozott arányban olyan esetben is elfogadjuk a neuronok értékváltását, ha ettől a rendszer magasabb energiájú állapotba kerül.

A következő fejezetben olyan módszert mutatunk be, amely ez utóbbi lehetőséget használja a globális minimum, de legalábbis egy kis energiájú lokális minimum megtalálására.

2.2. Szimulált lehűtés

A lokális minimumok elkerülésére leggyakrabban az ún. szimulált lehűtés módszert alkalmazzák a Hopfield hálónál. A hálózat ilyen kiterjesztését Boltzmann gépnek vagy szimulált lehűtés hálózatként hívjuk [Aar89].

A módosítás alap gondolata, hogy változtassuk sztochasztikussá a minimum-keresési eljárást úgy, hogy az lehetőséget adjon a lokális minimumokból történő kikerülésre. Ezzel tulajdonképpen bizonytalanságot, azaz zajt viszunk be a folyamatba, ami "kiűtheti" a kevésbé mély lokális minimumokból a rendszert, és így az nagyobb eséllyel stabilizálódhat a legmélyebb, globális minimumban [Pav05].

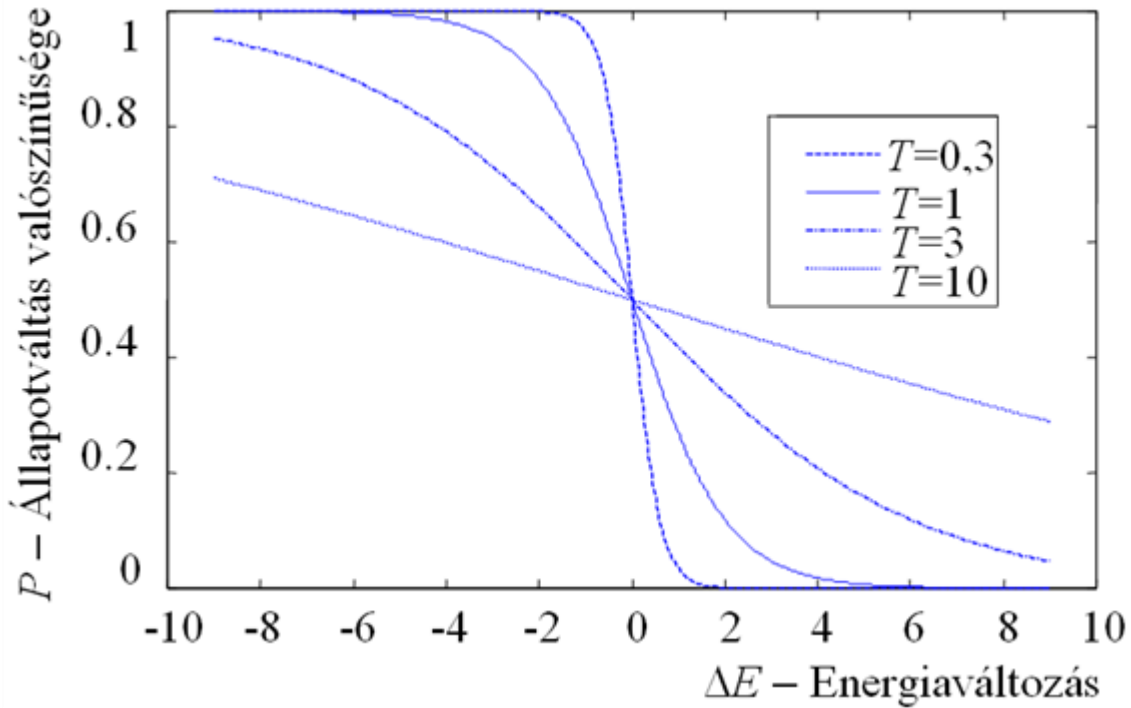
A módszer nevét a szilárd anyagok hőkezelésekor használt hasonló elvű módszerről kapta. Az anyagot először felhevítik, majd lassan lehűtik, és így nem a magasabb energiájú metastabil állapotokban, hanem a szabályos rácsszerkezethez tartozó alacsony energiájú állapotban szilárdul meg.

A szimulált lehűtés tehát a determinisztikus minimumkeresési eljárás helyett sztochasztikus keresést eredményez. A módszer alkalmazása a Hopfield hálózatok esetében a determinisztikus működésű neuronok sztochasztikus egységekkel való felváltását jelenti.

A modell neuronjainak működése egy sztochasztikus szabállyal írható le, amely meghatározza az értékváltások utáni lehetséges állapotok kialakulásának valószínűségét:

$$P(x_i = \pm 1) = f_T(s_i) = \frac{1}{1 + \exp\left(\frac{\mp s_i}{T}\right)}, \quad \text{ahol } s_i = \sum_j w_{ij} x_j + \Theta_i \quad (11.21)$$

11.5. ábra - Különböző T értékekhez tartozó szigmoid karakterisztikák



A neuronok véletlenszerű viselkedését szabályozó T paramétert az analógia alapján pszeudo-hőmérsékletnek, vagy csak egyszerűen hőmérsékletnek nevezik. A (11.21) szerinti szigmoid karakterisztika meredekségét a T paraméter határozza meg, magas T értéknél a karakterisztika lapos, míg ha a paraméter 0-hoz tart, akkor a görbe konvergál a determinisztikus hálózat transzfer függvényéhez, a (11.2) szerinti lépcsőfüggvényhez. Ez azt jelenti, hogy magas hőmérsékleten a neuronok a bemeneti gerjesztésüktől függetlenül véletlenszerűen váltanak értéket, míg a hőmérséklet paramétert csökkentve a rendszer egyre inkább a determinisztikus modellhez közelít. A hálózat működését most is tudjuk jellemezni az energiafüggvény felhasználásával. A Boltzmann gépnek a (11.21) definícióval ekvivalens leírása a következő:

$$P(x_i \rightarrow -x_i) = \frac{1}{1 + e^{\frac{\Delta E}{T}}}, \quad \text{ahol } \Delta E = E(-x_i) - E(x_i) \quad (11.22)$$

Az energiaminimum elérését vagy megközelítését a fizikai analógia alapján a szimulált lehűtés módszerével végezzük. A rendszert egy magas kezdeti hőmérsékletre indítva, a hőmérséklet paraméter lassú csökkentése mellett működtetjük, míg a hálózat stabil állapotba nem kerül. Ha a rendszert megfelelően lassan hűtjük, akkor egy adott hőmérsékleten egyensúlyi állapotba kerül.

A kialakuló egyensúlyban a hálózat egy E_k energiájú $\mathbf{x}_k$ állapotba való kerülésének valószínűségét a Boltzmann eloszlás írja le:

$$P(\mathbf{x} = \mathbf{x}_k) = q_k(T) = \frac{1}{Z(T)} e^{\frac{-E_k}{T}}, \quad (11.23)$$

ahol $q_k(T)$ stacionárius eloszlásban a $Z(T)$ partíciós függvény definíciója egy az összes lehetséges állapotban elvégzett összegzés:

$$Z(T) = \sum_j e^{\frac{-E_j}{T}} \quad (11.24)$$

Amennyiben a rendszer lehűtése során biztosítjuk az aktuális hőmérsékleten az egyensúlyi állapot kialakulását, akkor a hálózat konvergenciáját a következőképpen vizsgálhatjuk:

$$\lim_{T \rightarrow 0} q_k(T) = \lim_{T \rightarrow 0} \frac{e^{-\frac{E_k}{T}}}{\sum_j e^{-\frac{E_j}{T}}} = \lim_{T \rightarrow 0} \frac{e^{\frac{E_{\min} - E_k}{T}}}{\sum_j e^{\frac{E_{\min} - E_j}{T}}} \quad (11.25)$$

A zérus hőmérsékletre tartó határértéket külön felírva az $E_{\min}$ energiaminimummal rendelkező $\mathbf{x}_k \in X_{\min}$ globális minimumhelyekhez tartozó állapotokra és az $E_k > E_{\min}$ energiájú $\mathbf{x}_k \notin X_{\min}$ állapotokra a következőket kapjuk:

$$\begin{aligned} \forall \mathbf{x}_k \in X_{\min} : \quad \lim_{T \rightarrow 0} q_k(T) &= \lim_{T \rightarrow 0} \frac{1}{\sum_j e^{\frac{E_{\min} - E_j}{T}}} = \frac{1}{|X_{\min}|} \\ \forall \mathbf{x}_k \notin X_{\min} : \quad \lim_{T \rightarrow 0} q_k(T) &= \lim_{T \rightarrow 0} \frac{e^{\frac{E_{\min} - E_k}{T}}}{\sum_j e^{\frac{E_{\min} - E_j}{T}}} = 0 \end{aligned} \quad (11.26)$$

Tehát a hálózat a szimulált lehűtés ideális alkalmazásával aszimptotikusan konvergál az optimális megoldásokhoz tartozó állapotok halmazába, azaz az eljárás mentes a lokális minimumba való beragadás problémájától. Az alkalmazások szempontjából azonban fontos, hogy véges idejű lehűtést alkalmazhassunk. A szimulált lehűtés algoritmus tulajdonságai ilyen szempontból is kedvezőek. A lehűtés során az optimális megoldáshoz tartozó állapotba kerülés valószínűsége folyamatosan nő a hőmérséklet csökkentésével, ugyanakkor minden más állapothoz található egy olyan hőmérséklet küszöb érték, ami alatt az adott állapot előfordulási gyakorisága monoton csökken. Ez az eredmény az aktuális hőmérsékletekhez tartozó egyensúlyi állapotokra vonatkozik. Azonban általánosan is igaz, véges idejű lehűtés esetére is, hogy minél lassabb lehűtést alkalmazunk, annál nagyobb a valószínűsége annak, hogy a hálózat alacsony energiájú állapotba kerül.

A Boltzmann hálózat gyakorlati alkalmazásához meg kell határozni a hőmérséklet paraméter változtatásának algoritmusát, az ún. lehűtési stratégiát. A lehűtési stratégia a következő elemeket tartalmazza:

- kezdeti hőmérséklet (T_0)

-A rendszert olyan magas hőmérsékletre indítjuk, amelyen a neuronok véletlenszerűen változtatják állapotukat. A hőmérséklet csökkentését azonnal meg lehet kezdeni, mivel ezen a hőmérsékleten a rendszer kezdeti állapotban van.

- lehűtés szabályozó függvény, amelynek 2 komponense a következő:

-Az adott hőmérsékleten elvégzendő állapotváltozások száma,

-A hőmérséklet csökkentésének mértéke.

-A lehűtés sebességét meghatározó függvény paramétereit a feladat megoldására rendelkezésre álló idő és az adott számítási környezet által meghatározott állapotváltozási sebesség korlátozza a gyakorlatban. Egy elterjedt heurisztikus módszer, hogy a hálózat méretének kisszámú többszöröse számú (tipikusan $3N-15N$) neuron frissítése történjen meg egy adott hőmérsékleten, majd a hőmérsékletet egy 1-hez közeli faktorral szorozva csökkentjük ($T_{k+1} = cT_k$), ahol c értéke általában 0.8 és 0.99 közötti).

- megállási feltétel

-A szimulációt akkor lehet befejezni, ha a rendszer energiája nem változik egy megfelelően hosszú iteráció során. Mivel a hőmérséklet csökkentésének módszere általában olyan, hogy ez az érték aszimptotikusan konvergál a 0-hoz, ezért a minimumhelyekből való kijutás mindvégig lehetséges. Alacsony hőmérsékletnél

azonban ez a valószínűség már olyan kicsi, hogy feltételezhető, hogy a rendszer egy stabil állapotot talált meg. Ezt ellenőrizhetjük a leállás után a hőmérséklet paraméter 0-ra állításával, azaz a determinisztikus hálózattal.

Számos más lehűtési stratégia is ismert a szimulált lehűtés módszerhez, amelyek közül bármelyik felhasználható a Boltzmann hálózat működtetéséhez. A hőmérséklet csökkentésének mértékét szokásos még logaritmikus vagy polinom függvény formájában is definiálni.

3. Mean-field hálózatok

A Hopfield hálózat determinisztikus neuronjait sztochasztikus viselkedésű elemekkel felváltva lecsökkenthető a lokális minimumokba való beragadás valószínűsége. A szimulált lehűtés módszer alkalmazásának az ára viszont, hogy a gyorsan konvergáló Hopfield modell helyett, egy lassú, nagyszámú iterációt igénylő algoritmust kell használni. Bár a Boltzmann hálózat konvergencia-sebessége a lehűtési stratégia paramétereinek változtatásával szabályozható, a gyakorlatban eredményesen használható lehűtési módszerek jelentősen megnövelik a futtatás idejét.

A szimuláció során azonban nincs szükségünk a sztochasztikus rendszer elemei pillanatnyi állapotainak ismeretére, így ha a hálózat viselkedése a sztochasztikus neuronok aktuális értékének számítása nélkül is leírható, akkor ezzel gyorsítani lehet a globális minimum keresését. Bár egzakt megoldás nem ismert erre a problémára, nagyméretű hálózatok esetében a sztochasztikus változók várható értékét becslő közelítés, a mean-field approximáció sikeresen használható. A mean-field módszer jól ismert a statisztikus fizikában, ahol nagy szabadságfokú rendszerek (pl. mágneses anyagok) leírására gyakran alkalmazzák [Her91].

A mean-field közelítés felhasználásakor a rendszer diszkrét, sztochasztikus változóit folytonos, determinisztikus változókkal helyettesítjük [Nak97] sztochasztikus Boltzmann hálózat egy adott hőmérsékleten egyensúlyi állapotba konvergál, így ilyenkor a változók átlagértékei idő-függetlenek. A módszer lényege tehát, hogy az

egyensúlyi állapotban lévő sztochasztikus hálózat neuronjainak átlagértékét $\langle x_i \rangle$ – próbáljuk meghatározni (11.21) felhasználásával:

$$\begin{aligned} \langle x_i \rangle &\equiv P(x_i = 1)(1) + P(x_i = -1)(-1) \\ &= \frac{1}{1 + e^{-2s_i/T}} - \frac{1}{1 + e^{2s_i/T}} = \frac{e^{s_i/T}}{e^{s_i/T} + e^{-s_i/T}} - \frac{e^{-s_i/T}}{e^{-s_i/T} + e^{s_i/T}} = \\ &= \tanh(s_i / T) \end{aligned} \quad (11.27)$$

A mean-field közelítés alkalmazásakor véletlen változók függvényének átlagát közelítjük a változók átlagának függvényével. Helyettesítsük a 11.21-beli, a sztochasztikus neuronok hatását kifejező s_i -ket átlagértékükkel:

$$s_i \rightarrow \langle s_i \rangle = \left\langle \sum_j w_{ij} x_j + \Theta_i \right\rangle = \sum_j w_{ij} \langle x_j \rangle + \Theta_i \quad (11.28)$$

$$\langle x_i \rangle = \tanh(\langle s_i \rangle / T) = \tanh\left(\frac{1}{T} \sum_j w_{ij} \langle x_j \rangle + \Theta_i\right) \quad (11.29)$$

A termikus egyensúlyi állapotban lévő sztochasztikus hálózat neuronjai tehát a (11.29) által definiált N determinisztikus nemlineáris egyenlet megoldásai körül, mint átlagértékek körül oszcillálnak. A (11.29) egyenletet mean-field egyenletnek nevezzük. A fenti nemlineáris egyenletrendszer ekvivalens egy folytonos kimenetű (szigmoid aktivációs függvénnyel rendelkező) Hopfield hálózat modelljével. A megoldás a hálózat stabil állapotához tartozó neuron értékek, azaz a hálózat szimulációjával számítható, ha a (11.29) által definiált rendszer stabil. Ez lényegében egy gradiens alapú keresésnek felel meg. Hasonlóan a diszkrét értékű Hopfield hálózathoz, ehhez a rendszerhez is rendelhető energiafüggvény, amely biztosítja a stabilitást, mivel az energiafüggvény kielégíti Ljapunov stabilitási tételének a globális aszimptotikus stabilitásra vonatkozó feltételeit. A folytonos értékű Hopfield hálózatnak [Far90] a következő függvény energiafüggvénye, ha $\mathbf{W}$ szimmetrikus és $T > -\lambda_{\min}$:

$$E_f = -\frac{1}{2} \sum_i \sum_j w_{ij} x_i x_j - \sum_i \Theta_i x_i + \int_0^{x_j} \tanh^{-1}(x) dx \quad (11.30)$$

Ismert, hogy ha a súlymátrixnak van negatív sajátértéke, akkor a T paraméter kis értékeinél 2 periódusú határciklusba is kerülhet a hálózat. A diszkrét és folytonos hálózatok energiafüggvényének hasonlósága jelzi, hogy a két rendszert is hasonló tulajdonságok jellemzik és egy-egy arányú kapcsolat áll fenn az energia minimumok és maximumok között.

A mean-field algoritmus megvalósítására a leggyakrabban, a Boltzmann gépnél bemutatottakhoz hasonlóan itt is szimulált lehűtést alkalmaznak. Ilyenkor a lehűtés során minden hőmérsékleten ki kell számolni a mean-field változók értékeit, majd ezt a megoldást lehet felhasználni az alacsonyabb hőmérsékleten kezdeti értéként. Mivel a mean-field modell ekvivalens a folytonos Hopfield hálózattal, ezért a hőmérséklet csökkentésével ez a modell is a diszkrét Hopfield hálózathoz konvergál.

Mind a Boltzmann, mind a mean-field hálózatok a lokális keresést végző aszinkron Hopfield hálózat kiterjesztései, amelyek sztochasztikus keresés felhasználásával növelik meg a globális optimum megtalálásának esélyét. Mivel a módszerek egymásnak alternatívái, természetesen adódik a kérdés, hogy melyiket érdemes alkalmazni. Egyértelmű válasz megfogalmazása még egy konkrét probléma esetében is nehéz és bátorságot igénylő feladat. A következőkben az elméleti és tapasztalati ismeretek alapján több szempont szerint fogjuk vizsgálni és összehasonlítani a Boltzmann és mean-field neurális hálózatokat.

A módszerek elméleti háttére

A szimulált lehűtés módszerét közvetlenül alkalmazó Boltzmann hálózatról ismert, hogy aszimptotikusan konvergál az optimális megoldáshoz, sőt az eljáráshoz optimális sebességű lehűtési stratégia is ismert. Ezen felül, ehhez az algoritmushoz rendelkezésre áll olyan vizsgálati lehetőség, amely a megtett iterációk számához képes az aktuális megoldás várható költségének az optimális megoldás költségétől való eltérésére korlátot adni. Bár ezt a pontosság mértékét meghatározó eljárást a gyakorlatban jelentős számításiigénye miatt nem tudjuk alkalmazni, a korlát léte önmagában is eredmény. A mean-field módszer esetében ilyen igazolt tételek nem állnak a rendelkezésünkre. Ennek oka az, hogy a mean-field közelítés pontosságára nem ismert becslés véges méretű rendszerek esetében.

A megoldások minősége

A tapasztalatok azt mutatják, hogy a Boltzmann hálózat sok esetben az elméletileg szükségesnél nagyságrendekkel gyorsabb lehűtés alkalmazása esetén is igen jó minőségű megoldást ad. A mean-field módszer esetében a keresés az átlagolással és az átlag meghatározás pontatlanságával járó információvesztés miatt kevésbé érzékeny a probléma részleteire, ami az optimálisához közeli energiaértékű megoldások megtalálásának valószínűségét csökkenti. Különösen igaz ez, ha az energiafelület változatos, számos, viszonylag kis átmérőjű mély gödröt illetve csúcsot tartalmaz.

A keresés ideje

A sztochasztikus rendszer valódi állapotváltozásait nem követő, az átlagértékeket közvetlenül becsülő mean-field hálózat lényegesen kevesebb iteráció alatt konvergál az egyensúlyi állapot értékeihez, mint a Boltzmann hálózat. A tapasztalatok azt mutatják, hogy a mean-field módszer kevésbé érzékeny a lehűtési stratégia paramétereinek megválasztására és a lehűtés sebességének növelésével kevésbé romlik a hatékonysága. Ezt a tapasztalatot támasztja alá az energiafüggvények jellege közti különbség. A mean-field módszert megvalósító folytonos kimeneti függvényrel rendelkező Hopfield hálózat energiafüggvénye simább, mint a hasonló súlyokkal rendelkező diszkrét hálózaté, amelynek következtében a minimumok keresése gyorsabban is folytatható.

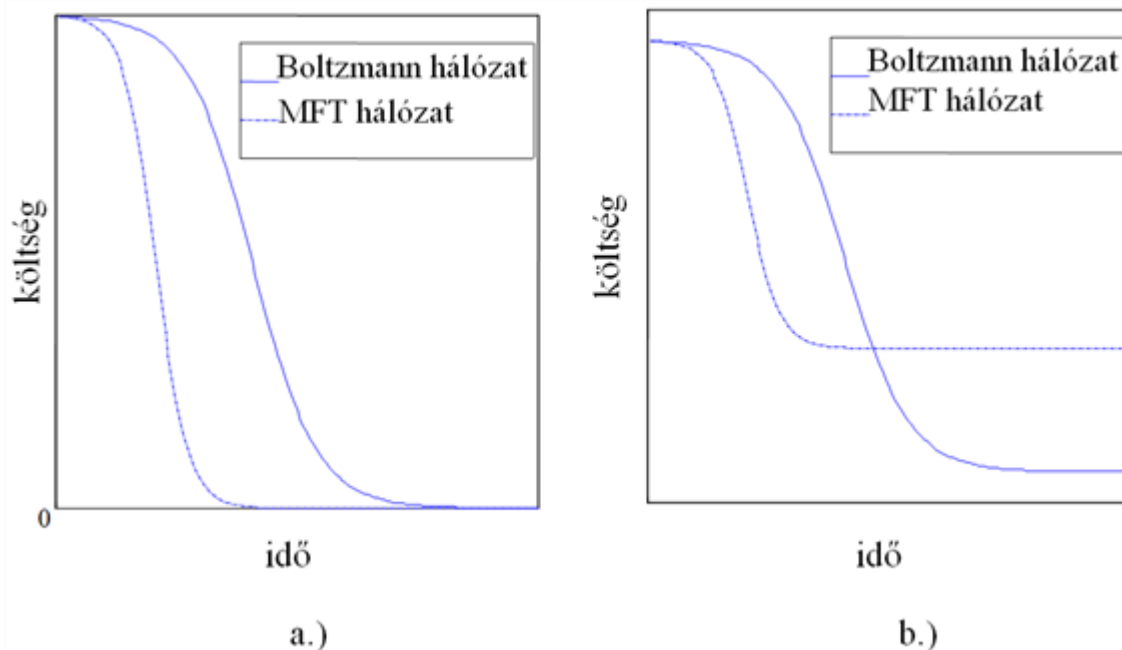
A probléma jellege

A Boltzmann hálózat egyaránt alkalmas kis és nagyméretű, illetve simának vagy változatosnak tekinthető energiafelületet eredményező feladatok megoldására. A mean-field módszer viszont azon a feltevésen alapszik, hogy megfelelő közelítés a nagyméretű homogén rendszer átlagértékeit meghatározni. Ezért kisebb méretű problémák vagy olyan bonyolult feladatok esetében, amikor ezek a feltevések nem igazak, akkor a mean-field egyenletek megoldása az eredeti probléma optimumától távoli eredményeket adhat.

Természetesen egy átfogó értékeléshez vagy egy alkalmazás esetében a megfelelő módszer kiválasztásához a fenti szempontokat együttesen kell figyelembe venni. Az eddigi érvelések a módszerek közti választáshoz mérlegelési szempontokat adhatnak, a tesztek megtervezéséhez iránymutatást nyújthatnak. A sok bizonytalanságot tartalmazó elemzés összefoglalásaként két jellemző görbe párt mutat be a 12.6 ábra. Látható, hogy a nagyobb méretű, de egyszerűbbnek tekinthető feladatok esetében jól kihasználható a mean-field közelítés gyorsasága, míg a bonyolultabb problémáknál a Boltzmann hálózat alacsonyabb költségű megoldást eredményez, bár ha egy adott időn belül kell választ adni, akkor létezik egy határérték, ami alatt a mean-field hálózat hatékonyabb.

Ha egy alkalmazási feladat esetében van lehetőség a két módszer között vizsgálatok alapján választani, akkor mind a két, különböző irányú megközelítés mellett lehet hasonló súlyú érveket felhozni. Kezdhetjük a vizsgálatokat a mean-field hálózattal, és ha nem sikerül megfelelő pontosságú eredményt elérni, akkor érdemes a Boltzmann hálózatot is alkalmazni. Vagy elemezhetjük először a Boltzmann hálózat eredményeit, és akkor döntünk a mean-field módszer vizsgálata mellett, ha elfogadható futási időn belül nem kapunk megfelelő minőségű megoldást.

11.6. ábra - Tipikus lehűtési görbék Boltzmann és mean-field hálózatokkal a.) homogén energiafüggvényen b.) változatos, sok lokális minimumot tartalmazó energiafüggvényen



4. Hopfield típusú hálózatok alkalmazása optimalizációs problémákra

Kombinatorikai optimalizációs feladatok megoldása nem jelent mást, mint megkeresni az optimálist a számos lehetséges megoldás közül. Rengeteg ilyen feladatra visszavezethető gyakorlati probléma ismert a számítástechnika, a VLSI tervezés, a közlekedésirányítás, az üzleti élet és számos egyéb területről.

Az utóbbi egy-két évtizedben ezek a feladatok előtérbe kerültek, mivel sok korábban kezelhetetlennek tűnő probléma részben vagy teljesen megoldhatóvá vált. Ennek oka a technika fejlődésén kívül, a problémák nehézségének meghatározásában elért eredmények és az egyre hatékonyabb megoldási algoritmusok megjelenése. A legjelentősebb felismerés a 60-as évek végén született, annak a máig nem bizonyított, de nem is cáfolt hipotézisnek a felállításával, amely azt mondja ki, hogy létezik a problémák egy osztálya, amelyre az jellemző, hogy nem lehet konstruálni olyan algoritmust, ami a probléma méret hatványával arányos lépésszámmal biztosan képes a megoldást megtalálni. Ezek a problémák az NP-teljes problémák körébe tartoznak. Az optimalizáció témakörébe tartozó feladatok jelentős része ebbe a kategóriába sorolható.

Az NP-teljes problémák megoldásánál kétféle stratégiát lehet követni. Ragaszkodhatunk az optimális megoldás megtalálásához, és ilyenkor kockáztatjuk egy nagyon hosszú, sok esetben kivárhatatlan eljárás esélyét, vagy

pedig egy viszonylag gyorsan megkapható megoldást választunk tudomásul véve, hogy esetleg az optimálisnál rosszabb minőségű megoldást találunk. Az első lehetőséghez tartoznak azok az optimalizációs eljárások, amelyek előzetes információk felhasználásával megállapított, hatékonynak ígérkező sorrendben megvizsgálják az összes lehetséges variációt. Ilyenek például a különböző kimerítő kereső eljárások. A másik csoportba sorolhatók a heurisztikát, véletlen elemeket használó módszerek. Ezek közös jellemzője, hogy valamilyen ismeret vagy elv szerint a lehetőségek egy részét figyelmen kívül hagyják. Ide tartoznak a különböző neurális hálózatokat alkalmazó módszerek is.

Először megmutatjuk, hogyan lehet neurális hálózatokat optimalizációs problémák megoldására használni, majd néhány jellegzetes feladat Hopfield hálózattal, illetve Boltzmann géppel történő megoldását vázoljuk fel.

Neurális hálózatokat egy optimalizációs probléma megoldására a következő két előfeltétel esetén tudunk használni:

1. A lehetséges megoldások száma legyen véges.
2. A megoldások minősége számszerűleg kifejezhető és bármely más megoldás minőségével összehasonlítható legyen.

A feltételeknek megfelelő optimalizációs problémákat formálisan egy (S, C) kettőssel lehet leírni, ahol a megoldási tér, S egy véges halmaz, amely tartalmazza az összes lehetséges megoldást, míg C a költségfüggvény vagy kritériumfüggvény egy leképzés:

$$C: S \rightarrow \mathbb{R}, \quad (11.31)$$

Tehát C minden megoldáshoz egy valós értéket, egy költséget rendel. Ezek után a feladat egy optimális $i_{\text{opt}} \in S$ megoldásnak a megkeresése, amelyre a minimalizációs probléma esetén igaz, hogy:

$$C(i_{\text{opt}}) \leq C(i), \quad \forall i \in S \text{-re} \quad (11.32)$$

(Maximalizációs probléma esetén az egyenlőtlenség természetesen fordítva áll.)

Például, a talán legismertebb, bár közvetlen gyakorlati jelentőséggel nem bíró, de tipikus optimalizációs feladat, az utazó ügynök probléma (*Traveling Salesman Problem, TSP*) a következőképpen írható le. A problémában egy ügynöknek adott városokat kell végiglátogatnia úgy, hogy minden városban csak egyszer járjon és a lehető legrövidebb utat megtéve érkezzen vissza a kiindulási pontba.

Legyen adott az n város közti távolságokat leíró $n \times n$ -es $[d_{pq}]$ mátrix. Egy, az ügynök által bejárandó, lehetséges útvonalat jelöljön a $P=(P(1), \dots, P(n))$ ciklikus permutáció, ahol $P(k)$ jelenti azt a várost, amelyik a k -adik után következik az úton ($P(k) \neq k$).

Így a lehetséges megoldások: $S = \{\text{az } n \text{ város összes } P \text{ ciklikus permutációjára}\}$

11.7. ábra - Egy jó és egy rossz megoldás az utazó ügynök problémára



A megoldási tér mérete: $|S|=(n-1)!$, és ismert, hogy ez is egy NP-teljes probléma. A feladathoz tartozó költségfüggvény, azaz az út hossza:

$$C(P) = \sum_{p=1}^n d_{p,P(p)} \quad (11.33)$$

Optimalizációs feladatok neurális hálózattal történő megoldásához a problémát le kell képeznünk a neurális struktúrára. Ennek lényege, hogy a hálózat minden konfigurációjához hozzárendelünk egy megoldást. Ha egy Boltzmann gép formális leírását egy $B(\mathbf{x}, \mathbf{W})$ kettőssel határozzuk meg, ahol $\mathbf{x}$ a neuronok állapotvektora és $\mathbf{W}$ a súlymátrix, akkor a probléma leképezését jelölhetjük az $m: \mathbf{x} \rightarrow \mathcal{S}$ függvénnyel.

Ezek alapján az optimalizációs problémák implementációjának stratégiája a következő 3 lépésben foglalható össze.

1. Alakítsuk át az optimalizációs feladatot 0-1 programozási feladattá. Ehhez az eredeti diszkrét változók helyett megfelelő számú új bináris változót kell bevezetni.
2. Definiáljunk egy Boltzmann gépet úgy, hogy minden egyes neuron a probléma egy változójának feleljen meg.
3. Határozzuk meg a hálózat súlyait úgy, hogy a hálózathoz tartozó energiafüggvényre igaz legyen a következő két feltétel:

I. Az energiafüggvény minimumpontjaihoz tartozó konfigurációknak a probléma változóinak olyan értékei feleljenek meg, amelyekhez optimális megoldások tartoznak (a költségfüggvény minimumpontjai)

$$m(\mathbf{x}_{\text{opt}}) = \mathcal{S}_{\text{opt}} \quad (11.34)$$

II. Az energiafüggvény és a költségfüggvény legyen nagyságrendtartó:

$$C(m(x_1)) \langle C(m(x_2)) \Rightarrow E(x_1) \langle E(x_2) \quad (11.35)$$

Ha az előzőekben leírtak szerint hozunk létre egy Boltzmann gépet optimalizációs probléma megoldására, akkor a hálózat egy véletlen kiindulási konfigurációból elindítva olyan állapotban stabilizálódik, amely a probléma egy jó megoldását adja. Az utolsó pontban szereplő feltételek biztosítják, hogy ha a hálózat globális minimumpontban stabilizálódik, akkor optimális megoldást kapunk, de ha egy alacsony energiájú lokális minimum lesz a stabil állapot, akkor is jó minőségű, alacsony költségű megoldáshoz jutunk.

4.1. Az utazó ügynök probléma

A már bemutatott utazó ügynök problémát a fentiek alapján könnyen lehet Boltzmann típusú neurális hálózatra implementálni. Először új bináris változókat vezetünk be a probléma leírására:

$$u_{ip} = \begin{cases} 1 & \text{ha az út során az } i\text{-edik város a } p\text{-edik lépésben kerül meglátogatásra} \\ 0 & \text{egyébként} \end{cases}$$

A bevezetett változókkal a feladat költségfüggvénye, az út hossza így írható le:

$$C(P) = \sum_{i,j,p,q=0}^{n-1} a_{ijpq} u_{ip} u_{jq} \quad (11.36)$$

ahol az a_{ijpq} együtthatókat a távolságok alapján a következőképpen definiáljuk:

$$a_{ijpq} = \begin{cases} d_{ij} & \text{ha } q=(p+1) \bmod n \\ 0 & \text{egyébként} \end{cases} \quad (11.37)$$

A költségfüggvény minimalizálásán kívül ügyelni kell, hogy a megoldás valódi utat írjon le, ezért be kell tartani a következő két kényszert:

$$\begin{aligned} \forall p\text{-re } \sum_{i=0}^{n-1} u_{ip} &= 1 \quad (\text{egy időpontban csak egy városban lehet az ügynök}), \\ \forall i\text{-re } \sum_{p=0}^{n-1} u_{ip} &= 1 \quad (\text{minden várost pontosan egyszer látogasson meg az ügynök}). \end{aligned} \quad (11.38)$$

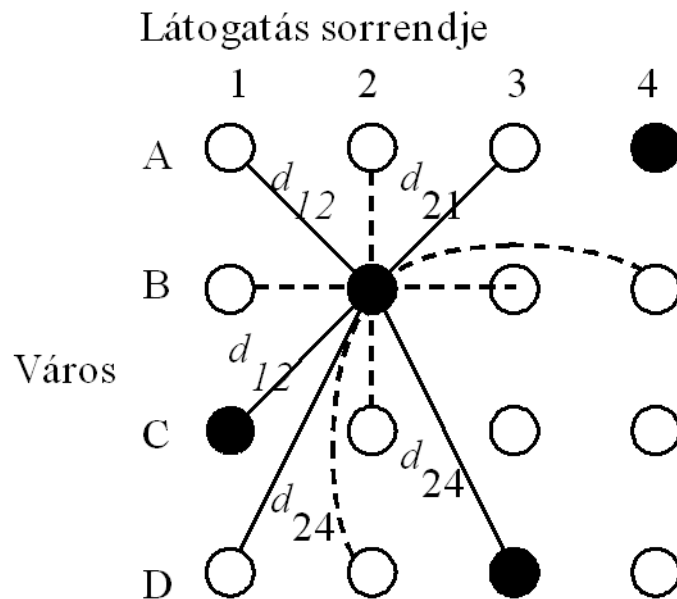
A problémában szereplő minden bináris u_{ip} változónak a hálózat egy x_{ip} neuronja fog megfelelni, így a hálózat n^2 neuront fog tartalmazni. A Hopfield modellről szóló fejezetben leírtak alapján a hálózat súlyait az energiafüggvény megadásával definiáljuk. Az utazó ügynök probléma megoldásához definiált függvény három tagból áll, melyekből az első a költségfüggvény minimalizálásáért felelős, a másik kettő pedig a problémában szereplő két kényszert realizálja:

$$E = \sum_{i,j,p,q=0}^{n-1} a_{ijpq} x_{ip} x_{jq} + c \left[\sum_{i=0}^{n-1} \left(1 - \sum_{p=0}^{n-1} x_{ip} \right)^2 + \sum_{p=0}^{n-1} \left(1 - \sum_{i=0}^{n-1} x_{ip} \right)^2 \right] \quad (11.39)$$

A hálózat súlyait az energiafüggvény tagjainak kiszorítása után közvetlenül megkapjuk, mivel nem szerepel benne x_{ip} négyzeténél magasabb hatványfokú tag. Könnyű belátni, hogy a függvény minimumpontjai ott vannak, ahol az út minimális és a megoldás megfelel mind a két kényszernek, ezenkívül teljesíti a függvénnyel kapcsolatban korábban leírt feltételeket.

Az eredmények azt mutatják, hogy az utolsó két tagot súlyozó c paramétert helyesen megválasztva igen jó megoldást talál a hálózat a problémára. Hozzá kell tenni azonban, hogy a tapasztalatok szerint a megoldás minősége jelentősen függ c értékétől.

11.8. ábra - Egy 4 várost tartalmazó utazó ügynök probléma.



Az ábrán látható 16 neuronból a 4 aktív (fekete) processzáló elem egy lehetséges megoldást mutat. Az ábrán jelöltük még az n_{22} neuron súlyozott összeköttetéseit (a szaggatott vonalak a c súlyú kapcsolatokat jelölik).

A módszert hagyományos kombinatorikai algoritmusokkal összehasonlítva, a talált út hosszát vizsgálva a neurális hálózat a hagyományos algoritmusok megoldásainál valamivel gyengébb eredményt ad. Ennek alapvető oka, hogy a hálózat a véletlenszerű állapotból kiindulva részben olyan állapotokon keresztül jut el a stabil állapotig, amelyek nem valódi utaknak felelnek meg, míg a hagyományos eljárások közül a legjobbak csak ilyen vesznek figyelembe. Így a hálózat számára a problémátér, melyben az optimális megoldást keresi, lényegesen nagyobb. Ennek a gondnak a kiküszöbölésére javasolták a hálózat struktúrájának olyan módosítását, hogy a neuronokat csoportba szervezve ún. " k -ből 1 neuron"-okat alakítottak ki, ahol garantált, hogy a hálózat minden

neuron csoportjában mindig pontosan egy aktív van. Ilyenkor az energiafüggvény első tagjának módosítása után az utolsó két tag elhagyható. Ezzel a módszerrel a Boltzmann gépek mind minőség, mind sebesség tekintetében lényegesen jobb eredményeket értek el.

4.2. Rádiófrekvenciák kiosztása neurális hálózattal

Hasonló problémára vezet a haditechnikában és a polgári távközlésben is felmerülő feladat, a nagyszámú rádiókapcsolatok létesítésének feladata. Igen bonyolult probléma több száz, esetleg néhány ezer rádiókapcsolathoz frekvenciát rendelni úgy, hogy azok ne zavarják egymást. A gyakorlatban a frekvencia értékeket a használt eszközöktől függő véges halmazból kell választani, és ki kell elégíteni a rádiókapcsolatok földrajzi elhelyezkedésétől függő kényszereket. Az alapproblémában a kapcsolatokhoz $l_1, \dots, l_n$ frekvencia értékeket rendelünk az $f_1, \dots, f_p$ lehetséges értékek közül. A kapcsolatokhoz rendelt frekvenciaértékekre a kényszerek a következő alakban írhatók fel:

$$|l_i - l_j| \geq d_{ij} \quad (11.40)$$

Minden lehetséges (l_i, f_k) kettőshöz rendeljünk hozzá egy bináris v_{ik} változót:

$$\begin{aligned} v_{ik} &= 1 \text{ ha } l_i = f_k \\ v_{ik} &= 0 \text{ ha } l_i \neq f_k \end{aligned} \quad (11.41)$$

A v_{ik} változóknak feleltessük meg a neurális hálózat x_{ik} neuronjait, majd a neuronok közti kapcsolatok súlyait kell meghatározni. Itt is az energiafüggvény meghatározásával jutunk eredményre. A probléma költségfüggvénye legyen a ki nem elégített feltételek száma. Az ezt reprezentáló energiafüggvény tag a következő alakú:

$$E_1 = \sum_{i,p,j,q} (1 - w_{ipjq}) x_{ip} x_{jq} \quad , \quad (11.42)$$

ahol

$$\begin{aligned} w_{ipjq} &= 1 \text{ ha } |f_p - f_q| < d_{ij} \\ w_{ipjq} &= 0 \text{ egyébként} \end{aligned} \quad (11.43)$$

Az energiafüggvényt, az utazó ügynök problémával bemutatottakhoz hasonlóan, ki kell még egészíteni egy taggal, amely minimumpontjában azt garantálja, hogy egy rádiókapcsolathoz pontosan egy frekvenciát rendelünk, illetve itt is alkalmazható a neuronok csoportba rendezésével (k -ból 1 neuronok) az utóbbi feltételt automatikusan teljesítő módszer.

4.3. A/D konverter megvalósítása Hopfield hálózattal

A/D konverterek érdekes megvalósítását mutatta be az alkalmazott hálózat névadója, John Hopfield. Mivel az eddigiekben csak kétértékű neuronokkal foglalkoztunk ezért itt a szerző által leírt folytonos értékű neuronokat alkalmazó A/D konverter megoldás digitális neurális hálózattal megvalósított változatát mutatjuk be [Hop86]. A hálózat modell itt kiegészül egy analóg bemenettel (a digitalizálandó jel – u), amelyet – hasonlóan a többi neuron kimenetének visszacsatolásához – súlyozott kapcsolatokon keresztül vezetünk a neuronok bemenetére. Az A/D átalakító költségfüggvénye legyen az átalakítás hibájának a négyzete, és a neuronok bináris kimenetei a digitalizált jel bitjei.

Így a hálózat energiafüggvénye:

$$E = \left(u - \sum_i x_i 2^i \right)^2 \quad (11.44)$$

Ebből a hálózat súlyaira a következőket kapjuk:

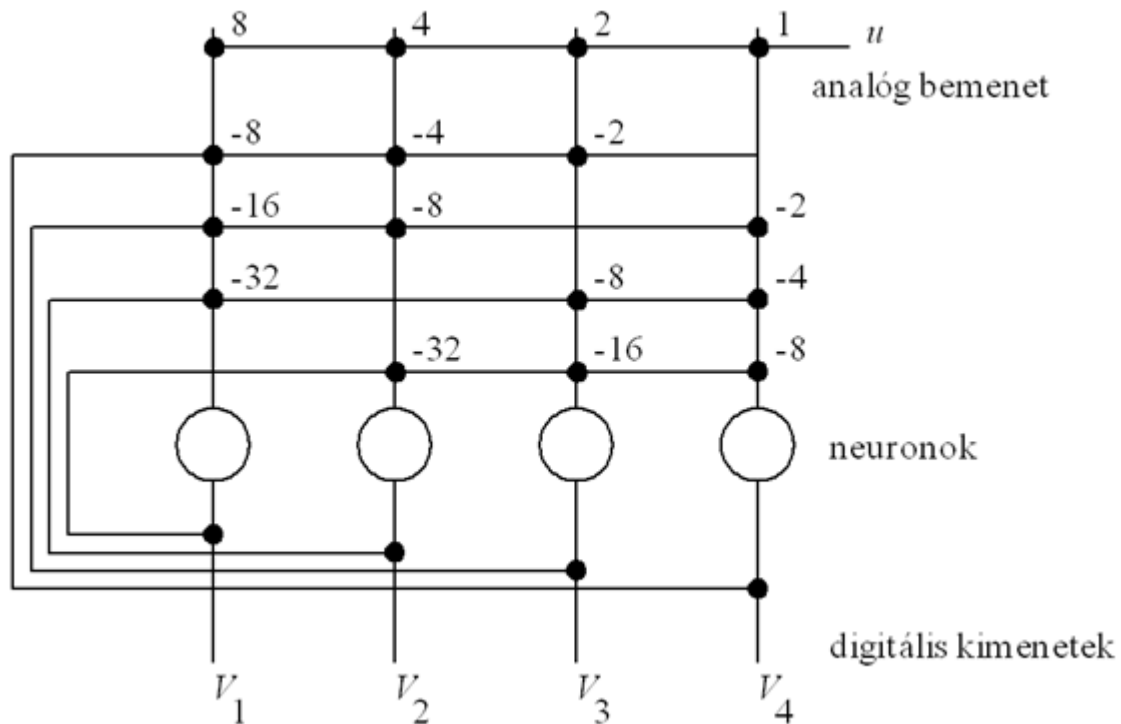
$$w_{ij} = 2^{i+j} \quad (11.45)$$

Az analóg bemenetet súlyozó tagokra:

$$t_i = -2^i \quad (11.46)$$

Egy 4-bites A/D konverter látható a 11.9 ábrán. Bár a megoldás nem hódított teret más A/D megvalósítások mellett, jól mutatja, hogy a feladatok milyen széles köre értelmezhető optimalizációs problémaként és oldható meg Hopfield típusú hálózattal.

11.9. ábra - 4 bites A/D konverter megvalósítása Hopfield hálózattal



Feladatok

11.1 Tervezzon Hopfield neurális hálózatot, amely megoldja az N -királynő problémát. A logikai feladvány lényege, hogy egy $N \times N$ -es sakktáblán kell elhelyezni N királynőt úgy, hogy egyik se álljon másik által ütésben.

Bizonyított, hogy minden $N > 3$ esetre létezik megoldás. Határozza meg egy N^2 neuront tartalmazó hálózat súlyait úgy, hogy a hálózat minden neuronja a sakktábla egy-egy mezőjének felel meg és egy neuron a hálózat stabil állapotában akkor vesz fel aktív értéket, ha rajta egy királynő található.

11.2 Adja meg annak a Hopfield neurális hálózatnak az energiafüggvényét, amely képes megoldani a következő gráf particionálási problémát. Válasszunk szét egy gráfot két egyenlő méretű részgráfra úgy, hogy a köztük található élek száma minimális legyen. A hálózat neuronjai reprezentálják a gráf csomópontjait úgy, hogy az energiafüggvény által meghatározott súlyokkal rendelkező hálózat stabil állapotában a neuronok aktív értéke jelölje az egyik, míg az inaktív érték a másik részgráfhoz való tartozást.

11.3 Mutassa meg, hogy az aszinkron Hopfield hálózatnak új hamis stabil állapotai jöhetnek létre, ha az önvisszacsatoló súlyok értékét növeljük.

11.4 Mutassa meg, hogy a szimmetrikus súlyokkal rendelkező szinkron Hopfield hálózatnál lehetséges, hogy két állapotváltozás után ugyanabba az állapotba kerül a hálózat (2 hosszúságú határciklus).

12. fejezet - Hibrid-neurális rendszerek

Láttuk, hogy a neurális hálózatok olyan rendszerek, melyek tudásukat, képességüket mintákból történő tanulással nyerik. A neurális hálózatok kialakításánál ezért alapvető fontosságú, hogy egy megoldandó problémáról megfelelő mennyiségű és minőségű ismeret adatok (mintapontok) formájában álljon rendelkezésünkre. Egy konkrét feladat kapcsán azonban a mintapontokon kívül szinte mindig van további ismeretünk is a megoldandó feladatról. Az *a priori* ismeret felhasználása alapvetően fontos a feladat megoldása szempontjából. Az *a priori* ismeret segít a megfelelő hálóstruktúra meghatározásában, de döntő szerepe lehet a megfelelő tanítókészlet, a *tanító adatbázis* kialakításánál is. Ennek alapján dönthetjük el, hogy egy feladatról mely rendelkezésre álló vagy megszerezhető jellemzők, mérési adatok alapján célszerű a neuronhálót megkonstruálni.

Az *a priori* ismeret a mintapontok által reprezentált leképezésről is adhat járulékos információt. Mint láttuk, a mintákból való tanulás rosszul definiált feladat, így bármilyen, a mintákon túl rendelkezésre álló ismeret felhasználása jobb megoldás elérését segítheti. Különösen fontos az *a priori* információ felhasználása, ha viszonylag kevés mintapont áll a rendelkezésünkre, valamint ha a mintapontok bizonytalan tudást képviselnek – pl. a mintapontokat jelentős zaj terheli –, vagy ha a mintapontok nem kellően fedik le a problémateret.

A járulékos ismeret különböző formában állhat rendelkezésünkre. Így birtokában lehetünk a megoldandó problémára vonatkozó bizonyos matematikai összefüggéseknek, vagy ismerhetünk a bemeneti és a kimeneti adatok kapcsolatát jellemző bizonyos szabályokat stb. Ilyen *a priori* ismeret lehet pl. hogy egy regressziós problémánál a bemenetek és a kimenetek között monoton kapcsolatnak kell fennállnia, esetleg tudunk valamit a bemenetek és a kimenetek közötti kapcsolat trendjéről, vagy ismerünk valamit a leképezés komplexitásáról vagy simaságáról. Hasonlóan, egy osztályozós feladathoz rendelkezésünkre állhat olyan *a priori* ismeret, hogy a minták bizonyos transzformációira (pl. eltolás, forgatás, tükrözés, esetleg adott korlátok közötti nagyítás, kicsinyítés stb.) az osztályozásnak invariánsnak kell lennie. Olyan *a priori* ismeretünk is lehet, amely korlátozza, hogy milyen tartományban vehetnek fel értékeket az együttesen előforduló bemeneti és/vagy kimeneti adatok.

A neuronhálók egyik „hiányossága” – legyen a háló bármilyen típusú is –, hogy közvetlenül csak mintapontok formájában meglévő ismereteket tudnak hasznosítani. Olyan módszerekre van tehát szükségünk, melyek lehetővé teszik más reprezentációjú tudás felhasználását is. Ez két úton érhető el: vagy lehetőségünk van arra, hogy a járulékos ismeretet közvetlenül felhasználjuk a neuronhálók konstrukciójában, vagy közvetve hasznosítjuk azáltal, hogy a meglévő minták kiegészítésére további, „virtuális” mintákat generálunk.

A közvetlen felhasználásnál a járulékos tudás hozzásegíthet a megfelelő háló architektúra, esetleg a háló méret minél pontosabb megválasztásához, vagy elősegítheti a háló-paraméterek kezdeti értékeinek jobb meghatározását, nagyságrendekkel megkönnyítve ezzel a megfelelő neuronhálós megoldás elérését. Egyes esetekben a megvalósítandó leképezésre vonatkozó formális összefüggések közvetlenül a tanuló eljárásba építhetők be.

A közvetett felhasználásnál a meglévő mintákból a járulékos információk felhasználásával generálunk további mintákat, melyek a megoldandó feladatot a problémátér olyan tartományain is reprezentálni fogják, melyekre az eredetileg rendelkezésünkre álló minták nem adnak információt.

A neuronhálók másik „hiányossága”, hogy ugyan a mintapontokból kinyerik a tudást, és így egy feladatot akár kiválóan meg is tudnak oldani, a megoldás értelmezésében nem segítenek. A neuronháló alapján véve egy fekete doboz, ami a minták által képviselt kapcsolatokat közelíti, és bármilyen jó választ is ad egy bemenetre, a válaszhoz nem nyújt indoklást. Ez a hiányossága számos alkalmazási területen nehezíti vagy akár meg is gátolja a felhasználást. Egy orvosi diagnosztikai vagy döntéstámogató rendszerben például hiába ad egy neuronháló adott bemenetekre (tünetek) jó választ (diagnózis), indoklás, magyarázat nélkül nehéz elfogadni a háló választ. A hálónak ez a hiányossága egyes, pl. biztonságkritikus alkalmazásoknál jelentős hátrány más tanuló rendszerekkel (pl. döntési fák) szemben.

Fontos ezért, hogy a neuronhálók képesek legyenek különböző reprezentációjú ismeret hasznosítására, a hálók konstrukciójánál ne csak a mintapontokat tudjuk felhasználni, továbbá, hogy egy megtanított hálóban „meglévő” tudás szimbolikus formában is kinyerhető legyen és, ezáltal a válaszokhoz magyarázatot is tudjunk rendelni.

Az olyan rendszerek, melyek a minták által képviselt ismeret mellett más reprezentációjú tudást is képesek felhasználni, kezelni, illetve melyekből a tudás szimbolikus formában is kinyerhető a **hibrid-neurális rendszerek** (*hybrid-neural systems*) [Wer00]. A hibrid-neurális rendszerek a hibrid intelligens rendszerek nagyobb családjához tartoznak. Hibrid intelligens rendszereknek nevezzük azokat a rendszereket, melyek a mesterséges intelligencia különféle eszközeit, megközelítéseit együttesen alkalmazzák. Hibrid intelligens eszközöknek szokás nevezni a **lágy számítástechnika** (*soft computing*) különböző (fuzzy, neurális, genetikus) megközelítéseit együttesen alkalmazó eszközöket is. A hibrid-neurális rendszerekben a neuronhálók szerepe meghatározó, de nem kizárólagos. Ezek a rendszerek tehát a megfelelő képességüket döntően továbbra is mintapontokból tanulás útján nyerik, de konstrukciójukban a járulékos a priori információ is jelentős szerepet kap. Azt is mondhatjuk, hogy egyes hibrid-neurális rendszerek a különböző ismeret-reprezentációk közötti konverzióra is képesek.

A következőkben a hibrid-neurális rendszerekhez tartozó két főbb területtel foglalkozunk röviden. Az első területen belül előbb az a priori információ virtuális minták generálásán keresztül történő felhasználásának fontosabb kérdéseit tekintjük át, majd arra adunk példát, hogy a járulékos információ hogyan építhető be magába a tanító eljárásba. A második terület a minták alapján történő tanuló rendszerek és a magyarázat alapú rendszerek ötvözésének egy viszonylag kiforrott eredményével foglalkozik. A **tudás alapú neuronháló** (*knowledge based artificial neural network, KBANN*) a megoldandó problémáról szabályok formájában rendelkezésünkre álló tudást a kiinduló neuronháló (MLP) létrehozásához használja fel, majd az így kialakított háló módosítása, pontosítása történik meg a hagyományoshoz hasonló tanuló eljárás során. A KBANN fontos jellemzője, hogy nemcsak lehetőséget ad szimbolikus formában meglévő ismeretek felhasználására, hanem azt is biztosítja, hogy a megtanított végleges hálóból szabályok formájában kinyerjük a tudást. Az a priori információ egy további hasznosítási lehetőségével, nevezetesen azzal, hogy a rendelkezésre álló jellemzők, megfigyelések közül milyen szempontok alapján válasszuk ki a neurális megoldás létrehozásánál felhasználándókat, a következő, 13. fejezetben fogunk kitérni.

1. Az a priori tudás felhasználása virtuális minták generálására

Virtuális minták generálása az a priori tudás felhasználásának az egyik legtermészetesebb módja. Az így létrehozott mintákkal kiegészítve a meglévő mintakészletet a hálók konstrukciója az eddig bemutatott eljárásoktól semmiben sem tér el. A virtuális mintákon keresztül a megoldandó problémáról további ismereteket ún. tárgy tartomány tudást (*domain knowledge*) tudunk a neuronháló által kezelhető formában hasznosítani. A virtuális minták lehetőséget adnak arra, hogy a problémátér olyan tartományairól is legyenek felhasználható mintáink, melyeket az eredetileg rendelkezésre álló minták nem fednek le, de azt is biztosíthatják, hogy a mintakészlet tükrözze a problémában eleve meglévő invariancia tulajdonságokat.

Az első esetre jó példa a 7. fejezetben már említett autonóm autóvezető rendszer, az ALVINN [Pom91]. Mint láttuk, ott kameraképek alapján kellett egy neuronhálónak beállítani egy terepjáró kormányát úgy, hogy az emberi kormányzás nélkül tudjon közlekedni egy autópályán. A tanító mintakészlet olyan kamerakép-kormánybeállítás párokból állt, melyeket egy emberi sofőr által vezetett autó normál menete közben vettek fel. Minthogy azonban egy ember „túl jó” sofőr, a begyűjtött minták extrém szituációkat nem reprezentáltak. Olyan helyzetekre tehát nem lehetett ezen minták alapján felkészíteni az automata kormányost, mikor az autó túlzottan letért a helyes nyomvonalról. Vagyis a problémátér egy jelentős tartományából nem álltak rendelkezésre minták. Bár a neuronhálók képesek általánosításra, ez az általánosítás inkább interpoláció, mint extrapoláció jellegű. Olyan szituációkra, melyek ismert helyzetek között helyezkednek el, egy háló általában jó válasszal reagál. Ha azonban a tanító minták által reprezentált ismert helyzetektől jelentősen eltérő szituáció fordul elő, a háló képtelen jó választ adni. Az ALVINN kapcsán az ilyen extrém vagy csak a normál menettől jelentősebben eltérő helyzetekre generáltak mintákat, felhasználva a problémáról meglévő a priori tudást. Ismerve a kamerának az autóhoz viszonyított helyzetét, és felhasználva a menet közben összegyűjtött mintákat, megfelelő transzformációval olyan virtuális mintákat – kameraképeket – lehetett előállítani, melyek akkor fordultak volna elő, ha az autó extrém helyzetbe került volna (pl. áttér az út bal oldalára, lefut az útpadkára, stb.). Az új képeknek és az autó ezekhez tartozó helyzetének a felhasználásával ki lehetett számítani a megfelelő kormánybeállítás értékeit, vagyis kiegészítő kamerakép-kormánybeállítás párokat lehetett generálni. A virtuális mintákkal kibővített mintakészlet tette lehetővé az ALVINN sikerét.

A bizonyos transzformációkkal szembeni invariancia szintén lehetőséget biztosít a priori ismeret virtuális minták generálása által történő hasznosítására. Számos – elsősorban osztályozási – feladatnál a bemeneti minták bizonyos transzformációi mellett is változatlan választ kell kapnunk. Tipikus példa erre a karakterfelismerés és

általában a különböző képi alakzatok felismerése. Egy karaktert akkor is fel kell ismerni, ha pozícióban eltolt, kismértékben elforgatott, korlátozottan nagyított vagy kicsinyített változata kerül egy neurális osztályozó bemenetére. Hasonlóan, ha egy vonalas ábrán a vonalak vastagsága bizonyos korlátok között változik, az ábra besorolása a lehetséges osztályok egyikébe nem változhat. A transzformációkkal szembeni invariancia, mint a priori ismeret felhasználható virtuális minták generálására. A 7. fejezetben bemutatott kézzel írt karakterek felismerésénél az igen jó eredmény elérése döntően annak volt köszönhető, hogy az eredetileg 60.000 elemből álló tanító mintakészletet 540.000 elemre növelték a képek egy-pixeles függőleges, vízszintes és átlós eltolásaival.

A virtuális minták generálása az a priori tudás hasznosításának általánosan alkalmazható módja [Niy98], ugyanis bármilyen mintákból tanuló rendszerrel, bármilyen hálótípusnál használható. Hátránya, hogy a tanuló eljárást jelentősen lelassíthatja. Ez két okból is bekövetkezhet. Egyrészt a jóval nagyobb mintakészlet a tanító lépések számát megsokszorozhatja, másrészt a mesterségesen generált minták között nagyobb lehet a korreláció, ami szintén a tanítás lassulását eredményezi.

A nagy mintakészlet a szupport vektor gépeknél különösen megnehezítheti a háló konstrukcióját, hiszen az SVM számítási komplexitását alapvetően a kernel mátrix mérete határozza meg, és a kernel mátrix egy $P \times P$ méretű mátrix, ahol P a tanítópontok száma. A következőkben röviden bemutatott virtuális szupport vektor eljárás ugyan felhasználja az a priori ismeret alapján létrehozott virtuális mintákat, azonban ezt úgy teszi, hogy a jelentősen megnövelt mintakészlet ne növelje meg a számítási komplexitást.

A szupport vektor gépek tárgyalásánál láttuk, hogy az SVM megoldása a szupport vektorokon múlik. A szupport vektorok mindazt az információt hordozzák, ami az osztályozási vagy regressziós feladat megoldásához a teljes kiinduló mintakészletben megtalálható. Ez a felismerés felveti annak a lehetőségét, hogy virtuális mintákat csak a szupport vektorok felhasználásával generáljunk. Egy osztályozási feladatnál a szupport vektorok a szeparáló felülethez legközelebb lévő mintapontok. Intuitíve is az az érzésünk, hogy nem sok haszonnal jár, ha azon mintapontokból generálunk virtuális mintákat, melyek a döntési felülettől távol vannak. A felvetés jogosságát valamilyen módon, például kísérletileg igazolni kell. Az ötlet és a kísérleti igazolás is Bernard Schölkopf-tól és munkatársaitól származik [Sch96b]. A módszer alkalmazása az alábbi lépéseket igényli:

- Tanítsunk meg egy szupport vektor gépet abból a célból, hogy a tanítópontok közül megkapjuk a szupport vektorokat.
- Generáljunk mesterséges mintákat úgy, hogy a megfelelő invariancia transzformációkat a szupport vektorokra alkalmazzuk. Ezeket a mesterséges mintákat nevezzük virtuális szupport vektoroknak.
- Tanítsunk meg egy újabb szupport vektor gépet az előbbieken létrehozott virtuális szupport vektorokkal kibővített szupport vektor készlettel.

Az eljárás akár iteratív módon több ciklusban is alkalmazható, azonban ekkor ügyelni kell arra, hogy az esetlegesen csak lokálisan érvényes invariancia ne váljék globális invarianciává.

Az eljárást a 7. fejezetben bemutatott karakter-felismerési problémán tesztelték. A nagyméretű (60.000 tanítóminta) tanítókészlet miatt, továbbá mivel egy SVM osztályozó kétosztályos feladat megoldására képes, a tízosztályos feladatot kétosztályos feladatokként oldották meg. A virtuális szupport vektor módszert úgy alkalmazták, hogy minden szupport vektor alapján további 4, 8, illetve 12 virtuális szupport vektort állítottak elő [Sch97], [DeC02]. A téves osztályozás aránya a 10.000 elemű teszt mintakészleten a három esetben rendre 0,8%, 0,7%, illetve 0,6%-ra adódott (ez utóbbi a valaha is publikált legjobb érték), miközben a virtuális szupport vektorok száma mindhárom esetben kevesebb volt, mint a kiinduló tanítókészlet elemszáma. (Összehasonlításként utalunk a 8.1 táblázatban közölt legjobb értékre, ahol a téves osztályozás aránya 0,9% volt mintánként 8 virtuális mintát felhasználva. Ugyanakkor meg kell jegyezni, hogy a kétféle megközelítésben felhasznált minták számát nehéz összevetni, mivel a 7. fejezetben bemutatott megoldás az eredeti 10 osztályos feladatot egyben oldotta meg, míg a virtuális SV módszerrel a problémát különálló kétosztályos feladatokra bontották.)

2. Az a priori tudás beépítése a tanuló eljárásba

Az a priori információnak az előbbiektől eltérő felhasználási lehetősége, ha az információt közvetlenül beépítjük a tanuló eljárásba. A probléma általános felvetése Yaser Abu-Mostafától származik, aki a járulékos információnak a tanuló eljárásba való beépítését, mint lehetőséget bevezette [Abu90], [Abu95]. Az általa

megfogalmazott „utalásokból” történő tanulás (*learning from hints*) tulajdonképpen mindazon ismeretnek a tanuló eljárásba való beépítését jelenti, mely ismeret nem mintapontok formájában áll rendelkezésre. Ebben az értelmezésben – még ha csak közvetve is –, de az előbbieken tárgyalt virtuális mintagenerálást is e körbe tartozónak kell tekinteni.

Azt a megközelítést, hogy a priori információt közvetlenül beépítünk a tanuló eljárásba valójában már korábban is alkalmaztuk. A tanuló eljárásba bármilyen járulékos információ beépítése legkönnyebben a kritériumfüggvény módosításán keresztül lehetséges. Amennyiben az optimalizálandó kritériumfüggvény nemcsak egy valamilyen értelemben vett hiba (veszteség, kockázat) minimalizálását fogalmazza meg, hanem járulékos ismeretekből származó feltételeket is megszab, az optimalizáló eljárás eredményeképp a járulékos ismeretek is „beépülnek” a megoldásba. Az előzőekben már alkalmazott regularizációs eljárások valójában pont ezt a megközelítést alkalmazzák. A „valódi” kritériumfüggvényt kiegészítjük egy regularizációs taggal, ahol ez a tag a tanító mintákon túl rendelkezésre álló ismeret figyelembevételét teszi lehetővé. Simasági feltétel beépítésére pl. már több megoldást is láttunk. Valójában a szupport vektor gépeknél alkalmazott súlyvektorhossz minimalizálási feltétel is egy simasági feltétel. A CMAC hálónál alkalmazott súlysimító regularizáció (ld. 5.2 alfejezet) is abból a feltételezésből (vagy a priori információból) indul ki, hogy azt várjuk, hogy a háló válasza a tanítópontok között minél kevésbé legyen változékony, és hirtelen ugrások a mintapontok közötti tartományokban ne forduljanak elő. A sima megoldás preferálása, mint általános cél – miközben természetesen a megoldás döntően a mintapontok által szolgáltatott ismereten kell hogy alapuljon – szintén a priori ismereten alapul, nevezetesen azon ismereten, hogy a természetben inkább sima leképezések fordulnak elő.

2.1. Monoton válaszü kernel regresszió

A simasági feltételen túl számos további feltétel megfogalmazása alapul a priori információ [Lau06]. Ilyen feltétel lehet, hogy egy regressziós feladatnál a megoldás monotonitását előírjuk. A monoton LS-SVM regressziós megoldásra találunk példát [Pel04]-ben. A 6. fejezetben bemutatott LS-SVM optimalizálási problémához egydimenziós esetben még a következő feltétel megfogalmazása adódik:

$$\mathbf{w}^T \boldsymbol{\varphi}(x_i) \leq \mathbf{w}^T \boldsymbol{\varphi}(x_{i+1}), \quad \text{ha } x_i < x_{i+1} \quad i=1, \dots, P-1, \quad (12.1)$$

ahol x_i jelöli az SVM bemenetét, $\mathbf{w}^T \boldsymbol{\varphi}(x_i)$ pedig a megfelelő választ. A járulékos információ a tanító eljárásba úgy épül be, hogy a Lagrange kritérium módosul:

$$\begin{aligned} L(\mathbf{w}, b, \mathbf{e}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = & \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{2} \sum_{i=1}^p e_i^2 - \sum_{i=1}^p \alpha_i \{ \mathbf{w}^T \boldsymbol{\varphi}(x_i) + b + e_i - d_i \} \\ & - \sum_{i=1}^{p-1} \beta_i \{ \mathbf{w}^T \boldsymbol{\varphi}(x_{i+1}) - \mathbf{w}^T \boldsymbol{\varphi}(x_i) \} \end{aligned} \quad (12.2)$$

Látható, hogy a szokásos Lagrange együtthatós tag mellett egy újabb tag jelent meg, amely szintén Lagrange multiplikátoros formában veszi figyelembe a monotonitásra vonatkozó feltételeket, ahol a $\{\beta_i\}_{i=1}^{p-1}$ együtthatók az új Lagrange multiplikátorok.

Más a priori ismeretet tükröző feltételek hasonló módon fogalmazhatók bele a tanuló eljárásba (a kiinduló kritériumfüggvénybe). Az eljárás hátránya, hogy minden újabb formálisan megfogalmazott a priori ismeret a teljes optimalizálási feladat származtatását igényli, ami egyes esetekben komoly matematikai nehézségekkel járhat. Az így kapott eljárások hatékonysága tehát jelentős mértékben függ attól, hogy milyen matematikai nehézségek merülnek fel az elsődleges, illetve a másodlagos optimalizálási feladatok megoldásánál. A monotonitást előíró feltételek figyelembevétele például kvadratikusan optimalizálási feladatra vezet annak ellenére, hogy a standard LS-SVM-nél csak lineáris egyenletrendszerrel kell megoldanunk.

2.2. Tudásalapú szupport vektor gép

A tudásalapú szupport vektor gépeknél (*knowledge-based support vector machines*) az a priori tudás a bemeneti tér valamilyen korlátozott tartományára vonatkozik. Glenn Fung és munkatársai olyan osztályozós esetet vizsgáltak, amikor a bemeneti tér egyes poliéder tartományai teljes egészében az egyik osztályhoz tartoznak [Fun03]. Az osztályozási problémát ebben az esetben tehát egyszerűen az ismert besorolású tanítópontok, másrészt a poliéder tartományok (egyenesekkel, síkokkal illetve hipersíkokkal határolt tartományok) specifikálják. A poliéder tartományoknak, mint a priori információnak az osztályozásra gyakorolt hatását illusztrálja a 12.1 ábra.

A Fung és Mangasarian által javasolt SVM-ben a megszokott négyzetes hiba helyett egyes normájú hiba szerepel a célfeltételben, így a feladat megoldása (hasonlóan az LS-SVM-hez) nem igényel kvadratikus programozást, elegendő a létrejövő lineáris egyenlőtlenségrendszert lineáris programozással megoldani. Az említett poliéder tartományok ezt a lineáris egyenletrendszert egészítik ki extra egyenlőtlenségekkel.

Hasonló eljárást dolgoztak ki függvényapproximációs feladatokra is, ahol az a priori információ a függvényre vonatkozóan extra korlátokat jelent [Man04]. Az alábbiakban bemutatjuk a lineáris kernel modellt, ami az alapfeladatot leírja.

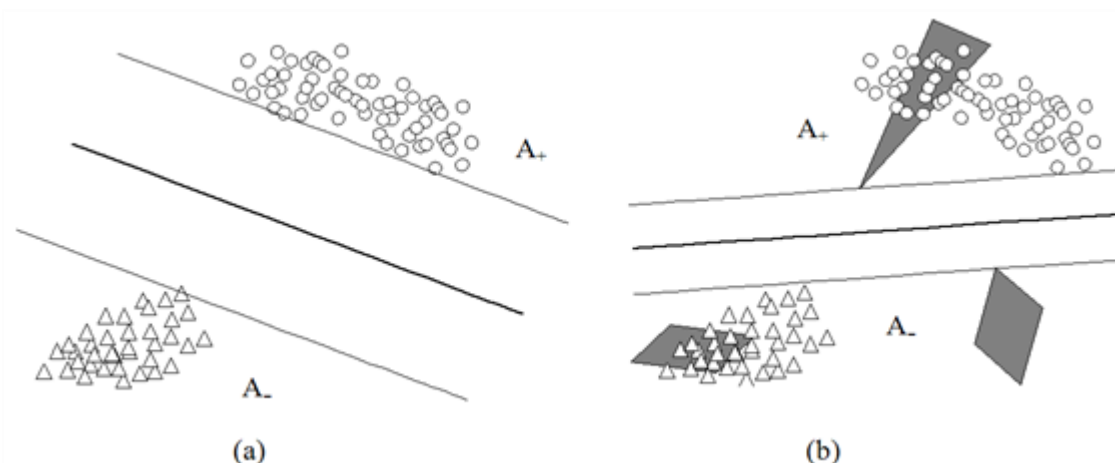
Mint az eddigi függvényapproximációs feladatoknál most is egy $f: \mathbb{R}^n \rightarrow \mathbb{R}$ függvényt szeretnénk közelíteni az

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b \quad (12.3)$$

lineáris kapcsolattal az $\{\mathbf{x}_i, d_i\}_{i=1}^p$ mintapontok felhasználásával. Jelölje a bemeneti mintavektorokat összefogó mátrixot

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_p^T \end{bmatrix}. \quad (12.4)$$

12.1. ábra - Egy kétszélyes osztályozási feladat. (a) az eredeti SVM-mel történő megoldás; (b) az a priori tudás (árnyékolt konvex sokszögek, mint az adott osztályhoz tartozó területek) figyelembevételével származtatott tudás alapú SVM megoldása



Ezzel a háló válasza az összes tanítópontra a következő formában is felírható:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}, \quad (12.5)$$

ahol $\mathbf{1}$ egy csupa egyesekből álló vektor. Feltételezve, hogy a háló válasza közelíti a kívánt válaszokat, az approximáció hibája közel nulla, vagyis:

$$\mathbf{X}\mathbf{w} + b\mathbf{1} - \mathbf{d} \approx \mathbf{0}, \quad (12.6)$$

ahol $\mathbf{d}$ a $d_i, i=1, \dots, p$ kívánt válaszokból képezett vektor. Ahogy azt a kernel gépeknél már láttuk, a súlyvektor $\mathbf{X}$ sorainak lineáris kombinációjaként is előállítható:

$$\mathbf{w} = \mathbf{X}^T \boldsymbol{\alpha}. \quad (12.7)$$

Itt $\boldsymbol{\alpha}$ a lineáris kombináció együtthatóinak vektora. (A szokásos Lagrange multiplikátoros felírásnál $\boldsymbol{\alpha}$ a Lagrange multiplikátorokból álló vektor.) Behelyettesítve a súlyvektor (12.7) kifejezését a (12.6) összefüggésbe, a lineáris kernel gépek szokásos formájú felírását kapjuk:

$$K(\mathbf{X}\mathbf{X}^T)\mathbf{a} + \mathbf{b}\mathbf{1} - \mathbf{d} \approx \mathbf{0} \quad (12.8)$$

ahol $K(\mathbf{X}, \mathbf{X}^T) = \mathbf{X}\mathbf{X}^T$ a lineáris kernel mátrix. A hibát az $\mathbf{e} \in \mathbb{R}^n$ vektorral elemenként mérjük, és ennek a hibának az 1-es normáját minimalizáljuk, ahol az 1-es norma jelentése:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i| \quad (12.9)$$

Az 1-es norma alkalmazásának következménye, hogy a megoldást a szupport vektor gépeknél (6. fejezet) látott kvadratikusan programozás helyett itt lineáris programozással megkaphatjuk. A megoldandó feladat a:

$$J(\mathbf{a}, \mathbf{b}, \mathbf{e}) = \|\mathbf{a}\|_1 + C\|\mathbf{e}\|_1 \quad (12.10)$$

kritériumfüggvény minimumát biztosító $\mathbf{a}$, $\mathbf{b}$ és $\mathbf{e}$ megtalálása, azzal a feltétellel, hogy teljesülnek az alábbi formában megfogalmazott egyenlőtlenségek:

$$-\mathbf{e} \leq K(\mathbf{X}, \mathbf{X}^T)\mathbf{a} + \mathbf{b}\mathbf{1} - \mathbf{d} \leq \mathbf{e} \quad (12.11)$$

A feltételes optimalizálási feladat az alábbi lineáris programozási feladatra vezethető vissza:

$$\begin{aligned} \min_{(\mathbf{a}, \mathbf{b}, \mathbf{e})} & \mathbf{1}^T \mathbf{a} + C\mathbf{1}^T \mathbf{e} \\ \text{sz.} & -\mathbf{e} \leq K(\mathbf{X}, \mathbf{X}^T)\mathbf{a} + \mathbf{b}\mathbf{1} - \mathbf{d} \leq \mathbf{e}, \\ & -\mathbf{a} \leq \mathbf{a} \leq \mathbf{a} \end{aligned} \quad (12.12)$$

Az eddigiek csupán azt illusztrálták, hogy egy kernel regressziós feladatnál a kritériumfüggvény a 6. fejezetben bemutatott kritériumfüggvényektől eltérő formában is megfogalmazható. A más megfogalmazást az indokolja, hogy így a kvadratikusan programozási feladat helyett a megoldás lineáris programozással megkapható. A lineáris programozás számítási komplexitásában egyértelműen előnyösebb, mint a kvadratikusan programozást igénylő „klasszikus” SVM.

A tudás alapú szupport vektor gépeknél járulékos információt is beépítünk az eljárásba. A fenti kiindulás mellett a lineáris egyenlőtlenségek formájában megfogalmazott a priori információ beépítése is lineáris programozási feladatra vezet. Az a priori információ itt azt jelenti, hogy a bemeneti tér egy poliéder tartományára további feltétel fogalmazható meg.

Ez a feltétel minden olyan $\mathbf{x} \in \mathbb{R}^M$ bemenetre vonatkozik, amely nem feltétlenül tartozik a tanítópontok közé, de melyre igaz, hogy egy

$$\mathbf{B}\mathbf{x} \leq \mathbf{g}, \quad (12.13)$$

által meghatározott nem üres poliéderen belül található, ahol $\mathbf{B} \in \mathbb{R}^{L \times M}$ valamilyen együttható készlet, $\mathbf{g} \in \mathbb{R}^L$ pedig valamilyen valós számokból álló vektor. A feltétel azt fogalmazza meg, hogy a poliéder tartományra az approximálandó függvénynek dominálnia kell egy adott lineáris függvényre, vagyis a járulékos információ az alábbi implikáció formájában fogalmazható meg:

$$\mathbf{B}\mathbf{x} \leq \mathbf{g} \Rightarrow \mathbf{w}^T \mathbf{x} + b \geq \mathbf{h}^T \mathbf{x} + \beta, \quad (12.14)$$

A lineáris függvény $\mathbf{h}$ és β paramétereit az a priori információ szolgáltatja. A súlyvektor (12.7) összefüggését felhasználva az implikáció

$$\mathbf{B}\mathbf{x} \leq \mathbf{g} \Rightarrow \mathbf{a}^T \mathbf{X}\mathbf{x} + b \geq \mathbf{h}^T \mathbf{x} + \beta, \quad (12.15)$$

formában is felírható. Ezt az implikációt kell még a (12.12) lineáris programozási feladathoz hozzávennünk.

Ennek érdekében az implikációt át kell alakítani lineáris egyenlőtlenségekké. Nem bizonyítjuk, de amennyiben a fenti implikációnak van megoldása, akkor az alábbi rendszer $\mathbf{u} \in \mathbb{R}^L$ megoldása ekvivalens az implikáció megoldásával:

$$\mathbf{B}^T \mathbf{u} + \mathbf{X}^T \mathbf{a} - \mathbf{h} = \mathbf{0}, \quad -\mathbf{g}^T \mathbf{u} + b - \beta \geq 0, \quad \mathbf{u} \geq \mathbf{0} \quad (12.16)$$

Az itt megfogalmazott feltételeket hozzávéve a (12.12)-ben megadott lineáris programozási feladathoz megkapjuk azt a lineáris programozási feladatot, mely a mintaponto α , b , e , a és u ulékos ismeretet is fig $u \geq 0$ nbe veszi. A minimalizálandó kritériumfüggvény, melynél a minimumot szerint keressük, ahol , az alábbi:

$$J(\alpha, b, e, a, u \geq 0) = \mathbf{1}^T \mathbf{a} + C \mathbf{1}^T \mathbf{e}, \quad (12.17)$$

a mellékfeltételek pedig:

$$\begin{aligned} -\mathbf{e} &\leq K(\mathbf{X}, \mathbf{X}^r) \alpha + \mathbf{b} \mathbf{1} - \mathbf{g} \leq \mathbf{e}, \\ -\mathbf{a} &\leq \alpha \leq \mathbf{a} \\ -\mathbf{a} &\leq \mathbf{B}^T \mathbf{u} + \mathbf{X}^T \alpha - \mathbf{h} \leq \mathbf{a} \\ -\mathbf{g}^T \mathbf{u} &\geq \beta - b \end{aligned} \quad (12.18)$$

A kapott megoldás meglehetősen speciális esetre érvényes, hiszen lineáris függvényapproximációs feladathoz indultunk ki és az a priori információ is a bemenet lineáris függvényeként van megfogalmazva. Az eljárás nemlineáris esetre történő általánosítása viszonylag természetes módon megtörténhet. Ekkor egyrészt a lineáris kernel helyett nemlineáris kernellel dolgozunk, másrészt az a priori információt is mint a bemenet nemlineáris függvényét fogalmazzuk meg. Ennek részleteire azonban itt nem térünk ki.

A priori ismeretek beépítésére az eddig bemutatott eljárásokon kívül számos további eljárást is kidolgoztak. Ilyen eljárásokat ismertet pl. [Yu04], [Sun05] és [Le06], valamint az a priori információ beépítésének egy további lehetőségére ad példát a következőkben bemutatott súlyozott margójú SVM is.

2.3. Súlyozott margójú szupport vektor gép

A súlyozott margójú szupport vektor gépeknél (*Weighted Margin Support Vector Machines*, *WMSVM*) [Wu04] minden tanító mintához a rendelkezésre álló a priori információ alapján egy konfidencia értéket is rendelünk. A konfidencia értékek hatása, hogy az egyes tanító minták különböző súllyal vesznek részt az SVM tanításában. A súlyozott szupport vektor gép működését lineárisan szeparálható esetre mutatjuk be, de értelemszerűen az eredmények megfelelő kernelfüggvény-választást követően érvényesek a nemlineárisan szeparálható esetre is.

Adott egy lineárisan szeparálható tanítóhalmaz $\{\mathbf{x}_i, d_i\}_{i=1}^P$, és minden mintához rendelkezésünkre áll egy $v_i \in [0, 1]$ konfidenciaérték, ami a d_i kívánt válasz konfidenciaszintjét jellemzi. A konfidenciához olyan intuitív értelmezés adható, hogy minél nagyobb egy minta konfidenciája, annál nagyobb margót szeretnénk biztosítani az adott mintánál. A súlyozott SVM és a standard SVM közötti különbséget illusztrálja a 12.2 ábra. Az ábrán látható kétosztályos osztályozási feladatnál különböző méretű körök és négyzetek jelölik a két osztályba tartozó mintapontokat, és a körök, illetve a négyzetek mérete képviseli a mintaponthoz rendelt konfidenciát. Az ábrán vékony folytonos vonal jelzi az SVM szeparáló egyenesét. Látható, hogy ez nem veszi figyelembe a mintapontokhoz rendelt konfidenciát, míg a vastag vonal olyan elválasztó felületet képvisel, mely a nagyobb konfidenciájú pontoknál nagyobb távolságot (nagyobb margót), a kisebb konfidenciájú pontoknál pedig kisebb távolságot biztosít.

A súlyozott margójú megoldás származtatása a standard SVM 6. fejezetben bemutatott származtatásához hasonlóan történik. Itt is azt a $\mathbf{w}$ súlyvektorral és b eltolás értékkel jellemezhető hipersíkot keressük, ahol $\|\mathbf{w}\|^2$ minimális, és ahol az osztályozásra vonatkozó

$$f(v_i) d_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad i = 1, \dots, P \quad (12.19)$$

feltételek teljesülnek. A (12.19) összefüggésben szereplő $f(\cdot) \in [0, 1]$ monoton csökkenő függvény biztosítja, hogy a „megbízhatóbb”, nagyobb konfidenciájú mintapontok nagyobb hatást gyakorolnak az elválasztó hipersíkra, ezeknél a pontoknál nagyobb margót kapunk.

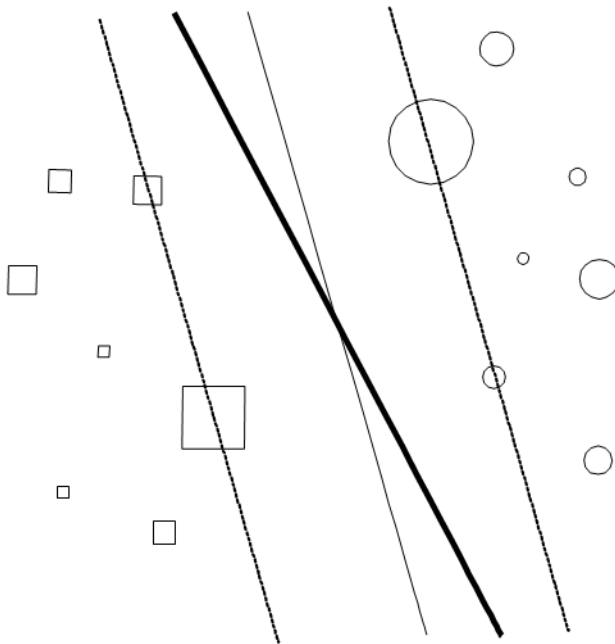
A részletektől eltekintve a megoldás súlyvektorára a következő adódik:

$$\mathbf{w}^* = \sum_{i=1}^P \alpha_i^* d_i f(v_i) \mathbf{x}_i \quad (12.20)$$

ahol az $\alpha_i^* \geq 0$ együtthatók most is a Lagrange multiplikátorok.

A standard SVM-hez hasonlóan itt is származtatható egy lágy margójú változat, ha egy gyengítő változót vezetünk be. Ekkor a megfelelő kritériumfüggvény a következő lesz

12.2. ábra - Az WMSVM működésének a standard SVM-től való eltérésre. A háló négyzetekkel illetve körökkel jelölt mintapontok osztályozását végzi, ahol a objektum mérete a konfidenciát jellemzi. A vékony vonal az SVM válasza, a vastag fekete vonal a WMSVM válasza.



$$J(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^P g(v_i) \xi_i, \quad (12.21)$$

Ennek $\mathbf{w}$ szerinti minimumát keressük azzal a feltétellel, hogy

$$f(v_i) d_i(\mathbf{w}^T \mathbf{x}_i + b^*) \geq 1 - \xi_i, \quad i = 1, 2, \dots, P. \quad (12.22)$$

ahol $f(\cdot) \in [0, 1]$ egy monoton csökkenő és $g(\cdot) \in [0, 1]$ egy monoton növekedő függvény és a $\xi_i \geq 0, i = 1, 2, \dots, P$ értékek a gyengítő változó értékei. A szokásos lépések eredményeképpen itt is megkapjuk a megoldás súlyvektorát:

$$\mathbf{w}^* = \sum_{i=1}^P \alpha_i^* d_i f(v_i) \mathbf{x}_i, \quad (12.23)$$

ahol teljesülnie kell még a következő összefüggéseknek:

$$\sum_{i=1}^P d_i \alpha_i f(v_i) = 0, \quad 0 \leq \alpha_i \leq g(v_i) C, \quad i = 1, 2, \dots, P. \quad (12.24)$$

2.3.1. A priori tudás beépítése WMSVM-be

A súlyozott margójú szupport vektor gépeknél a priori tudást egyrészt a konfidencia értékek meghatározásával, másrészt virtuális minták generálásával is beépíthetünk az eljárásba. A kiinduló mintákhoz tartozó konfidencia értékek például a mintákat terhelő zajtól függhetnek. A virtuális minták generálása itt nem invariancia transzformációk alkalmazását jelenti, hanem ismeretlen válaszu mintapontok beiktatását, ahol a kívánt válasz és annak konfidenciája származik az a priori információból. Konfidencia értéként annak a $P(d|\mathbf{x})$ feltételes

valószínűségnek a becslését használhatjuk, amely megadja, hogy egy adott, ismeretlen osztályba tartozó x bemenet milyen valószínűséggel tartozhat egy adott osztályba. A módszer jelentősége, hogy bizonytalan a priori tudást is be tud építeni. Ugyancsak a priori ismeret alapján választhatjuk meg az $f(\cdot)$ és $g(\cdot)$ függvényeket. Ezeknek a függvényeknek a hatása nagymértékben függ a v_i konfidencia értékektől. Mivel a legtöbb valószínűsági problémán a v_i értékekre csak durva becslést tudunk adni és semmilyen elmélet nem támasztja alá a komplexebb függvények használatának előnyét, ezért a legtöbb esetben a legegyszerűbb függvényeket, pl. az $f(v) = 1/v$ és a $g(v) = v$ függvényeket használhatjuk.

Példaként tekintsünk egy lehetséges alkalmazást, különböző dokumentumok kategorizálását. A kiinduló tanítóhalmaz a már kategorizált dokumentumok halmaza. A virtuális tanítóhalmaz kialakításához szükséges konfidencia értékeket a már kategorizált tanítóhalmaz alapján szakértői tudás felhasználásával határozhatjuk meg. A szakértők a tanítóhalmazba tartozó dokumentumok alapján olyan szavakat választanak ki, melyek szerintük az adott kategóriát jól jellemzik, vagyis egy kiválasztott szó nagy valószínűséggel fordul elő egy adott kategóriába tartozó dokumentumokban. Ezeket a szavakat tekintsük kulcsszavaknak. Például, ha a kategória a *kardiológia*, akkor a kulcsszavak a *vér*, *vérvénymérés*, *szív*, *véna*, *artéria* stb. A jelöletlen dokumentumok kategorizálása a kulcsszavak keresésével történik. A konfidencia értékeket többféleképpen is előállíthatjuk. Jelöljön x egy adott dokumentumot és c egy adott kategóriát. Az x mintának a c kategóriába tartozását jellemző konfidencia legyen egyszerűen $|x|_c / |x|$, ahol $|x|_c$ jelöli, hogy az x dokumentumban az adott kategóriát jellemző kulcsszavak közül hány fordul elő, $|x|$ pedig azt adja meg, hogy az adott kategóriát hány kulcsszóval jellemezzük (A jelölésnél a k_w index a kulcsszóra (*keyword*) utal.) Azokat a dokumentumokat, melyek egyetlen kulcsszót sem tartalmaznak, figyelmen kívül hagyjuk.

Az SVM konstrukciója ezek után a két tanítóhalmaz együttes felhasználásával történik. Amennyiben az eredeti mintahalmaz pontjaihoz is rendeltünk konfidencia értékeket, a két mintahalmaz eltérő kezelésére nincs szükség. Ha azonban az eredeti tanítóhalmaz mintáit zajmentesnek, pontosnak tekinthetjük, akkor célszerű, ha ezen mintákhoz nem rendelünk konfidencia értékeket (pontosabban ezekhez egységnyi konfidencia értékeket rendelünk), vagyis a két mintahalmaz pontjait a kritériumfüggvényben különbözőképpen vesszük figyelembe. Amennyiben az első P_1 elemű mintahalmaz az eredeti ismert kategóriájú tanítómintákból áll és a maradék $P - P_1$ minta képezi a virtuális mintahalmazt, akkor a következő módosított kritériumfüggvényt célszerű használni:

$$J(w) = \frac{1}{2} w^T w + C \sum_{i=1}^P \xi_i + \eta C \sum_{i=P_1+1}^P g(v_i) \xi_i \quad (12.25)$$

Itt C az SVM-nél megszokott paraméter, amely a biztonsági sáv mérete és a sávon belülre eső pontok száma közötti egyensúlyért felelős, míg az η paraméter a két tanítóhalmaz relatív fontosságának beállítására szolgál. Értelemeszerűen, ha az eredetileg ismert kategóriájú dokumentumok száma kicsi, akkor nagy η -t érdemes választani. Ahogy a valóban kategorizált dokumentumok száma növekszik, úgy érdemes a virtuális tanítóhalmaz hatását csökkenteni, hiszen az eredeti tanítóhalmaz „jobb minőségű”.

3. KBANN, a tudás alapú neurális hálózat

A tudás alapú neurális háló (*Knowledge Based Artificial Neural Network*, *KBANN*) egy lehetséges megoldást ad arra, hogy hogyan ötvözhető a magyarázat alapú és a mintapontok alapján tanuló rendszerek előnyei [Tow91]. A KBANN rövid bemutatása előtt összefoglaljuk a kétféle ismereten alapuló rendszerek legfontosabb előnyeit és hátrányait.

Míg a neuronhálók minták formájában rendelkezésre álló ismeretet tudnak feldolgozni, a magyarázat alapú rendszerek egy adott tématerületre – a tárgyterületre (*domain*) vonatkozó – szimbolikus szabályhalmazzal fogalmazzák meg az adott problémáról rendelkezésre álló ismereteket. Például a neuronhálókval ellentétben egy élőlény madár voltát nem rengeteg példán keresztül határozzák meg, hanem egy előre meghatározott szabályrendszer – mint tollas-e, van-e szárnya stb. – alapján. A szabályrendszert – amit szakértők állítanak fel – a tanuló rendszerek irodalmán belül szokás a tárgyterületre vonatkozó elméletnek, domén elméletnek (*domain theory*) is nevezni. A továbbiakban mi is ezt az elnevezést fogjuk használni.

A minták alapján, induktív tanulással (empirikus tanulás, ET) létrehozott és a magyarázat alapú rendszerek (MA) alapvetően eltérő tudásrepresentációt alkalmaznak. Mindkét megközelítésnek megvannak az előnyei és hátrányai, melyeket a 12.1. táblázatban foglalunk össze. Az előnyös tulajdonságokat (+), a hátrányosakat (-) jellel jelöltük. Ha összevetjük a két tudásrepresentáció problémáit jól látható, hogy a két megközelítés kiegészíti

egymást, azaz a kombinációjukból adódó hibrid-neurális rendszerben az egyik reprezentáció erősségeivel felülkerekedhetünk a másik gyengeségein. A minták formájában meglévő tapasztalati tudás és a szabályok formájában rendelkezésre álló domén elmélet együttes felhasználására alkalmasak a tudás alapú neuronhálók (KBANN). A KBANN alapvető megközelítése, hogy a domén ismeretet a kiinduló neuronháló konstrukciójánál használja fel: a feladatról rendelkezésünkre álló szabálykészlet alapján olyan háló architektúrát és kiinduló hálópárparamétereket (súlyokat) határoz meg, hogy a kiinduló háló leképezése teljes mértékben megegyezzen a bemenetek és kimenetek között szabályok formájában megfogalmazott kapcsolattal.

12.1. táblázat - A különböző tudásreprezentációs tanuló rendszerek egymást kiegészítő erősségei (+) és gyengeségei (-)

Magyarázat alapú rendszerek (MA)	Mintákból történő tanuló rendszerek (ET)
(-) teljes és korrekt domén elméletet tételez fel (bizonyos feladatokra a domén ismeret megfogalmazása nemcsak hogy nehéz, de egyenesen lehetetlen)	(+) az ET rendszer nem vagy alig igényel elméleti tudást
(-) valójában nem tanul, lényegében csak átfogalmaz, ezért a kiinduló elmélet hibáit nem képes kijavítani	(+) az ET rendszer képes új ismereteket befogadni, képes tanulni
(-) a domén elméletekre nem jellemző a "fokozatos romlás" : ha a tudáshatárt átlépi a feladat, a megoldóképesség drasztikusan romlik, emellett a tudáshatár átlépésének detektálása is problematikus	(+) az ET rendszer "fokozatosan romlik"
(-) a domén elmélet túl terjedelmes/bonyolult is lehet (idő/memória korlátok)	(+) egy megtanított ET rendszer gyors működésű
(-) a domén elméletet valakinek meg kell fogalmaznia	(+) az ET rendszer kevés domén ismeretet igényel
(+) az MA nem használja a példák közötti korrelációt	(-) a példákban található álkorrelációk helytelen osztályozáshoz vezethetnek (pl. ha a példahalmazban minden madár fekete tollazatú, a rendszer kikövetkeztetheti, hogy minden madárnak feketének kell lennie)
(+) a domén tudásban tetszőleges mennyiségű alhalmazt/altudást tudunk reprezentálni (speciális, kivételes esetek kezelése)	(-) sok pozitív példa mellett is elképzelhető, hogy a kivételes eseteket nem, vagy nem kellő mértékben reprezentálják a példák
(+) a kontextus szerepelhet a	(-) az osztályozás szempontjából

szabályrendszerben	lényeges vonások környezetfüggők lehetnek, a kontextust valahogy reprezentálni kell
(+) a releváns tulajdonságokat a domén tudás specifikálja	(-) bármelyik objektum elvileg végtelen sok különféle attribútummal írható le, a lényegi vonások kiszűréséhez tudás kell
(+) a bonyolultabb jellemzők a végállapot és kezdeti szabályok közötti "köztes következtetések"	(-) a kezdeti primitív tulajdonságokból bonyolultabb jellemzők kinyerése segítené a tanulást, ez a kinyerés azonban nem automatizált és garantált

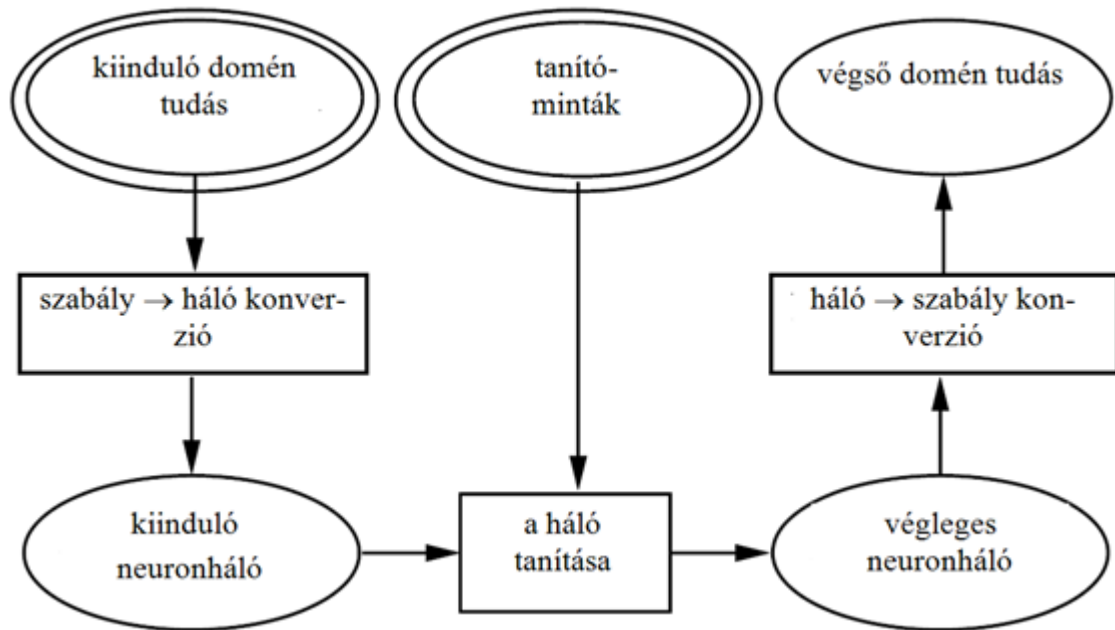
A szabályok ugyanakkor nem szolgálnak teljes ismerettel a megoldandó feladatról, további ismeret mintapontok formájában áll rendelkezésünkre. Ezt az ismeretet használjuk fel a kiinduló háló finomítására, paramétereinek pontosabb beállítására, sőt esetleg a kiinduló háló struktúrájának a megváltoztatására is. A KBANN nemcsak arra képes, hogy a kétféle reprezentációjú tudást a háló kialakításánál hasznosítsa, hanem arra is, hogy a megtanított hálóból a kiinduló szabályokhoz hasonló szabályok formájában szimbolikus ismeretet is kinyerjünk.

3.1. Apriori tudás használata a kezdeti hipotézis kialakításában: KBANN

A KBANN konstrukciója három fázisból áll (ld. 12.3. ábra). Az első fázis a kezdeti domén elmélet alapján egy kiinduló neuronháló – MLP – létrehozása. Ez a fázis a szimbolikus szabályokat egy neurális hálózatba konvertálja. Az így keletkezett hálózat garantáltan ugyanazt a választ adja, mint amit a kiinduló szabályok adnának.

A második fázisban a létrehozott neuronhálót tanítóminták segítségével tanítjuk. A tanítás módja alapvetően megegyezik az MLP tanításánál bemutatott eljárással, tehát itt is a hibavisszaterjesztéses algoritmust alkalmazzuk. Ugyan a tanítási algoritmus a megszokott, ezt azonban nem egy szokványos hálón végezzük, hanem a szabályokból nyert, a domén tudásból létrehozott speciális struktúrájú hálón, és ez a tanításra bizonyos korlátokat ad (ld. bővebben a 12.3.4 alfejezet).

12.3. ábra - Az információ terjedése KBANN-ban



A végső, harmadik fázis során történik meg a módosított neuronhálóból a módosított szabályrendszer kinyerése. A kapott elméleti tudás már emberi elemzésre is alkalmas, megoldva ezzel a neurális háló egyik legnagyobb hiányosságát. A kapott finomított domén tudást akár kiinduló tudásként is alkalmazhatjuk egy következő KBANN-ben, így az információ visszaáramoltatással egy iterációs alkalmazáshoz is juthatunk.

Az ábrán az ellipszisek adatstruktúrákat, a négyszögek algoritmusokat jelentenek. A dupla ellipszisek a kiinduló ismereteket jelölik. A KBANN tehát kétféle kiinduló ismeretből építkezik: egyrészt a kezdeti domén tudásból, másrészt tanító mintapontokból. Az alábbiakban röviden bemutatjuk a KBANN alapváltozatának a konstrukciójához szükséges lépéseket. Az alapváltozaton kívül a KBANN-nek számos továbbfejlesztett változatát dolgozták ki.

3.2. Az elméleti tudás neurális hálózatba történő konvertálása

A KBANN szimbolikus szabályokból indul ki, a szabályokat egy többrétegű perceptron neuronjaira képezi le. A szabály $\rightarrow$ neuronháló konverzió a szabályokra nézve meglehetősen szigorú korlátokat állít fel. A KBANN csak Horn klózként megfogalmazott szabályokból tud hálót építeni. A Horn klóz az ítéletkalkulus fontos fogalma. Egy Horn klóz literálok diszjunkciója, ahol a literálok közül legfeljebb egy lehet pozitív. Egy literál egy atom vagy annak negáltja, ahol az atom az ítéletkalkulus osztatlan szintaktikai eleme. (Részletesebben ld. [Rus05]).

A KBANN által felhasználható szabályok a Boole-algebra nyelvén is megfogalmazhatók. E szerint egy szabály olyan logikai kifejezés, amely bemeneti logikai változók konjunktív és diszjunktív kapcsolataiból áll. A logikai kifejezés bemenetére kerülő logikai változók képezik a szabályok feltétel részét, a logikai kifejezés kimenete pedig a szabály következmény részét. A kiinduló szabályok olyan logikai kifejezéseknek felelnek meg, ahol a kimeneti logikai változó a bemeneti változók ÉS kapcsolatainak VAGY kapcsolataként írható fel (ezt szokás konjunktív normál alaknak (*conjunctive normal form*) is nevezni). A bemeneti logikai változók ponált vagy negált értékkel szerepelhetnek. A KBANN kiinduló szabályainál korlátozás még, hogy a szabályok csak aciklikus szabályok lehetnek (azaz nincs olyan szabály, vagy szabály kombináció, ahol egy szabály következménye a feltételben is szerepel). Az aciklikusság követelménye azonban nem zárja ki a hierarchikus szabályok alkalmazhatóságát, amikor egy szabály következményrész egy következő szabály feltételrészeként jelenik meg. A hierarchikus felépítésű szabályrendszerből kiindulva a konverzió rejtett neuronokat is tartalmazó többrétegű hálót eredményez.

A KBANN konstrukciója a következő 7 fő lépésből áll.

1. Szabályok átírása

A fentiekben definiált szabályok közvetlenül nem minden esetben konvertálhatók neuronokká. A konverzió úgy biztosítható, ha előbb a szabályokat olyan formába írjuk át, hogy a diszjunkciónál a feltételrészben csak egyetlen logikai változó szerepeljen. Ez az átírás lehetővé teszi a szabályok egyértelmű konvertálását neurális

háló elemekre (neuronokra) és egyben tisztázza is a hierarchikus struktúrát. Tekintsük a következő példát: legyen a kiinduló szabályrendszerünk:

$A :- B, C, D$

$A :- D, E, F, G$

ahol A,B,C,D,E,F és G mind logikai változók. A fenti szabályrendszer Boole-algebrai megfelelő felírása:

$A = BCD + DEFG$

vagyis az A kimeneti logikai változó akkor vesz fel IGAZ értéket, ha VAGY a B,C és D bemeneti logikai változók mindegyike IGAZ értéket vesz fel (ÉS kapcsolat) VAGY D,E,F és G mindegyike vesz fel IGAZ értéket.

A szabály $\rightarrow$ háló konverzióhoz a következő átírássra van szükség:

$A :- A'$

$A' :- B, C, D$

$A :- A''$

$A'' :- D, E, F, G$

vagy Boole-algebrai felírással

$A = A' + A''$

$A' = BCD;$

$A'' = DEFG$

Az átírás azért szükséges, mert másképpen nem tudjuk a szabályból származtatott neuronok eltolásértékeit (*bias*), úgy beállítani, hogy a konverzió eredményeképpen létrejött háló megfeleljen a szabály logikai működésének.

2. Szabálystruktúra leképezése neurális hálóvá

A második lépésben az előzőekben már átírt szabályok mindegyikét egy-egy neuronra képezzük le. A neurális háló bemenetei a szabályok feltétel részének, az előzményeknek, míg a kimenetei a végső következményeknek felelnek meg. A fentiekből már jól látszik, hogy a neurális háló rejtett elemei a szabályrendszer közbülső következményeit reprezentálják.

A szabályok nemcsak az egyes neuronok felépítését (bemenetek száma) határozzák meg, hanem a neuronok súlyait és az eltolásértékét is. A súlyok megválasztása – amit a későbbiekben ismertetünk – a konvertálandó szabály típusától függ. A kapott hálózat már megfelel a kezdeti domén elméletnek: a kimenete csak akkor aktív, amikor a szabályrendszer alapján is igaz az állítás.

3. Címkézés

Címkézést hajtunk végre, melynek eredményét a későbbi lépések hasznosítják. A címkézés során minden rejtett és kimeneti neuronhoz egy, a neuron rétegének megfelelő számot (címkét) rendelünk úgy, hogy a bemenettől azonos „rétegtávolságra” lévő neuronok azonos számot kapnak. (A bemenetek a 0. réteget képviselik.) A címkézés végrehajtásához szükséges, hogy minden érvelési pálya teljes legyen, azaz minden közbülső konklúzió egy olyan pályán legyen, amely egy vagy több bemeneti egységből egy vagy több kimeneti egységhez vezet. Nem teljes pályák esetén ezeket a szabály $\rightarrow$ neuronháló konverter kiegészíti igen kis súlyú kapcsolatok bevezetésével.

4. Rejtett egységek hozzáadása a felhasználó által specifikált szinteken

Ahhoz, hogy a neurális hálónk a tanulás során az eredeti domén tudáson túlmutató új kapcsolatokat tudjon megtanulni, szükség lehet rejtett egységek hozzáadására. Mivel a hozzáadott rejtett egységekre nem mindig van szükség (a kezdeti információkészlet elegendő), ezért ez a lépés opcionális.

5. Bemeneti egységek hozzáadása

Ha elfogadjuk, hogy a domén elmélet csak közelítőleg helyes, vagyis lehetnek olyan bemenetek, melyek a kimenetekre hatnak, de melyek nem szerepelnek a szabályokban feltételként, akkor a szabályokban nem szereplő bemeneteket, mint új hálóbemeneteket a kiinduló háléhoz hozzá kell adni.

6. Szabályokban nem specifikált kapcsolatok hozzáadása szomszédos szinteken

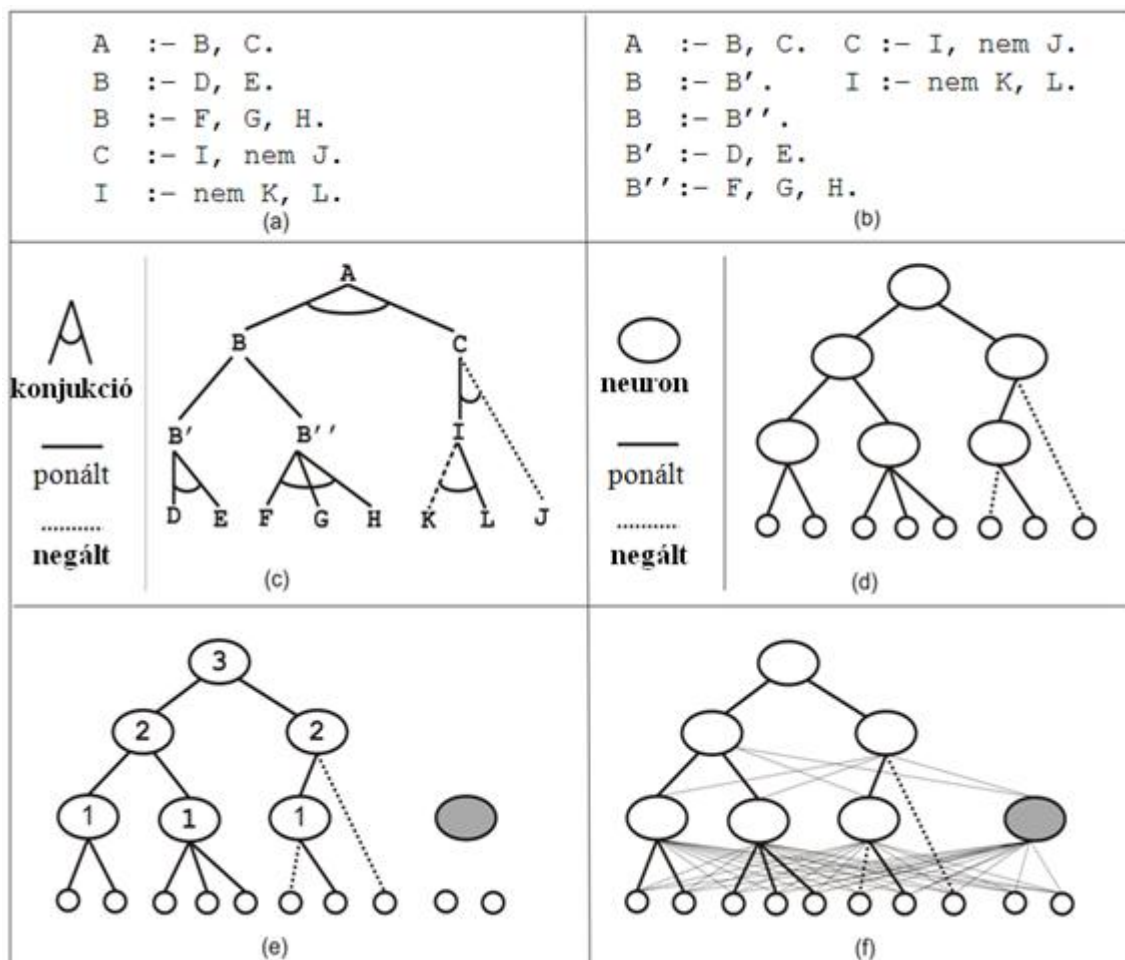
Ebben a lépésben a címkézés eredményének felhasználásával a hálóba újabb éleket (kapcsolatokat) iktatunk, melyekhez nulla kezdeti súlyértékeket rendelünk. A kapcsolat beiktatása lehetővé teszi, hogy a későbbi, minták alapján történő tanítás során ott valóban érdemi kapcsolat alakuljon ki, a nulla kezdeti érték ugyanakkor azt biztosítja, hogy a beiktatást követően a háló által megvalósított leképezés az eredeti szabályrendszer által meghatározott leképezéshez képest ne változzon.

Az új élek beiktatása az alábbi elv szerint történik. Kössünk össze egy-egy éllel minden $(n-1)$ címkéjű egységet (bemenetet vagy neuront) minden (n) címkéjű egységgel. Az összeköttetések beiktatásánál a 4. és az 5. lépésben beiktatott bemeneteket illetve rejtett neuronokat is vegyük figyelembe.

7. Súlyok perturbálása.

Módosítsuk a háló összes súlyát úgy, hogy mindegyikhez hozzáadunk egy kisértékű véletlen számot. A kismértékű módosítás nem változtatja meg érdemben a háló leképezését, de elegendő ahhoz, hogy a háló későbbi tanításánál bizonyos szimmetria-kialakulásból adódó problémákat elkerüljünk.

12.4. ábra - Mintapéllda KBANN kialakítására



12.1 példa

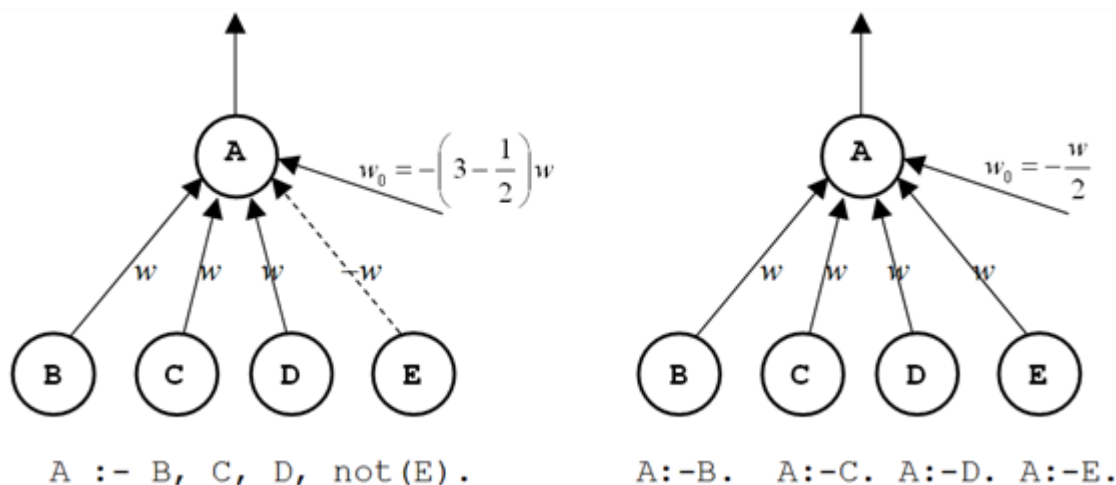
A 12.4 ábrán bemutatott példán végigkövethetjük a KBANN konstrukciójának lépéseit.

Az ábra (a) és (b) ablaka a kezdeti domén elméletet megadó szabályokat mutatja az eredeti (a) és a konstrukció által megkövetelt átírt (b) formában. A (c) és (d) ablak illusztrálja a szabályok neuronhálóba való átültetését. Látható, hogy egy olyan többretegű hálóarchitektúrát kapunk, amely nem teljesen összekötött, hiszen a bemenetek és a neuronok, illetve az egyes neuronok közötti összeköttetéseket a kiinduló szabályok határozzák meg. Az (e) ablakban a struktúrabővítés látható, itt egy rejtett egységet és két bemeneti egységet adunk a hálózathoz. Ezen az ábrán láthatók a címkézések is, ahol számozás a bemenettől indul és a címkék a bemenettől számított leghosszabb út hosszúságát reprezentálják. Az (f) ablakban a nulla súlyú extra élek felvétele látható. Figyeljük meg, hogy a később hozzáadott egységeknek a hálózat többi eleméhez való hozzákapcsolása szintén megtörténik.

3.3. Szabály → neuron konverzió

A szabályfordítás egyszerűsítéséért feltesszük, hogy csak bináris értékű bemenetekkel kell számolnunk. Mivel a többértékű jellemzők átalakíthatók bináris jellemzők logikai kapcsolatává, ezért a fenti egyszerűsítés nem jelent korlátozást. Könnyen belátható, hogy a konjunktív és a diszjunktív szabályok a 12.5 (a) illetve (b) ábrán látható módon konvertálhatók neuronokká.

12.5. ábra - Szabálykonverzió. Konjunktív (a) illetve diszjunktív (b) szabály leképezése



Az ábra olyan neuronokat mutat, melyek a szokásos módon a bemenetek súlyozott összegét képezik, majd az összegre egy szigmoid típusú nemlinearitást alkalmaznak. A konverzió során a pozitív bemeneteket (pozitív feltételek) azonos $w > 0$ súlyokkal, a negált bemeneteket (negált feltételek) pedig $-w$ súlyokkal vesszük figyelembe. Ha a feltételezésünknek megfelelően bináris bemenetekkel dolgozunk (vagy legalább azt biztosítjuk, hogy a bemenetek közel 0 vagy közel 1 értéket vegyenek fel) és a kimeneteken is közel bináris értékeket szeretnénk kapni, akkor logisztikus szigmoid függvényt kell alkalmaznunk. A w súly értékét – megfelelő feltételek betartása esetén – viszonylag egyszerűen, elemi számítások felhasználásával határozhatjuk meg. A részletek bemutatásától eltekintünk csak a feltételeket és az eredményt adjuk meg.

Tételezzük fel, hogy az egyes bemenetek és kimenetek vagy aktív vagy inaktív állapotban vannak. Aktív állapotról akkor beszélünk ha a megfelelő érték a $[C_*, 1]$ intervallumban, inaktív állapotról pedig, ha a $[0, C_*]$ intervallumban van, ahol C_* az a minimális aktiváció, ami mellett egy egység még aktívnek és C_* az a maximális aktiváció, ami mellett az egység még inaktívnek tekinthető. Alkalmazzuk a következő logisztikus

szigmoid függvényt: $\text{sgm}(x) = \frac{1}{1 + e^{-x}}$, ahol σ olyan konstans, melynek értéke a szigmoid függvény $x=0$ -hoz tartozó meredekségét határozza meg, és jelölje K_p a szabály pozitív feltételeinek, K_n a negált feltételeinek, $K = K_p + K_n$ pedig az összes feltételnek a számát. Megmutatható, hogy ha ezeket a feltételeket betartjuk és a $C_* = 1 - C_*$, valamint a $K C_*, < 1/2$ feltételek is teljesülnek, akkor a súlyértékre a következő korlátokat kapjuk:

$$w \geq \frac{\ln\left(\frac{1}{C_*} - 1\right)}{K C_* - \frac{1}{2}} \quad (12.26)$$

és

$$w \geq \frac{\ln\left(\frac{1}{C_i} - 1\right)}{C_i - \frac{1}{2}} \quad (12.27)$$

Ha w értékét a (12.26) és a (12.27) összefüggéseknek megfelelően választjuk meg, a konjunktív szabályok konvertálásánál

$$w_0 = -\frac{K_0 - 1}{2} w, \quad (12.28)$$

a diszjunktív szabályok konvertálásánál pedig

$$w_0 = -\frac{w}{2} \quad (12.29)$$

eltolásértéket (*bias*) kell alkalmaznunk.

3.4. A KBANN háló finomítása

A KBANN háló tanítása hibavisszaterjesztéses eljárással történik. Azonban a speciális hálókészítés és kezdeti súlybeállítás miatt a KBANN esetében célszerű a standard hibavisszaterjesztéses eljárást kismértékű módosítása.

A KBANN háló tanításánál a speciális helyzetet az idézi elő, hogy a neuronok bemenetei és kimenetei 0-hoz vagy 1-höz közeli értékeket vesznek fel. Egy szigmoid nemlinearitással felépülő neuron kimenetén akkor kapunk közel 0 vagy 1 értéket, ha a szigmoid telítési szakaszán vagyunk. A telítési szakaszon viszont a szigmoid függvény deriváltja közel 0, így a súlymódosítás, amely a hiba és a szigmoid deriváltjának a szorzatával arányos, még akkor is nagyon kicsivé válhat, ha egyébként a kimeneten a hiba nagy. A szigmoid aktivációs függvénnyel dolgozó MLP-knél ez a jelenség természetesen mindig felléphet, de komolyabb következménye csak akkor van, ha a telítési szakaszon való tartózkodás a háló „normál” működésének a jellemzője. A jelenség következménye, hogy a háló nem vagy nagyon lassan tanul, annak ellenére, hogy a bináris kimeneten épp a kívánt válasszal ellentétes választ kapunk (0 helyett 1-et vagy fordítva), tehát a hiba nagy. A KBANN esetében ez a szituáció már a kiinduló hálónál is előfordulhat, így a háló súlyain a hagyományos hibavisszaterjesztéses algoritmus már keveset változtat, függetlenül attól, hogy a háló működése a tanító minták alapján helyes vagy sem. Ezért a KBANN esetében általában a hibavisszaterjesztéses algoritmus Hinton-féle módosított változatát használjuk [Hin89]. Ebben a hagyományos négyzetes hibafüggvény helyett a keresztentropia hibafüggvényt használjuk:

$$C(\mathbf{w}) = -\sum_{i=1}^N [d_i \ln(y_i) - (1 - d_i) \ln(1 - y_i)] \quad (12.30)$$

ahol

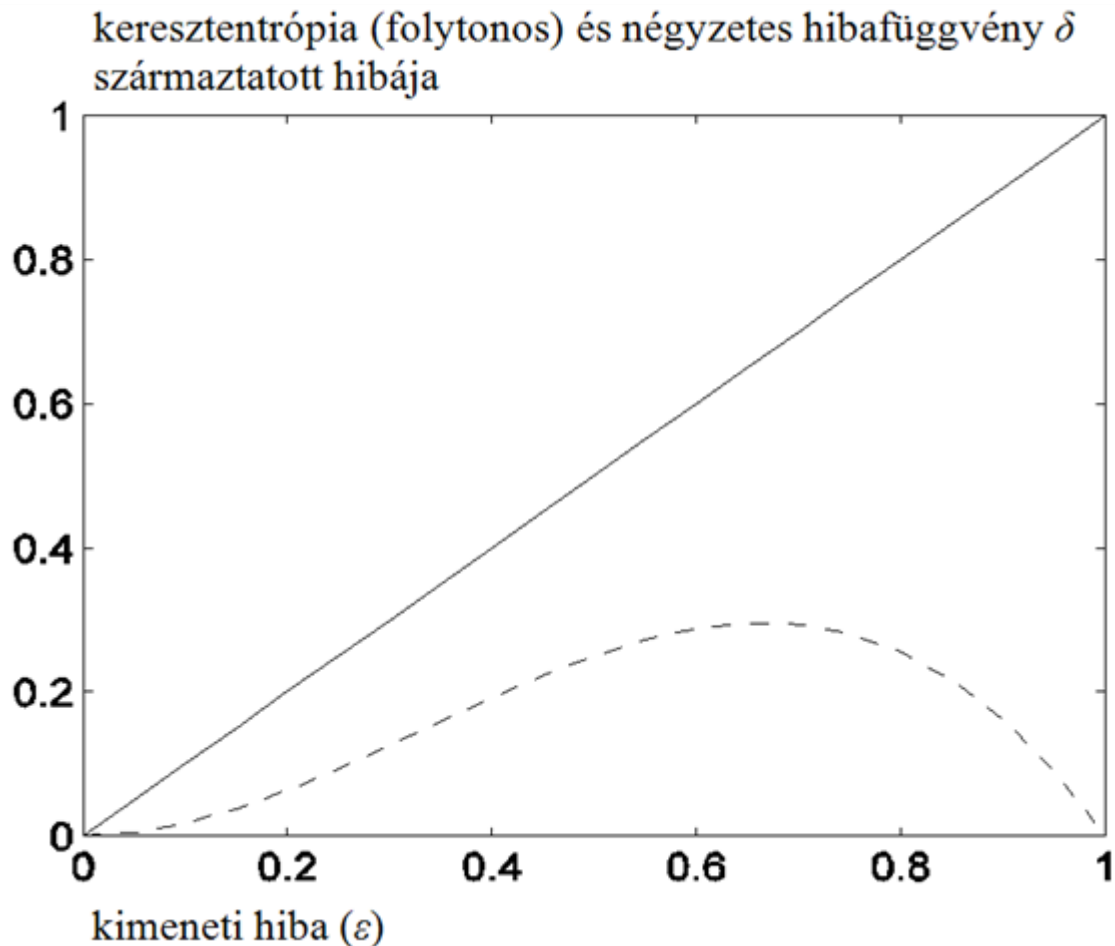
d_i az i -edik neuron számára előírt kívánt válasz

$y_i = y_i(\mathbf{w})$ a neuron kimeneti értéke

A keresztentropia hibafüggvény alkalmazása azt eredményezi, hogy a hibavisszaterjesztéses algoritmusban a gradiens számításból származó $\delta = \text{sgm}'(\cdot)$ értékek (ld. az 4. fejezetben a (4.6), (4.12) és (4.14) összefüggéseket) akkor sem adnak közel nulla értéket, ha a szigmoid függvény telítési szakaszán vagyunk. Ez jól látható a 12.6 ábrán. Az ábra bináris kívánt választ feltételezve δ alakulását mutatja a kimeneti hiba függvényében. A szaggatott vonal a szokásos négyzetes hibafüggvényre, a folytonos vonal a keresztentropia hibafüggvényre érvényes δ -t mutatja. Ezt az eredményt könnyen származtathatjuk, ha a (12.30) összefüggéssel megadott kritériumfüggvény alapján meghatározzuk a hibavisszaterjesztéses algoritmus megfelelő δ -ira vonatkozó összefüggéseket.

A KBANN tanításánál a standard hibavisszaterjesztési eljáráshoz képest egy további módosítás alkalmazása célszerű. Mivel a tanuló eljárás során a háló kiinduló súlyai megváltoznak, a háló által reprezentált szabályok is változni fognak. Ha a kiinduló szabályrendszer kellően megbízható, azt szeretnénk, ha az eredeti szabályok a tanítás után is érvényesek maradnának, a tanítás következtében pedig esetlegesen további szabályok lennének megfogalmazhatók. A kiinduló szabályok csak akkor maradnak érvényben, ha a kiinduló hálóarchitektúra rész súlyai nem módosulnak jelentős mértékben.

12.6. ábra - Kimeneti neuron származtatott hibája keresztentrópia (folytonos) és négyzetes (szaggatott) hibafüggvény mellett



Ennek biztosítására a kritériumfüggvényhez egy regularizációs tagot vehetünk hozzá, amelynek szerepe a kiinduló súlyok jelentős módosításának megnehezítése. A regularizációs tag egy olyan büntető tag, mely az eredeti súlyok (w_{init}) megváltozását bünteti.

$$\mathcal{Q}(\mathbf{w}) = \sum_i \frac{(w_i - w_{init,i})^2}{1 + (w_i - w_{init,i})^2} \quad (12.31)$$

Ekkor a kritériumfüggvényünk a következő lesz:

$$C_r(\mathbf{w}) = C(\mathbf{w}) + \lambda \mathcal{Q}(\mathbf{w}) \quad (12.32)$$

ahol $C(\mathbf{w})$ a (12.30) összefüggéssel megadott eredeti kritériumfüggvény, λ pedig a regularizációs együttható.

3.5. Szabályok kinyerése – finomított domén elmélet

A szabályok kinyerése fázis arra szolgál, hogy a most már mintákkal is tanított hálóban „meglévő tudást” szabályok formájában is meg tudjuk fogalmazni. Ez a fázis teszi lehetővé, hogy a háló válaszához magyarázatot is tudunk rendelni. A szabálykinyeréssel visszatérünk a kiinduló tudás reprezentációra azzal a különbséggel,

hogy kezdetben a szabályok egy kiinduló domén ismeretet tükröztek, az eljárás végén kinyert szabályok viszont már a tanítómintákból származó ismeretet is tartalmazzák.

Az alább részletezett szabálykinyerő módszer két fő feltételezéssel él. Egyrészt feltesszük, hogy a KBANN tanítása nincs nagy befolyással az egységek szimbolikus jelentésére. Tanított hálókön végzett vizsgálatok azt mutatják, hogy a jelentések általában stabilak. Ha mégis történik jelentésváltozás, akkor az „elmozdulás” a legkevésbé biztos szabályoknál figyelhető meg. Másrészt feltesszük, hogy a megtanított KBANN-nél majdnem

minden egység vagy aktív, vagy inaktív (közel 1 vagy 0). Ez a feltevés megvalósítható, hiszen a standard $\frac{1}{1+e^{-\sigma x}}$ logisztikus aktivációs függvény paramétereinek változtatásával egységugrás függvényt tudunk közelíteni (minél nagyobb σ értéke, annál meredekebben vált át a szigmoid függvény az egyik telítési tartományból a másik telítési tartományba).

A két feltevés következménye, hogy egy szabályt ki lehet nyerni csupán a súlyok értékei alapján, megvizsgálva, milyen súlykombináció mellett igaz a szabály. Azaz a szabálykinyerő módszerek általános elve olyan súlycsoportok keresése, melyeknek az összege meghaladja az eltolás (*bias*) értékét. (Az eltolásértéket itt szokás küszöbértéknek is nevezni, ami természetesen adódik a működés jellegéből.)

A fenti elvek és feltevések jelentős mértékben csökkentik a keresési tér méretét. A továbbiakban egy specifikus szabálykinyerő módszerről lesz szó.

3.5.1. NofM módszer

Az NofM szabálykinyerő algoritmus az alábbi általános formájú szabályokat keres:

ha (N a következő M feltétel közül IGAZ) akkor ...,

Az NofM szabálykinyerő módszer lépései a következők:

1. Csoportosítás. Minden rejtett és kimeneti egység súlyait csoportosítsuk hasonló értékek szerint.

Mivel a tanulás során inkább laza súlyklaszterek keletkeznek, mint határozott ekvivalencia osztályok, ezért klaszterezésre van szükség. Több különböző klaszterező megoldást ismerünk.

Az egyik ilyen megoldás az **egyesít** (*join*) eljárás, ahol a klaszterek egyesítése, egyelemű klasszterektől kezdve történik. A leállás akkor következik be, ha nincs már két olyan klaszter, amely közelebb lenne, mint egy megadott küszöb-távolság. Ez a klaszterező eljárás $O(n^3)$ számítási komplexitású, ahol n a súlyok száma.

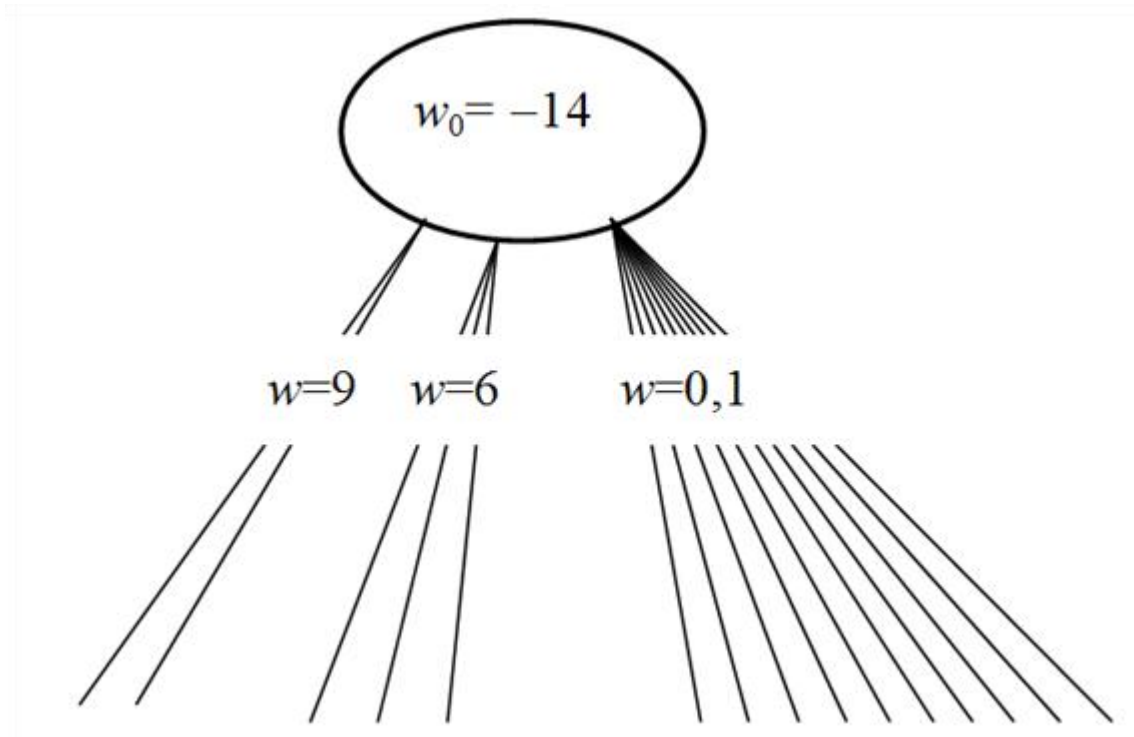
A másik klaszterező eljárás a **rendez** (*sort*) eljárás. Ebben a súlyokat előzetesen rendezzük, majd a rendezett súlyokon egylépésben végigmegyünk. A sorbarendezett súlyokat egyenként elővesszük és az aktuális klaszterhez adjuk egészen addig, amíg vagy az új súly távolabb esik az aktuális klaszter középpontjától, mint egy megadott t távolság, vagy az új súly hozzáadása egy meglévő súlyt a megváltozott klaszter középponttól t távolságon kívülre helyez. Szinte kizárólag a *rendez* eljárást alkalmazzák, mivel ennek számítási komplexitása csak $O(n \log n)$.

2. Súlyok átlagolása. A csoport súlyértékeinek átírása a súlyok átlagértékére.

3. Bemeneti klaszterek elhagyása. Azon csoportok kizárása, amelyek nem befolyásolják, hogy az egység aktív, vagy inaktív lesz.

Az **elimináció** célja a befolyással nem bíró klaszterek megkeresése. Ehhez minden klaszter totális aktivációját kell kiszámítanunk, majd egy összehasonlításos elemzést kell végezni a többi súlyértékkel. A vizsgált klaszter eliminálható, ha a totális értékét hiába adjuk hozzá egyetlen másik szabálycsoport súlyösszegéhez, az nem haladja meg a szükséges küszöbértéket. Jó példa az eliminációra a 12.7 ábra. Látható, hogy a teljes C klaszter szumma aktivációja ($10 \cdot 0,1 = 1$) nem befolyásolja az egység aktív vagy inaktív voltát.

12.7. ábra - Példa az eliminációra.



4. Eltolásérték újratanítása. Az eltolásértékek optimalizálása a hibavisszaterjesztéssel úgy, hogy közben a többi súly értékét nem változtatjuk.

Erre a lépésre azért van szükség, mert a szükségtelen bemenetek törlése és a súlyok átlagolása megváltoztathatja az egység aktivációját. Ennek eredményeképpen egy KBANN háló hiba aránya szignifikánsan nagyobb lehet az NofM 3. lépése után, mint az eredeti tanított formájában.

5. Szabály képezés. Ebben a lépésben minden rejtett és kimeneti egységhez külön szabályt rendelünk, azaz „leírjuk” a hálóban található szabályokat. A szabály akkor lesz igaz, ha a súlyozott igaz bemenetek összege meghaladja az adott neuron eltolásértékét.

6. Szabály egyszerűsítés. Ha lehetséges, egyszerűsítsük a szabályokat az eltolásértékek és súlyok eliminálásával. Az egyszerűsítés során megállapítjuk a csoportok lehetséges kombinációit, melyek meghaladják a szabály küszöbértékét, a biast. Ez a lépés gyakran több szabályt eredményez. Tehát a kezdeti bonyolult szabály helyett több egyszerűbb szabályt alkotunk, így az egyedi szabály komplexitását a szabályhalmazból eredő komplexitással váltjuk fel.

Példaként nézzük a következő szabály egyszerűsítését:

- $Z : -18 < 9,1 * \text{AktívElemekSzám}(\{A, B, C, D\})$
- $+7,5 * \text{AktívElemekSzám}(\{E, F, G\})$

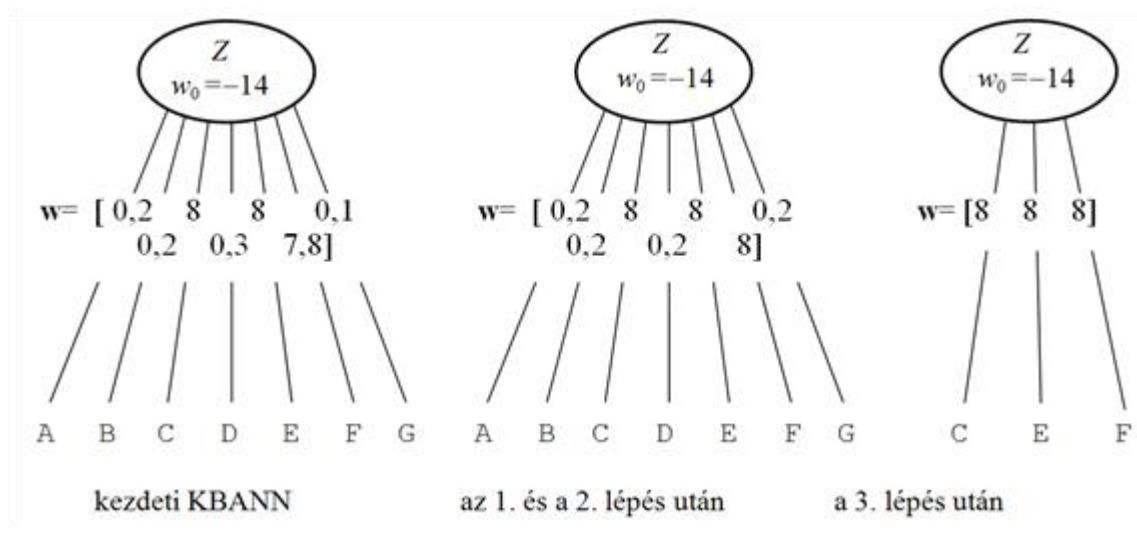
ahol az AktívElemekSzám függvény azt adja vissza, hogy az argumentumában szereplő feltételek közül hány vesz fel igaz értéket.

Az egyszerűsítés eredményeképp a következő három szabály fogalmazható meg:

- $Z : -2$ az $\{A, B, C, D\}$ közül
- $Z : -1$ az $\{A, B, C, D\}$ közül és 2 az $\{E, F, G\}$ közül
- $Z : -E, F, G$

A 12.8 ábra az NofM eljárás lépéseit mutatja egy egyszerű példán keresztül. A kiindulás egyetlen neuron hét bemenettel. Az eljárás olyan szabályt eredményezett, amely szerint a következmény igaz, ha három feltétel közül kettő igaz.

12.8. ábra - Példa az NofM működésére egyszerűsítéssel



A 4. és 5. lépés után:

Ha $8 * \text{AktívElemekSzám}\{C, E, F\} > 14$ akkor Z

A 6. lépés után:

Ha 2 a $\{C, E, F\}$ közül IGAZ akkor Z

3.6. Új csomópontok dinamikus hozzáadása a KBANN hálózathoz: TopGen algoritmus

A KBANN konstrukciójának eddig bemutatott lépései alapvetően a kiinduló szabályok háló architektúrára való konvertálására és az így kapott hálónak a tanítómintákkal történő tanítására vonatkoztak. A konstrukció lépései között volt azonban két olyan lépés – a 4. és az 5. lépés –, melyek a kiinduló háló strukturális módosítását tették lehetővé rejtett neuronok illetve újabb bemenetek beiktatása révén. Újabb bemenetek beiktatása a priori ismeret alapján lehetséges, illetve annak alapján, hogy a tanító adatok olyan új bemeneti komponens(ek)re is vonatkoznak, mely(ek) az eredeti szabályokban nem szerepel(nek). A rejtett elemek közbeiktatása azonban további megfontolásokat igényel. A következőkben egy olyan eljárást ismertetünk, mely a strukturális módosításoknak egy lehetséges módját adja meg.

A strukturális módosítás célja, hogy új szabályok megfogalmazása lehetségessé váljon. Erre különösen akkor van szükség, ha a kiinduló domén elmélet nem teljes, a kiinduló szabályok a tárgyterományra vonatkozó szükséges ismereteknek csak egy részét reprezentálják. A KBANN-nel szerzett tapasztalatok szerint az általánosítóképesség nagymértékben csökken, ha a konstrukció nem „teljes” domén elméleten alapul, azaz hiányoznak szabályok vagy tények a tanult domén elméletből [Tow91]. Ugyanakkor, ha egy „teljes” domén elméletet bővítünk további szabályok hozzáadásával a KBANN hatékonyan kiszűri ezeket a feladathoz nem illő szabályokat.

A TopGen algoritmus több lépésből áll, melyek legfőbb célja a domén elméletre illetve a tanítómintákra legjobban illeszkedő háló megkeresése. A TopGen ehhez felderíti azokat a csomópontokat, amelyek a hibás kimenetekhez vezetnek és ezen csomópontok „javításával” és a háló újratanításával iterál a legjobb megoldáshoz.

A TopGen algoritmus lépései:

1. Válasszuk szét a rendelkezésre álló mintahalmazt tanító halmazra és két kiértékelő halmazra. Legyen a két kiértékelő halmaz: validációs-halmaz-1 és validációs-halmaz-2.
2. Tanítsuk meg a kezdeti KBANN hálót az eddig ismertetett módon és a megtanított hálót tegyük egy HÁLÓK nevű listára.
3. A megállási feltétel (hiba egy adott küszöb alá esik a validációs halmazon, vagy elértünk egy időkorlátot) eléréséig:

válasszuk ki és töröljük a HÁLÓK listáról a validációs-halmaz-2 szerinti legjobb hálót,

a kiválasztott hálón alkalmazzuk a **CsomópontÉrtékelő** eljárást, hogy meghatározzunk N olyan csomópontot, ahol a leginkább szükség volna a topológia bővítésére,

készítsük el az N módosított hálót, mindegyiket tanítsuk újra és adjuk a hálókat a HÁLÓK listához,

rendezzük sorba a HÁLÓK listát a validációs-halmaz-2-n értékelt teljesítményük alapján.

a HÁLÓK listán csak az M legjobban teljesítő hálót tartsuk meg, a többit töröljük.

Válasszuk ki a HÁLÓK listáról a validációs-halmaz-2 szerinti legjobb hálót.

A **CsomópontÉrtékelő** eljárás alkalmazásának célja, hogy kiválogassuk azokat a csomópontokat (neuronokat), melyek módosításával a teljes hálónak a validációs-halmaz-1-re adott válaszai javíthatók. Olyan csomópontokat keresünk, ahol a módosítás hatására a javulás a lehető legnagyobb mértékű. A **CsomópontÉrtékelő** eljárás alkalmazása alatt a neuronok aktivációs függvényét ideiglenesen az ugrásfüggvényre cseréljük. Ez a lépés azt biztosítja, hogy egy neuron válasza mindenképpen vagy 0 vagy 1 legyen.

A **CsomópontÉrtékelő** eljáráshoz definiálnunk kell, hogy mit nevezünk egy adott csomópont szempontjából javítható esetnek. Egy eset javítható, ha a háló adott mintára hibás választ ad, de az adott csomópont választát ellenkezőjére változtatva a háló válasza helyes lesz. A javítható hibákon belül eltérő módon javítható a hiba, ha tévesen-pozitív illetve tévesen-negatív esetről van szó. Az alábbi eljárásban ezért a javítható hibák két típusát külön-külön számba kell venni.

A **CsomópontÉrtékelő** eljárás a következő lépésekből áll:

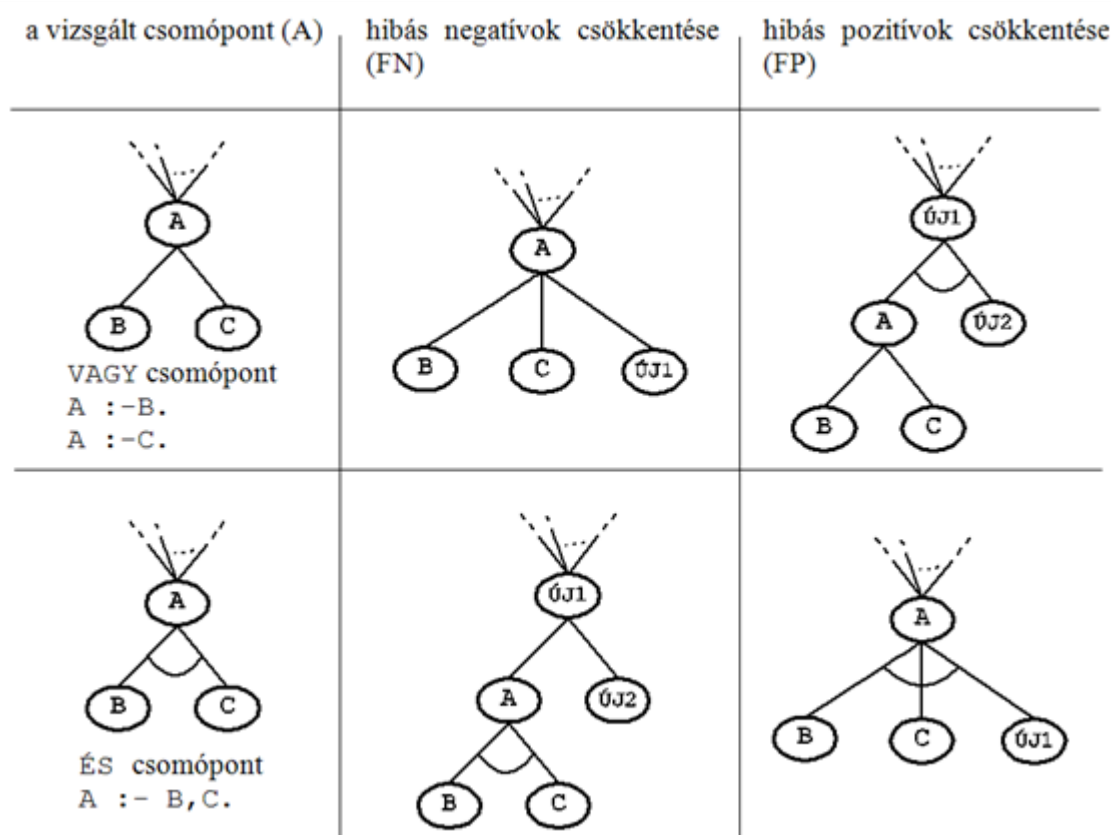
1. Értékeljük minden egyes csomópontot a következők szerint.
 - rendeljük minden csomóponthoz két számlálót, melyek a javítható-tévesen-pozitív (*false positive*, *FP*) és a javítható-tévesen-negatív (*false negative*, *FN*) eseteket számlálják. Állítsuk a számlálók kezdeti értékeit nullára.
 - menjünk végig egy, a validációs-halmaz-1-be tartozó rosszul osztályozott mintapont esetén a háló összes csomópontján, és vizsgáljuk meg, hogy megváltoztatva egy konkrét csomópont kimenetét (0-ról 1-re vagy fordítva), a módosított háló már helyesen osztályozza-e az adott példát. Amennyiben igen, akkor az adott csomópont megfelelő számlálóját inkrementáljuk.
 - rendezzük sorba a javítandó csomópontokat a számlálók csökkenő értékei szerint.
2. Válasszunk ki egy csomópontot a sorbarendezett csomópontok közül és módosítsuk a csomópontot úgy, hogy a válasza az adott mintánál ellenkezőjére változzon.

A módosítás többféleképpen elvégezhető: lehetséges a csomópont bővítésével, illetve a csomópont egyszerűsítésével. A csomópont egyszerűsítése a standard tanító eljárás következtében is bekövetkezhet (például egy súly a tanítás hatására nullává válik), bővítés viszont csak strukturális kiegészítéssel lehetséges. Ezért az eljárás csak a bővítésre ad megoldást. A szükséges módosítás függ attól, hogy a csomópont konjunktív (ÉS) vagy diszjunktív (VAGY) kapcsolatot valósított-e meg. Ennek megfelelően 4 lehetséges esetet kell megkülönböztetnünk.

A csomópont korrekcióhoz javasolt bővítés 4 alapesetét a 12.9 ábra mutatja. Látható, hogy mindegyik esetben bővítésre kerül sor. Téves negatív esetek javítása új VAGY kapcsolati ág beiktatásával történhet. Ez VAGY kapcsolatot megvalósító csomópontnál a meglévő csomópontokhoz egy újabb bemenet beiktatását igényli, míg ÉS kapcsolatnál egy új VAGY csomópontra van szükség, ami egyben egy új réteget is jelent. Téves pozitív esetek

javításához az ÉS kapcsolatot megvalósító csomópontok bővíthetők új bemenettel, míg a VAGY kapcsolathoz egy új rétegben egy új ÉS csomópont beiktatása szükséges.

12.9. ábra - Példa a TopGen beszűrési algoritmusára



3.7. A KBANN háló topológiájának megváltoztatása genetikus módszerrel: REGENT

A KBANN kialakításánál jelentős szerepet kaphatnak a genetikus algoritmusok is. Minthogy a kiinduló hálót a priori szabályokból hozzuk létre, valójában minden egyes neuron egy szabálynak felel meg. A tanítással történő háló finomítása – amennyiben csak a háló súlyainak hangolását eredményezi – nem alkalmas újabb szabályok beépítésére. Ez csak úgy lehetséges, ha a háló struktúráját is módosítjuk, a hálót újabb neuronnal, neuronokkal bővítjük. Az előzőekben bemutatott TopGen eljárás a háló strukturális módosításán keresztül igyekszik a háló képességét javítani. A KBANN konstrukciójánál bevezetett REGENT (*Refining with Genetic Evolution Network Topologies*) algoritmus a struktúra módosítás egy újabb lehetőségét teremti meg [Opi97].

A REGENT algoritmus két fázisban dolgozik: egyrészt genetikus kereséssel végignézi a háló topológia terét, majd az egyes hálókat hibavisszaterjesztéssel algoritmusmal tanítja. A REGENT mindkét fázisban felhasználja a domén ismereteket. A domén ismeretek a topológia térben való keresést segítik, irányítják, a tanításnál pedig biztosítják, hogy jó kezdő súlyokkal induljon a hibavisszaterjesztéssel algoritmus.

Az algoritmus első lépésként a rendelkezésre álló mintapontokat tanító és kiértékelő (validációs) mintakészletre bontja. A validációs készlet szerepe most is – mint minden neuronháló tanításánál – az egyes hálók teljesítőképességének, általánosítóképességének meghatározása.

A kezdeti neuronháló populáció egyedeit a kiinduló és a szabályokból létrehozott háló perturbációjával hozzuk létre. A REGENT eljárás a perturbáció során törölhet egyes neuronokat, illetve újabb neuronokat iktathat be. A neuronok törlése vagy beiktatása valójában a kiinduló háló alapját képező szabálykészlet módosítását jelenti.

A perturbációval előállított kiinduló populációra a szokásos genetikus operátorokat, vagyis a keresztezést és a mutációt alkalmazzuk. A keresztezés két kiinduló hálótopológiából (a szülőkből) két utód hálót hoz létre. Ehhez

a kiinduló hálók rejtett neuronjait két csoportba osztja. Jelöljük ezeket A és B csoportnak. Így tehát létrejönnek A_1, B_1 és A_2, B_2 neuroncsoportok. A keresztezés eredményeképp létrejött utódok az A_1-A_2 és a B_1-B_2 neuronokat használják fel. Amennyiben több rejtett réteg van, a rejtett neuronok két csoportra bontása rétegenként történik a kimenethez közelebbi rétegtől indítva. A rejtett neuronok szétszétását az eljárás úgy végzi, hogy törekszik az erős kapcsolatban lévő neuronok együtt-tartására. Mivel ezek a neuronok egy szabály neuronháló megfogalmazásai, ha két csoportba kerülnének, az általuk képviselt szabály semmisülne meg. A REGENT keresztezés operátora ezért lényegében szabályok keresztezését végzi.

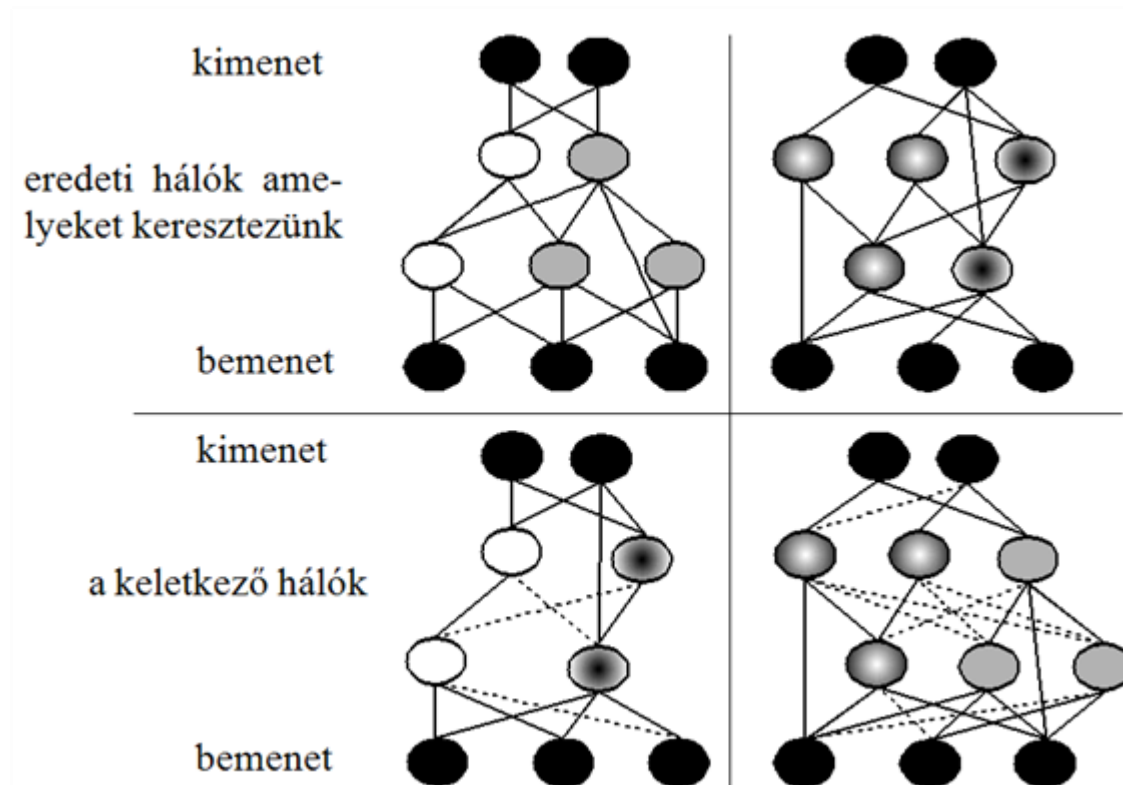
Az utódháló összeköttetéseinek meghatározása az alábbiak szerint történik:

- egy utódháló örökli a w_{ij} súlyt, ha a megfelelő (i -edik és j -edik) neuronok is öröklődnek, illetve ha ezek bemeneti vagy kimeneti neuronok,
- a nem bekötött neuronokat a szomszéd szinteken lévő neuronokhoz kötjük közel nulla értékű súlyokkal,
- beállítjuk a neuronok eltolás (*bias*) értékét, hogy az egyes neuronok által képviselt szabály által megvalósított eredeti logikai kapcsolat (ÉS illetve VAGY) megmaradjon.

A keresztezésre mutat példát a 12.10 ábra. Az ábrán a neuronok öröklődése követhető. A szaggatott vonalak a közel nulla súlyú kapcsolatokat reprezentálják.

A REGENT algoritmus során alkalmazott mutáció csak növelni tudja a hálót azáltal, hogy újabb neuront iktat be. Mivel minden újabb neuron beiktatása valójában egy újabb szabály beiktatását jelenti, a mutációval a szabálybázis bővítése érhető el. Annak oka, hogy a REGENT csak bővíteni képes a hálót az, hogy az eredeti KBANN tanítása alkalmatlan újabb szabályok beépítésére, miközben a tanítás eredményeképpen létrejöhet olyan háló (a súlyok olyan érték-eloszlása), mely egyes szabályok megszűnését eredményezi.

12.10. ábra - A REGENT eljárásnál alkalmazott keresztezés



A REGENT algoritmussal működő KBANN hibrid rendszert számos különböző komplexitású feladatban alkalmazták. A feladatok egyike az emberi géntérkép elkészítésre irányuló Human Genome Project-hez kapcsolódott, ahol a DNS szekvenciákban belül gének helyét kellett megállapítani. A REGENT algoritmussal működő KBANN eredményét összevetették az egyszerű KBANN eredményével, illetve a klasszikus neuronháló megoldással. Minden összehasonlításból a REGENT algoritmust alkalmazó megoldás került ki

győztesen. A javulás a standard neuronhálós megoldáshoz képest mintegy 20-40% volt, és még a genetikus algoritmus nélküli KBANN-hez képest is 10-15% javulás volt tapasztalható.

A KBANN-nek számos további módosított illetve kibővített változatát dolgozták ki. A bővítés egyrészt a képességek, modellező erő kiterjesztését szolgálják, illetve a szabálykinyerés további lehetőségeit adják meg. A modellező erő kiterjesztését az ún. FS-KBANN rendszer [Mac95] biztosítja, ahol a tudásalapú háló nemcsak statikus leképezésre képes, hanem végesállapotú rendszer modellezésére is. Az FS-KBANN esetében a kiinduló domén elmélet sem egyszerű szabályok formájában, hanem procedurális ismeret formájában áll rendelkezésre, és a neuronháló sem egyszerű előrecsatolt MLP, hanem visszacsatolt háló. A szabálykinyerést döntési fák felhasználása segítségével oldja meg a TREPAN algoritmus [Cra96].

Bár a KBANN és változatai csak meglehetősen korlátozott formájú domén tudás felhasználását teszik lehetővé, jelentőségük, hogy mindezidáig talán a legsikeresebben oldották meg a különböző reprezentációjú ismeretek együttes felhasználását.

13. fejezet - Gyakorlati feladatmegoldás: adatelőkészítés, lényegkiemelés

Neurális hálózatok segítségével elvileg csaknem tetszőleges modellezési, feldolgozási feladat megoldható, ha kielégítő mintahalmaz áll rendelkezésünkre, amely tartalmazza a keresett információt. Nincs másra szükség, mint megtalálni a megfelelő típusú és architektúrájú hálózatot, majd a minták segítségével tanítani. Ezen problémákkal, pl. a hálózattípus és méret megválasztásával, a megfelelő tanítási algoritmusokkal az előző fejezetekben foglalkoztunk.

A gyakorlati feladatok megoldásának menete azonban tipikusan jóval bonyolultabb ennél. A tapasztalatok szerint, ha egyszerűen a nyers, mért adatokat használjuk a háló tanítása során, akkor a hálózat vagy rendkívül lassan konvergál a megoldás irányába, vagy akár divergenciát is tapasztalhatunk. A problémák egy részének egyik oka az adatokat gyakran torzító, meghamisító zaj lehet, ezért rendszerint célszerű a nyers adatokat hihetőség vizsgálatnak alávetni, kiszűrni vagy korrigálni a hibás adatokat, szűréssel javítani a zajos adatokat. Másrészt a megfelelő tanító és tesztminta halmaz kialakítására sem mindig a legcélszerűbb eljárás a korrekt elsődleges mért adatok összegyűjtése: informatív származtatott adatok előállítása sokszor nagymértékben felgyorsítja a megoldás megtalálását. (Triviális példa, hogy amikor a rendszer állapota a disszipálódo teljesítménytől függ, akkor nem feltétlenül a mért feszültség és áram értéke jellemzi legjobban a helyzetet, hanem a kettő szorzataként származtatott mennyiség, a teljesítmény. A háló ugyan képes megtalálni ezt a kapcsolatot, de jelentősen lelassíthatja a megoldás elérését, növelheti a szükséges modell komplexitást, ha olyan összefüggések felismerését is a hálóra bízuk és tanítással kívánjuk elérni, melyeknek eleve birtokában voltunk.)

A gyakorlati feladatok megoldásánál tehát nagy jelentőségű, gyakran a legfontosabb lépések közé tartozik, hogy megfelelő adatelőkészítéssel, lényegkiemeléssel, utófeldolgozással tehermentesítsük a hálózatot. Általánosságban kimondható: minden olyan ismeretet, amely *a priori* rendelkezésünkre áll, célszerű valamilyen módon az elő- vagy utófeldolgozásba beépíteni, hogy a tanítás során arra a feladatra lehessen koncentrálni, melyről csak a mintákban kódolt módon áll rendelkezésünkre információ.

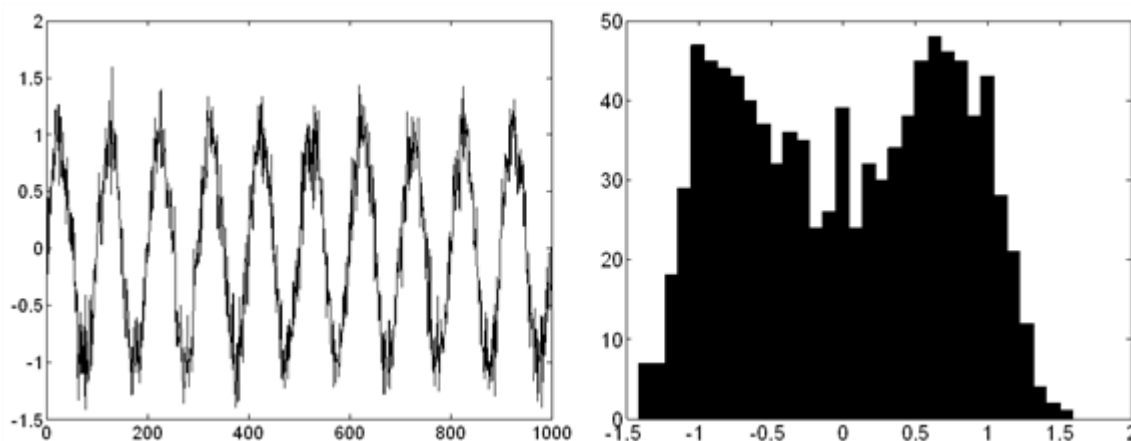
1. Zajos adatok

A zajos adatok kezelésével, a zajok fajtáival, a szűrésekkel ebben a könyvben még elnagyoltan sincs lehetőségünk foglalkozni, ezeknek a témáknak önmagukban könyvtárnyi irodalma van. Ugyanakkor a tanításnál felhasznált mintahalmazok kapcsán érdemes rámutatni egy-két speciálisan fontos jelenségre.

1.1. Nem Gauss eloszlású zajok hatása

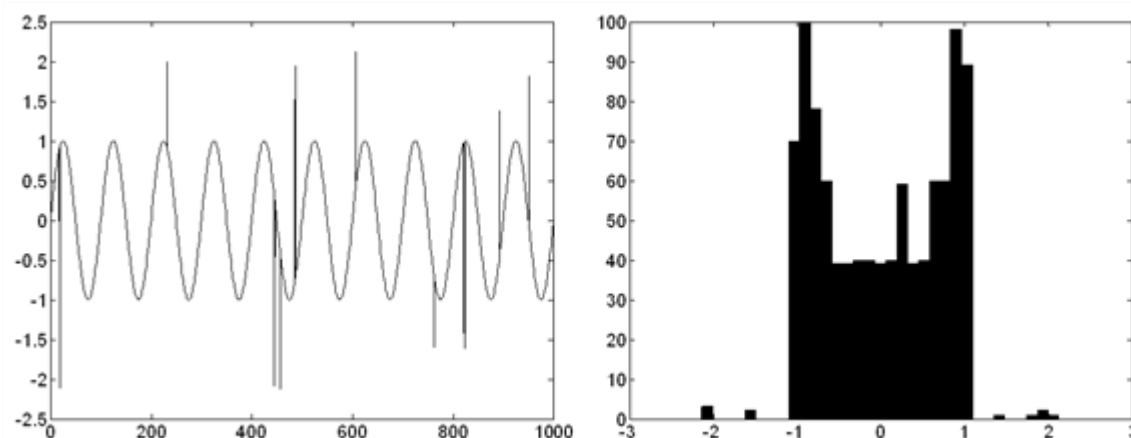
A jelfeldolgozás, szűrés területén legtöbbször abból indulunk ki, hogy a zajok Gauss eloszlásúak, ami végső soron gyakran az átlagolás típusú feldolgozási, zajcsökkentési eljárásokhoz vezet. (Megmutatható, hogy additív Gauss fehér zaj esetén a lineáris szűrési eljárások közül legkisebb négyzetes értelemben az átlagolás optimális eljárás.) Ugyanakkor a gyakorlati feladatmegoldás kapcsán gyakran találkozunk az úgynevezett kilógó adatok problémájával, a 13.3 pontban ezzel részletesebben is foglalkozunk majd. Ezek a kilógó adatok (*outliers*) nem modellezhetők jól Gauss zajként, és feldolgozásukat sokszor nem célszerű átlagolással megközelíteni.

13.1. ábra - Gauss zajjal terhelt szinuszos jel időfüggvényének egy szakasza és hisztogramja



Az impulzus zajok kezelésére gyakran az átlagolásnál jobb eredményt ad az úgynevezett medián szűrés vagy annak valamely változata. A medián szűrés a rendezett statisztika szűrők (*Order Statistic Filter, OSF*) legismertebb alelete. Idősorok esetén az egy időablakba eső értékeket először sorbarendezzük, majd a sor középső elemét vesszük (természetesen a medián szűrés nemcsak idősoroknál, hanem sokféle mérési sorozatnál alkalmazható). Egyszerűen belátható, hogy ha az ablakhossz $W=2L+1$, akkor az L -nél nem szélesebb impulzusok sohasem jutnak el a rendezés során a középső pozícióba, így nem befolyásolják a kimenetet. A 13.2 ábrán látható idősor átlagolással javított változatának egy részletét láthatjuk a következő, 13.3 ábra ábrán.

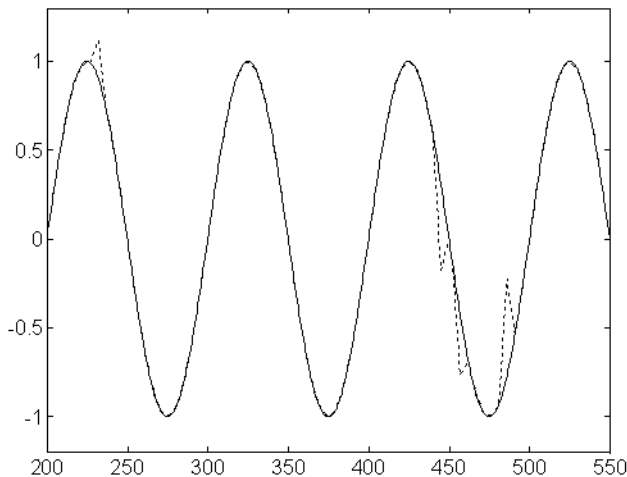
13.2. ábra - Ugyanazon szinuszos jel impulzuszajjal terhelt időfüggvénye, illetve hisztogramja



Az átlagolás az impulzuszerű zajok amplitúdóját csökkentette ugyan, de időben "szétkente" a zajt, az eredetileg zajjal nem terhelt pontok is torzultak. (Ugyanis az impulzusok helyén megjelent az átlagoló szűrő impulzusválasz függvénye.) Ugyanezt a jelet ugyanakkora (jelen esetben 5 pont ablakszélességű) medián szűrővel javítva az eredeti jelet közel hibátlanul visszaállíthatjuk. (A medián szűrő olyan módon torzít, hogy levágja a jel csúcsait, 5 pontos szűrő esetén 2-2 pontot a csúcsonál. Ez a szinusznál igen kis torzítást eredményez, mert a csúcsonál lapos a függvény.) Természetesen, ha maga az alapjelünk is impulzusjellegű lenne - a zajhoz hasonló szélességű impulzusokkal - vagy rendelkezne ilyen szakaszokkal, akkor a medián szűrés nem lenne alkalmazható.

Nagyon lényeges tehát, hogy vegyük figyelembe a rendelkezésünkre álló (akár kvalitatív) információkat a jelről és a zajról, optimális vagy közel optimális eljárást csak a jel- és zajmodell figyelembevételével tudunk választani.

13.3. ábra - Az impulzus zajjal torzított szinuszos jel egy részlete - 5 pont széles ablakú mozgó - átlagolással javítva (folytonos vonal: ideális jel, pontozott vonal: szűréssel kapott jel)



1.2. Zajos adatok felhasználása az adatok megfelelő súlyozásával (EIV módszer)

A tanításra használt adatokat torzító zaj hatásának csökkentésére több lehetőség is kínálkozik a tanítás során. Ezek közül az egyik, de nem az egyetlen, az előző pontban bemutatott zajszint csökkentés (a jel/zaj viszony javítása) valamilyen – legtöbbször lineáris – szűréssel, ami kétségtelenül a legismertebb és leginkább elterjedt módszer.

A problémának most egy ritkábban vizsgált aspektusát mutatjuk be, ahol a kiinduló kérdés, hogy milyen következményekkel jár, milyen megoldásokat tesz szükségessé, ha a zaj nemcsak a megfigyelt kimenet(ek)nél jelentkezik, amely a legtöbbször vizsgált eset, hanem a rendszer bemenetén/bemenetein is csak zajos megfigyelésre van lehetőségünk. A vizsgálatok során az derült ki, hogy a bemeneti zaj bizonyos értelemben nagyobb problémát jelent a modell kialakításnál, a tanításnál, mint a kimeneti. A szokásos feltételek mellett (nulla várható értékű Gauss zaj) a kimeneti zaj aszimptotikusan nem okoz torzítást az eredményben, ugyan lassítja a konvergenciát, számos problémát vet fel, de torzítatlan becsléshez juthatunk. Nem ez a helyzet, ha a bemeneten is megfigyelési zajjal kell küzdenünk, a becslés ilyenkor még a szokásos "kellemes tulajdonságú" (nulla várható értékű Gauss) zaj esetén is torzított becsléshez vezet. A kimeneti és bemeneti zaj tanulás során történő kezelését szolgálja az EIV (*Errors In Variables*) módszer, aminek ugyan számos problémája van, de a kérdés elvi jelentősége miatt mégis érdemes megismerkedni legalább az alapgondolatával [Dei86].

A zaj figyelembevétele – mint az előző pontban is kifejtettük – akkor történhet meg hatékonyan, ha rendelkezünk a zajra vonatkozó valamilyen ismerettel, pl. ismerjük az egyes tanítómintákat eltérő mértékben torzító zaj szintjét (szórását). Ez az ismeret valamilyen fizikai megfontolás alapján, esetleg *a priori* információkból nyerhető. Másik lehetőség – és a következőkben ezt mutatjuk be –, amikor a mérések alapján a jellel együtt a zajt is modellezni próbáljuk. Például úgy, hogy egy-egy szituációt többször is megpróbálunk előidézni, ugyanazt a helyzetet többször is megmérjük, és ebből nyerjük ki a különböző zajokra vonatkozó információkat. Természetesen ezt is akkor tudjuk hatékonyabban megtenni, ha legalább részleges, kvalitatív ismeretünk van a zajok jellegéről. Másrészt látni fogjuk, hogy a bonyolultabb modell (rendszer+zaj) felállítására több szabad paraméter meghatározását igényli, így több mérésre lehet szükségünk.

Tegyük fel, hogy az ismeretlen – neurális hálóval modellezni kívánt – rendszer bemenetét és kimenetét nem tudjuk zajmentesen megfigyelni (ld. 13.4 ábra). A valóságban mindig ez a helyzet, csak egyes esetekben elhanyagolhatóan kicsi lehet a megfigyelési zaj. Összesen P darab ($p=1,2,\dots,P$) $\{\mathbf{x}_M^{(p)}, \mathbf{y}_M^{(p)}\}$ megfigyelt mintapár áll rendelkezésünkre a tanításhoz. A p -edik minta bemeneti jeleiből képzett vektort az összes komponensében egy-egy $\sigma_{\mathbf{x},p}$ szórású, nulla várhatóértékű Gauss eloszlású zaj terheli (ezt az ábrán $\mathbf{n}_{\mathbf{x}}^{(p)}$ zajvektor jelöli). A p -edik minta kimeneti jeleiből képzett vektort pedig az összes komponensében egy-egy $\sigma_{\mathbf{y},p}$ szórású, ugyancsak nulla várhatóértékű Gauss eloszlású zaj terheli (ezt az ábrán $\mathbf{n}_{\mathbf{y}}^{(p)}$ jelöli).

Az egyszerűbb leíró formalizmus érdekében azzal a – tulajdonképpen nem életszerű – feltevessel éltünk, hogy egy-egy szituációban a bemeneti és kimeneti jelekből képzett vektorok összes komponensét azonos szórású zaj

terheli (ezért lehet két skalárral, $\sigma_{\mathbf{x},p}$ -vel és $\sigma_{\mathbf{y},p}$ -vel jellemezni az adott mintát terhelő zajt). Ezt kézenfekvő módon továbbfejleszthetjük olyan módszerre, amelyben a komponens zajok is eltérő szintűek, de ennek hatására egy újabb index és újabb szumma jelenik meg az EIV megközelítésnél alkalmazott kritériumfüggvényben (ld. (13.1)). A gondolatmenet ugyanaz maradna, így az egyszerűség kedvéért a komponensenként azonos zajszintet feltételező eset bemutatását választottuk. Ugyanakkor viszont azt tesszük fel, hogy az egyes minták esetén a zaj különböző lehet (ezért indexeljük, az $\mathbf{n}$ zajokat p -vel), éppen ez ad esélyt a tanítás eredményeképpen előálló becslés javítására.

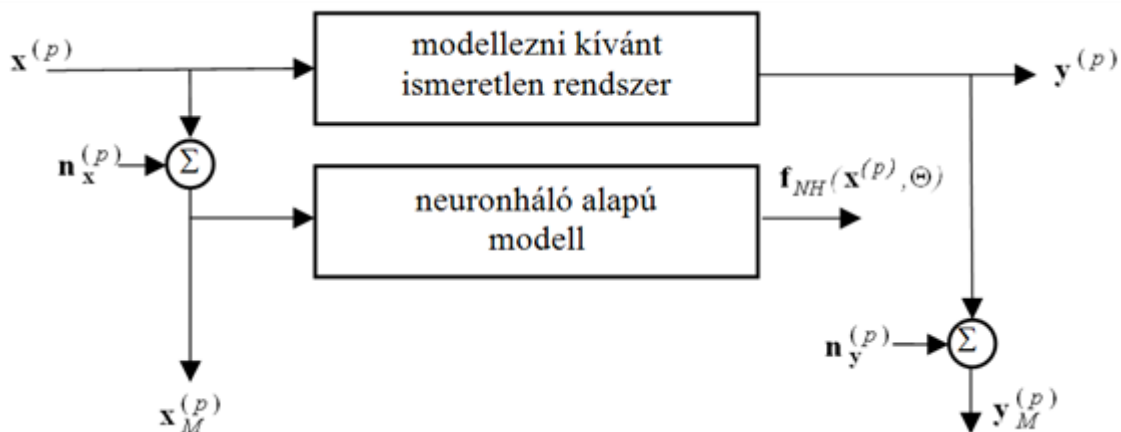
Ekkor az EIV módszer által javasolt lehetőség az, hogy a tanítás során használt kritériumfüggvényt a következőképpen módosítsuk:

$$C_{EIV} = \frac{1}{P} \sum_{p=1}^P \frac{\left(\mathbf{y}_M^{(p)} - \mathbf{f}_{NH}(\mathbf{x}^{(p)}, \Theta) \right)^T \left(\mathbf{y}_M^{(p)} - \mathbf{f}_{NH}(\mathbf{x}^{(p)}, \Theta) \right)}{\sigma_{\mathbf{y},p}^2} + \frac{\left(\mathbf{x}_M^{(p)} - \mathbf{x}^{(p)} \right)^T \left(\mathbf{x}_M^{(p)} - \mathbf{x}^{(p)} \right)}{\sigma_{\mathbf{x},p}^2} \quad (13.1)$$

A kritériumfüggvényben két dolgot kell hangsúlyoznunk:

- az ismeretlen (megfigyelhetetlen) torzítatlan bemeneti értékek – az $\mathbf{x}^{(p)}$ -k – is szerepelnek a kritériumfüggvényben, tulajdonképpen ezek olyan további szabad paraméterek, amelyeket az optimalizálásban a bemeneti zaj hatásának modellezésére használunk,
- az egyes mintákat torzító zaj szórásnégyzetével súlyozzuk a hibát, tehát a tudottan nagyobb zajjal terhelt minták kisebb súllyal kerülnek az optimalizálandó függvénybe.

13.4. ábra - A bemeneten és kimeneten is megfigyelési zajjal terhelt rendszer és neurális modellje



Az optimalizálandó kritériumfüggvényben a szummázott kifejezés első tagja tulajdonképpen a hagyományos LS (legkisebb négyzetes, *least squares*) hibakritériumnak felel meg annyi módosítással, hogy az egyes mintákhoz tartozó négyzetes hibaértékeket a kimeneti zaj szórásnégyzetének a reciprokával súlyozzuk. Tehát, ha tudjuk (*a priori* ismeretből, mérésből, bárholonnan), hogy az egyik minta zajosabb, mint a másik, akkor annak kisebb lesz a hatása a modell tanítás során történő kialakítására. A második tag a bemeneti mért és az – ismeretlen – tényleges mintavektorok eltérését igyekszik minimalizálni, ismét a mintához tartozó bemeneti zaj szórásnégyzetének a reciprokával súlyozva. Tulajdonképpen azt a járulékos feladatot viszi be a problémába, hogy "milyen lehetett a zajmentes, megfigyelhetetlen bemenet, a zajos értékek alapján becslve?".

Az összehasonlítás kedvéért tekintsük át:

- A hagyományos LS módszernél az $\mathbf{x}_M^{(p)}$ és $\mathbf{y}_M^{(p)}$ adatok jelentik a konkrét optimalizálás (tanítás) szempontjából konstans értékeket, amelyek a konkrét feladatot jellemzik, a Θ paramétervektor pedig az optimalizálásban felhasználható változókat.

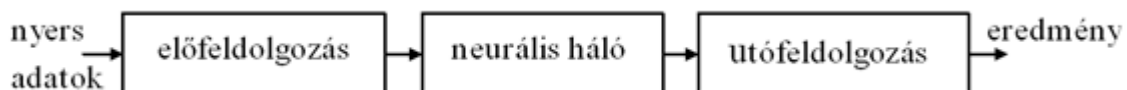
- Az EIV kritériumfüggvényében az $\mathbf{x}_M^{(p)}$ és az $\mathbf{y}_M^{(p)}$ adatok, valamint a $\sigma_{\mathbf{x},p}$ és a $\sigma_{\mathbf{y},p}$ szórásértékek jelentik a konkrét optimalizálás (tanítás) szempontjából konstans értékeket, a Θ paramétervektor és az ismeretlen zajmentes bemenetnek ($\mathbf{x}^{(p)}$ ($p=1,2,\dots,P$) komponensei) pedig az optimalizálásban felhasználható változókat.

Ez az összehasonlítás már mutatja az EIV módszer egyik hátrányát is, nagyon megnő a szabad paraméterek száma, ezáltal az eljárás különösen hajlamossá válik a túltanulásra. A másik hátránya, hogy az eljárás bonyolultabb, mint az egyszerű LS modellalkotás. Előnye az eljárásnak ugyanakkor, hogy nem elhanyagolható bemeneti zajok kezelésére is képes [Van00].

2. Az adatok előfeldolgozása

A gyakorlati feladatok megoldása során szinte mindig célszerű a nyers adatokon valamilyen előfeldolgozást alkalmazni, és a feldolgozott adatokat adni a hálózat bemenetére, majd a hálózat választását megfelelő utófeldolgozással a kívánt eredménybe transzformálni.

13.5. ábra - A gyakorlati problémák tipikus feldolgozási folyamata



Az elő- és utófeldolgozás az egyszerű transzformációktól (pl. lineáris adatskálázás) a bonyolult adaptív algoritmusokig terjedhet. Az eljárásokat legtöbbször a konkrét adatokra, zajokra vonatkozó *a priori* információ, vagy egyéb szakértői ismeret felhasználásával választjuk ki, de gyakran a neurális háló tanítása során nyerünk olyan tapasztalatokat, melyek segítségével eldönthető, hogy milyen új adatelőkészítési módszert kell alkalmaznunk, vagy az eddig alkalmazott módszereinket hogyan kell módosítsuk. Ezen esetekben a neurális modellezés folyamata oly módon válik iteratívvá, hogy a tanító és teszt adatkészletek (amit a továbbiakban általában adatbázisoknak nevezünk) kialakítása, majd a hálózat tanítása egy vagy több lépésben ciklikusan történik.

Mivel a háló tanítása önmagában is iteratív eljárás, legtöbbször célszerű az egész rendelkezésünkre álló adatbázison elvégezni a szükséges transzformációkat, hogy közvetlenül a háló bemenetére kapcsolt, illetve a kívánt kimenetként felhasznált adatokhoz jussunk. On line tanítás esetén erre nincs lehetőség, a transzformációkat pontonként kell elvégeznünk. Ki kell emelnünk, hogy ellenőrzött tanítás esetén az utófeldolgozást jelentő transzformáció inverzét is elő kell tudnunk állítani, hogy a mért kimeneti értékekből a háló kimenetére értelmezett mennyiségekhez jussunk. Többek között ezért is ritkább az utófeldolgozás alkalmazása, mint a szinte mindig jelenlévő előfeldolgozásé.

2.1. Adatok hihetőségvizsgálata

Gyakorlati problémák megoldása során – különösen, ha zajos ipari környezetben mért, esetleg kézzel regisztrált adatok adják a mintákat – nagy jelentőségű az adatok hihetőségének vizsgálata. Erre lehetőséget nyújt többek közt az, hogy a valós környezetben vizsgált fizikai folyamatok általában természetes módon adnak alsó és felső korlátokat egy-egy mennyiségre. (Pl. egy tartály mérete automatikusan korlátozza a beletölthető anyag térfogatát, a mérőeszköz érzékenysége pedig alsó korlátot ad a mért mennyiségre, a méréshatára pedig a legnagyobb mérhető értékre stb.) A nyers adatbázisban az összes paramétert meg kell vizsgálni, hogy a hihetőségi határok közé esnek-e. Érdemes továbbá a technológiát ismerő szakemberekkel megvizsgáltatni a paraméterek mért értékének eloszlását (hisztogramját) a várt elvi eloszláshoz viszonyítva.

Amennyiben az egyes mért paraméterek között összefüggés mutatható ki, akkor a paraméterek együttes hihetőségvizsgálatát is el kell végezni, előfordulhat ugyanis, hogy külön-külön minden paraméter értéke hihető, de együtt mégis hibára mutatnak. (Az emberek magassága 150 és 210 cm között hihető, ugyancsak hihető, hogy

a súlyuk 35 kg és 200 kg között van, mégis a 210 cm-es 35 kg-os ember nagyon valószínűtlen.) Az adatok között található összefüggések vizsgálatára a 13.4 pontban még visszatérünk.

További vizsgálatok végezhetők a neurális modell kialakítása, tanítása során, amikor a rosszul modellezett (kiugró) adatokat megvizsgálhatjuk hihetőség szempontjából is, és ennek figyelembevételével is szűrhetjük az adatbázist.

2.2. Az adatok normalizálása

Az előfeldolgozás legelterjedtebb, gyakorlatilag szinte mindig alkalmazott formája a bemeneti adatok egyszerű, lineáris skálázása. Különösen fontos ez a lépés, ha a bemeneti adatvektor jellegükben eltérő komponensekből áll, melyek több nagyságrenddel különböző értékeket vehetnek fel. (Pl. egy konkrét ipari folyamat modellezésénél – a 7. fejezetben említett LD-konverteres acélgyártási eljárás modellezésénél – közel azonos fontosságú paraméterként, bemeneti adatként szerepelt egy adalékanyag tömeg, melynek tipikus értéke 4000-8000 kg volt, és egy anyag mangán tartalma, melynek értéktartománya a 0,05-0,1% tartományba esett. A hálók egy része – bemeneti rétegének megfelelő súlyaival – elvileg kompenzálni képes ezt a nagyságrendi eltérést, de a tanulás konvergencia sebessége olyan szintre csökkenhet, mely a megoldás elérését gyakorlatilag lehetetlenné teszi.)

Az adatnormalizálást legtöbbször akkor is el kell végeznünk, ha a bemeneti paraméterek mind azonos nagyságrendbe esnek. Például egy MLP háló a szokásos (kis véletlen értékekkel történő) inicializálás után nem – vagy csak nagyon nehezen – képes normalizálatlan értékeket megtanulni, ha a bemeneti értéktartomány az origótól távolabb eső, azt nem tartalmazó intervallum.

A bemeneti adatok normalizálása legtöbbször úgy történik, hogy mindegyik komponens minimális és maximális értékét a normált tartományban azonosra választjuk (tipikusan a $[0,1]$ illetve a $[-1,1]$ tartományok valamelyikét használjuk). Pl., ha az $\mathbf{x}=[x_1, \dots, x_N]$ bemeneti vektor i -edik komponensét a $[0,1]$ tartományra akarjuk normálni, akkor a következő lineáris transzformációt kell az összes mért értéken végrehajtani (a (p) felső index arra mutat, hogy az $1,2,\dots,P$ minták közül a p -dik minta komponenseiről van szó) :

$$\hat{x}_i^{(p)} = \frac{x_i^{(p)} - \min\{x_i\}}{\max\{x_i\} - \min\{x_i\}} \quad (13.2)$$

Másik szokásos eljárás, hogy valamennyi bemeneti paramétert a normálás során nulla várható értékűvé és egységnyi szórásúvá transzformálunk. Ehhez szükségünk van a mintahalmaz tapasztalati átlagára és szórására komponensenként. A tapasztalati átlag az alábbi egyenlettel számítható:

$$\bar{x}_i = \frac{1}{P} \sum_{p=1}^P x_i^{(p)} \quad (13.3)$$

A szórás becslése:

$$\hat{\sigma}_i^2 = \frac{1}{P-1} \sum_{p=1}^P \left(x_i^{(p)} - \bar{x}_i \right)^2 \quad (13.4)$$

Ezen értékek felhasználásával az i -edik komponens nulla várható értékűre és egységnyi szórásúra normált értékei a következő összefüggéssel állíthatók elő (megjegyezzük, hogy ez a megállapítás csak közelítőleg igaz, mivel a normalizálásnál mind a várhatóérték, mind a szórás becsült értékével dolgozunk):

$$\hat{x}_i^{(p)} = \frac{x_i^{(p)} - \bar{x}_i}{\hat{\sigma}_i} \quad (13.5)$$

A leggyakrabban használt MLP hálózatok esetén a fenti normálásnak megfelelő lineáris transzformáció tulajdonképpen a tanulás során is kialakulhat a bemeneti lineáris rétegben, de a tapasztalatok szerint célszerű a normálást egy előfeldolgozási lépésben elvégezni. Ugyanakkor pl. az RBF hálózatoknál az adat normálás különösen nagy jelentőségű, hiszen a neuronok aktiválása legtöbbször nem az egyes bemeneti komponensek

súlyozott összegével történik, hanem az euklideszi távolság alapján. Ha a bemeneti vektor N dimenziós, akkor az alap RBF háló k -edik neuronját aktiváló érték a p -edik minta esetén:

$$d_k^{(p)} = \|\mathbf{x}^{(p)} - \mathbf{c}_k\| = \sqrt{\sum_{i=1}^N \left(x_i^{(p)} - c_{ki}\right)^2} \quad (13.6)$$

Látható (13.6)-ból, hogy ha az $\mathbf{x}$ bemeneti vektor komponensei nagyságrendekkel eltérő értékeket vesznek fel, akkor a kis értékekkel rendelkező paraméterekre a hálózat nagyon érzéketlen lesz. Mivel – szemben az MLP hálózattal – itt nem találunk bemeneti lineáris súlyozó réteget, az RBF háló nem képes a tanulás során kompenzálni ezt a problémát.

A fent bemutatott lineáris skálázási eljárásokban a bemeneti $\mathbf{x}$ vektor komponenseit egyenként, egymástól függetlenül normáltuk. Lehetőség van a vektor elemeinek együttes normálására, melynek során az esetleg fellépő keresztkorrelációkat is figyelembe vesszük.

$$\bar{\mathbf{x}} = \frac{1}{P} \sum_{p=1}^P \mathbf{x}^{(p)} \quad (13.7)$$

$$\mathbf{C} = \frac{1}{P-1} \sum_{p=1}^P \left(\mathbf{x}^{(p)} - \bar{\mathbf{x}}\right) \left(\mathbf{x}^{(p)} - \bar{\mathbf{x}}\right)^T \quad (13.8)$$

Írjuk fel az $\mathbf{C}$ kovariancia mátrix sajátérték-sajátvektor egyenletét:

$$\mathbf{C}\boldsymbol{\varphi}_j = \lambda_j \boldsymbol{\varphi}_j \quad (13.9)$$

és definiáljuk az alábbi mátrixokat:

$$\boldsymbol{\Phi} = [\boldsymbol{\varphi}_1, \dots, \boldsymbol{\varphi}_N] \quad (13.10)$$

$$\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_N) \quad (13.11)$$

Ezen mátrixok segítségével a következő lineáris transzformációt hajthatjuk végre:

$$\tilde{\mathbf{x}}^{(p)} = \boldsymbol{\Lambda}^{-1/2} \boldsymbol{\Phi}^T \left(\mathbf{x}^{(p)} - \bar{\mathbf{x}}\right) \quad (13.12)$$

Az eredmény egy olyan adathalmaz, melynek átlaga a nulla vektor, míg kovariancia mátrixa egységmátrix.

Bizonyos esetekben – elsősorban regressziós problémáknál – célszerű a bemeneti adatok mellett a kimeneti adatokat is normálni. Egyes hálónál a kimeneti adatok normálása vagy megfelelő határok közé transzformálása nemcsak célszerű, hanem kötelező is. Gondoljunk csak egy olyan MLP-re, ahol a kimeneten szigmoid nemlinearitás található. Egy ilyen háló kimenete nem vehet fel a $(-1, +1)$ (vagy logisztikus nemlinearitás esetén a $(0, 1)$) intervallumon kívül eső értékeket. Ha a kívánt válaszok ezen tartományon túllógnak, a háló képtelen a feladat megtanulására. Sőt a kívánt válaszok kissé nagyobb mértékű korlátozására is szükség van, hiszen a nemlinearitás telítődő jellege miatt az intervallumok szélső értékeit már nagyon nehezen veszik fel ezek a hálók. Így a gyakorlatban jól működő megoldást nyerünk, ha pl. tangens hiperbolikus nemlinearitás esetén a kívánt válaszokat a $[-0,8, +0,8]$ intervallumra normáljuk.

2.3. Diszkrét értékekkel reprezentált szimbolikus adatok kezelése

A hagyományos mérnöki feladatokban a bemeneti adatokat legtöbb esetben úgy kezeljük, mint amelyek egy folytonos változó valamilyen értéktartományban felvett értékeinek halmazából állnak. Így általában értelmezett a távolságfogalom és beszélhetünk arról, hogy a halmaz egyik eleme közelebb van egy adott másikhoz, mint a halmaz bármely egyéb eleméhez. Ennek megfelelően rendszerint a közeli számértékekkel reprezentált adatok hasonló szituációkat takarnak. (Például, ha egy folyamatot modellezünk, és bemeneti modellparaméter a mért

hőmérséklet, akkor a 20°C mintaérték és a 21°C mintaérték tipikusan egymáshoz inkább hasonló szituációkat takar, mint a 70°C-os mintaértékkel jellemzett helyzet.)

Előfordulnak azonban olyan feladatok is, ahol számok diszkrét halmaza segítségével szimbolikus értékek ábrázolását végezzük. Ez különösen gyakran előfordul olyan feladatoknál, ahol emberi intelligenciát igénylő problémákat akarunk megoldani, hiszen ezekben a látás, képfeldolgozás, a szimbolikus feldolgozás keveredik a numerikus jellegű feldolgozással. A számokon természetes módon értelmezett távolságfogalom ilyenkor meghamisíthatja a szituáció értelmezését. Tegyük fel pl., hogy valamilyen modellben az anyag színének is jelentősége van, bizonyos szituációkat csak így tudunk megkülönböztetni. Amennyiben a piros, barna, sárga, kék, stb. színeket rendre 1, 2, 3, 4, ... számokkal reprezentáljuk, a neurális háló tanítása során azt a feltehetően hamis információt sugalljuk, hogy a sárga közelebb van a kékhez (egyiket 3-mal, másikat 4-gyel kódoltuk), mint pl. a piroshoz (amit 1-gyel kódoltunk). Ilyenkor a dimenziószám növelését is vállalva az inkább az N -ből 1 kódolást kell alkalmaznunk (vagy a bemenetek közötti hasonlóságot tükröző metrikát kell használnunk). Tehát pl. a fenti négy szín esetén egy bemeneti változó helyett négyet használunk: [1, 0, 0, 0] jelöli a pirosat, [0, 1, 0, 0] a barnát, stb. A dimenziószám-növelés hatását kompenzálja, hogy a távolság félrevezető használatát kiküszöböltük, másrészt a bináris bemenetek (és kimenetek) jelenléte általában gyorsabb konvergenciát eredményez, mint a folytonos értéktartományban, vagy diszkrét többértékű halmazon értelmezett be- és kimenetek használata. Ez a helyzet különösen gyakori osztályozási feladatok esetén, ezért az osztályozó hálók kimenetén szinte mindig az N -ből 1 kódolást célszerű alkalmazni. (Természetesen kivételt jelent a kétértékű osztályozási feladatok egy része, amelyeknél a távolság-probléma nem lép fel, így itt a kimeneten 0/1-gyel kódolható a két osztály.)

2.4. Invariáns tulajdonságok elérését segítő előfeldolgozás

Az alkalmazások jelentős körében – elsősorban, de nem kizárólag, az alakfelismerési feladatokban – fontos lehet, hogy az adatok valamilyen transzformációjával – pl. a térbeli eltolással vagy elforgatással – szemben érzéketlen legyen az eljárásunk. Például ha postai irányítószámokat kívánunk felismerni, akkor egy adott számjegy a kijelölt négyzeten belül különböző helyeken fordulhat elő, és bizonyos határok között elfordulással és egyéb hatásokkal is számolnunk kell. Természetesen a számjegyet ezen transzformációktól függetlenül fel kell ismernünk, tehát a felismerési, osztályozási eljárásnak eltolással és bizonyos mértékű elforgatással szemben érzéketlennek kell lennie. Más feladatokban is előfordul, hogy azt szeretnénk, hogy bizonyos paraméterekre legyen érzéketlen a megoldás. Az osztályozási feladatoknál bizonyos transzformációkkal szembeni invariancia kérdését a virtuális minták generálásához kapcsolódóan az előző fejezetben már érintettük. Itt a kérdés kezelésére két további lehetőséget is megemlítünk.

Amennyiben neurális hálót alkalmazunk olyan feladatra, ahol a bemeneti minta valamilyen transzformációjával szemben invariáns feldolgozásra van szükség, három lehetőség között választhatunk:

- A neurális háló tanítását, pl. a tanító adatbázis vagy a költségfüggvény kialakítását végezzük úgy, hogy az eredmény szükséges invarianciáját biztosítsuk.
- A feldolgozási algoritmust két részre bontjuk, az első lépésben egy előfeldolgozási eljárás keretében a transzformációval szemben invariáns jellemzőket (*features*) állítunk elő, majd a második lépésben a neurális modellt ezekkel az invariáns jellemzőkkel tanítjuk.
- A neurális modellt úgy strukturáljuk, hogy elősegítsük a transzformációval szembeni invarianciát.

A fenti három eljárásból az első speciális adatelőkészítéssel, adatbázis kialakítással oldja meg a problémát, a második egy előfeldolgozási lépést iktat be, míg a harmadik eljárás tulajdonképpen nem az a fejezeten belül tárgyalt módszereket használja.

Két lehetőséget ismertettünk, amelyekkel a tanítás során invariáns eredmény elérésére törekedhetünk.

Egyik lehetőség, hogy úgy alakítjuk ki a tanító adatbázist, hogy nagy számban tartalmazzon eltolt (illetve szükség esetén elforgatott, stb.) mintákat. Ha nem képfeldolgozás a feladat, akkor csak azon paraméterben, jellemzőben különböző, egyébként azonos mintákat állítunk elő, amely paraméterre, jellemzőre érzéketlen megoldást szeretnénk kapni.) Ha nem állnak rendelkezésünkre kellő számban ilyen mért adatok, akkor a meglevő adatainkból a feladat szempontjából fontos transzformációk felhasználásával előállíthatunk eltolt/elforgatott adatvektorokat. (Ezt a lehetőséget a 12. fejezetben, a járulékos információ felhasználása kapcsán is említettük.) Ezek után a neurális modellt erre az adatbázisra alapozva tanítjuk, tulajdonképpen az adatbázisba felvett, speciális szempontok alapján válogatott/módosított adatokkal biztosítva a cél elérését. A

módszer alkalmazásánál azonban óvatosan kell eljárni, mert az adatbázis nem megfelelő kibővítése, a gyakorlatban sokszor igen rossz eredményeket ad (lassú konvergencia, rossz teszt eredmények, stb.).

Másik lehetőség, hogy a regularizációhoz hasonlóan (lásd 2. fejezet) az optimalizálni kívánt költségfüggvényben vesszük figyelembe a kívánt invariáns tulajdonság hiányát vagy meglétét. Tehát a tanító mintákra való illeszkedést leíró hiba helyett a C_r összetett, regularizált költségfüggvény minimalizálását végezzük:

$$C_r = C + \lambda C_i \quad (13.13)$$

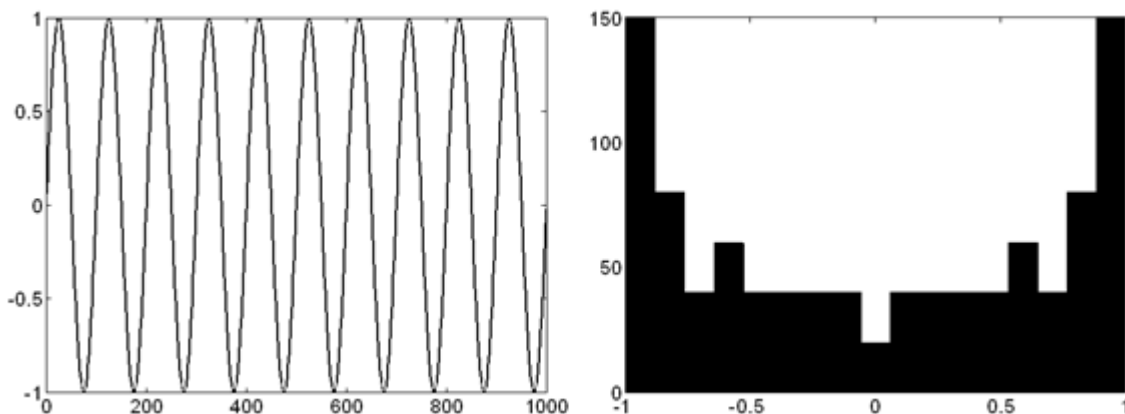
ahol C a szokásos – a tanító minták illeszkedési hibájára jellemző – költségfüggvény tag, C_i az invariancia hiányát leíró költségtag, λ a regularizációs együttható.

3. Kilógó adatok

A hiányzó adatok mellett igen sok problémát okoznak a zajok vagy a mérési hibák miatt fellépő, a valós adatoktól erősen eltérő, úgynevezett kilógó adatok (*outliers*) is. Ebben az alfejezetben alapvetően az adatok minősítésével foglalkozunk; hogyan lehet felismerni, modellezni, hogy az adathalmazból melyeket tekintünk kilógóknak, melyek tekinthetők "csupán" a szokásos módon zajosnak. A következő alfejezetben ismertetett, hiányzó adatok pótlására szolgáló eljárások alkalmasak lehetnek arra, hogy javítsuk az adatbázisunkat. A detektált kilógó adatokat ugyanis egyszerűen úgy tekinthetjük, hogy ezekben a pontokban hiányzik a releváns információ.

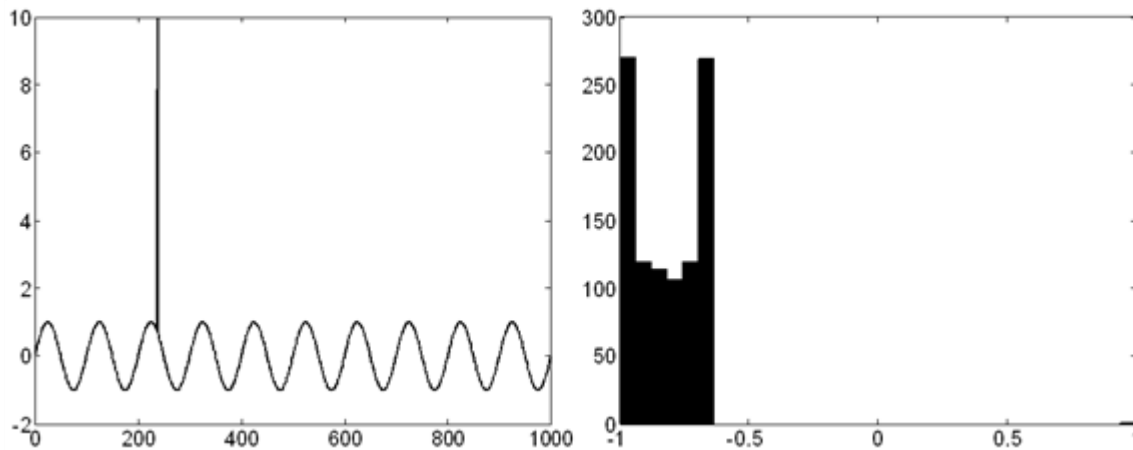
Kézi adatrögzítés esetén tipikus "mérési hiba" az adatrögzítés hibája, pl. rossz helyre tett tizedesvessző. A hatásuk több szempontból is káros. Egyrészt a tanítás folyamatában az impulzus zajoknál (lásd 13.1.1) ismertetett eredményre vezethetnek (a modellben általában nem impulzusszerű, de egy tágabb környezetben elkenet torzítást okozhatnak). Másrészt tönkreteszhetik az előfeldolgozásnál nagyon gyakran alkalmazott és rendkívül hasznos normalizálási lépést. Az 13.6 ábrán egy zajmentes, $[-1,1]$ tartományra normált szinuszjel és hisztogramja látható.

13.6. ábra - Egy $[-1,1]$ -re normalizált szinuszjel, és hisztogramja



A következő ábra azt mutatja, hogy ha valamilyen okból a jel egyetlen pontjában egy erősen hibás (kilógó) értéket mérünk, akkor az a normalizáláskor összenyomja a hasznos jeltartományt. Ha ezzel a normalizált paraméterrel tanítunk egy hálót, akkor azt "sugalljuk", hogy lényegében $+1$ és -1 értékek fordulnak elő ezen a bemeneten (a $+1$ -es érték csak egyszer, a -1 körüli értékek sokszor), mintha bináris jelünk lenne. Természetesen a háló képes lehet ezt kiküszöbölni, de minden "félrevezetés" rontja a megoldás minőségét és a tanulás konvergencia sebességét.

13.7. ábra - Egyetlen kilógó értékkel terhelt szinuszjel $[-1,1]$ -re normalizálva, és a normált jel hisztogramja



3.1. Kilógó értékek modellezése klaszterezéssel, az EM algoritmus

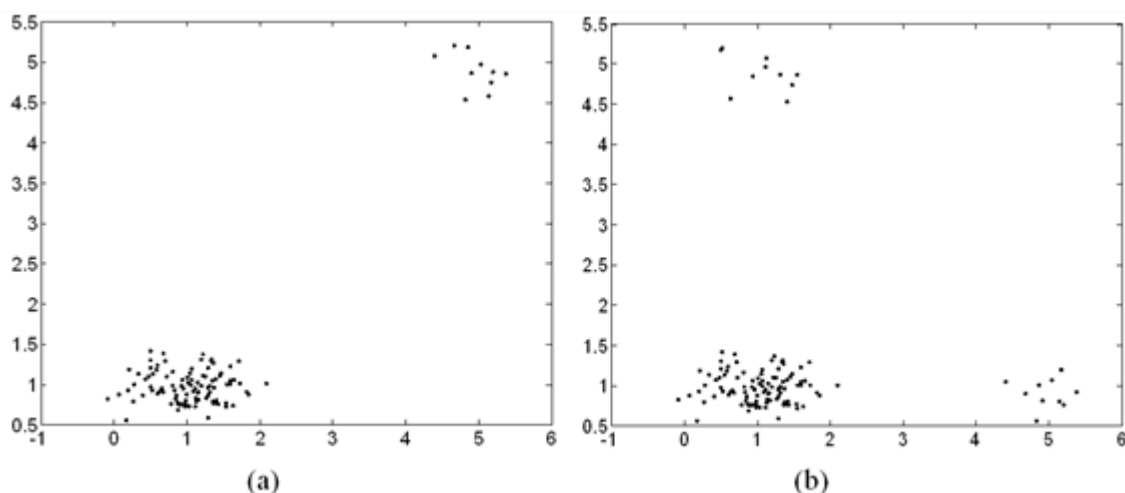
A kilógó értékek káros hatását csökkentő eljárások alapvető problémája, hogy honnan vesszük észre egy adatról, hogy az alapvetően használt jelmodellünknek megfelel-e, ami lehet valamilyen szokásos mértékben zaj által torzított adat, vagy valamilyen rendkívüli, nagymértékű torzítás történt.

Az egyik lehetőség az adatok nemellenőrzött tanítással történő osztályozása, klaszterezése. (Ellenőrzött tanítás nem jöhet szóba, hiszen nem tudjuk az egyes adatokról, hogy melyik csoportba tartoznak.)

Feltételezhetjük, hogy ha a torzítás nagymértékű, akkor az adatok osztályozása során a kilógó adatok a tipikusaktól eltérő klaszterbe kerülnek. Elvileg nincs más dolgunk, mint klaszterezni és eldönteni, hogy melyik klaszter tartalmazza a normál és melyik vagy melyek a kilógó adatokat. Általában azzal az egyszerű feltételezéssel élünk, hogy a normál adatok vannak többségben, és a nagy elemszámú csoportokat (klasztereket) normálként kezeljük. Ugyanakkor a helyzet különösen problematikus akkor, ha nagyon összetett (sokdimenziós) adatokat használunk, és a kilógó adat sokféleképpen jöhet létre. A következő ábrán demonstráljuk a problémát egy kétváltozós adathalmazon.

Látható, hogy ha a kétváltozós adathalmazunk mindkét paraméterére egyszerre hat a nagymértékű torzítás, amely kilógó adatokat eredményez, akkor két klaszter jött létre: egy a normál adatokból és egy a kilógókból. Amennyiben a torzítás hol ezt, hol azt a paramétert éri (pl. hol az egyik, hol a másik adatot regisztrálják hibásan), akkor 3 klaszter alakul ki. Ez azért fontos kérdés, mert a klaszterezési eljárásaink nagy része kötött, előre rögzített számú klaszter esetén működik jól. Ha felülbecsüljük a klaszterek számát, és pl. a 13.8 (a) ábrában 3 klasztert keresünk, akkor fennáll a veszély, hogy a normál adatok egy részét is leválasztjuk és kilógónak tekintjük. Ha viszont alulbecsüljük a klaszterek számát, pl. a 13.8 (b) ábrában 2 klasztert keresünk, akkor jó eséllyel az egyik kilógó csoportot normálnak fogjuk tekinteni.

13.8. ábra - A kilógó adatokat mindkét paraméterre egyszerre (a), illetve külön-külön ható (b) torzítás hozta létre (mindkét ábrán a vízszintes tengelyen az egyik, a függőleges tengelyen a másik paraméter értéke látható)



A feladatunk tehát abban áll, hogy önszerveződő osztályozással, klaszterezéssel eltérő paramétereloszlású csoportokat keressünk, és ennek alapján döntjük el, hogy melyik adat kilógó és melyik normál adat.

3.1.1. Az EM-algoritmus

Az EM-algoritmust (*expectation-maximization algorithm*) először általánosságban tárgyaljuk, majd konkretizáljuk egy olyan feladatra, amely kevert minták szétválasztását célozza¹.

Általánosságban az EM-algoritmust olyan feladatok megoldására fejlesztették ki, amelyekben ismeretlen, de konkrét, elvileg rögzített paraméterek becslését szeretnénk meghatározni valamilyen adathalmaz alapján úgy, hogy bizonyos adatelemeket nem ismerünk (pl. nem tudjuk ezeket mérni), csak valószínűség-eloszlásukról van információnk [Dem77], [Mit97].

Az EM-algoritmus szemléltetésére több példa is felhozható. Első példánk legyen a következő: meg akarjuk becsülni egy sokdimenziós adathalmaz átlagát úgy, hogy az adataink hiányosak, bizonyos adatvektorok egy vagy több komponense hiányzik. (Pl. egy – valamilyen elektromos vagy elektronikus berendezésekből álló – termékhalmazon mérték a termék áramfelvételét, két meghibásodás közötti időt stb., de néhány termékénél egyik-másik adat hiányzik, elveszett stb.) Egy másik lehetséges példa, amelyet későbbiekben részletesebben is tárgyalunk a következő: van egy adatsorunk, melyben az adatok több eloszlásból származhatnak. Az adatok alapján szeretnénk megbecsülni az egyes eloszlások várható értékét, de anélkül, hogy tudnánk, hogy az egyes adatok melyik eloszlásból származnak. Egy hétköznapi példaként gondoljunk arra, hogy az amerikaiak magasság, vérnyomás stb. adatai alapján azt szeretnénk megmondani, hogy az amerikaiakon belül az egyes népcsoportok (fehér, néger, mexikói, ázsiai) átlagmagassága, vérnyomásának átlaga és szórása stb. mekkora. Mindezt azonban úgy szeretnénk meghatározni, hogy az egyes méréseknél nem lett feljegyezve, hogy ki melyik népcsoportba tartozik. Van tehát egy olyan adathalmazunk, mely pl. a magasságértékeket tartalmazza, de az egyes értékek mellett nincs meg az az információ, hogy melyik magasságérték melyik népcsoportba tartozó ember magassága, vagyis hiányos adatokkal van dolgunk. Itt az ismeretlen paraméter, a hiányzó adat az, hogy melyik adat melyik csoportból származik, a keresett paramétervektor pedig a csoportok átlagos magasságából képzett vektor. (Természetesen ennek a demográfiai illetve népegészségügyi példának számos műszaki analógiáját lehet találni, de ez talán egy könnyen érthető példa.)

Az EM-algoritmust általános formájában a következőképpen szokták megfogalmazni (pl. [Mit97]). Egy valószínűség eloszlás valamilyen θ paramétervektorát akarjuk becsülni csupán részlegesen megfigyelhető adatok alapján. Legyen $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L\}$ a megfigyelt adatkomponensek halmaza és $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_L\}$ azon adatkomponensek halmaza, amelyeket nem tudtunk megfigyelni. Tehát az $Y = X \cup Z$ lenne a teljes adathalmaz, és ha mind X , mind Z ismert volna a feladatot könnyen meg lehetne oldani. Minthogy Z -t nem ismerjük, ezt hiányzó ismeretnek nevezzük. A nem ismert Z komponenseket valószínűségi változóként kezelhetjük, amelyek eloszlása az ismeretlen θ paramétervektortól függ. Hasonlóképpen Y is valószínűségi változó, hiszen X és Z segítségével definiáltuk. A Z hiányzó adatokat jellemző valószínűség eloszlás függ a keresett paramétértől. Ha ismernénk Z valószínűség eloszlását, akkor az X megfigyelt adatok alapján a θ paramétervektor becslését meg tudnánk határozni. Ugyanis a θ paramétervektor becsléséhez Z teljes ismerete nem szükséges Z várható értéke

¹ Ebben a szakaszban a valószínűségek jelölésére használt P helyett a mintaszámot ismét L -lel jelöljük

alapján a becslés meghatározható (Meg tudnánk fogalmazni egy maximum likelihood becslési feladatot, melynek megoldásaként θ olyan becslését határozhatnánk meg, mely mellett a megfigyeléseink a legnagyobb valószínűségűek.) A θ becslés segítségével a hiányzó adatok valószínűség eloszlását pontosabban meg tudjuk adni, ami (a maximum likelihood becslésen keresztül) a θ paraméter jobb becslését eredményezi. Az eljárást folytatva a paramétervektor egyre jobb becslését kapjuk. Megfelelő feltételek mellett ez az iteratív eljárás konvergens és a θ becslések sorozata a valódi θ paraméterhez konvergál [Dem77].

Általánosabban az eljárást a következőképpen fogalmazhatjuk meg. Van egy kiinduló hipotézisünk az ismeretlen paramétréről. Jelöljük ezt h -val. A h aktuális hipotézis megad egy aktuálisan igaznak tartott paramétervektort. Egy iteráció során egy új hipotézist, h' -t állítunk fel, mely a paramétervektor egy jobb

becslését adja. Olyan új hipotézist keresünk, mely mellett a (teljes) megfigyeléseinkre vonatkozó $P(Y|h')$ feltételes sűrűségfüggvény maximumát a tényleges adatainknál veszi fel. Ez egy maximum likelihood becslési feladat lenne, ahol a feltételes sűrűségfüggvény a likelihood függvény. Azonban mivel az Y teljes adathalmazt nem ismerjük, a $P(Y|h')$ likelihood függvény maximumát biztosító hipotézist sem tudjuk meghatározni.

Meghatározhatjuk azonban a $P(Y|h')$ likelihood függvény várhatóértékét $E\{P(Y|h')\}$ -t, és a várhatóérték

alapján fogalmazhatunk meg egy maximum likelihood becslési feladatot. Az $E\{P(Y|h')\}$ meghatározásához az Y teljes adatok ismeretére nincs szükségünk, elegendő, ha Y eloszlása ismert, ami azt jelenti, hogy a hiányzó adatok ismerete nélkül, de a hiányzó adatok eloszlásának ismeretében a várhatóérték likelihood függvény megfogalmazható. A várhatóértéket az Y -t generáló valószínűségi eloszlás felett képezzük, amelyet viszont a θ paramétervektor határoz meg. Az aktuális hipotézis (az ehhez tartozó aktuális paraméterbecslés) alapján

meghatározható a várhatóérték likelihood függvény, $E\{P(Y|h')\}$, majd az ennek maximumát biztosító új

hipotézis. (Legtöbbször praktikus okokból e helyett az $E\{\log(P(Y|h'))\}$ -t maximáljuk, de ez a lényegen nem változtat, a két kifejezés maximuma egybeesik.)

Definiáljunk egy $Q(h'|h)$ függvényt, ami $E\{\log(P(Y|h'))\}$ -t mint h' függvényét adja meg azzal a feltételezéssel, hogy az aktuális hipotézisünk, h szerint a paramétervektor értéke θ , valamint, hogy az adataink megfigyelhető része X :

$$Q(h'|h) = E\{P(Y|h') \mid h, X\} \quad (13.14)$$

Ezekután az EM-algoritmus az alábbi két lépést ismétli, amíg a konvergencia be nem következik:

E-lépés: várhatóérték-képzés (expectation). Meghatározzuk a $Q(h'|h)$ függvényt a jelenlegi h hipotézis (θ paramétervektor) és a megfigyelt X adathalmaz alapján, az Y felett nyert valószínűségi eloszlás várható értékét képezve:

$$Q(h'|h) \leftarrow E\{P(Y|h') \mid h, X\} \quad (13.15)$$

M-lépés: maximumkeresés (maximization). A h hipotézist javítjuk, megkeressük azt a h' -t (a paramétervektor θ' új becslését), amely maximálja az E -lépés során nyert Q -t:

$$h \leftarrow \underset{h'}{\operatorname{argmax}} Q(h'|h) \quad (13.16)$$

Az M -lépés után az új paramétervektorral újra meghatározzuk az Y feletti eloszlás (amely változott a paramétervektor változása miatt) várható értékét, majd újra maximáljuk a Q -t stb.

Az általános bemutatás után szemléltessük az EM-algoritmust a fent már említett, legismertebb alkalmazási példájával, a kevert eloszlások paraméterbecslésével. Tegyük fel, hogy L darab többdimenziós mintánk van $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L\}$, amelyek K különböző eloszlásból származnak a következő módon. Először valamilyen valószínűséggel eldől, hogy melyik eloszlásból jön a következő minta (bár mi ezeket a valószínűségeket nem

ismerjük). Azt az eseményt, hogy a k -dik eloszlásból származik a következő minta, jelölje ω_k , ennek a priori bekövetkezési valószínűsége $P(\omega_k)$. Ezt követően az adott, k -dik eloszlásra jellemző $p(\mathbf{x}|\omega_k, \boldsymbol{\theta}_k)$ valószínűségeloszlás alapján előáll az adott érték. A $\boldsymbol{\theta}_k$ $k = 1, 2, \dots, K$ ismeretlen paramétervektorok együtt adják az összetett eloszlás $\boldsymbol{\theta}$ ismeretlen eredő paramétervektorát, amely meghatározza a minták eredő eloszlását.

Egy adott – bár ismeretlen – paramétersor (paramétervektor) esetén a kevert eloszlást a következő összefüggés írja le:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^K p(\mathbf{x}|\omega_k, \boldsymbol{\theta}_k) P(\omega_k) \quad (13.17)$$

A rögzített, de ismeretlen $\boldsymbol{\theta}$ mellett a mért adathalmazunk valószínűségét a következő összefüggés, az úgynevezett likelihood függvény adja meg, ami nem más, mint az egyes értékek valószínűségeinek szorzata:

$$p(X|\boldsymbol{\theta}) = \prod_{j=1}^L p(\mathbf{x}_j|\boldsymbol{\theta}) \quad (13.18)$$

A $\boldsymbol{\theta}$ ismeretlen paramétervektor maximum likelihood (ML) becslése az a $\hat{\boldsymbol{\theta}}$ becslő, amely a $p(X|\boldsymbol{\theta})$ -t maximálja.

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} p(X|\boldsymbol{\theta}) \quad (13.19)$$

Ez az összefüggés azt fejezi ki, hogy a ténylegesen mért adatsor a különböző $\boldsymbol{\theta}$ -k esetén különböző valószínűségű, és érthető módon azt a $\boldsymbol{\theta}$ -t választjuk ezek közül, amely mellett a mért adatsor a legvalószínűbb. Tegyük fel, hogy $p(X|\boldsymbol{\theta})$ differenciálható $\boldsymbol{\theta}$ szerint, jelöljük a likelihood logaritmusát $LIKELIHOOD$ -lel, a logaritmikus likelihood függvény $\boldsymbol{\theta}_i$ szerinti gradiensét pedig $\nabla_{\boldsymbol{\theta}_i}(LIKELIHOOD)$ -lel, ekkor:

$$LIKELIHOOD = \sum_{j=1}^L \log(p(\mathbf{x}_j|\boldsymbol{\theta})) \quad (13.20)$$

és

$$\nabla_{\boldsymbol{\theta}_i}(LIKELIHOOD) = \sum_{j=1}^L \frac{1}{p(\mathbf{x}_j|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}_i} \left[\sum_{k=1}^K p(\mathbf{x}_j|\omega_k, \boldsymbol{\theta}_k) P(\omega_k) \right] \quad (13.21)$$

Tegyük fel, hogy az összes $\boldsymbol{\theta}$ és $\boldsymbol{\theta}_k$ paramétervektorok elemei függetlenek egymástól, amennyiben $i \neq k$. Mivel $P(\omega_i)$ nem függ $\boldsymbol{\theta}_i$ -től:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}_i}(LIKELIHOOD) &= \sum_{j=1}^L \frac{1}{p(\mathbf{x}_j|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}_i} p(\mathbf{x}_j|\omega_i, \boldsymbol{\theta}_i) P(\omega_i) = \\ &= \sum_{j=1}^L \frac{p(\mathbf{x}_j|\omega_i, \boldsymbol{\theta}_i) P(\omega_i)}{p(\mathbf{x}_j|\boldsymbol{\theta})} \frac{1}{p(\mathbf{x}_j|\omega_i, \boldsymbol{\theta}_i)} \nabla_{\boldsymbol{\theta}_i} p(\mathbf{x}_j|\omega_i, \boldsymbol{\theta}_i) \end{aligned} \quad (13.22)$$

$$\frac{P(\mathbf{x}_j | \omega_i, \boldsymbol{\theta}_i)}{P(\mathbf{x}_j | \omega_i, \boldsymbol{\theta}_i)} = 1$$

A kifejezést bővítettük - gyel. A Bayes tétel alapján a szumma utáni első tényező nem más, mint az esemény a posteriori valószínűsége. Tehát

$$P(\omega_i | \mathbf{x}_j, \boldsymbol{\theta}_i) = \frac{P(\mathbf{x}_j | \omega_i, \boldsymbol{\theta}_i) P(\omega_i)}{P(\mathbf{x}_j | \boldsymbol{\theta})} \quad (13.23)$$

Ez annak valószínűségét adja meg, hogy a megfigyelt $\mathbf{x}_j$ minta az i -edik eloszlásból származik adott $\boldsymbol{\theta}$ (és ezen belül $\boldsymbol{\theta}_i$) mellett. Ezek után az (13.22) egyenlet a következő alakra hozható:

$$\nabla_{\boldsymbol{\theta}_i} (LIKELIHOOD) = \sum_{j=1}^L P(\omega_i | \mathbf{x}_j, \boldsymbol{\theta}_i) \nabla_{\boldsymbol{\theta}_i} \log(P(\mathbf{x}_j | \omega_i, \boldsymbol{\theta}_i)) \quad (13.24)$$

Mikor elérjük a maximumot, akkor a gradiens 0-vá válik, tehát az ML becslés ki kell elégítse a következő összefüggéseket:

$$\sum_{j=1}^L P(\omega_i | \mathbf{x}_j, \hat{\boldsymbol{\theta}}_i) \nabla_{\boldsymbol{\theta}_i} \log(P(\mathbf{x}_j | \omega_i, \hat{\boldsymbol{\theta}}_i)) = \mathbf{0} \quad i = 1, \dots, K \quad (13.25)$$

Tehát ezen egyenletrendszer megoldásai között kell keressük az ML becslőt.

Ha tudnánk, hogy melyik minta melyik eloszlásból származik, akkor az ismeretlen paramétereket (pl. az eloszlások várható értékét, szórását stb.) viszonylag egyszerűen tudnánk becsülni. Ezek viszont a nem megfigyelt, nem megfigyelhető változók. Így viszont az alábbi nemlineáris egyenletrendszert kéne megoldanunk (amennyiben sem az a priori eloszlások, sem az egyes $\boldsymbol{\theta}_i$ eloszlásparaméterek nem ismertek):

$$\hat{P}(\omega_i) = \frac{1}{L} \sum_{j=1}^L \hat{P}(\omega_i | \mathbf{x}_j, \hat{\boldsymbol{\theta}}_i) \quad (13.26)$$

$$\hat{P}(\omega_i | \mathbf{x}_j, \hat{\boldsymbol{\theta}}_i) = \frac{P(\mathbf{x}_j | \omega_i, \hat{\boldsymbol{\theta}}_i) \hat{P}(\omega_i)}{\sum_{k=1}^K P(\mathbf{x}_j | \omega_k, \boldsymbol{\theta}_k) \hat{P}(\omega_k)} \quad (13.27)$$

$$\sum_{j=1}^L \hat{P}(\omega_i | \mathbf{x}_j, \hat{\boldsymbol{\theta}}_i) \nabla_{\boldsymbol{\theta}_i} \log(P(\mathbf{x}_j | \omega_i, \hat{\boldsymbol{\theta}}_i)) = \mathbf{0} \quad i = 1, \dots, K \quad (13.28)$$

Foglaljuk össze, hogy mit ismerünk, és mit kívánunk megbecsülni ebben az egyenletrendszerben. Két dolgot ismerünk:

- az $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L\}$ mintahalmazt,

az egyes eloszlások struktúráját (legtöbbször pl. azt tudjuk vagy feltételezzük, hogy normál eloszlásról van szó).

Ezt fejezi ki az egyenletrendszerben a $P(\mathbf{x}_j | \omega_i, \hat{\boldsymbol{\theta}}_i)$ szerepeltetése, amely $\boldsymbol{\theta}_i$ paraméterekkel jellemezhető.

Nem ismerjük:

- az egyes eloszlások a priori valószínűségeit (ezért használunk az összefüggésekben $\hat{P}(\omega_i)$ -t, ami a valószínűség becsült voltára utal),

- az egyes eloszlások paramétereit (erre utal a $\hat{\theta}$ használata),

nem tudjuk, hogy melyik minta melyik eloszlásból származik – ez tulajdonképpen az az adat, amit nem tudunk megfigyelni –, és természetesen nem ismerjük az a posteriori valószínűségeket sem ($\hat{P}(\omega_i | \mathbf{x}_j, \hat{\theta}_i)$).

Az egyenletrendszer gyakran nem oldható meg vagy a megoldás nagyon nagy számításigényű. Ennek megfelelően az EM-algoritmus iteratív megoldást kínál, a következő két lépés ismételt alkalmazásával. Első lépésben a (13.26) és a (13.27) összefüggések alapján, a pillanatnyilag használt θ segítségével kiszámítjuk az a

posteriori valószínűség eloszlást, és a várható értékét képezzük – $E\left\{\hat{P}(\omega_i | \mathbf{x}_j, \hat{\theta}_i)\right\}$ (expectation-lépés). Ezek után az így kiszámított várható értékeket írjuk be a (13.28) logaritmikus likelihood egyenletbe, és megkeressük azt az új θ értéket, amely maximálja a logaritmikus likelihood-ot (maximization-lépés). Ezt a két lépést ciklikusan valamilyen kezdeti értékekből kiindulva, valamilyen leállási feltétel eléréséig folytatjuk. A kezdetiérték választás, a konvergencia, a leállási feltételek kérdéseinek tárgyalása meghaladják a jelen könyv kereteit.

A legtöbbször vizsgált eset, amikor az egyes részeloszlások Gauss-jellegűek. Ilyenkor a következő – nagyon logikus – eredményre jutunk az ismeretlen átlagok (és szórások) becslésénél. Az eredményeket – a jobb áttekinthetőség kedvéért – skalár részeloszlások esetére közöljük, amikor csak az eloszlások várható értéke (m_i) ismeretlen, a szórásuk azonos és ismert. (Természetesen az is ismeretlen, hogy a keverékből melyik adat melyik eloszláshoz tartozik.)

$$E\left\{\hat{P}(\omega_i | x_j, \hat{\theta}_i)\right\} = \frac{e^{-\frac{(x_j - m_i)^2}{2\sigma^2}}}{\sum_r e^{-\frac{(x_j - m_r)^2}{2\sigma^2}}} \quad (13.29)$$

$$m_i = \frac{\sum_j E\left\{\hat{P}(\omega_i | x_j, \hat{\theta}_i)\right\} x_j}{\sum_j E\left\{\hat{P}(\omega_i | x_j, \hat{\theta}_i)\right\}} \quad (13.30)$$

Látható, hogy ha tudnánk, hogy melyik minta melyik részeloszlásból származik, akkor visszakapnánk a várható érték becslésére a jól ismert átlagértéket – azon értékek átlagát, amelyek az adott részeloszláshoz tartoznak. (Az $E\left\{\hat{P}(\omega_i | x_j, \hat{\theta}_i)\right\}$ várhatóértékek 1, illetve 0 értéket vennének fel, attól függően, hogy az adott mérés a kiválasztott eloszláshoz tartozik-e vagy sem.)

4. Hiányzó adatok

Gyakorlati feladatok megoldásánál a mérőeszközök, illetve a technológia hibái miatt a mért adatok hiányosak lehetnek. (Természetesen a hiánnyal azonosan kezelendő problémára vezet, ha az adat előállt ugyan a mérés során, de értéke nem hihető.) Az egyszerűbb tárgyalhatóság kedvéért tegyük fel, hogy egy folyamat P mért adatából állítjuk elő a skalár kimenet becslését neurális modellünk segítségével. Ha ellenőrzött tanítással alakítjuk ki a modellt, akkor többféle megoldandó probléma fordulhat elő (azt feltételezzük, hogy a modell több bemenettel, de egyetlen kimenettel rendelkezik):

- a bemeneti adatsor teljesen hiányzik,
- a kimeneti adat hiányzik,
- a bemeneti adatsor egyes elemei hiányoznak.

Vizsgáljuk meg melyik hiba milyen problémát eredményez a tanítás során. (A tanított hálózat használata során a bemeneti adatok hiánya azonos jellegű problémákat vet fel, mint a tanítás során. A kimeneti adat hiányának viszont az ellenőrzött tanítás során nincs értelme, ez esetben nem tudunk tanítani ezzel a mintával. Ez alól kivétel a félig ellenőrzött tanulás, mely során épp az a cél, hogy olyan mintákat is hasznosítani tudjunk, melyeknél a kívánt válasz nem ismert. A tanított háló működtetésekor viszont természetes, hogy nincs kimeneti adat.) A feladat jellegétől függően alapvetően kétféle helyzettel kerülünk szembe. Amennyiben statikus feladattal foglalkozunk, akkor a hiányzó adatok a tanítás során csupán a modellezni kívánt mintatér lefedettségét rontják, de a többi mintára nincs hatásuk. Ha dinamikus feladattal foglalkozunk, akkor a hiányzó adatok a meglévő helyes adatok használhatóságát is zavarják. (Példa lehet a dinamikus modellezésre, ha modellünk idősorok jóslásával foglalkozik: az előző N időpontban mért értékek alapján jósolja a következő értékvektort. Egy mért érték kimaradása ilyenkor a következő N érték becslését is lehetetlenné teszi.)

Statikus eset

Statikus esetben ha a teljes bemeneti adatsor vagy a kimeneti adat hiányzik (*missing data*), akkor nem tehetünk mást, mint hogy töröljük a mintát a tanító (teszt) készletből. (Statikus eseten itt nem azt értjük, hogy a neurális modell statikus, hanem azt, hogy az egymás után generált adatsorok között nincs kapcsolat, az egymás után generált adatok korrelálatlanok.)

Amennyiben az N elemű bemeneti adatvektor egy vagy több eleme hiányzik, megkísérelhetjük ezek pótlását (*imputation*). Természetesen ez általában csak akkor kecsegtet sikerrel, ha a hiányzó elemek száma jóval kisebb N -nél. A hiányzó adat pótlására az adatsor jellegétől függően két lehetőség kínálkozik. Általában is célszerű, de ha hiányzó adatok pótlására van igény, akkor mindenképp meg kell vizsgálnunk, hogy van-e összefüggés a bemeneti adatvektor elemei között. Ezt a kovariancia mátrix vizsgálatával végezhetjük el.

A (13.7), (13.8) egyenletekkel definiált $\mathbf{C}$ kovariancia mátrix elemeiből képezhető a korrelációs együtthatók mátrixa, melynek elemei:

$$\tilde{C}(i, j) = \frac{C(i, j)}{\sqrt{C(i, i) C(j, j)}} \quad (13.31)$$

A $\tilde{C}(i, j)$ korrelációs együttható 1-hez közeli értéke az x_i és x_j komponensek közötti erős kapcsolatra utal, míg ha értéke közel 0, akkor a bemeneti vektor ezen két komponense korrelálatlannak tekinthető.

Ha egy adatkomponens egyetlen másik komponenssel sincs kapcsolatban, akkor nincs más lehetőségünk, mint hogy a hiányzó bemeneti változó értéket ugyanezen változó többi mért értéke alapján becsüljük. Tulajdonképpen a hiányzó bemeneti paraméterre konstans + zaj jelmodellt állítunk fel, és ez alapján becsüljük a hiányzó értéket. Így a hiányzó i -edik adatkomponens becslése a következő összefüggéssel történhet:

$$\hat{x}_i = \bar{x}_i + \sigma_i \xi \quad (13.32)$$

ahol $\bar{x}_i$ és σ_i az (13.3) és (13.4) összefüggéssel definiált, ξ pedig egy alkalmas (tipikusan 0 várható értékű, 1 szórású Gauss vagy egyenletes eloszlású) valószínűségi változó. Természetesen használhatunk σ_i -nél kisebb vagy nagyobb szórást is. Gyakran egyszerűen az átlagértékkel pótoljuk a hiányzó adatokat, ekkor viszont a mintahalmaz statisztikai jellemzői megváltoznak az adatpótlás hatására.

Amennyiben a korrelációs együttható mátrix bizonyos főátlón kívüli elemei 1-hez közeli értéket vesznek fel, akkor ezen elemek közötti kapcsolat lehetőséget teremt arra, hogy az egyik elem hiánya esetén a többi komponens értékéből következtethessünk a hiányzó adat valószínű értékére. Ehhez további vizsgálatokat kell végeznünk, hiszen a korrelációs együttható csak a kapcsolat meglétét mutatja, de nem mond semmit a kapcsolat jellegére. Tulajdonképpen egy regressziós problémával állunk szemben: ha az i -edik változó hiányzik, és tudjuk,

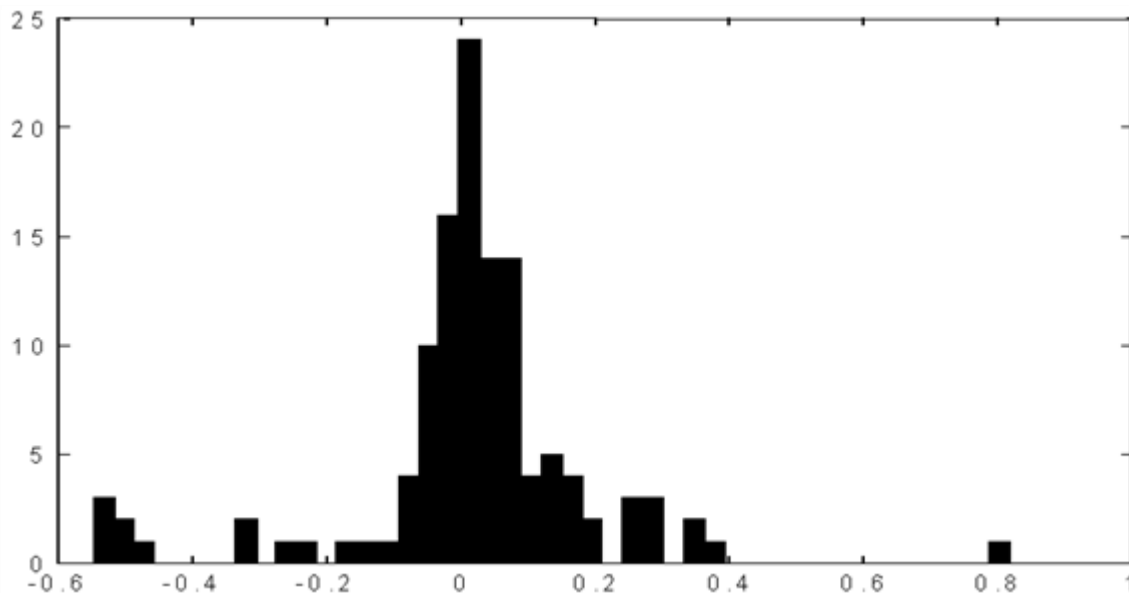
hogy az i -edik változó kapcsolatban van a j -edik változóval, akkor az $x_i^{(p)} - x_j^{(p)}$ $p = 1, 2, \dots, P$ ponthalmazra illeszkedő $x_i = \hat{f}(x_j)$ függvényt kell megkeresnünk. Amennyiben ezt a regressziós függvényt megtaláltuk, akkor a hiányzó $x_i^{(h)}$ értéket az:

$$\hat{x}_i^{(h)} = \hat{f}\left(x_j^{(h)}\right) \quad (13.33)$$

összefüggéssel becsülhetjük. Természetesen a fenti legegyszerűbb eseten kívül előfordul, hogy x_i nem csupán x_j -vel mutat szorosabb kapcsolatot, hanem közvetlenül vagy közvetve (x_i kapcsolatban van x_j -vel, x_j kapcsolatban van x_k -val, stb.) más bemeneti komponensekkel is. Ezen esetekben egy többdimenziós regressziós problémával állunk szemben, és az $\hat{f}(x_j, x_k, \dots)$ regressziós függvényt kell megtalálnunk a hiányzó adat becsléséhez.

A 13.9 ábra egy konkrét ipari folyamat modellezése során végzett vizsgálat eredményét mutatja: a felhasznált 16 bemeneti paraméterből képzett korrelációs együttható mátrix elemeinek hisztogramja látható az ábrán. (A 13.31 összefüggésből látszik, hogy a mátrix főátlójában minden elem 1 és a mátrix szimmetrikus, így az ábrán csak a főátló alatti elemek hisztogramját vettük fel.

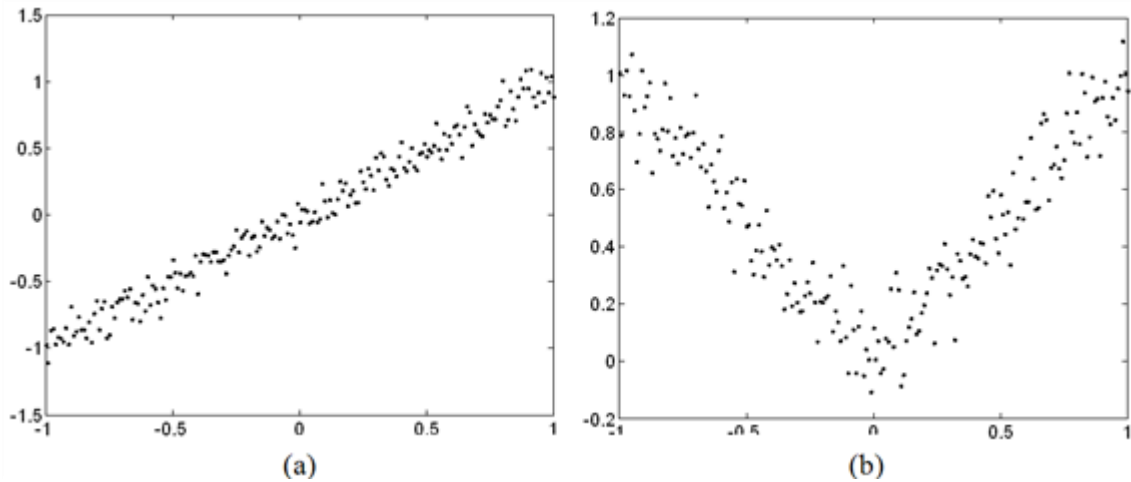
13.9. ábra - A korrelációs együttható mátrix elemeinek hisztogramja egy konkrét feladatban



Az ábrán látható, hogy a lehetséges $\binom{16}{2} = 90$ paraméterpárból nagyságrendileg 5-10 olyan pár van, amelyek erősebb korrelációt mutatnak (jelen esetben az együttható 0,5-nél nagyobb abszolút értékét vettük szignifikánsnak).

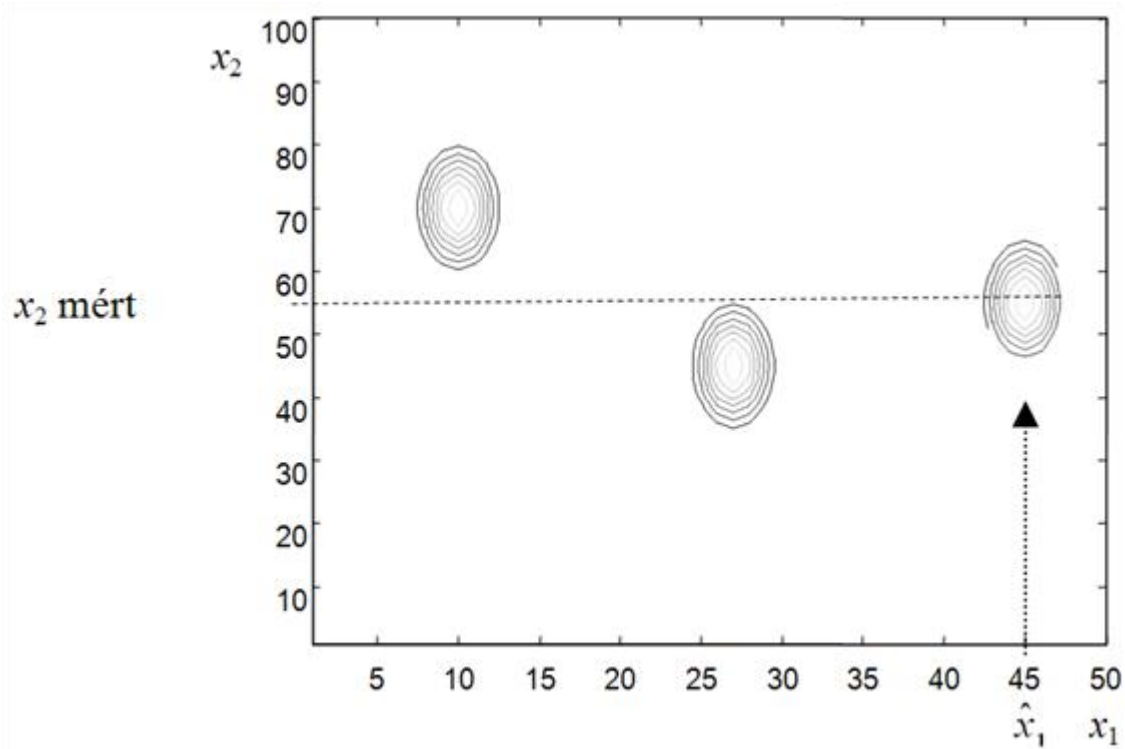
A korreláció ugyanakkor nem minden esetben mutatja meg kellő módon a komponensek közötti összefüggést. Mindkét alábbi ábrán az látható, hogy a két ábrázolt paraméter között szoros kapcsolat van, mégis a 13.10 (a) ábrán látható helyzetben a korrelációs együttható értéke nagy (0,9 feletti), míg a 13.10 (b) ábrán kicsi (0,1 alatti). A 13.3.1 pontban ismertetett EM algoritmus ugyanakkor eszközt adhat a kapcsolat felfedezésére, a pótlás segítésére.

13.10. ábra - Egymással szoros kapcsolatban álló, de különböző korrelációs koefficienset adó paraméterpárok (az (a) ábrán látható esetben közel 1, míg a (b) ábrán látható esetben közel 0)



Az 13.11 ábrán demonstrációs céllal bemutatunk egy esetet, amikor a két paraméter (x_1 és x_2) között van kapcsolat, de ez egyszerű korrelációs módszerekkel nem mutatható ki jól. Ugyanakkor, ha tudunk is arról, hogy van kapcsolat a két változó között, a kapcsolat jellege más, attól függően, hogy a mintapont melyik klaszterbe tartozik. Ha ezt el tudjuk dönteni, pl. EM algoritmus segítségével, akkor az ismert x_2 alapján (x_2 mért, szagattot vonal) jó becslést tudunk adni hiányzó x_1 -re. Ha a mintapontok eloszlásáról van ismeretünk (pl. az ábrán látható módon azok 3 klaszterben helyezkednek el) és az egyes klaszterek a priori előfordulásai azonosak, akkor a feltételes valószínűségek alapján az ismert x_2 -höz a hiányzó x_1 pótlására a legnagyobb valószínűséggel $\hat{x}_1$ felel meg.

13.11. ábra - Egymással szoros - de klaszterenként eltérő - kapcsolatban álló paraméterpár. Az ábrán a két paraméter különböző értékeinek előfordulási gyakorisága látható, szintvonalas ábrázolással. (Tehát 3 különböző csúcsa van a gyakoriságnak)



Dinamikus eset

Amennyiben dinamikával rendelkező folyamat modellezésénél találkozunk hiányzó adatokkal, akkor a statikus esetben használtakon túlmenően új lehetőségeink nyílnak az adatok pótlására.

Alapvetően azon változók, változócsoporthoz hiányzó értékeit tudjuk a statikus esethez képest jobban pótolni, amelyeknél legalább rövidtávú trendeket fedezhetünk fel a változó időbeli alakulásában. Az időbeli összefüggések felderítésére egyszerű esetekben az időfüggvény tanulmányozása is elegendő, de általában az autokorrelációs függvény vizsgálata szolgálja ezt a célt.

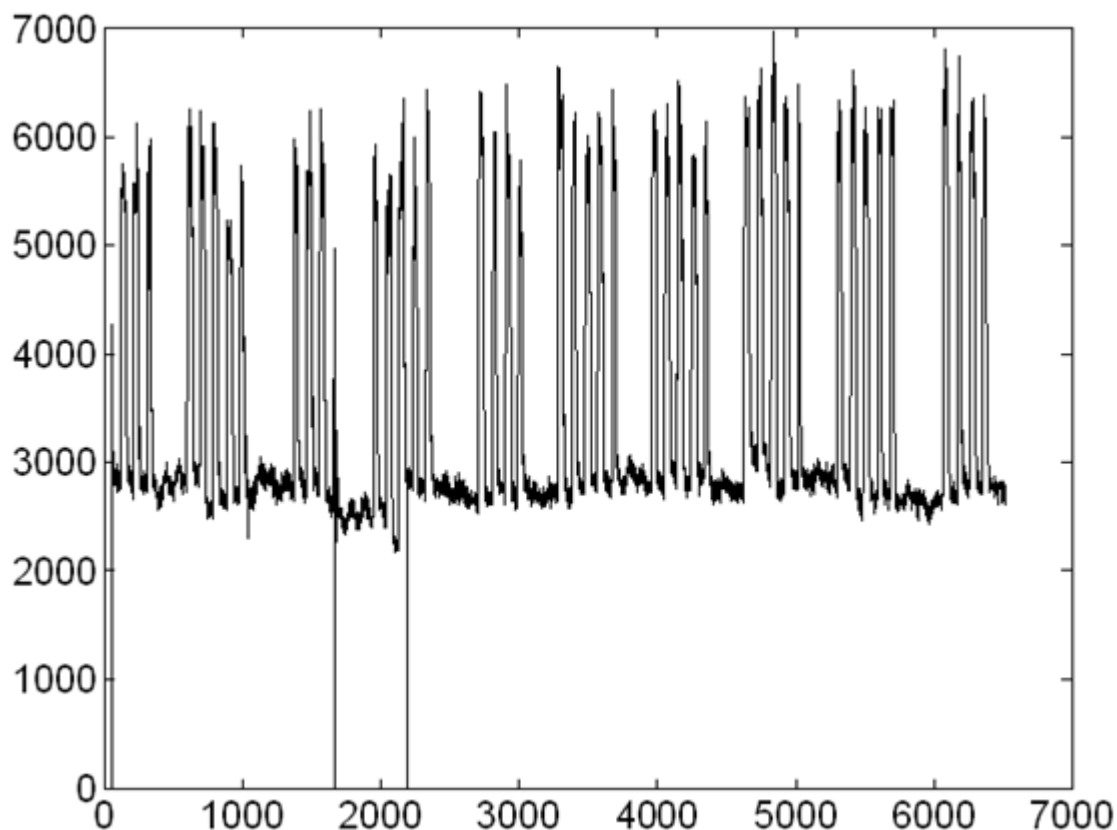
$$R_i(t, \tau) = E\{x_i(t) x_i(t + \tau)\} \quad (13.34)$$

Ha az autokorrelációs függvény a τ valamilyen tartományában nagy értékeket vesz fel, akkor a hiányzó $x_i(t)$ az $x_i(t + \tau)$ segítségével pótolható.

Gyakran találkozunk olyan helyzettel, mikor az autokorrelációs függvény τ valamilyen tartományában – például 0-hoz közeli τ -k esetén – t -től függetlenül nagy érték. Ez arra utal, hogy rövid távon a változó erős trendet mutat: ilyenkor a tanító mintakészletnél lehetőségünk van a hiányzó értéket az időben megelőző és követő értékek alapján interpolálni, illetve a tanított hálózat működése során az időben megelőző minták alapján extrapolálhatunk a hiányzó aktuális értékre.

Tehát ha az adatokat valamilyen – legalább rövidtávú trendeket generáló – folyamat hozza létre, akkor a teljes N elemű bemeneti vektor, vagy a kimeneti érték hiánya is pótolhatóvá válik. Természetesen a statikus esetre ismertetett módszerek a paraméter időbeli alakulását figyelembe vevő módszerekkel kombinálhatók.

13.12. ábra - Egy konkrét – villamos energiafogyasztást jellemző – idősor, mérési problémák miatt hiányzó (0 értékkel feltüntetett) adatokkal



5. Lényegkiemelés

5.1. A dimenziócsökkentés (lényegkiemelés) célja

Sokszor az előfeldolgozás egyik fontos célja a (bemeneti) adatvektorok dimenziójának csökkentése. Míg a neurális háló elvileg tetszőleges dimenziószámmal képes dolgozni, a feleslegesen nagy adatdimenzió a hálózatot elbonyolítja, a szükséges súlyok és minták számának indokolatlan növekedéséhez vezet, és rendszerint a konvergenciát is jelentősen rontja. A nagy dimenzió okozta problémákat a dimenzió átka (*curse of dimensionality*) névvel szokták illetni.

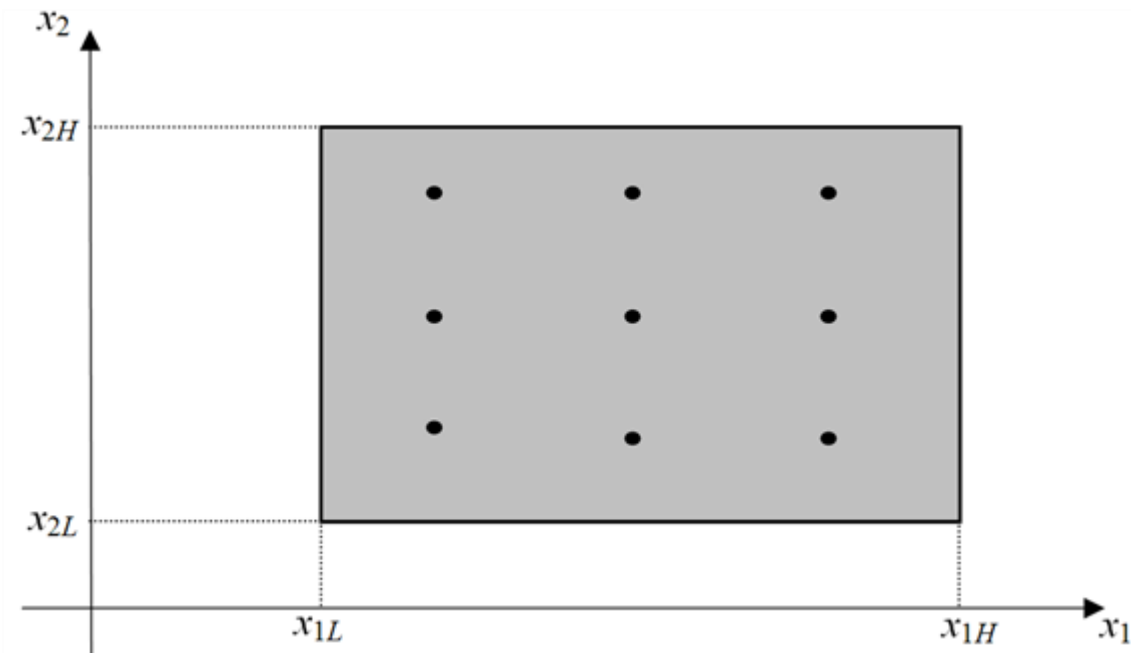
Tegyük fel, hogy egy N -bemenetű egykimenetű hálózattal dolgozunk, és az $\mathbf{x} = [x_1, x_2, \dots, x_N]$ bemeneti vektor komponenseinek mindegyike valamilyen határok közt vehet fel értékeket: $x_{iL} \leq x_i \leq x_{iH}$. Akár regressziós (függvény approximációs) feladatot akarunk megoldani, akár osztályozási feladatot, rendszerint arra törekszünk, hogy azt az N -dimenziós térrészt, melyben minták előfordulhatnak egyenletesen lefedjük tanító (és teszt) mintákkal. Az egyenletes lefedés ugyanis biztosítja, hogy a tanított hálózat tényleges felhasználásakor bármilyen szituáció is fordul elő, egy ahhoz hasonló szituációt jelentő nem túl távol eső mintaponttal már tanítottunk hálózatot. Egydimenziós esetben, ha P ponttal le akarjuk fedni a használt tartományt, P mintára van szükségünk, kétdimenziós esetben ugyanolyan sűrűségű lefedettséghez már P^2 minta kell. Általánosságban N dimenzió esetén a dimenzióként P ponttal történő egyenletes lefedéshez szükséges minták száma P^N . A 13.13 ábra $N=2$ esetén mutatja a dimenzióként $P=3$ ponttal való egyenletes lefedést, ami általában a szoba jöhető térrész elég hézagosságot jelent, de két dimenzió esetén már ehhez is 9 mintára van szükség, míg 10 dimenzió (bemenet) esetén már közel 60000 minta kell az ilyen sűrűségű lefedéshez.

A szükséges tanító minták száma a dimenzió számmal tehát exponenciálisan nő, ami gyakorlati feladatok megoldása során különösen nagy problémákat vet fel. (Szimulációs feladatoknál is gondot jelenthet a minták előállításának ideje, a szükséges tároló kapacitás, stb., de fizikai folyamatok modellezésénél, vagy gyakorlati osztályozási problémáknál további minták gyűjtése rendkívül drága, sőt, sok esetben nem is lehetséges.)

A dimenziószám-csökkentés legegyszerűbb módja, ha a mért paraméterek egy részét elhagyjuk. Kézenfekvő példákat nyújt erre a problémára és eljárásra az, amikor ipari folyamatok modellezését kívánjuk elvégezni mért adatok alapján. A mért adatok köre a megvalósított technológiában rendszerint jóval tágabb, mint ami a konkrét modellezési cél eléréséhez szükséges. (Pl. tipikusan mért és naplózott paraméter a technológiai lépések kezdeti és végidőpontja, ami legtöbbször irreleváns a modellezés szempontjából.) Elvileg persze megtehetjük azt is, hogy valamennyi mért paramétert felhasználjuk a modellezésben, a háló képes lehet a bemeneti súlyainak csökkentésével figyelmen kívül hagyni az érdektelen adatokat. Viszont lényegesen kisebb hálóméretet és jobb konvergenciát remélhetünk, ha a felesleges adatokat előzetes vizsgálatok során kiszűrjük, és csak a valóban fontosakra koncentrálunk.

Az egyszerű adatelhagyás nem, de a dimenziószám-csökkentés megvalósítható, ha a mért paraméterek valamilyen lineáris kombinációja hordozza a keresett információt, vagy legalábbis az információ jelentős részét. Pl. ha egy folyamat hőegyensúlyát kívánjuk modellezni, akkor elég lehet a különböző – de közel azonos fajhővel rendelkező – anyagok tömegének összegét figyelembe venni, ami az ehhez hasonló összevonásokkal együtt érdemi paraméterszám csökkenést eredményezhet. Az ilyen típusú dimenziócsökkentés történhet a folyamat fizikai jellemzőinek elemzésével, illetve statisztikai módszerek (pl. főkomponens analízis) segítségével.

13.13. ábra - Dimenzióként 3-3 ponttal lefedett kétdimenziós térrész



5.2. Problémafüggetlen, általános eljárások (PCA, ICA)

A 10. fejezetben már bemutattuk azokat a legelterjedtebb, általános, problémafüggetlen eljárásokat, amelyeket a dimenziószám-csökkentésben, lényegkiemelésben is alkalmazni szoktunk. A két leggyakrabban alkalmazott eljárás a főkomponens analízis (PCA), illetve a független komponens analízis (ICA). Mindkét eljárással a bemeneti vektor komponenseiből – rendszerint lineáris transzformációval – egy új (azonos méretű) vektort hozunk létre. Lényegkiemeléskor ennek a vektornak csak egyes komponenseit őrizzük meg, a többit eldobjuk (ld. 13.14 ábra).

Mint az említett fejezetben láttuk, a PCA eljárás azt célozza, hogy az N -dimenziós bemeneti vektorokat olyan új $M < N$ dimenziós térben ábrázoljuk, ahol négyzetes értelemben kicsi az eltérés. Tehát a bemeneti vektorokat transzformáljuk, majd elhagyjuk $N-M$ komponensüket, vagy eleve olyan transzformációt használunk, amely létrehozza ezt a végeredményt. A transzformáció:

$$\mathbf{x}' = \mathbf{T}\mathbf{x} \quad (13.35)$$

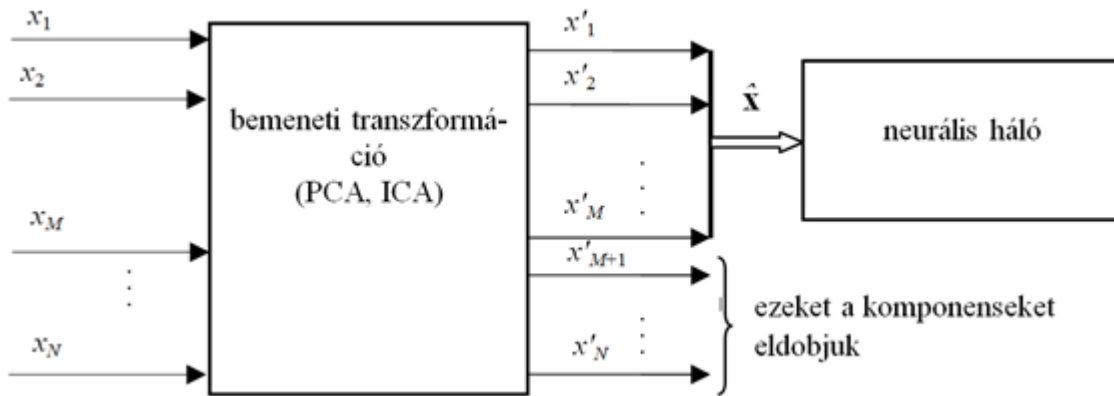
A $\mathbf{T} = [\phi_1, \phi_2, \dots, \phi_N]^T$ transzformációs mátrixot úgy alakítjuk ki (a 11. fejezetben ismertetett módszerekkel), hogy $\mathbf{x}'$ -nek az első M komponenssel való közelítése várható értékben (átlagosan) a minimális négyzetes hibát adja.

Tehát:

$$\hat{\mathbf{x}} = [\phi_1, \phi_2, \dots, \phi_M]^T \mathbf{x} \quad M < N \quad (13.36)$$

A PCA közelítés eredménye a dimenziószám-csökkentés – minimális átlagos négyzetes hiba mellett.

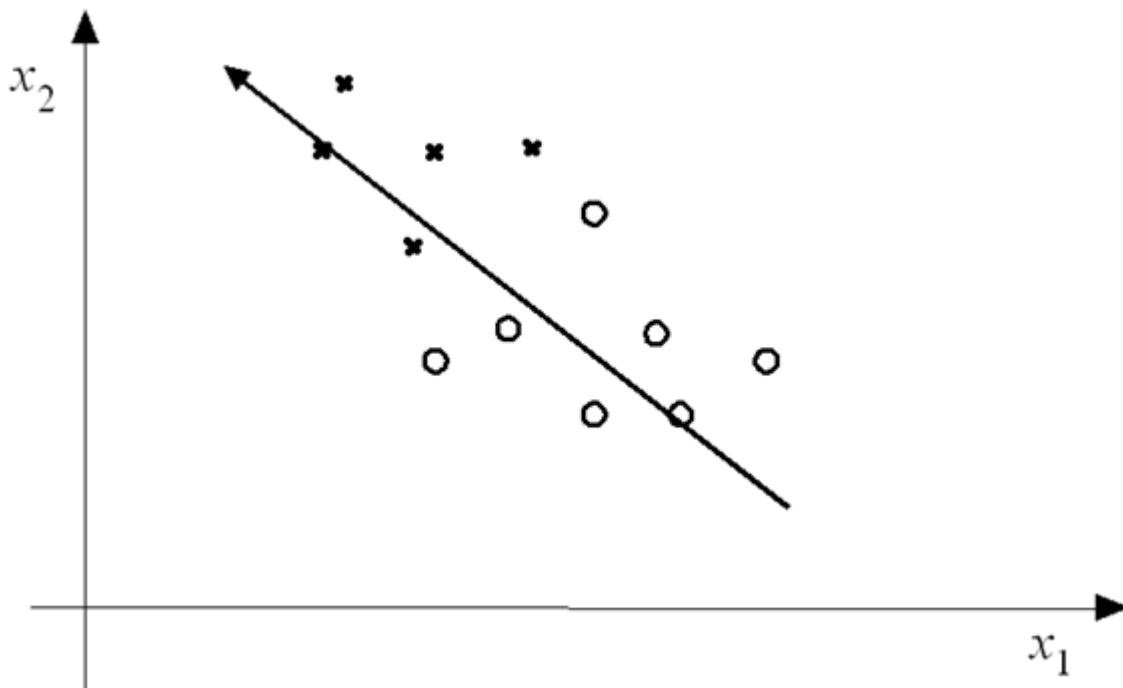
13.14. ábra - Bemeneti - tipikusan PCA, illetve ICA - transzformációval történő lényegkiemelés



A 13.15 ábra egy egyszerű kétdimenziós osztályozási feladaton mutatja be a főkomponens analízisen alapuló dimenziószám-csökkentés lehetőségét. Az ábrán látható két osztály mintái önmagában a primer módon mért két változó egyikének használatával sem választhatók szét, de ha egy egyszerű lineáris transzformációval bevezetjük az $\tilde{x} = x_2 - x_1$ változót, ami éppen a mintahalmaz egyik főkomponens iránya, akkor ezen egyetlen változó segítségével az osztályozás elvégezhető.

Ugyanakkor konkrét gyakorlati feladatoknál meg kell vizsgálni, hogy a legkisebb átlagos négyzetes hiba valóban kis információvesztést jelent-e. Triviális példával élve: ha arcképeket akarunk felismerni, akkor a kis eltérések (pl. egy-egy anyajegy, forradás) viszonylag kis négyzetes hibát jelenthetnek a mintahalmazból kinyerhető "átlagos arcképhez" képest. Mégis éppen azt veszítjük el ezek elhanyagolása esetén, ami az arcképet könnyebben felismerhetővé teszi. Az összes képből nagy súllyal jelenlévő "átlagos arc" elhanyagolása a felismerhetőség szempontjából sokkal kevésbé lenne problematikus, mint a négyzetes értelemben kis súlyú részleteké.

13.15. ábra - A mintahalmaz főkomponense mentén végzett osztályozása



Az ICA eljárással történő lényegkiemelés csak annyiban különbözik a PCA módszerrel történőtől, hogy más előfeldolgozási transzformációt alkalmazunk. Ugyanakkor jelentős különbség, hogy itt nem a vektor közelítő ábrázolásának minimális négyzetes hibája a cél, hanem a transzformációs mátrix vektorainak függetlensége, amelyet az ICA alkalmazásakor az eloszlások magasabbrendű momentumaival mérünk. Itt is elmondható, hogy

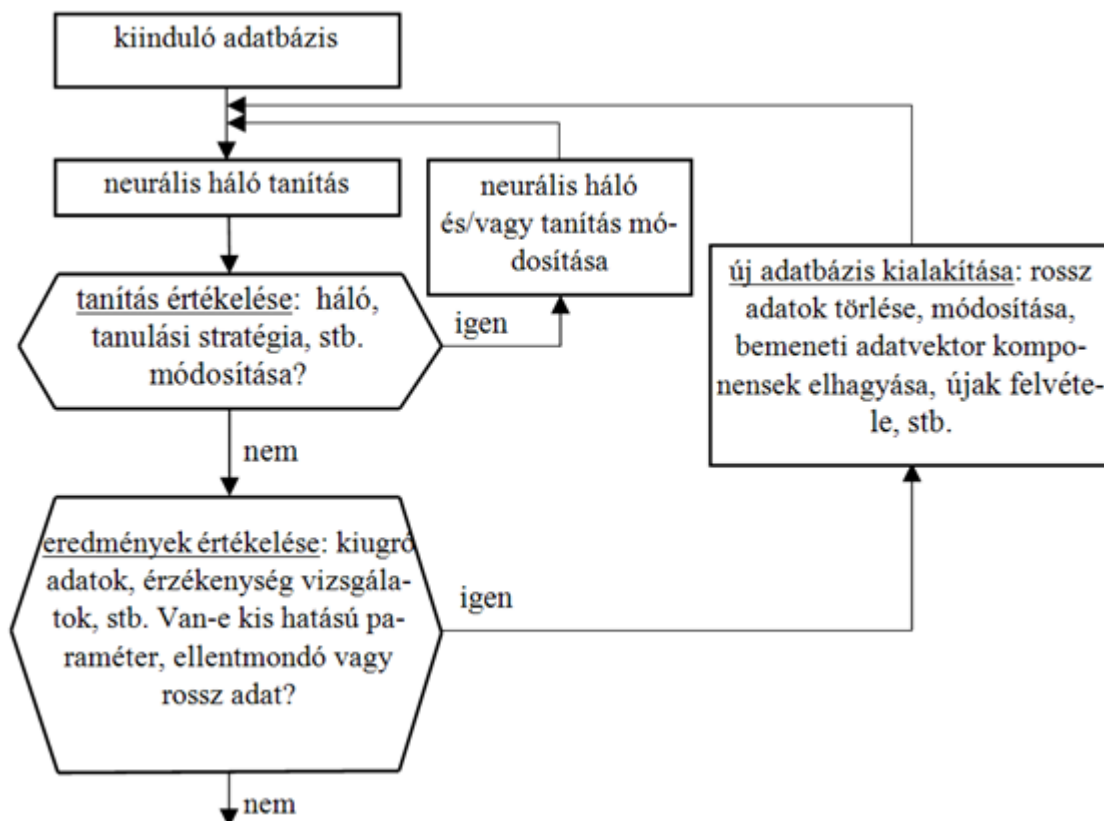
önmagában minden feladatnál meg kell vizsgálni, hogy ez a függetlenség valóban a hasznos információk kiemelését jelenti-e.

A fentiekhez még két megjegyzést kell tennünk. A PCA/ICA/egyéb transzformációk előtt legtöbbször célszerű a normalizálást végrehajtani, különben a transzformáció a viszonylag nagy értékű és változékonyságú eredeti komponenseket fogja kiemelni. A transzformáció elvégzése után a kapott komponenseket a kívánt kimeneti értékkel célszerű lehet korreláltani. Valószínűleg a korreláció abszolútértékének nagysága alapján jobban el lehet dönteni, hogy mely komponenseket őrizzünk meg és melyeket hanyagoljunk el, mint pl. PCA-nál a legkisebb négyzetes hibakritérium alapján.

Természetesen a dimenziócsökkentés akkor egyszerűbb, ha a paraméterek között lineáris a kapcsolat, de elérhető az esetben is, ha az adatok közötti összefüggés nemlineáris, ilyenkor az összevonandó paraméterek kiválasztása és az összevonás elvégzése tipikusan nagyságrendekkel nehezebb feladat. Ez esetben neurális módszerek használata is felmerül a dimenziócsökkentésre. Példaként az autoasszociatív hálóval való adattömörítési módszereket vagy a kernel PCA eljárást említhetjük (ld. 10. fejezet).

A 13.16 ábrán látható folyamatábra azt a kettős iterációs ciklust mutatja, melyben a neurális háló tanításán túl az adatbázis átalakítása: dimenziószám módosítás, hihetőség vizsgálat is folyik a fejlesztés során.

13.16. ábra - A neuronháló tanítása közben iteratíván végzett dimenziómódosítás



Az ábrán bemutatott alapgondolat a következő: amennyiben a neurális modellt megtanítottuk, akkor az elért teszt eredmények elemzését felhasználhatjuk az adatbázis, illetve az egyes adatok minősítésére is. A bemeneti vektor egyes komponenseit a modellen érzékenységvizsgálattal tesztelhetjük, tehát azt vizsgáljuk, hogy az adott bemenet változtatása, mekkora kimeneti változást idéz elő. Ha olyan bemeneteket találunk, melyek hatása a kimenetre elhanyagolható, akkor ezen bemenetek törlésével dimenziószám-csökkentést érhetünk el. Ez az eljárás rokon a hálózat méretének csökkentésével (pruning, ld. 4. fejezet), amikor felesleges súlyok vagy neuronok elhagyása a cél. Amennyiben a kiugró adatokat vizsgáljuk (tehát azokat, melyekre a neurális modell válasza nagyon rossz), akkor két olyan szituációval is találkozhatunk, mely az adatbázis minőségének javítását lehetővé teszi. Egyrészt kiderülhet, hogy a kiugró adat hibás, ekkor az adatvektort vagy töröljük az adatbázisból, vagy hibás komponensét megpróbáljuk korrigálni a 13.4 pontban leírt módszerek valamelyikével. Másrészt kiderülhet, hogy az adatbázis ellentmondást tartalmaz, tehát azonos (vagy közel azonos) bemeneti mintákra lényegesen eltérő kimenetet ad két vagy több adatvektor esetén. Ennek gyakran az az oka, hogy elhagytunk

olyan releváns paramétert (a dimenziócsökkentés során), mely ezt a két szituációt megkülönböztetné. (Például ellentmondásos adatbázishoz jutunk, ha a 13.15 ábra osztályozási feladatát az x_2 változó elhagyásával, tehát csak x_1 -re támaszkodva, akarjuk megoldani. Azonos bemeneti értékhez különböző osztályokba sorolt minták fognak tartozni ebben a csökkentett dimenziós térben.) Ekkor a dimenziószám-növelést is vállalva fel kell vegyük ezt a paramétert is a bemeneti változók közé.

Feladatok

13.1 Készítsen egy kétbemenetű egykimenetű MLP hálót, amelynek első rejtett rétegében 5 tangens hiperbolikus aktivációs függvényű nemlineáris feldolgozó neuron, kimeneti rétegében pedig egyetlen, lineáris neuron van. Tanítsa a hálót a következő Matlab kóddal megadott bemeneti és kimeneti jellel:

```
x1=5+6*randn(1,100);           % egyik bemeneti komponens
x2=-200+30*randn(1,100);        % másik bemeneti komponens
d=x1.*x1+x2+0.1*randn(1,10) % kívánt válasz:
0);
```

Végezze el a tanítást

- nem normált bemeneti és kimeneti jelekkel,
- [-1,+1] tartományra normált bemeneti, de nem normált kimeneti jelekkel,
- egyaránt [-1,+1] tartományra normált be- és kimeneti jelekkel,
- egyaránt 0 átlagértékűre és egységnyi szórásúra normált be- és kimeneti jelekkel.

A kiindulásnak javasolt tanítási paraméterek (Matlab környezetben):

```
MaxNumofEpochs= % a tanítási ciklusok megengedett
200;              maximális száma

CelHiba=0.001;    % a tanítás leállítását előidéző
                  célhibaszint
```

Hasznos lehet, ha több maximális tanítási ciklusszám és célhiba paraméterrel is kísérletezik. Egy adott beállításnál végezze el a kísérletet többször (pl. 10-szer) és az egyes esetekben kapott átlagos négyzetes hibák sorozatait hasonlítsa össze! Mit tapasztal?

Vigyázat, pl. Matlabban a kijelzett átlagos négyzetes hiba (MSE) abszolút hiba jellegű, ezért a hiba értéke – még egyformán jó approximációk esetén is – értelemszerűen függ attól, hogy mekkora a kimeneti tartomány. Az összehasonlítást ezért úgy végezze el, hogy a megtanított hálót futtassa le újra a hozzá tartozó bemeneti adatokkal (Matlab sim() függvény), a kapott kimeneti eredményt transzformálja vissza az eredeti tartományba, és ott nézze meg a kívánt kimenettől való átlagos négyzetes eltérést!

13.2 Oldja meg a 13.1 feladatot úgy, hogy néhány ponton impulzuszerű zajt ad a bemeneti és kimeneti jelhez. Pl. az alábbi Matlab kóddal módosítva a fentiekben megadott jeleket:

```
x1(1,23)=-40;
x1(1,52)=+42;
x1(1,88)=+35;
x2(1,2)=-50;
x2(1,47)=-600;
d(1,66)=d(1,66)+50;
```

13.3 Oldja meg a 13.1 feladatot RBF háló alkalmazásával is! (Matlab: ne a newrbe(), hanem a newrb() függvényt használja.) A maximális első rétegbeli neuronszám legyen 30. Vizsgálja meg az approximációt az első rétegbeli (Gauss aktivációs függvényű) neuronok $\sigma=1$ és $\sigma=10$ értéke mellett is!

13.4 Készítsen egy kétbemenetű egykimenetű MLP hálót, amelynek első rejtett rétegében 5 tangens hiperbolikus aktivációs függvényű nemlineáris feldolgozó neuron, kimeneti rétegében pedig egyetlen, lineáris neuron van. Tanítsa a hálót a következő Matlab kóddal megadott kétfajta bemeneti adattal. A kimeneti adatok mindkét esetre legyenek ugyanazok:

```
x1=5+3*randn(1,100);  
  
x2=-8+30*randn(100);  
  
X=[x1;x2];           %egyik fajta bemeneti jel, a  
                      %bemenetek függetlenek  
  
T=[1 -1; 1 1];       %transzformáló mátrix  
  
XT=T*X;              %másik bemeneti jel, a bemenetek  
                      %keverték, összefüggének  
  
d=x1.*x1+x2+0.1*randn(1  
,N);
```

A kiindulásnak javasolt tanítási paraméterek (Matlab környezetben):

```
MaxNumofEpochs=2 % a tanítási ciklusok megengedett  
00;               % maximális száma  
  
CelHiba=0.001;    % a tanítás leállítását előidéző  
                  % célhibaszint
```

Hasznos lehet, ha több maximális tanítási ciklusszám és célhiba paraméterrel is kísérletezik. A 13.1 feladathoz hasonlóan a kísérletet itt is végezze el többször (pl. 10-szer), és hasonlítsa össze az egyes esetekben kapott átlagos négyzetes hibák sorozatait! Mit tapasztal?

A. függelék - Összegzés, várható fejlődési irányok

A neurális számítástechnika viszonylag rövid fejlődése során komoly viták tárgya volt, megítélése a szélsőségektől sem volt mentes. A neurális számítástechnika fejlődésének mintegy 50 évét komoly sikerek és nagy kudarcok kísérték. A sikerek általában újszerű alkalmazásokhoz kötődtek, melyek során azt lehetett bemutatni, hogy a „neurális” feladatmegoldás sokszor nagyságrendekkel könnyebb, (kevesebb erőfeszítéssel, gyorsabban ad jó eredményt), mint a „hagyományos” megközelítés. Néhány látványos eredmény azt mutatta, hogy a komplex feladatok megoldásához más sem kell, mint kellő mennyiségű adat, a többit a neuronháló megoldja; a neuronháló az adatokból megtanulja a megoldást. Egyes sikeres alkalmazásokat már akkor kidolgoztak, amikor azok az elméleti eredmények még meg sem születtek, melyek megalapozzák ezen sikeres alkalmazásokat. A sikerek mellett azonban hamar kiderült az is, hogy a megoldandó feladatok alapos ismerete nélkül összetett valódi gyakorlati problémák neurális megoldása nagyon gyakran kudarcra vezet. A megoldandó feladat elemzése nélkül, tudatosan összeállított adatbázis nélkül, az adatok megfelelő reprezentálása, értelmezése, előkészítése nélkül, esetleg előfeldolgozása nélkül az adatokból való tanulás sem lehet hatékony. Mind a sikerek, mind a kudarcok sürgették az elméleti alapok tisztázását: annak meghatározását, hogy milyen hálóarchitektúrák milyen képességekkel rendelkeznek, hogyan lehet az adatokban meglévő tudást „átadni” a neuronhálóknak, hogyan lehet hatékony tanuló eljárásokat konstruálni.

A könyvben igyekeztünk bemutatni a neurális számítástechnika kialakulásának és gyors fejlődésének indítékait, legfontosabb jellemzőit – előnyeit, hátrányait –, a mind elméleti, mind gyakorlati szempontból sikeres neurális architektúrákat, a neurális hálózatok általános képességeit és néhány fontosabb alkalmazási területet. Összefoglaltuk a neurális hálózatoknál alapvető szerepet betöltő tanulás főbb fajtáit és a legfontosabb tanulási algoritmusokat.

A bemutatott hálókön kívül jónéhány egyéb háló architektúra létezik, kidolgoztak más tanuló eljárásokat is, és főképp az alkalmazások köre jóval bővebb, mint ami a könyvben megjelent. A könyvben – több-kevesebb részletességgel – arra is utaltunk, hogy a neurális számítástechnika milyen elméleti alapokra támaszkodik, illetve megemlítettünk egy sor olyan kérdést, melyekre kielégítő megoldás ma még nem létezik. Viszonylag nagy teret adtunk a kernel gépek bemutatásának, és ezen belül is az utóbbi tíz év talán legfontosabb eredményének, a szupport vektor gépeknek, valamint röviden bemutattuk a szupport vektor gépek konstrukciójának elvi hátterét is megadó statisztikus tanuláselmélet alapjait.

A neurális számítástechnika fejlődésében több irány figyelhető meg. Az egyik fő irány a témakör minél egzaktabb matematikai megalapozására törekszik. Ez elsősorban matematikai kutatásokat igényel, a hálózatok képességeinek minél pontosabb meghatározását célozza, illetve arra irányul, hogy adott feladathoz (adott képességeket igénylő feladathoz) a megfelelő, netán az optimális háló architektúra egzakt módon meghatározható legyen. Ezen kutatások gyakorlati eredménye lehet, hogy a neuronhálók "tervezésére" születnek eljárások. E téren – elsősorban a kernel gépek témakör kialakulásával és az itt elért eredményekkel – az elmúlt évtizedben jelentős előrelépés történt. A szupport vektor gépek és általában is a kernel gépek területén elért eredményeket – annak ellenére, hogy ezek a gépek nem tekinthetők klasszikus neurális hálózatoknak – a neurális számítástechnika jelentős eredményeiként kell értelmezni. Az elkövetkező évek valószínűleg e téren további fontos eredményeket hoznak. Olyan kérdések megválaszolása várat magára, mint a kernel gépek hiperparamétereinek egzakt meghatározása, a kernel függvények alkalmazásfüggő megválasztása, de nagyon fontos kérdések a hálók által szolgáltatott megoldások minősítésének elvi kérdései is. A megoldások minősítésére vonatkozó elméleti eredmények – pl. statisztikus tanuláselmélet általánosítási hibára vonatkozó korlátai – bár jelentős eredmények, de még nem biztosítják a feladathoz igazított optimális tanuló rendszer tervezésének lehetőségét. Általánosságban az mondható, hogy a matematikai eredmények és a gyakorlati alkalmazások által igényelt „apró pénzre váltott eredmények” között ma még meglehetősen nagy a távolság. Az alkalmazások során ma szükséges kísérletezés, próbálkozás szerepének mérséklése csak akkor várható, ha az elméleti eredmények a gyakorlati alkalmazások igényeihez jobban igazodnak.

Egy másik fontos fejlődési irányt az alkalmazások által támasztott igények szabják meg. Bár a könyvben igyekeztünk bemutatni, hogy egyes neuronháló architektúrák univerzális feladatmegoldó eszközök, mégis összetett, bonyolult gyakorlati problémák megoldásában legtöbbször önmagukban nem alkalmazhatók, inkább csak részfeladatok megoldásában jelentenek nagy segítséget. Általában elmondható, hogy egy komplex feladat megoldása az esetek döntő többségében csak különböző megközelítések együttes alkalmazásával lesz sikeres. Az utóbbi években ezért egyre inkább megfigyelhető az a tendencia, amely a különböző eszközök ötvöztetésére

törekszik. A neurális számító eszközöket mind gyakrabban alkalmazzák együtt más, "szoft" számítási megközelítésekkel, a hagyományos mesterséges intelligencia eszközeivel, de a hagyományos algoritmikus megoldásokkal is, létrehozva a hibrid intelligens számító rendszereknek nevezett eszközöket. A könyvben bemutatott néhány hibrid megoldás is ilyen igényeket igyekszik kielégíteni. Jelenleg azonban – az adatokon, mintapontokon túl – meglehetősen korlátozott reprezentációjú tudás beépítésére, felhasználására van lehetőség. További gondot jelent, hogy a járulékos információ beépítése sokszor igen nehezen megoldható matematikai problémákra vezet. Sok feladat merül fel a bizonytalan ismeretek, hiányos, zajos adatok felhasználása terén is.

A neuronhálók és általában is a mintákból tanuló rendszerek egy jelentős mértékben eltérő megközelítése a Bayes statisztikát, illetve a sztochasztikus folyamatok elméletét használja fel. A Bayes neurális hálók és a Gauss folyamatok témakörök egyáltalán nem szerepelnek a könyvben, miközben új szemléletet hoznak be az adatokból való tanulásba. Különösen érdekes a kernel gépek és a Gauss folyamatok kapcsolata. Várható, hogy a közeljövőben megtörténik ezen kapcsolatok minél pontosabb tisztázása, ami az a priori ismeretek lényegesen hatékonyabb kezelésére is vezethet.

Az elvi kérdéseken túl fontos kérdés az implementáció hatékonyságának növelése, a szükséges számítási komplexitás redukálása is. Ugyan az elmúlt néhány évben a számítási komplexitás csökkentése terén is fontos eredmények születtek – gondoljunk csak a kvadratus programozás hatékonyságát célzó eredményekre – még sok tennivaló van. Az igényelt számítási komplexitás csökkentése – és a számítógépek teljesítőképességének növekedése egyértelműen neurális megoldások szélesebbkörű elterjedését fogják eredményezni. A számítási komplexitás csökkentése, illetve a hatékonyabb implementáció azért is fontos, mert csak így lesz lehetséges a mainál jóval komplexebb, a mai lehetőségeknél nagyságrendekkel több neuronból álló komplex architektúrákat létrehozni. A háló képessége ugyanis nyilvánvaló kapcsolatban áll a komplexitásával. Ehhez a „természetes neuronhálók” kellő mintát szolgáltatnak. Az implementáció kapcsán azt is meg kell jegyezni, hogy – annak ellenére, hogy döntően ma is általános célú számítógépeken történő szimuláció az implementáció legelterjedtebb formája – a célhardverek egyre gyakoribb alkalmazása várható. Egyes részfeladatok neurális megoldása ugyanis akkor lehet hatékony, ha a neurális megoldást kis méretben, olcsón tudjuk megvalósítani. Sok lokális, tanulni képes „lokális” intelligenciával rendelkező egység együttműködéséből hozhatunk létre komplex intelligens rendszereket.

B. függelék - Függelék

1. Mátrixok és vektorok [Gol96b], [Róz91]

Egy mátrix az $\{a_{ij}; i=1, 2, \dots, m; j=1, 2, \dots, n\}$ elemek alábbi elrendezése, ahol az elemek általában számok vagy függvények:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \quad (\text{F.1})$$

Az $\mathbf{A}$ mátrix tömör megadása:

$$\mathbf{A} = [a_{ij}]_{m \times n} \quad (\text{F.2})$$

ahol a_{ij} az i -edik sor j -edik eleme. Egy $m \times 1$ mátrix egy oszlopvektor. Az $\mathbf{a}_i$ vektor az $\mathbf{A}$ mátrix i -edik ($i=1, 2, \dots, n$) oszlopvektora. Egy $\mathbf{A}$ mátrix felírható az oszlopvektorai segítségével:

$$\mathbf{A} = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \dots \quad \mathbf{a}_n] \quad (\text{F.3})$$

Mátrixok összeadása, konstanssal való szorzása

Ha az $\mathbf{A}$ és $\mathbf{B}$ mátrixok mérete azonos ($m \times n$), akkor értelmezhető a két mátrix összeadása:

$$\mathbf{C} = \mathbf{A} + \mathbf{B} \quad \text{ahol} \quad c_{ij} = a_{ij} + b_{ij} \quad (\text{F.4})$$

Az összeadás kommutatív:

$$\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A} \quad (\text{F.5})$$

Egy mátrix skalárral való szorzása a mátrix elemeinek skalárral való szorzását jelenti:

$$\mathbf{B} = k\mathbf{A} \quad \text{ahol} \quad b_{ij} = k a_{ij} \quad (\text{F.6})$$

Mátrixok szorzása

Ha $\mathbf{A}$ egy ($m \times p$)-s mátrix és $\mathbf{B}$ egy ($p \times n$)-es mátrix, akkor a két mátrix összeszorozható. A $\mathbf{C}$ szorzatmátrix egy ($m \times n$)-es mátrix lesz:

$$\mathbf{C} = \mathbf{AB} \quad \text{ahol} \quad c_{ij} = \sum_{k=1}^p a_{ik} b_{kj}, \quad \forall i, j = 1, \dots, n \quad (\text{F.7})$$

A szorzás nem kommutatív, tehát általában:

$$\mathbf{AB} \neq \mathbf{BA} \quad (\text{F.8})$$

A szorzás asszociatív:

$$\mathbf{ABC} = \mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C} \quad (\text{F.9})$$

A mátrix transzponáltja

Egy $\mathbf{A}$ mátrix transzponáltja az a $\mathbf{B} = \mathbf{A}^T$ mátrix melynek elemeire igaz, hogy:

$$b_{ij} = a_{ji}, \quad \forall i, j = 1, \dots, n \quad (\text{F.10})$$

A transzponált mátrixokra fennállnak a következő összefüggések:

$$\begin{aligned}(\mathbf{A}^T)^T &= \mathbf{A} \\ (\mathbf{AB})^T &= \mathbf{B}^T \mathbf{A}^T \\ (\mathbf{A} + \mathbf{B})^T &= \mathbf{A}^T + \mathbf{B}^T\end{aligned}\quad (\text{F.11})$$

Szimmetrikus mátrix, diagonálmátrix, egységmátrix

Legyen $\mathbf{A}$ egy $(n \times n)$ -es mátrix. Ha $\mathbf{A} = \mathbf{A}^T$, akkor az $\mathbf{A}$ mátrix szimmetrikus. Ha $a_{ji} = 0 \quad \forall i \neq j$, akkor $\mathbf{A}$ diagonálmátrix.

$$\mathbf{A} = \text{diag}(a_{11}, a_{22}, \dots, a_{nn}) \quad (\text{F.12})$$

Ha a diagonálmátrix minden diagonális eleme egységnyi, akkor a mátrix az egységmátrix.

$$\mathbf{I} = \text{diag}(1, 1, \dots, 1) \quad (\text{F.13})$$

Az egységmátrixszal végzett szorzásra igaz, hogy

$$\mathbf{AI} = \mathbf{IA} = \mathbf{A} \quad (\text{F.14})$$

A mátrix determinánsa

Az $\mathbf{A}$ $(n \times n)$ -es kvadratikus mátrix determinánsa $|\mathbf{A}|$, melyet (az első sor szerint kifejtve) a következőképpen kapunk meg:

$$\begin{aligned}|\mathbf{A}| &= a_{11} \begin{vmatrix} a_{22} & a_{23} & \dots & a_{2n} \\ a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n2} & a_{n3} & \dots & a_{nn} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n3} & \dots & a_{nn} \end{vmatrix} \\ &+ a_{13} \begin{vmatrix} a_{21} & a_{22} & a_{24} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{34} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n4} & \dots & a_{nn} \end{vmatrix} - \dots\end{aligned}\quad (\text{F.15})$$

Szinguláris mátrix, a mátrix inverze, pszeudoinvert

Legyen $\mathbf{A}$ egy $(n \times n)$ -es kvadratikus mátrix. A mátrix szinguláris, ha $|\mathbf{A}| = 0$. Ha a kvadratikus mátrix nonszinguláris, akkor és csak akkor létezik egy olyan $\mathbf{B}$ mátrix melyre

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I} \quad (\text{F.16})$$

A $\mathbf{B} = \mathbf{A}^{-1}$ mátrixot az $\mathbf{A}$ mátrix (multiplikatív) inverzének nevezzük [Pen55].

Fontos tulajdonságok az inverzre:

$$\begin{aligned}(\alpha \mathbf{A})^{-1} &= \alpha^{-1} \mathbf{A}^{-1} \\ (\mathbf{A}^{-1})^T &= (\mathbf{A}^T)^{-1} \\ \text{Ha } \mathbf{A}_1 \text{ és } \mathbf{A}_2 \text{ } (n \times n)\text{-es nonszinguláris mátrixok,} \\ \text{akkor } (\mathbf{A}_1 \mathbf{A}_2)^{-1} &= (\mathbf{A}_2)^{-1} (\mathbf{A}_1)^{-1}\end{aligned}\quad (\text{F.17})$$

Minden $\mathbf{A} \in \mathbb{R}^{m \times n}$ mátrixhoz létezik egy egyértelmű $\mathbf{A}^+$ mátrix, melyet az $\mathbf{A}$ mátrix pszeudoinvertnek vagy Moore–Penrose invertnek nevezünk. A pszeudoinvertre a következő összefüggések teljesülnek:

$$\begin{aligned} \mathbf{A}\mathbf{A}^{\dagger}\mathbf{A} &= \mathbf{A} \\ \mathbf{A}^{\dagger}\mathbf{A}\mathbf{A}^{\dagger} &= \mathbf{A}^{\dagger} \\ (\mathbf{A}\mathbf{A}^{\dagger})^r &= \mathbf{A}\mathbf{A}^{\dagger} \\ (\mathbf{A}^{\dagger}\mathbf{A})^r &= \mathbf{A}^{\dagger}\mathbf{A} \end{aligned} \quad (\text{F.18})$$

Ha $\mathbf{A}$ nonszinguláris, kvadrátikus mátrix, akkor a pszeudoinverz megegyezik a mátrix inverzével, vagyis: $\mathbf{A}^{\dagger} = \mathbf{A}^{-1}$.

Ha $\mathbf{A}^r\mathbf{A}$ vagy $\mathbf{A}\mathbf{A}^r$ nonszinguláris, a pszeudoinverz definiálható mint:

$$\mathbf{A}^{\dagger} = (\mathbf{A}^r\mathbf{A})^{-1}\mathbf{A}^r \quad (\text{F.19})$$

illetve, mint

$$\mathbf{A}^{\dagger} = \mathbf{A}^r(\mathbf{A}\mathbf{A}^r)^{-1} \quad (\text{F.20})$$

Az előbbi esetben szokás a mátrix pszeudoinverzét az alábbi módon is definiálni:

$$\mathbf{A}^{\dagger} = \lim_{\mu \rightarrow 0} (\mathbf{A}^r\mathbf{A} + \mu^2\mathbf{I})^{-1}\mathbf{A}^r, \quad (\text{F.21})$$

míg az utóbbi esetben az alábbi definíció alkalmazható:

$$\mathbf{A}^{\dagger} = \lim_{\mu \rightarrow 0} \mathbf{A}^r(\mathbf{A}\mathbf{A}^r + \mu^2\mathbf{I})^{-1} \quad (\text{F.22})$$

Néhány hasznos összefüggés a pszeudoinverzre:

$$\begin{aligned} (\mathbf{A}^{\dagger})^{\dagger} &= \mathbf{A} \\ (\alpha\mathbf{A})^{\dagger} &= \alpha^{-1}\mathbf{A}^{\dagger} \\ (\mathbf{A}^{\dagger})^r &= (\mathbf{A}^r)^{\dagger} \\ \mathbf{A}\mathbf{A}^r(\mathbf{A}^{\dagger})^r &= \mathbf{A} \\ \mathbf{A}^{\dagger}\mathbf{A}\mathbf{A}^r &= \mathbf{A}^r \\ (\mathbf{A}^{\dagger})^r\mathbf{A}^r\mathbf{A} &= \mathbf{A} \\ \mathbf{A}^r\mathbf{A}\mathbf{A}^{\dagger} &= \mathbf{A}^r \end{aligned} \quad (\text{F.23})$$

Mátrix inverziós lemma

A Sherman-Morrison-Woodbury formula számos esetben hasznos lehet $(\mathbf{A} + \mathbf{B}\mathbf{C})$ inverzének meghatározásánál, amennyiben $\mathbf{A}$ és $(\mathbf{I} + \mathbf{C}\mathbf{A}^{-1}\mathbf{B})$ nonszingulárisak.

$$(\mathbf{A} + \mathbf{B}\mathbf{C})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{I} + \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} \quad (\text{F.24})$$

Vektorok lineáris függetlensége

Legyen egy $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$, $\mathbf{a}_i \in \mathbb{R}^n$ vektorkészletünk az és legyenek $\alpha_1, \alpha_2, \dots, \alpha_n$ skalárok.

$\mathbf{c} = \sum_{i=1}^n \alpha_i \mathbf{a}_i$ vektort az $\{\mathbf{a}_i\}$ vektorok lineáris kombinációjának nevezzük. Az $\{\mathbf{a}_i\}$ vektorok lineárisan függetlenek, ha a $\mathbf{c} = \mathbf{0}$ akkor és csak akkor áll fenn, ha $\alpha_i = 0 \quad \forall i = 1, \dots, n$.

Egy $\mathbf{A} \in \mathbb{R}^{n \times n}$ mátrix oszlopvektorai akkor és csak akkor lineárisan függetlenek, ha $\mathbf{A}^r\mathbf{A}$ nonszinguláris, a sorvektorai pedig akkor és csak akkor lineárisan függetlenek, ha $\mathbf{A}\mathbf{A}^r$ nonszinguláris.

A mátrix rangja

Az $\mathbf{A}$ mátrix rangja a maximális számú lineárisan független oszlopvektorainak a száma, vagy alternatív definícióként a maximális számú lineárisan független sorvektorainak a száma.

Egy $\mathbf{A}$ $(m \times n)$ -es mátrix teljes rangú, ha

$$\text{rang}(\mathbf{A}) = \min\{m, n\}. \quad (\text{F.25})$$

Megmutatható, hogy egy tetszőleges $\mathbf{A}$ mátrixra

$$\text{rang}(\mathbf{A}) = \text{rang}(\mathbf{A}^T) = \text{rang}(\mathbf{A}^T \mathbf{A}) = \text{rang}(\mathbf{A} \mathbf{A}^T) \quad (\text{F.26})$$

Egy $\mathbf{A}$ egy $(n \times n)$ -es kvadratikusan mátrix akkor és csak akkor nonszinguláris, ha teljes rangú, vagyis $\text{rang}(\mathbf{A}) = n$.

Sajátértékek, sajátvektorok

Legyen $\mathbf{A}$ egy $(n \times n)$ -es kvadratikusan mátrix. Ha létezik olyan λ skalár és olyan nemzérus $\mathbf{v}$ vektor, melyre fennáll, hogy

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v} \quad (\text{F.27})$$

akkor λ -t a mátrix sajátértékének, $\mathbf{v}$ -t pedig a megfelelő sajátvektornak nevezzük.

A mátrix összes sajátértéke megkapható a

$$|\mathbf{A} - \lambda\mathbf{I}| = 0 \quad (\text{F.28})$$

karakterisztikus polinom gyökeiként. Ha a sajátértékek egyszeres gyökök, akkor minden sajátértékhez egy sajátvektor-készlet tartozik, melynek elemei csak egy nullától különböző szorzóban térnek el egymástól. Többszörös gyökök esetén egy sajátértékhez több sajátvektor is tartozhat. Ezek a sajátvektorok nem konstans szorzóban különböznek. Diagonálmátrix sajátértékei maguk a főátlóban lévő elemek.

Ha az $\mathbf{A}$ mátrix nonszinguláris, akkor összes sajátértéke nullától különböző és a mátrix inverzének sajátértékei a mátrix sajátértékeinek reciprokai.

Egy szimmetrikus mátrix sajátértékei mind valós számok. A szimmetrikus mátrix sajátvektorai ortogonális vektor-készletet alkotnak, vagyis

$$\begin{aligned} \mathbf{v}_i^T \mathbf{v}_j &= 0 \quad \text{ha } i \neq j \\ \mathbf{v}_i^T \mathbf{v}_i &= 1 \end{aligned} \quad (\text{F.29})$$

A maximális és minimális sajátértékekre fennállnak az alábbi összefüggések (Rayleigh hánnyados)

$$\lambda_{\max}(\mathbf{A}) = \max_{\mathbf{v} \neq 0} \frac{\mathbf{v}^T \mathbf{A} \mathbf{v}}{\mathbf{v}^T \mathbf{v}} \quad \text{és} \quad \lambda_{\min}(\mathbf{A}) = \min_{\mathbf{v} \neq 0} \frac{\mathbf{v}^T \mathbf{A} \mathbf{v}}{\mathbf{v}^T \mathbf{v}} \quad (\text{F.30})$$

Ha egy $\mathbf{A}$ szimmetrikus mátrix minden sajátértéke szigorúan pozitív ($\lambda_i > 0$, $i=1, 2, \dots, n$), a mátrix pozitív definit. Ha a sajátértékek nemnegatívak, akkor a mátrix pozitív szemidefinit. A mátrix nemdefinit, ha van legalább egy pozitív és legalább egy negatív sajátértéke.

Ha $\mathbf{A}$ egy olyan mátrix, hogy $\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T$, akkor a mátrix az alábbi szorzat formájában előállítható:

$$\mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T \quad (\text{F.31})$$

ahol $\mathbf{V}$ a mátrix sajátvektoraiból, mint oszlopvektorokból képezett ortogonális mátrix, $\mathbf{\Lambda}$ pedig a sajátértékek diagonálmátrixa.

A mátrix nyoma

Egy kvadratikusan $\mathbf{A}$ $(n \times n)$ -es mátrix nyoma a főátlóbeli elemeinek az összege. A mátrix nyoma a sajátértékeinek összegeként is meghatározható.

$$\text{tr}(\mathbf{A}) = \sum_{i=1}^n a_{ii} = \sum_{i=1}^n \lambda_i \quad (\text{F.32})$$

2. Gauss elimináció [Gol96b]

Az $\mathbf{Ax}=\mathbf{b}$ lineáris egyenletrendszer megoldására gyakran használt univerzális módszer a Gauss elimináció. Az egyenletrendszer mátrixai az alábbiak:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad (\text{F.33})$$

ahol $\mathbf{A}$ az együttható mátrix, $\mathbf{b}$ egy adott ismert értékekből álló vektor, az $\mathbf{x}$ vektor pedig az ismeretleneket tartalmazó vektor. Egyszerűbb esetben az $\mathbf{A}$ mátrix $n \times n$ -es, de az algoritmus $n \times m$ -es esetre is alkalmazható.

A megoldás bemutatása során a változók felírását gyakran elhagyjuk és az egyenletrendszert az alábbi kibővített mátrix segítségével írjuk fel:

$$[\mathbf{A} \ \mathbf{b}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{bmatrix}. \quad (\text{F.34})$$

Látható, hogy a kibővített mátrix az együttható mátrix kiegészítve a $\mathbf{b}$ vektorral. A kiegészített $[\mathbf{A} \ \mathbf{b}]$ mátrix és az eredeti $\mathbf{A}$ mátrix rangjára vonatkozóan:

- ha $\text{rang}(\mathbf{A}) = \text{rang}([\mathbf{A} \ \mathbf{b}]) = n$, akkor egy egyértelmű megoldás van.

- ha $\text{rang}(\mathbf{A}) = \text{rang}([\mathbf{A} \ \mathbf{b}]) < n$, akkor az egyenletnek végtelen sok megoldása van. Ilyen esetben lehet csökkenteni az ismeretlenek számát, majd valamilyen optimális megoldást (pl. a négyzetes hibaminimalizáló megoldás) keresni.

- ha $\text{rang}(\mathbf{A}) < \text{rang}([\mathbf{A} \ \mathbf{b}])$ akkor nincs megoldás.

A mátrixon az egyenletrendszer eredményének változása nélkül az alábbi elemi sor-műveletek végezhetők:

- egy vagy több egyenlet (sor) szorzása egy c konstanssal,
- két vagy több egyenlet (sor) felcserélése,
- egy egyenlet (sor) c -szeresének hozzáadása egy másik sorhoz.

A fentiek mellett az alábbi, oszlopok cseréjével járó művelet is elvégezhető:

az egyenletrendszerben a változók sorrendjét felcseréljük (ez oszlopok cseréjét jelenti, kibővített mátrix esetén az utolsó oszlop kivételével).

Az alap eljáráshoz elegendő az első három elemi sor-művelet, de bizonyos esetekben (például a pontosság növeléséhez) az utolsó műveletet is alkalmazzuk. Erről a későbbiekben lesz szó (ld. pivotálás).

A fenti műveletek segítségével a mátrixot, illetve az egyenletrendszert egy olyan egyszerűbb alakra (pl. háromszög-, diagonál mátrix) hozhatjuk, amely esetén egyszerű a megoldás számítása.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{nn} & b'_n \end{bmatrix} \quad (\text{F.35})$$

az eredeti egyenletrendszert egy $\mathbf{Rx}=\mathbf{d}$ felső háromszögmátrixot ($\mathbf{R}$) tartalmazó egyenletrendszerré alakítjuk, ami már egyszerű behelyettesítéssel megoldható. Az algoritmus egy köztes lépésében a

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a'_{22} & \dots & a'_{2n} & b'_2 \\ 0 & a'_{32} & \dots & a'_{3n} & b'_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n2} & \dots & a'_{nn} & b'_n \end{bmatrix}, \quad (\text{F.36})$$

az a'_{32} kinullázásához a második sort elosztjuk az a'_{22} generáló „pivot” elemmel majd ennek a'_{32} szorosát kivonjuk a harmadik sorból. Amennyiben a pivot elem helyén 0 áll, sorcserét kell alkalmazni (ha nincs nem 0 sor, akkor az adott oszlop megfelel az elvárásoknak).

A “pivot” vagy generáló elem, amint azt láthattuk fontos szerephez jut az algoritmus során. A számítási pontosság növelésére szolgál a pivotolás, amikor ezt a pivot elemet tudatosan választjuk ki, ahol a cél egy nagy értékű elem használata. Ez az elem sor illetve oszlop cserével mozgatható a megfelelő pozícióba. A részleges pivotolás (*partial pivoting*) esetén az adott oszlop abszolút értékben legnagyobb elemét használjuk, míg a teljes pivotolás (*complete pivoting*) esetén a mátrix oszlopait és sorait is cserélve mozgadjuk az abszolútértékben legnagyobb elemet a megfelelő pozícióba. Ebben az esetben (oszlopok cseréjénél) az ismeretlenek sorrendje is változik. A leggyakrabban elegendő az egyszerűbb részleges pivotolás alkalmazása.

A fentiekhez hasonlóan egy másik algoritmus, a Gauss-Jordan elimináció célja az alábbi diagonális mátrix alak:

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 & b_1 \\ 0 & a'_{22} & \dots & 0 & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & a'_{nn} & b'_n \end{bmatrix} \quad (\text{F.37})$$

amit a Gauss eliminációnál alkalmazott műveletekkel érhetünk el. Ebben az esetben nincs szükség visszahelyettesítésekre, az eredmény azonnal számítható.

3. A reduced row echelon alak

A mátrix redukált sor echelon alakja (RRE alak) (*Reduced Row Echelon Form – RREF*) a következő feltételeket teljesíti:

- Minden nem csupa 0-ból álló sor legelső nem 0 eleme 1-es (ezt vezető 1-esnek nevezzük).

Ha egy sor vezető 1-ese az r -edik oszlopban van, akkor

- az r -edik oszlop össze többi eleme 0.
- a későbbi sorok vezető 1-esei az r -edik oszloptól jobbra kell legyenek

A csak nulla elemeket tartalmazó sorok a mátrix aljára csoportosítottak.

Egy mátrix RRE alakja egyértelmű.

Az alábbi mátrix egy példa a reduced row echelon alakra:

$$\begin{bmatrix} 1 & 0 & 2 & 0 & 4 \\ 0 & 1 & 3 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (\text{F.38})$$

Látható, hogy ez az alak a Gauss eliminációnál tárgyalt elemi műveletekkel előállítható.

Az LS-SVM redukciójánál (az LS²-SVM support vektorainak meghatározásához) a RRE alakot biztosító algoritmust egészítjük ki. Az algoritmus során figyeljük a pivot elem értékét és amennyiben az egy általunk meghatározott ϵ^f tolerancia értéknél kisebb, az adott oszlopot kinullázzuk (nem vesszük figyelembe). Az algoritmus menete – melynek alapja a részleges pivotolással végzett Gauss-Jordan elimináció – az alábbi. Iteratívan lépdeljünk végig a mátrix főátlója mentén:

1. Legyen j az első nem csupa nulla oszlop indexe.

2. Határozzuk meg a j -edik oszlopban az $i \geq j$ sorindexű elemek közül a legnagyobbat. Jelöljük ezt az elemet p -vel.

3. Ha $p \leq \epsilon'$ (ahol ϵ' a tolerancia érték) akkor nullázzuk ki az oszlop alsó részét (a j -edik oszlop azon elemeit, melyekre a sorindex $i \geq j$);

egyébként jegyezzük meg az oszlop indexét, mert bázis, és ha $i > j$ cseréljük fel a p elemet tartalmazó sort a j -edik sorral. Így az oszlop legnagyobb eleme, p – az ún. generáló elem, ami egyben az i -edik sor balról első nem nulla eleme, a pivot elem – a főátlóba kerül. Osszuk el ezt a sort p -vel, ezáltal p helyére 1 kerül, és vonjuk ki ennek a sornak megfelelő konstans-szorosait a többi sorból, hogy ezáltal kinullázzuk a többi sor j -edik pozícióban lévő elemeit.

4. Növeljük meg j -t: $j = j + 1$ és folytassuk az egész eljárást a második lépéstől.

Ez az eljárás visszaadja az ϵ' tolerancia értelmében lineárisan független oszlopok listáját (azaz az LS²-SVM-ben használt szupport vektorokat).

4. Feltételes szélsőérték-keresés, Lagrange multiplikátoros módszer [Fle86, Boy04]

A Lagrange multiplikátoros módszer egy hatékony eljárás a feltételes szélsőérték-keresési feladatok megoldására.

Legyen adott egy $f(\mathbf{x})$ függvény $\mathbf{x} \in \mathbb{R}^n$ és adottak a $g_i(\mathbf{x}) = 0$ feltételek. Keressük $f(\mathbf{x})$ szélsőértékét (minimumát vagy maximumát) úgy, hogy közben a $g_i(\mathbf{x}) = 0$ feltételek is teljesüljenek.

Definiáljuk az alábbi Lagrange függvényt:

$$L(\mathbf{x}, \boldsymbol{\alpha}) = f(\mathbf{x}) - \sum_i \alpha_i g_i(\mathbf{x}) \quad (\text{F.39})$$

ahol az α_i együtthatók a Lagrange multiplikátorok. Figyeljük meg, hogy mind a szélsőértékfeladat, mind a feltételek figyelembevétele szerepel a Lagrange függvény szélsőértékére vonatkozó feladatban. Ugyanis

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}) = \nabla_{\mathbf{x}} f(\mathbf{x}) - \nabla_{\mathbf{x}} \sum_i \alpha_i g_i(\mathbf{x}) \quad (\text{F.40})$$

vagyis

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\alpha}) = 0 \Leftrightarrow \nabla_{\mathbf{x}} f(\mathbf{x}) = \nabla_{\mathbf{x}} \sum_i \alpha_i g_i(\mathbf{x}) \quad (\text{F.41})$$

$$\nabla_{\boldsymbol{\alpha}} L(\mathbf{x}, \boldsymbol{\alpha}) = 0 \Leftrightarrow g_i(\mathbf{x}) = 0 \quad (\text{F.42})$$

A következő egyszerű kétdimenziós feladaton illusztráljuk a Lagrange multiplikátoros módszer működését. Legyen a kétdimenziós függvényünk $f(\mathbf{x}) = 2x_1^2 + x_2^2$. Keressük ennek a függvénynek a minimumát azzal a feltétellel, hogy $x_1^2 + x_2^2 = 1$, vagyis $g(\mathbf{x}) = x_1^2 + x_2^2 - 1 = 0$. Az $f(\mathbf{x})$ függvény feltétel nélküli minimuma az $\mathbf{x} = [0, 0]^T$ pontban van. Ez a pont azonban nyilvánvalóan nem elégíti ki a $g(\mathbf{x}) = 0$ mellékfeltételt. A megfelelő Lagrange függvény:

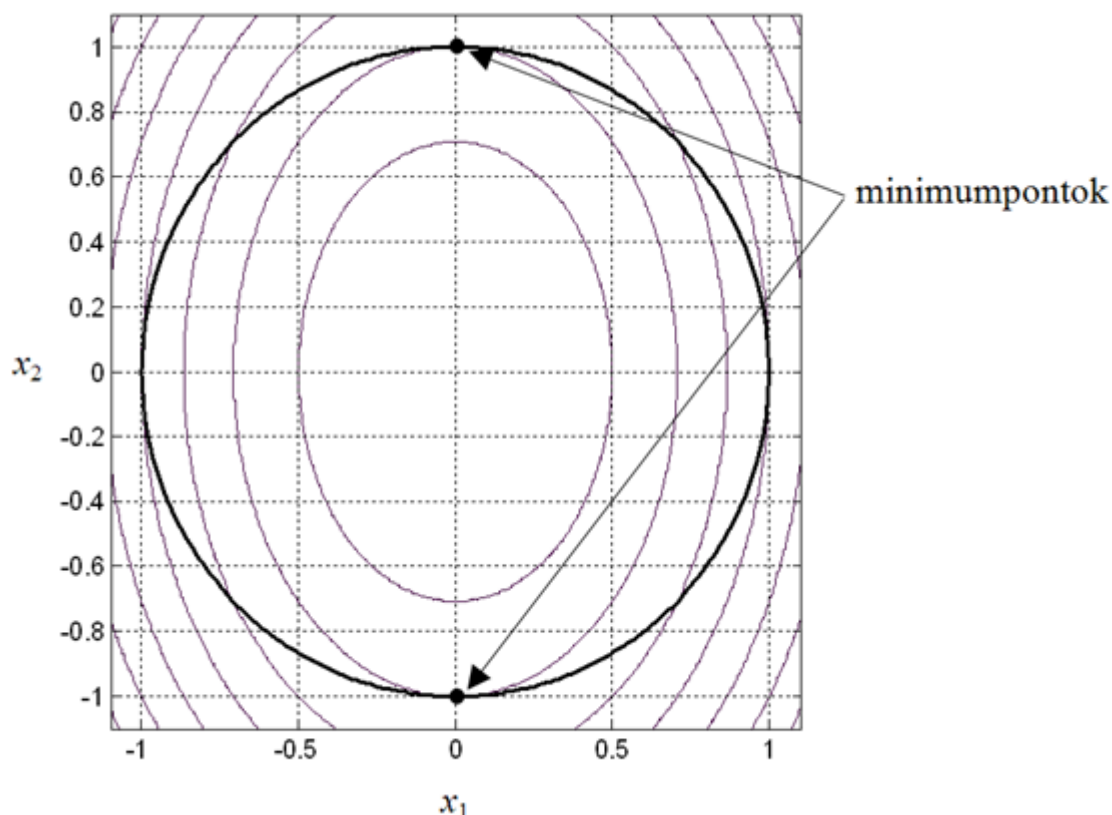
$$L(\mathbf{x}, \boldsymbol{\alpha}) = 2x_1^2 + x_2^2 - \alpha(x_1^2 + x_2^2 - 1) \quad (\text{F.43})$$

Melynek $\mathbf{x}$ szerinti gradiense

$$\begin{aligned}
\frac{\partial L(\mathbf{x}, \alpha)}{\partial x_1} &= 4x_1 - 2\alpha x_1 = 0 \\
\frac{\partial L(\mathbf{x}, \alpha)}{\partial x_2} &= 2x_2 - 2\alpha x_2 = 0 \\
\frac{\partial L(\mathbf{x}, \alpha)}{\partial \alpha} &= -(x_1^2 + x_2^2 - 1) = 0
\end{aligned}
\quad (\text{F.44})$$

A két első egyenletből $\alpha = 2$ illetve $\alpha = 1$ adódik, mely értékek mellett $f(\mathbf{x})$ minimumát az $x_1 = 0; x_2 = \pm 1$ pontban kapjuk (F.3.1 ábra).

F.3.1. ábra - Mintapéllda a feltételes szélsőérték-keresésre. A vékonyabb vonallal rajzolt ellipszisek a minimalizálandó $f(\mathbf{x})$ függvény szintvonalai, a vastagabb vonallal rajzolt kör pontjai a mellékfeltételnek megfelelő pontok. A két fekete pont a feltételes minimumot biztosító pontok.



Az ábrán jól látható, hogy az $f(\mathbf{x})$ függvény által meghatározott szintvonalak gradiensvektorai a feltételes minimumpontokban azonos irányúak, mint a mellékfeltételt megadó kör gradiensvektorai, vagyis a minimumpontokban a két görbe érinti egymást.

5. Karush-Kuhn-Tucker feltételek [Cri00], [Boy04]

A Karush-Kuhn-Tucker (KKT) feltételek nemlineáris optimalizálási feladatok megoldását biztosító feltételek. A problémakör a Lagrange multiplikátoros módszer általánosításának tekinthető. Míg a Lagrange multiplikátoros módszer olyan feltételes szélsőérték-keresési feladatok megoldásánál alkalmazható, ahol a feltételek egyenlőség formájában vannak megfogalmazva, a KKT optimalizálásnál az egyenlőség típusú feltételek helyett vagy azok mellett egyenlőtlenség típusú feltételek (is) szerepelnek.

$f(\mathbf{x})$ $\mathbf{x} \in X$ a következő nemlineáris optimalizálási feladatot egy $X \subseteq \mathbb{R}^n$ konvex tartománnyal, $g_i(\mathbf{x}) = 0$ egyen egy konvex¹ folytonosan differenciálható függvény és legyenek a $i=1, \dots, M$ és a $j=1, \dots, L$ függvények affin² függvények.

Keressük

$$\begin{aligned} \min_{\mathbf{x} \in X} f(\mathbf{x}) \text{-et, } \mathbf{x} \in X \text{ úgy, hogy közben a} \\ g_i(\mathbf{x}) \geq 0 \text{ és a} \\ h_j(\mathbf{x}) = 0 \text{ feltételek is teljesüljenek.} \end{aligned} \quad (\text{F.45})$$

Definiáljunk egy általánosított Lagrange függvényt a következő formában:

$$L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) - \sum_{i=1}^M \alpha_i g_i(\mathbf{x}) - \sum_{j=1}^L \beta_j h_j(\mathbf{x}) \quad (\text{F.46})$$

ahol $\alpha_i \geq 0$ és $\beta_j \in \mathbb{R}$ Lagrange multiplikátorok.

Annak szükséges és elégséges feltétele, hogy egy $\mathbf{x}^* \in X$ a fenti feladat optimauma legyen az, hogy létezzenek olyan $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$, $\boldsymbol{\alpha}^* \in [0, \infty)^M$, $\boldsymbol{\beta}^* \in \mathbb{R}^L$ változók, hogy

$$\begin{aligned} \nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) &= \frac{\partial L(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{x}} = \mathbf{0} \\ \nabla_{\boldsymbol{\beta}} L(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) &= \frac{\partial L(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = \mathbf{0} \\ \alpha_i^* g_i(\mathbf{x}^*) &= 0, \quad i = 1, \dots, M \\ g_i(\mathbf{x}^*) &\geq 0, \quad i = 1, \dots, M \\ \alpha_i^* &\geq 0, \quad i = 1, \dots, M, \end{aligned} \quad (\text{F.47})$$

$(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ -ra igaz, hogy bármely $\mathbf{x} \in \mathbb{R}^n$ és $\boldsymbol{\alpha} \in [0, \infty)^M$ és $\boldsymbol{\beta} \in \mathbb{R}^L$ mellett

$$L(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \leq L(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \leq L(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \quad (\text{F.48})$$

vagyis $\mathbf{x}^*$ egy nyeregpont (*saddle point*).

Elsődleges és másodlagos feladat

Az SVM-nél felmerülő elsődleges (*primal*) optimalizálási feladat minimumkeresési feladat egyenlőtlenség típusú mellékfeltételekkel.

Keressük

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \text{-et, } \mathbf{x} \in X \text{ úgy, hogy közben a} \\ g_i(\mathbf{x}) \geq 0 \text{ feltételek is teljesüljenek.} \end{aligned} \quad (\text{F.49})$$

A megfelelő általánosított Lagrange függvény

$$L(\mathbf{x}, \boldsymbol{\alpha}) = f(\mathbf{x}) - \sum_{i=1}^M \alpha_i g_i(\mathbf{x}) \quad (\text{F.50})$$

Feltételezve, hogy $(\mathbf{x}^*, \boldsymbol{\alpha}^*)$ egyértelmű minimum, minden rögzített $\boldsymbol{\alpha} \geq \mathbf{0}$ mellett definiálható egy Lagrange függvény,

¹ Egy valós értékű $f(\mathbf{x})$ $\mathbf{x} \in \mathbb{R}^n$ függvény konvex, ha $\forall \mathbf{x}, \mathbf{u} \in \mathbb{R}^n$ és bármely $\lambda \in (0, 1)$ mellett $f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{u}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{u})$. Egy kétszer differenciálható függvény konvex, ha a második deriváltakból képezett Hesse mátrix pozitív definit.

² Egy affin függvény felírható az alábbi formában: $f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, ahol $\mathbf{A}$ valamilyen mátrix és $\mathbf{b}$ egy vektor. Egy affin függvény egyben konvex is.

$$Q(\alpha) := \min_{\mathbf{x}} L(\mathbf{x}, \alpha) \quad (\text{F.51})$$

mely egyedül α függvénye.

Ennek megfelelően megfogalmazható a duális (*dual*) optimalizálási feladat:

$$\begin{aligned} &\text{maximalizáljuk } Q(\alpha)\text{-t} \\ &\text{úgy hogy } \alpha_i \geq 0 \quad i=1,2,\dots,M, \end{aligned} \quad (\text{F.52})$$

ahol $Q(\alpha)$ -t duális kritériumfüggvénynek nevezzük. A duális feladat megoldása nem feltétlenül egyszerűbb, mint az eredeti feladat megoldása. A duális feladat előnye, hogy a duális kritériumfüggvény csak α függvénye.

Az SVM esetén a duális feladat az α_i Lagrange multiplikátorok kvadratikusan függvénye.

$$Q(\alpha) = \alpha^T \mathbf{1} - \frac{1}{2} \alpha^T \mathbf{K} \alpha \quad (\text{F.53})$$

az α_i Lagrange multiplikátorokra vonatkozó feltételekkel: $\alpha_i \geq 0$ és $\sum_{i=1}^P \alpha_i d_i = 0$, ahol d_i $i=1,2,\dots,P$ az SVM feladatnál megfogalmazott kívánt válaszokat jelöli. Az SVM duális feladatát kvadratikusan programozással oldhatjuk meg.

C. függelék - Jelölések

Szimbólumok

$\mathbf{a}$	vektor
$\mathbf{a}^T$	transzponált vektor
$\ \mathbf{a}\ $	az a vektor euklideszi normája, a vektor hossza
$\ \mathbf{a}\ _p$	az a vektor p -normája
$\mathbf{A}$	mátrix
$\mathbf{A}^{-1}$	inverz mátrix
$\mathbf{A}^\dagger$	pszeudó-inverz (Moore-Penrose inverz) mátrix
$ \mathbf{A} $	az $\mathbf{A}$ mátrix determinánsa
$\text{LT}(\mathbf{A})$	az $\mathbf{A}$ mátrix alsó háromszögmátrixa
$\mathcal{b}$	eltolás
$\mathbf{C}$	autókovariancia mátrix
$C(\cdot)$	kritérium függvény
C^N	az N -dimenziós folytonos függvények halmaza
$\mathcal{d}$	kívánt válasz
E	várható érték operátor
h	VC dimenzió
$\mathbf{H}$	Hesse mátrix
$\mathbf{I}$	egységmátrix
$J(\mathbf{w})$	elsődleges kritériumfüggvény SVM-nél
k	diszkrét időindex
$\mathbf{K}$	kernel mátrix
$K(\mathbf{x}, \mathbf{z})$	kernel függvény

$L(\mathbf{w})$	veszteségfüggvény, hibafüggvény
$L_\varepsilon(\mathbf{w})$	ε érzéketlenségi sávval rendelkező veszteségfüggvény
$L(\mathbf{w}, b, \alpha)$	Lagrange függvény
<i>LIKELIHOOD</i>	likelihood függvény
$\mathbf{m}$	várható érték vektor
M	bázisfüggvények száma
N	a bemeneti minták dimenziója
p	valószínűség, sűrűségfüggvény
$p\{x \Theta\}$	feltételes valószínűség sűrűségfüggvény
P	mintapontok száma
$P\{x \Theta\}$	feltételes valószínűség
$Q(\alpha)$	duális kritériumfüggvény osztályozós SVM-nél
$Q(\alpha, \alpha')$	duális kritériumfüggvény regressziós SVM-nél
$\mathbf{R}$	autókorrelációs mátrix
$\mathbb{R}$	valós számok halmaza
$R(\mathbf{w})$	kockázat, mint a súlyvektor függvénye
$\mathbb{R}^n$	valós szám N -esek halmaza
$\text{tr}(\mathbf{A})$	az $\mathbf{A}$ mátrix nyoma
$\mathbf{x} \in \mathcal{X}$	bemenet és a bemeneti tér
$\bar{x}$	az x változó átlaga (súlyozott átlaga)
$\langle x \rangle$	az x változó középértéke
$\mathbf{y} \in \mathcal{Y}$	kimenet és a kimeneti tér
$\mathcal{Y}$	egydimenziós kimenet

$\mathbf{w}$	súlyvektor
$\Delta \mathbf{w}$	a súlyvektor megváltozása, módosítása
w_0	eltolás
$\mathbf{W}$	súlymátrix
$O(.)$	ordó jelölés
ξ	gyengítő változó
λ	sajátérték
μ	tanulási tényező
σ	szélesség paraméter
α	Lagrange multiplikátor
β	Lagrange multiplikátor
δ	általánosított hiba, érzékenység
ε	hiba
$\in$	eleme egy halmaznak szimbólum
$\subseteq$	tartalmazás szimbólum
$\cup$	únió szimbólum
$\cap$	metszet szimbólum
$\lceil a \rceil$	az a -nál nagyobb vagy azzal egyenlő legkisebb egész
$\lfloor a \rfloor$	a egész része
∇	gradiens operátor
$\delta(.,.)$	Kronecker delta

Rövidítések

AIC	Akaike information criterion
APEX	Adaptive principal component extraction
BP	back-propagation

BPTT	back-propagation through time
CART	classification and regression tree
CMAC	cerebellar model articulation controller
DCT	discrete cosine transform
EIV	errors-in-variables
EM	expectation-maximization
ERM	empirical risk minimization
FIR	finite impulse response
FIR-MLP	finite impulse response multilayer perceptron
FPE	final prediction error
GHA	generalised Hebbian algorithm
HMOE	hiererchical mixture of experts
ICA	independent component analysis
IIR	infinite impulse response
KBANN	knowledge based artificial neural network
KKT	Karush-Kuhn-Tucker
KLT	Karhunen-Loève transform,
LMS	least mean squares
LS	least squares
LS-SVM	least squares support vector machine
LS ² -SVM	LS - least squares support vector machine
LVQ	learning vector quantization
MDL	minimum description length
ML	maximum likelihood
MLP	multilayer perceptron
MOE	mixture of experts

MRAC	model reference adaptive control
MSE	mean squared error
NARX	nonlinear autoregressive with exogenous input
NARMAX	nonlinear autoregressive moving average with exogenous input
NFIR	nonlinear FIR
NIC	neural information criterion
NJB	nonlinear Jenkins-Box
NOE	nonlinear output error
OBD	optimal brain damage
OBS	optimal brain surgeon
OLS	orthogonal least squares
PCA	principal component analysis
RBF	radial basis function
RLS	recursive least squares
RREF	reduced row echelon form
RRKRR	reduced rank kernel ridge regression
RSVM	reduced support vector machine
RTRL	real time recursive learning
SLT	statistical learning theory
SMO	sequential minimal optimization
SOM	self-organizing map
SRM	structural risk minimization
SSVM	smooth support vector machine
SVM	support vector machine
TDNN	time delay neural network
TSP	traveling salesman problem

VC Vapnik-Chervonenkis

WMSVM weighted margin support vector machine

D. függelék - Irodalom

- [Aar89] Aart, E. - Korst, J. "Simulated Annealing and Boltzmann Machines", John Wiley, New York, 1989.
- [Abu90] Abu-Mostafa, Y. S. "Learning from Hints in Neural Networks", *Journal of Complexity*, Vol. 6. pp. 192-198, 1990.
- [Abu95] Yaser S. Abu-Mostafa, "Hints" *Neural Computation*, Vol. 7. pp. 639-671, 1995.
- [Aga01] Agarwal, R. P. – Meehan, M – O'Regan, D. "Fixed Point Theory and Applications", Cambridge University Press, 2001.
- [Aka69] Akaike, H. "Fitting Autoregressive Models for Prediction", *Ann. Ins. Stat. Math.* Vol. 21. pp. 243-347.
- [Aka74] Akaike, H. "A New Look at the Statistical Model Identification", *IEEE Trans. on Automatic Control*, AC-19. No. 6. pp. 716-723. 1974.
- [Alb75] Albus, J. S. "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)", *Transaction of the ASME*, Sep. 1975. pp. 220-227.
- [Ama95] Amari, S. - Cichocki, A. - Yang, H. H. "Recurrent Networks for Blind Separation of Sources", *Proc. of the International Symposium on Nonlinear Theory and its Applications, NOLTA-95*, Las Vegas, USA, pp. 37-42. 1995.
- [Ama97] Amari, S. - Chen, T. - Cichocki, A. "Stability Analysis of Learning Algorithms for Blind Source Separation", *Neural Networks*, Vol. 10. No. 8. pp. 1345-1351. 1997.
- [Ama99] Amari, S.- Wu, S. "Improving support vector machine classifiers by modifying kernel functions", *Neural Networks*, Vol. 12. pp. 783-789. 1999.
- [Bab89] Baba, N. "A New Approach for Finding the Global Minimum of Error Function of Neural Networks", *Neural Networks*, Vol. 2. pp. 367-373. 1989.
- [Bac91] Back, A. D. - Tsoi, A. C. "FIR and IIR Synapses, a New Neural Network Architecture for Time Series Modeling", *Neural Computation*, Vol. 3. pp. 375-385. 1991.
- [Bal89] Baldi, P. - Hornik, K. "Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima", *Neural Networks*, Vol. 2. No. 1. pp. 53-58. 1989.
- [Bal00] Balestrino, A. - Caiti, A. "Approximation of Hammerstein/Wiener Dynamic Models," *IEEE International Joint Conference on Neural Networks IJCNN'00*, Vol. 1, pp. 1070-1074., 2000.
- [Bar93] Barron, A. R. "Universal Approximation Bounds for Superposition of Sigmoidal Functions", *IEEE Trans. on Information Theory*, Vol. 39. No. 3. pp. 930-945. 1993.
- [Bat92] Battiti, R. "First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method", *Neural Computation*, Vol. 4. pp. 141-166. 1992.
- [Bau89] Baum, E. B. - Haussler, D. "What Size Net Gives Valid Generalization?", *Neural Computation*, Vol. 1. No. 1. pp. 151-160. 1989.
- [Bau01] Baudat G. - Anouar, F. "Kernel-based methods and function approximation". *International Joint Conference on Neural Networks*, pp. 1244-1249, Washington, DC, 2001.
- [Bea94] Beaufays, F. - Wan, E. "Relating Real-Time Backpropagation and Backpropagation-Through-Time: An Application of Flow Graph Interreciprocity", *Neural Computation*, Vol. 6. pp. 296-306. 1994.
- [Bek93] Bekey, G. - Goldberg, K. Y. "Neural Networks in Robotics", Kluwer Academic Publisher, Boston 1993.

- [Bel95] Bell, A. - Sejnowsky, T. "Blind Separation and Blind Deconvolution: an Information/ Theoretic Approach", *Proc. of the 1995 IEEE International Conference on Acoustics Speech and Signal Processing*, Detroit, USA, pp. 3415-3418. 1995.
- [Ber01] Berényi, P. - Horváth, G. - Pataki, B. – Strausz, Gy. "Hybrid-Neural Modeling of a Complex Industrial Process" *Proc. of the IEEE Instrumentation and Measurement Technology Conference, IMTC'2001*. Budapest, Vol. III. pp.1424-1429. 2001.
- [Bis95] Bishop, C. M.: "Neural Networks for Pattern Recognition", Clarendon Press, Oxford, 1995.
- [Bis06] Bishop, C. M.: "Pattern Recognition and Machine Learning ", Springer, 2006.
- [Bos92] Boser, B. - Guyon, I. - Vapnik, V. "A Training Algorithm for Optimal Margin Classifier" *Proc. of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144-152. 1992.
- [Bos97] Bossley, K. M. "Neurofuzzy Modelling Approaches in System Identification", *Ph.D Thesis*, University of Southampton, 1997.
- [Bou95] Boutayeb, M., - Darouach, M. "Recursive Identification Method for MISO Wiener-Hammerstein Model", *Automatica*, Vol. 40. No. 2. pp. 287-291. 1995.
- [Boy04] Boyd, S. - Vandenberghe, L. "Convex Optimization", Cambridge University Press, 2004.
- [Bre84] Breiman, L. - Friedman, J. H. - Olshen, R. A. - Stone, C. J. "Classification and Regression Trees", *Technical Report*, Wadsworth International, Monterey, CA. 1984.
- [Bre96] Breiman, L. "Bagging Predictors", *Machine Learning*, Vol. 24. No. 2, pp. 123-140, 1996.
- [Bro93] Brown, M. - Harris, C. J. - Parks, P. "The Interpolation Capability of the Binary CMAC", *Neural Networks*, Vol. 6. No. 3. pp. 429-440. 1993.
- [Bro94] Brown, M. - Harris, C. "Neurofuzzy Adaptive Modelling and Control", Prentice Hall, New York, 1994.
- [Bru90] Bruck, J. "On the Convergence Properties of the Hopfield Model", *Proceedings of the IEEE*, Vol. 78. No. 10. pp. 1579-1585. 1990.
- [Bry69] Bryson, A. E. and Ho, Y.C. „Applied Optimal Control”, Blaisdell, New York, 1969.
- [Car89] Carroll, S. M. - Dickinson, B. W. „Construction of Neural Nets Using the Radon Transform” *Proc. of the Int. Conference on Neural Networks*, Vol. I, pp. 607-611. New York, 1989.
- [Car94] Cardoso, J. F. - Laheld, B. "Equivariant Adaptive Source Separation", *IEEE Trans. on Signal Processing*, 1994.
- [Caw02] Cawley, G. C. - Talbot, N. L. C. "Reduced Rank Kernel Ridge Regression" *Neural Processing Letters*, Vol. 16. No. 3. pp. 293-302, 2002.
- [Cha87] Chan, L. W. - Fallside, F. "An Adaptive Training Algorithm for Backpropagation Networks", *Computer Speech and Language*, Vol. 2. pp. 205-218. 1987.
- [Cha02] Chapelle, O. - Vapnik, V. - Bousquet, I. - Mukherjee, S. "Choosing multiple Parameters for Support Vector Machines", *Machine Learning*, Vol. 46. No. 1. pp. 131-159. 2002.
- [Cha06] Chapelle, O. – Schölkopf, B. – Zien, A. "Semi-Supervised Learning", The MIT Press, Cambridge, MA. 2006.
- [Che89] Chen, S. - Billings, S. A. - Luo, W. „Orthogonal Least Squares Methods and their Application to Non-linear System Identification”, *International Journal of Control*, Vol. 50. pp. 1873-1896, 1989.
- [Che91] Chen, S. - Cowan, C. - Grant, P. „Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks”, *IEEE Trans. on Neural Networks*, Vol. 2. pp. 302-309, 1991.

- [Che04] Chen, B., Chang, M., Lin, C., „Load Forecasting Using Support Vector Machines: A Study on EUNITE Competition 2001”, *IEEE Transactions on Power Systems*, Vol. 19, No. 4, pp. 1821-1830. 2004.
- [Cho97] Cho, S. - Cho, Y. - Yoo, S. "Reliable Roll Force Prediction in Cold Mill Using Multiple Neural Networks", *IEEE Trans. on Neural Networks*, Vol. 8. No. 4. pp. 874-882. 1997.
- [Cho98] Choi, S. - Cichocki, A. - Amari, S. "Flexible Independent Component Analysis," *Proc. of the IEEE Workshop on Neural Networks for Signal Processing*, Cambridge, England, 31. August - 3. September, 1998.
- [Cic93] Cichocki, A. - Unbehauen, R. "Neural Networks for Optimization and Signal Processing", John Wiley & Sons, New York, 1993.
- [Ciz04] Cizek, P., "Asymptotics of least trimmed squares regression", *Discussion Paper 72*, Tilburg University, Center for Economic Research. 2004.
- [Com94] Comon, P. "Independent Component Analysis, A New Concept?", *Signal Processing*, Vol. 36. No. 2. pp. 287-314. 1994.
- [Cov65] Cover, T. M "Geometrical and Statistical Properties of Systems of Linear Inequalities with Application in Pattern Recognition", *IEEE Trans. on Electronic Computers*, EC-14. pp. 326-334. 1965.
- [Cra96] Craven, M. W. "Extracting Comprehensible Models from Trained Neural Networks" *Ph.D. thesis*, University of Wisconsin, Madison, USA, 1996.
- [Cri99] Cristianini, N. - Campbell, C. - Shawe-Taylor, J. "Dynamically Adapting Kernels in Support Vector Machines" in: Kearns M., Solla S., Cohn D., (eds.) *Advances in Neural Information Processing Systems (NIPS)* Vol. 11, MIT Press. 1999,
- [Cri00] Cristianini, N. - Shawe-Taylor J. "Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, 2000.
- [Cyb89] Cybenko, G. "Approximation by Superpositions of a Sigmoidal Function", *Mathematical Control Signals Systems*, Vol. 2. pp. 303-314, 1989.
- [DeC02] DeCoste, D. - Schölkopf, B. "Training Invariant Support Vector Machines" *Machine Learning*, Vol. 46. pp. 161-190. 2002.
- [Dei86] Deistler, M. "Linear Dynamic Errors-in-Variables Models", *Journal of applied Probability*, Vol. 23. pp. 23-39. 1986.
- [Dem77] Dempster, A. P. - Laird, N. M. - Rubin, D. B. "Maximum Likelihood from Incomplete Data via the EM Algorithm (with discussion)", *Journal of the Royal Statistical Society (Series B)*, Vol. 39. No. 1. pp. 1-38, 1977.
- [Dia96] Diamantaras, K. I. - Kung, S. Y. "Principal Component Neural Networks Theory and Applications", John Wiley and Sons, New York. 1996.
- [Dre05] Dreyfus, G. "Neural Networks. Methodology and Applications", Springer, 2005.
- [Dru93] Drucker, H. - Schapire, R. E. - Simard, P. "Boosting Performance in Neural Networks", *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 7. No. 4, pp. 705-719. 1993.
- [Dru99] Druckner, H., Wu, D., Vapnik, V., „Support Vector Machines for Spam Categorization”, *IEEE Transactions on Neural Networks*, Vol. 10, No. 5, pp. 1048-1054. 1999.
- [Dun93a] Dunay, R. - Horváth, G. "Modified CMAC Neural Network Architectures for Nonlinear Dynamic System Modelling", *Proceedings of the International Conference on Artificial Neural Networks (ICANN 93)*, Amsterdam, Holland, p. 473. 1993.
- [Dun93b] Dunay, R. - Pataki, B. - Horváth, G. "Some Further Possibilities of the Application of Neural Networks for Nonlinear Dynamic System Modelling", *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies, EUFIT 93*, Aachen, Germany, Vol. 2. pp.930-936. 1993.

- [Egm02] Egmont-Petersen M. - de Ridder D. - Handels H. "Image Processing With Neural Network – A Review", *Pattern Recognition*, Vol. 35, No. 10, pp. 2279-2301, 2002,
- [Eld97] Eldracher, M. - Staller, A. - Pompl, R. „Adaptive Encoding Strongly Improves Function Approximation with CMAC”, *Neural Computation*, Vol. 9. pp. 403-417. 1997.
- [Ell91] Ellison, D.: "On the Convergence of the Multidimensional Albus Perceptron", *The International Journal of Robotics Research*, Vol. 10. No. 4. pp. 338-357. 1991.
- [Far90] Farrell, J. A. - Michel, A. N. "A Synthesis Procedure for Hopfield's Continuous-Time Associative Memory", *IEEE Trans. on Circuits and Systems*, Vol. 37. No.7. pp. 877 884. 1990.
- [Fle88] Fletcher, R. "Practical Methods of Optimization", John Wiley, 1988.
- [Föl89] Földiák, P. "Adaptive Network for Optimal Linear Feature Extraction" *Proc. of the 1989 Intnt. Joint Conference on Neural Networks*, Vol. 1. pp. 401-405. 1989.
- [Fre95] Freund, Y. "Boosting a Weak Learning Algorithm by Majority" *Information and Computation*, Vol. 121. No. 2. pp. 256-285, 1995.
- [Fre97] Freund Y. - Schapire, R. E. „A Decision-Theoretic Generalization of On-line Learning and its Application to Boosting”, *Journal of Computer and System Sciences*, Vol. 55. No. 1. pp. 119-139, 1997.
- [Fuk88] Fukushima, K. "Neocognitron: A Hierarchical Neural Network Capable of Pattern Recognition", *Neural Networks*, Vol. 1. No. 2. pp. 119-130. 1988.
- [Fun89] Funahashi, K. I. "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, Vol. 2. No. 3. pp. 183-192. 1989.
- [Fun03] Fung, G. - Mangasarian, O. L. - Shavlik, J. W. "Knowledge-Bases Support Vector Machine Classifiers" In: S. Becker, S. Thrun, K. Obermayer (eds.) *Advances in Neural Information Processing Systems*, Vol. 15, pp. 521-528. MIT Press, Cambridge, MA. USA. 2003.
- [Gal89] Galar, R. "Evolutionary search with soft selection", *Biological Cybernetics*, Vol. 60. pp. 357-364. 1989.
- [Gir96] Girolami, M - Fyfe, C. "Higher Order Cumulant Maximisation Using Nonlinear Hebbian and Anti-Hebbian Learning for Adaptive Blind Separation of Source Signals", *Proc. of the IEEE/IEE International Workshop on Signal and Image Processing, IWSIP-96*, Advances in Computational Intelligence, Elsevier publishing, pp. 141 - 144, 1996.
- [Gol89a] Goldberg, D. E. "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1989.
- [Gol89b] Golub H. - Van Loan, C. F. "Matrix Computations", Third Edition, Gene Johns Hopkins University Press, Baltimore and London, 1996.
- [Gol96] Golden, R. M. "Mathematical Methods for Neural Network Analysis and Design", MIT Press, Cambridge, MA. 1996.
- [Gon98] Gonzalez-Serrano, F. - Figueiras-Vidal, A. R. - Artés-Rodríguez, A. „Generalizing CMAC Architecture and Training”, *IEEE Trans. on Neural Networks*, Vol. 9. pp. 1509-1514, 1998.
- [Goo95] Goonatilake, S. - Khebbal, S. (Editors) "Intelligent Hybrid Systems", John Wiley and Sons, London, 1995.
- [Gra03] Granas, A. - Dugundji, J. "Fixed Point Theory" Springer-Verlag, New York, 2003.
- [Gre87] Grefenstette, J. "Genesis 4.5." GA software from gref@aic.nrl.navy.mil 1987.
- [Gup03] Gupta, M. M - Jin, L. - Homma, N. "Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory", Wiley-IEEE Press, 2003.

- [Guy96] Guyon, I. „A Scaling Law for the Validation-set Training-set Ratio”, *Technical Report*, AT&T Bell Laboratories. 1996.
- [Had1902] Hadamard, J. “Sur les problèmes aux dérivées partielles et leur signification physique”, *Princeton University Bulletin*, 49--52. 1902.
- [Ham90] Hambaba, M. L. "Nonlinear Principal Component Using Feedforward Neural Networks", *Proc. of the Joint Conference on Neural Networks*, 1990. pp. 3246-3248.
- [Han89] Han, J. Y. - Sayeh, M. R. - Zhang, J. "Convergence and Limit Points of Neural Network and Its Application to Pattern Recognition", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 19. No. 5. 1989. pp. 1217-1222.
- [Hao04] Hao, X. - Zheng, P. - Xie, Zh. - Du, G. - Shen, F. "A predictive modeling for blast furnace by integrating neural network with partial least squares regression", *Proc. of the IEEE International Conference on Industrial Technology CIT '04*. Vol. 3. pp. 1162-1167. 2004.
- [Har93] Harrer, H. - Galias, Z. - Nossek, J. A. "On the Convergence of Discrete Time Neural Networks", *International Journal of Circuit Theory and Applications*. Vol. 21. 1993. pp. 191-195.
- [Has92] Hassibi, B. - Stork, D. G. "Second Order Derivatives for Network Pruning: Optimal Brain Surgeon", in: Giles, C. L., Hanson, S. J., Cowan, J. D. (Editors) *Advances in Neural Information Processing Systems*, Vol. 5. San Mateo, CA: Morgan Kaufman, pp. 164-171. 1993.
- [Has95] Hassoun, M. H.: "Fundamentals of Artificial Neural Networks", MIT Press, Cambridge, MA. 1995.
- [Has97] Hashem S. "Optimal Linear Combinations of Neural Networks", *Neural Networks*, Vol. 10. No. 4. pp. 599-614. 1997.
- [Has09] Hastie, T. - Tibshirani, R. - Friedman, J. "The Elements of Statistical Learning. Data Mining, Inference, and Prediction", Springer, 2009.
- [Hay99] Haykin, S. "Neural Networks. A Comprehensive Foundation", Second Edition, Prentice Hall, 1999.
- [He93] He, X. - Asada, H. "A New Method for Identifying Orders of Input-Output Models for Nonlinear Dynamic Systems", *Proc. of the American Control Conference*, San Francisco, CA. USA. pp. 2520-2523. 1993.
- [Hea02] Heath, M. T. "Scientific Computing, An Introductory Survey", McGraw-Hill, New York. 2002.
- [Heb49] Hebb, D. O. "The Organization of Behaviour", John Wiley and Sons, New York, 1949.
- [Hec87] Hecht-Nielsen, R. „Kolmogorov's Mapping Neural Network Existence Theorem" *Proc. of the Int. Conference on Neural Networks*, Vol. III. pp. 11-13. IEEE Press, N.Y. 1987.
- [Hec89] Hecht-Nielsen, R. "Neurocomputing", Addison-Wesley Publishing Co. 1989.
- [Her91] Hertz, J. - Krogh, A. - Palmer, R. G. "Introduction to the Theory of Neural Computation", Addison-Wesley Publishing Co. New York. 1991.
- [Hin89] Hinton, G. E. "Connectionist Learning Procedures", *Artificial Intelligence*, Vol. 41. pp. 185-234, 1989.
- [Hir91] Hirose, Y. - Yamashita, K. - Hijiya, S. "Back-Propagation Algorithm Which Varies the Number of Hidden Units", *Neural Networks*, Vol. 4. No. 1. pp. 61-66. 1991.
- [Hoe70] Hoerl, A. E. - Kennard, R. W. "Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, Vol. 12. No. 3. pp. 55-67, 1970.
- [Hol75] Holland, J. H. "Adaptation in Natural and Artificial Systems", Ann Arbor, MI: University of Michigan Press, 1975.
- [Hol77] Holland, P. W. - Welsch, R. E. "Robust Regression Using Iteratively Reweighted Least-Squares," *Communications in Statistics: Theory and Methods*, A6, pp. 813-827. 1977.

- [Hol92] Holland, J. H. "Genetikai algoritmusok", *Tudomány*, 1992. szept. pp. 28-34.
- [Hop82] Hopfield, J. J. "Neurons Networks and Physical Systems with Emergent Collective Computational Abilities", *Proc. Natl. Acad. Science*. Vol. 79. pp. 2554-2558. 1982.
- [Hor98] Horváth, G. (szerk), "Neurális hálózatok és műszaki alkalmazásai" Egyetemi tankönyv, Műegyetemi Kiadó, Budapest. 1998.
- [Hor99] Horváth, G. - Pataki, B. - Strausz, Gy. "Black box modeling of a complex industrial process", *Proc. of the 1999 IEEE Conference and Workshop on Engineering of Computer Based Systems*, Nashville, TN, USA. pp. 60-66. 1999.
- [Hor07] Horváth, G. - Szabó, T. „Kernel CMAC with Improved Capability”, *IEEE Trans. on Systems Man and Cybernetics, Part B*. Vol. 37. pp. 124-138. 2007.
- [Hu03] Hu, S. - Liao, X. - Mao, X. "Stochastic Hopfield Neural Networks", *Journal of Physics A: Mathematical and General*. Vol. 35. 2003. pp. 2235-2249.
- [Hua88] Huang, W. Y. - Lippmann, R. P. "Neural Nets and Traditional Classifiers", in: D. Z. Anderson (ed), *Neural Information Processing Systems*, (Denver, 1987) American Institute of Physics, pp. 387-396. New York, 1988.
- [Hub81] Huber, P. J. "Robust Statistics", Wiley, 1981.
- [Hun99] Hung S. L. - Jan, J. C. "MS_CMAL Neural Network Learning Model in Structural Engineering" *Journal of Computing in Civil Engineering*, pp. 1-11. 1999.
- [Hyv97] Hyvärinen A. - Oja, E. "One-unit Learning Rules for Independent Component Analysis", *Advances in Neural Information Processing Systems*, Vol. 9. pp. 480-486. Cambridge, MA. The MIT Press, 1997.
- [Hyv00] Hyvärinen A. - Oja, E. "Independent Component Analysis: Algorithms and Applications. *Neural Networks*, Vol. 13. No.4-5. pp. 411-430. 2000.
- [Hyv01] Hyvärinen, A. - Karhunen, J. - Oja, E. "Independent Component Analysis. John Wiley & Sons, New York, 2001.
- [Hyv06] Hyvärinen A. – Shimizu, S. "A Quasi-stochastic Gradient Algorithm for Variance-dependent Component Analysis", *Proc. International Conference on Artificial Neural Networks (ICANN2006)*, Athens, Greece, pp. 211-220, 2006.
- [Iig92] Iiguni, H. - Sakai, Y. "A Real-Time Learning Algorithm for a Multilayered Neural Network Based on the Extended Kalman Filter", *IEEE Trans. on Signal Processing*, Vol. 40. No. 4. pp. 959-966. 1992.
- [Ish96] Ishikawa, M. "Structural Learning with Forgetting", *Neural Networks*, Vol. 9. No. 3. pp. 509-521. 1996.
- [Jac91] Jacobs, R.A. - Jordan, M. I. - Nowlan, S. J. - Hinton, G. E. "Adaptive Mixture of Local Experts", *Neural Computation*, Vol. 3. pp. 79-89. 1991.
- [Jan01] Jan J. C. - Hung, S. L. "High-Order MS_CMAL Neural Network," *IEEE Trans. on Neural Networks*, vol. 12. pp. 598-603, May, 2001.
- [Jen93] Jenkins, R. E. - Yuhas, B. P. „A Simplified Neural Network Solution Through Problem Decomposition: The Case of the Truck Backer-Upper”, *IEEE. Trans. on Neural Networks*, Vol. 4. No. 4. pp. 718-720. 1993.
- [Joh92] Johansson, E. M. - Dowla, F. U. - Goodman, D. M. "Backpropagation Learning for Multilayer Feed-Forward Neural Networks Using the Conjugate Gradient Method", *International Journal of Neural Systems*, Vol. 2, No. 4. pp. 291-301. 1992.
- [Jor94] Jordan, M. I. - Jacobs, R.A. "Hierarchical Mixture of Experts and the EM Algorithm", *Neural Computation*, Vol. 6. pp. 181-214. 1994.
- [Jut88] Jutten, C. - Herault, J. "Independent component analysis versus PCA", *Proceedings of EUSIPCO*, pp. 643 - 646, 1988.

- [Kar94] Karhunen, J. - Joutsensalo, J. "Representation and Separation of Signals using Nonlinear PCA Type Learning", *Neural Networks*, Vol. 7. No. 1. pp. 113-127. 1994.
- [Kar97] Karhunen, J. - Hyvärinen, A. - Vigarío, R. - Hurri, J. - and Oja, E. "Applications of Neural Blind Source Separation to Signal and Image Processing", *Proc. of the IEEE 1997 International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*, April 21 - 24. Munich, Germany, pp. 131-134. 1997.
- [Kar98] Karhunen, J. - Pajunen, P., - Oja, E. "The Nonlinear PCA Criterion in Blind Source Separation: Relations with Other Approaches", *Neurocomputing*, Vol. 22. No. 1. pp. 5-20. 1998.
- [Kea96] Kearns, M. „A Bound on the Error of Cross Validation using the Approximation and Estimation Rates, with Consequences for the Training-Testing Split”, *NIPS'95*, D.S. Touretzky, M. C. Mozer, M. E. Hasselmo (eds.) *Advances in Neural Information Processing Systems*, MIT Press, MA. USA, 1996
- [Ker95] Ker, J. S. - Kuo, Y. H. - Liu, B. D. "Hardware Realization of Higher-order CMAC Model for Color Calibration", *Proceedings, of the IEEE International Conference on Neural Networks*, Perth, Vol. 4. pp. 1656-1661. 1995.
- [Kim90] Kim, S. P. - Bose, N. K. "Reconstruction of 2-D Bandlimited Discrete Signals from Nonuniform Samples", *IEEE Proceedings, Part F*. Vol. 137. No. 3. pp. 197-204. 1990.
- [Kne92] Knerr, S. - Personnaz, L. - Dreyfus, G. "Handwritten Digit Recognition by Neural Networks with Single-Layer Training", *IEEE Trans. on Neural Networks*, Vol. 3. 1992. pp. 962-969.
- [Knu88] Knuth, D. E. "A számítógép programozás művészete 3. Keresés és rendezés", Műszaki Könyvkiadó, Budapest, 1988.
- [Koh82] Kohonen, T. "Self-organized Formation of Topologically Correct Feature Maps", *Biological Cybernetics*, Vol. 434.. pp. 59-69. 1982
- [Koh86] Kohonen, T. "Learning Vector Quantization for Pattern Recognition", *Technical Report TKK-F-A601*. Helsinki University of Technology. 1986.
- [Koh88] Kohonen, T. "The 'Neural' Phonetic Typewriter", *Computer*. Vol. 21. 1988. pp. 11-22.
- [Koh89] Kohonen, T. „Self-Organization and Associative Memory”, Third edition, Springer, New York, 1989.
- [Koh90] Kohonen, T. "Improved Versions of Learning Vector Quantization", *International Joint Conference on Neural Networks*. Vol. 1. pp. 545-550. 1990.
- [Koh00] Kohonen, T. - Kaski, S. - Lagus, K. - Salojärvi, J. - Honkela, J. - Paatero, V. - Saarela, A. „Self Organization of a Massive Document Collection”. *IEEE Transactions on Neural Networks*, Special Issue on Neural Networks for Data Mining and Knowledge Discovery, Vol. 11, No. 3. pp. 574-585. 2000.
- [Kol57] Kolmogorov, A. N. "On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and addition", (Oroszul) *Dokl. Akad. Nauk. USSR*. Vol. 114. 1957. pp. 953-956.
- [Kro95] Krogh, A. - Vedelsby, J. "Neural Network Ensembles, Cross Validation, and Active Learning”, G. Tesauero, D. S. Touretzky and T. K. Leen (eds.): *Advances in Neural Information Processing Systems 7*. pp. 231-238. MIT Press, Cambridge, MA: 1995.
- [Kun90] Kung, S. Y. - Diamantaras, C.I. "A Neural Network Learning Algorithm for Adaptive Principal Component Extraction (APEX)", *Proc. of the International Conference on Acoustics, Speech and Signal Processing*, Vol. 2. pp. 861-864. 1990.
- [Kûr92] Kûrková, V. „Kolmogorov Theorem and Multilayer Neural Networks”, *Neural Networks*, Vol. 5. pp. 501-506. 1992.
- [Kûr97] Kûrková, V. - Kainen, P. C. - Kreinovich, V. "Estimates of the Number of Hidden Units and Variations with Respect to Half-Spaces", *Neural Networks*, Vol. 10. No. 6. pp. 1061-1068. 1997.

- [Kwo03] Kwok, J. T. - Tsang, I. W. "Linear Dependency Between ε and The Input Noise in ε -Support Vector Regression" *IEEE Trans. on Neural Networks*, Vol. 14. No.3. pp. 544-553. 2003.
- [Lag02] Lagus, K. „Text retrieval using self-organized document maps”, *Neural Processing Letters*. Vol. 15, No. 1. pp. 21-29. 2002.
- [Lag04] Lagus, K. - Kaski, S. - Kohonen, T. „Mining massive document collections by the WEBSOM method” *Information Sciences*, Vol 163. No. 1-3. pp. 135-156. 2004.
- [Lan84] Landau, I. D. "A Feedback System Approach to Adaptive Filtering", *IEEE Trans. on Information Theory*, Vol. IT 30. No 2. pp. 251-267. 1984.
- [Lan88] Lang K. J. -Hinton, G. E. "The development of the time-delay neural network architecture for speech recognition" *Technical Report*, CMU-CS-88-152, Carnegie-Mellon University, Pittsburg, PA.
- [Lan92] Lane, S. H. - Handelman, D. A. - Gelfand, J. J "Theory and Development of Higher-Order CMAC Neural Networks", *IEEE Control Systems Magazine*, Vol. 12. No. 2. pp. 23-30. 1992.
- [Lau06] Lauer, F. - Bloch, G. "Incorporating Prior Knowledge in Support Vector Machines for Classification: a Review", HAL-CCSD-CNRS, 2006.
<http://hal.ccsd.cnrs.fr/docs/00/06/35/21/PDF/LauerBlochNeurocomp06.pdf>.
- [Le06] Le, Q. V. - Smola, A. J. - Gärtner, T. "Simpler Knowledge-based Support Vector Machines", *Proc. of the 23rd International Conference on Machine Learning*, Pittsburg, PA. pp. 521-528. 2006.
- [LeC89] LeCun, Y. - Boser, B. - Denker, J. S. "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, Vol. 1. pp. 541-551. 1989.
- [LeC90] Le Cun, Y. - Denker, J. S. - Solla, S. "Optimal Brain Damage", in: Touretzky, D. (Ed.) *Advances in Neural Information Processing Systems*, Vol. 2. San Mateo, CA. Morgan Kaufman, pp. 598-605. 1990.
- [Lee01a] Lee, Y-J. - Mangasarian, O. L. "SSVM: A Smooth Support Vector Machine", *Computational Optimization and Applications*, Vol. 20. No. 1. pp. 5-22. 2001.
- [Lee01b] Lee, Y-J. - Mangasarian, O. L., "RSVM: Reduced Support Vector Machines", *Proc. of the First SIAM International Conference on Data Mining*, Chicago, 2001.
- [Lee03] Lee, H. M. Chen C. M. - Lu, Y. F. "A Self-Organizing HCMAC Neural-Network Classifier," *IEEE Trans. on Neural Networks*, vol. 14. pp. 15-27. Jan. 2003.
- [Lei91] Leighton, R. - Conrath, B. "The Autoregressive Backpropagation Algorithm", *Proc. of the 1991 International Joint Conference on Neural Networks, IJCNN'91*. Vol. 2. pp. 369-377. 1991.
- [Les93] Leshno, M. - Lin, V. Y. - Pinkus, A. - Schocken, S. "Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function", *Neural Networks*, Vol. 6. pp. 861-867. 1993.
- [Leu01] Leung, C. S. - Tsoi, A. C., "Two-regularizers for Recursive Least Squared Algorithms in Feedforward Multilayered Neural Networks", *IEEE Trans. on Neural Networks*, vol. 12, pp. 1314-1332. 2001.
- [Lew96] Lewis, F. Yesildirek, A. Liu, K. "Multilayer neural-net robot controller with guaranteed tracking performance". *IEEE Trans. on Neural Networks*, Vol. 7. No. 2. pp. 388-399. 1996.
- [Li04] Li C. K. - Chiang, C. T. "Neural Networks Composed of Single-variable CMACs," *Proc. of the 2004 IEEE International Conference on Systems, Man and Cybernetics*, The Hague, The Netherlands, pp. 3482-3487, 2004.
- [Li05] Li, H., Shi, K., McLaren, P., „Neural-Network-Based Sensorless Maximum Wind Energy Capture with Compensated Power Coefficient", *IEEE Trans. on Industry Applications*, Vol. 41, No. 6. pp. 1548-1556. 2005.
- [Lin96] Lin C. S. - Li, C. K. "A Low-Dimensional-CMAC-Based Neural Network", *Proceedings of IEEE International Conference on Systems, Man and Cybernetics* Vol. 2. pp. 1297-1302. 1996.

- [Lin98] Lindsey, C. S. - Lindblad, T. "Review of Hardware Neural Networks: A User's Perspective", <http://msia02.msi.se/~lindsey/> Oct. 1998.
- [Lip92] Lipschitz, S. P. - Wannamaker, R. A. "Quantization and Dither a Theoretical Survey", *Journal of the Audio Engineering Society*, Vol. 40. No. 5. pp. 355-375. 1992.
- [Lju99] Ljung, L.: "System Identification, Theory for the User", Second edition, Prentice Hall, Upper Saddle River, N. J. 1999.
- [Lor76] Lorentz, G. G. "The 13-th Problem of Hilbert", *Proc. of Symposia in Pure Mathematics*, Vol. 28. pp. 419-430. 1976.
- [Lu99] Lu, B. L. Ito, M. „Task decomposition and Module Combination Based on Class Relations: A Modular Neural Network for Pattern Classification" *IEEE. Trans. on Neural Networks*, Vol. 10. No. 5. pp. 1244-1256. 1999.
- [Mac95] Maclin, R. F., "Learning from Instruction and Experience: Incorporating Procedural Domain Theories into Knowledge-Based Neural Networks", *Ph.D Thesis*, University of Wisconsin, Madison, USA, 1995.
- [Mai99] Maiorov, V. - Pinkus, A. "Lower Bounds for Approximation by MLP Neural Networks", *Neurocomputing*, Vol. 25, pp. 81-91.1999.
- [Mal96] Malthouse, E. C. "Some Theoretical Results on Nonlinear Principal Components Analysis", ftp from <http://skew2.kellogg.nwu.edu/~ecm>, 24 p. 1996.
- [Man04] Mangasarian, O. L. - Shavlik, J. - Wild, E. W. "Knowledge-Based Kernel Approximation", *Journal of Machine Learning Research*, Vol. 5. pp. 1127-1141. 2004.
- [Mát65] Mátyás, J. "Random Optimization", *Automation and Remote Control*, Vol. 26. pp. 246-253. 1965.
- [McC43] McCulloch, W. S. - Pitts W. "A Logical Calculus of Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, pp. 115-133. 1943.
- [Mer06] Merler, S. - Jurman, G. "Terminated Ramp-Support Vector Machines: A nonparametric data dependent kernel", *Neural Networks*, Vol. 19. No. 10. pp. 1597-1611. 2006.
- [Mha94] Mhaskar, H. N. „Approximation of Real Functions using Neural Networks", in: H. P. Dikshit, C. A. Micchelli (eds.) *Advances in Computational Mathematics*, New Delhi, India. World Scientific, Singapore, pp. 267-278. 1994.
- [Mik99] Mika, S. - Schölkopf, B. - Smola, A. - Müller, K-R. - Schultz, M. - Rätsch, G. "Kernel PCA and De-Noising in Feature Spaces", in: M. S. Kearns, S. A. Solla, D. A. Kohn (eds.) *Advances in Neural Information Processing Systems*, Vol. 11. pp. 536-542. Cambridge, MA. The MIT Press, 1999.
- [Mil91] Miller, W. T. - Box, B. A. - Whitney E. C. "Design and Implementation of a High Speed CMAC Neural Network Using Programmable CMOS Logic Cell Arrays", *Advances in Neural Information Processing Systems* 3. pp. 1022-1027. 1991.
- [Min69] Minsky, M. - Papert, S. "Perceptrons", MIT Press, Cambridge MA. 1969.
- [Mit97] Mitchell, T. "Machine Learning", McGraw-Hill, 1997.
- [Mon89] Montana, D. L. - Davis, L "Training Feedforward Neural Networks Using Genetic Algorithms", *Proc. of the Eleventh IJCAI*, 1989. pp. 762-767. 1989.
- [Moo88] Moody, J - Darken, C. "Learning with Localized Receptive Fields", *Proceedings of the 1988 Connectionist Models Summer School*, Pittsburg, D Touretzky, G. Hinton, T. Sejnowsky (eds.) pp. 174-185. San Mateo, Morgan Kaufmann, 1988.
- [Moo89] Moody, J. "Fast Learning in Multi-Resolution Hierarchies", *Advances in Neural Information Processing*, 1. Touretzky (Ed.) Los Altos, CA. Morgan Kaufmann, pp. 474-481. 1989.

- [Mor77] Moré, J. J. "The Levenberg-Marquardt Algorithm, Implementation and Theory" G. A. Watson (ed.) Numerical Analysis, Lecture Notes in Mathematics, Vol. 630. pp. 105-116, Springer, 1977.
- [Muk97] Mukherjee, S. - Osuna, E. - Girosi, F. "Nonlinear Prediction of Chaotic Time Series using a Support Vector Machine", *IEEE Workshop on Neural Networks for Signal Processing, NNSP'97*, pp. 511-520. 1997.
- [Mur94] N. Murata, S. Yoshizawa and Shun-Ichi Amari, Network Information Criterion - Determining the Number of Hidden Units for an Artificial Neural Network Model, *IEEE Trans. on Neural Networks*, Vol. 5. No. 6. pp. 865-871. 1994.
- [Mül97] Müller, K. R. - Smola, A. - Schölkopf, B. - Rätsch, G. - Kohlmorgen, J. - Vapnik, V. "Predicting Time Series with Support Vector Machines", *Proc. of the Artificial Neural Networks, ICANN'97*, pp. 999-1004. Springer, 1997.
- [Nak97] Nakanishi, K. - Tkayama, H. "Mean-field theory for a spin-glass model of neural networks: TAP free energy and the paramagnetic to spin-glass transition ", *Journal of Physics A: Mathematical and General*. Vol. 30. 1997. 8085-8094
- [Nar83] Narayan, S. S. - Peterson, A.M. - Narashima, M. J. „Transform domain LMS algorithm" *IEEE Trans. on Audio Speech and Signal Processing*, Vol. 31, pp. 631-639. 1983.
- [Nar89b] Narendra, K. S. - Thathachar, M. A. L "Learning Automata: An Introduction", Prentice Hall, Englewood Cliffs, N. J. 1989.
- [Nar90] Narendra, K. S. - Pathasarathy, K. "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Trans. Neural Networks*, Vol. 1. pp. 4 27. 1990.
- [Nar91] Narendra, K. S. - Pathasarathy, K. "Identification and Control of Dynamic Systems Using Neural Networks", *IEEE Trans. on Neural Networks*, Vol. 2. pp. 252-262. 1991.
- [Nar05] Narendra, K. S. - Annaswamy, A. M. "Stable Adaptive Systems", Dover Publications, 2005.
- [Ner93] Nerrand, O. - Roussel-Ragot, P. - Personnaz, L. - Dreyfus, G. " Neural Networks and Nonlinear Adaptive Filtering: Unifying Concepts and New Algorithms", *Neural Computation*, Vol. 5. pp. 165-199. 1993.
- [Ngu89] Nguyen, D. - Widrow, B. "The Truck Backer-Upper: An Example of Self-Learning" *Proc. of the Neural Networks IEEE International Joint Conference on Neural Networks, IJCNN'89*, Vol. II, p. II-357-II-364, IEEE, 1989.
- [Ngu90] Nguyen, D. - Widrow, B. „Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights", *Proc. of the Intl. Joint Conference on Neural Networks*, Vol. III. pp. 21-26. 1990.
- [Nil65] Nilsson, N. J. "Learning Machines", McGraw-Hill, New York, 1965.
- [Niy98] Niyogi, P. - Girosi, F. - Poggio, T. "Incorporating Prior Information in Machine Learning by Creating Virtual Examples" *Proc. of the IEEE*. Vol. 86. No. 11. pp. 2196-2209. 1998.
- [Nør00] Nørgaard, M. Ravn, O. - Poulsen N. K. - Hansen, L. K. "Neural Networks for Modelling and Control of Dynamic Systems", Springer-Verlag, London, 2000.
- [Oja82] Oja, E. "A Simplified Neuron Model as a Principal Component Analyzer", *Journal of Mathematical Biology*, Vol. 15. pp. 267-273. 1982.
- [Oja83] Oja, E. "Subspace Methods for Pattern Recognition", Research Studies Press, Letchworth, Hertfordshire, England, 1983.
- [Opi95] Opitz, D. W. "An Anytime Approach to Connectionist Theory Refinement: Refining the Topologies of Knowledge-Based Neural Networks", *Ph.D Thesis*, University of Wisconsin, Madison, USA, 1995.
- [Osu99] Osuna, E. - Girosi, F. "Reducing run-time complexity in SVMs", in: B. Schölkopf, C. Burges, A. J. Smola (eds.) *Advances in Kernel Methods – Support Vector Learning*. MIT Press, Cambridge, MA. pp. 271-284. MIT Press. 1999.

- [Par91] Park, J. - Sandberg, I. W. „Universal Approximation using Radial Basis Function Networks” *Neural Computation*, Vol. 3. pp. 246-257, 1991.
- [Par92] Parks, P. C. - Militzer, J. "A Comparison of Five Algorithms for the Training of CMAC Memories for Learning Control Systems", *Automatica*, Vol. 28. No. 5. pp. 1027-1035. 1992.
- [Par93] Park, J. - Sandberg, I. W. "Approximation and Radial-Basis-Function Networks", *Neural Computation*, Vol 5. No. 2. pp. 305-316. 1993.
- [Pat98] Pataki, B. - Horváth, G. - Strausz, Gy. "Effects of Database Characteristics on the Neural Modeling of an Industrial Process", *Proc. of the International ISCS/IFAC Symposium on Neural Computation (NC'98)*. Vienna. pp. 834-840. 1998.
- [Pat00] Pataki, B., Horváth, G., Strausz, Gy., Talata, Zs. "Inverse Neural Modeling of a Linz- Donawitz Steel Converter" *e & i Elektrotechnik und Informationstechnik*, Vol. 117. No. 1. pp. 13-17. 2000.
- [Pav05] Pavlovic, V. - Schonfeld, D. – Friedman G. "Stochastic Noise Process Enhancement of Hopfield Neural Networks", *IEEE Trans. on Circuits and Systems*, Vol. 52. No. 4., pp. 213-217, 2005.
- [Pel04] Pelckmans, K. - Espinoza, M. - De Brabanter, J. - Suykens, J. A. K. - De Moor, B. "Primal-Dual Monoton Kernel Regression” *Neural Processing Letters*, Vol. 4. No. 6. pp. 17-31. 2004.
- [Pen55] Penrose, R. "A Generalized Inverse for Matrices" *Proc. of the Cambridge Philosophical Society*, Vol. 51. pp. 406-413. 1955.
- [Pet72] Peterson, W. W. – Weldon, E. J. "Error-correcting Codes", second edition, MIT Press, Cambridge, MA. 1972.
- [Pie99] Pietruschka, U., Brause, R., „Using Growing RBF-Nets in Rubber Industry Process Control”, *Neural Computing & Applications*, No. 8, 1999, pp. 95-105.
- [Pin01] Pintelon, R. - Schoukens, J. "System Identification, The Frequency Domain Approach", MIT Press. 2001.
- [Pla86] Plaut, D. C. - Nowlan, S. J. - Hinton, G. E. "Experiments on Learning with Back-propagation", *Technical Report*, CMU-CS-86-126. Carnegie-Mellon University, 1986.
- [Pla99] Platt, J. C. "Fast Training of Support Vector Machines using Sequential Minimal Optimization", in: B. Schölkopf, C. Burges, A. J. Smola (eds.) *Advances in Kernel Methods – Support Vector Learning*. MIT Press, Cambridge, MA. pp. 185-208. 1999.
- [Plu93] Plumbly, M. "A Hebbian /anti-Hebbian network which Optimizes Information Capacity by Orthonormalizing the Principal Subspace” *Proc. IEE Conference on Artificial Neural Networks*, Brighton, UK. pp. 86-90. 1993
- [Pog90] Poggio, T. - Girosi, F. „Networks for approximation and Learning”, *Proc. of the IEEE*, Vol. 78. pp. 1481-1497. 1990.
- [Pom91] Pomerleau, D. A. "Efficient Training of Artificial Neural Networks for Autonomous Navigation", *Neural Computations*, Vol. 3. No. 1. pp. 88-97. 1991.
- [Pra96] Prabhu, S. M. - Garg, D. P. "Artificial Neural Network Based Robot Control: An Overview” *Journal of Intelligent and Robotic Systems*, Vol. 15, No. 4, pp. 333-365. 1996.
- [Pre02] Press, W. H. - Teukolsky, S. A. - Wetterling W. T. - Flannery, B. P. "Numerical Recipes in C", Cambridge University Press, Books On-Line, <http://www.nr.com>, 2002.
- [Rad93] Radcliffe, N. "Genetic Set Recombination and its Application to Neural Network Topology Optimisation", *Neural computing and application*, Vol. 1. No. 1. pp. 67-90. 1993.
- [Ral04] Ralaivola, L. - d’Alché-Buc, F. "Dynamical Modeling with Kernels for Nonlinear Time Series Prediction” *Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, B. Schölkopf (Eds.) MIT press, MA, 2004.

- [Ris78] Rissanen, J. "Modelling by Shortest Data Description", *Automatica*, Vol. 14. pp. 465-471, 1978.
- [Riv94] Rivals, L. - Cannas, D. - Personnaz, L. - Dreyfus, G. "Modelling and Control of Mobile Robots and Intelligent Vehicles by Neural Networks", *Proc. of the IEEE Conference on Intelligent Vehicles*, pp. 137-142. 1994.
- [Rob91] Robinson, T. - Fallside, F. "A recurrent error propagation network speech recognition system," *Computer Speech & Language*, Vol. 5. No. 3. pp. 259-274. 1991.
- [Ros58] Rosenblatt, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization of the Brain", *Psychol. Rev.*, Vol. 65. pp. 386-408. 1958.
- [Róz91] Rózsa P. „Lineáris algebra és alkalmazásai”, 3. átdolgozott kiadás, Tankönyvkiadó, Budapest, 1991.
- [Rum86] Rumelhart, D. E. - Hinton, G. E. - Williams, R. J. "Learning Internal Representations by Error Propagation", in Rumelhart, D. E. - McClelland, J. L. (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1. MIT Press. pp. 318-362. 1986.
- [Rus05] Russel, S. - Norvig, P. "Mesterséges intelligencia modern megközelítésben", magyar fordítás Panem, 2005
- [San89] Sanger, T. "Optimal Unsupervised Learning in a Single-layer Linear Feedforward Neural Network", *Neural Networks*, Vol. 2. No. 6. pp. 459-473. 1989.
- [San93] Sanger, T. "Relation Between the APEX Algorithm and GHA", Manuscript on ftp 1993.
- [Sca92] Scalero, R. S. - Tepedelenlioglu, N. "A Fast New Algorithm for Training Feedforward Neural Networks", *IEEE Trans. on Signal Processing*, Vol. 40. No. 1. pp. 202-210. 1992.
- [Sca98] Scarselli, F. - Tsoi, A. H. "Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods and Some New Results", *Neural Networks*, Vol. 11. No. 1. pp.15-37. 1998.
- [Sch96a] Schölkopf, B. - Sung, K. - Burges, C. - Girosi, F. - Niyogi, P. - Poggio, T. - Vapnik, V. "Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers", AIM-1599. MIT. 1996. <ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-1599.pdf>
- [Sch96b] Schölkopf, B. - Burges, C. - Vapnik, V. "Incorporating Invariances in Support Vector Learning Machines" in: C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, B. Sendhoff (eds): *Artificial Neural Networks- ICANN'96*, Springer Lector Notes in Computer Sciences. Vol. 1112, Berlin pp. 47-52. 1996.
- [Sch96c] Schölkopf, B. - Smola, A. - Müller, K.-R. "Nonlinear Component Analysis as a Kernel Eigenvalue Problem", *Technical Report No. 44*. Max-Planck-Institut für biologische Kibernetik. 1996.
- [Sch97] Schölkopf, B. "Support Vector Learning", R. Oldenburg Verlag, Munich, 1997.
- [Sch99] Schölkopf, B. - Burges, C. - Smola, A. "Advances in Kernel Methods Support Vector Learning", MIT Press, MA. USA. 1999.
- [Sch02] Schölkopf, B. - Smola, A. "Learning with Kernels Support Vector Machines, Regularization, Optimization and Beyond", MIT Press, MA. USA. 2002.
- [Scha90] Schapire, R. E. "The Strength of Weak Learnability", *Machine Learning*, Vol. 5. pp. 197-227. 1990.
- [Scha99] Schapire, R. E., "A Brief Introduction to Boosting", *Proc. of the Sixteenth International Joint Conference on Artificial Intelligence, (IJCAI-99)* Vol. 2. pp. 1401-1406, 1999.
- [Scha02] Schapire, R. E., "The Boosting Approach to Machine Learning: An Overview", *MSRI Workshop on Nonlinear Estimation and Classification*, Berkeley, CA, 2002.
- [Sej87] Sejnowski, T. J. - Rosenberg, C. R. "Parallel Networks that Learn to Pronounce English Text", *Complex Systems*, Vol. 1. pp. 145-168. 1987.

- [Šer96] Šerbedžija, N. B. "Simulating Artificial Neural Networks on Parallel Architectures", *Computer IEEE*, March, pp. 56-63. 1996.
- [Sha92] Shah, S. - Palmieri, F. - Datum, M. "Optimal Filtering Algorithms for Fast Learning in Feedforward Neural Networks", *Neural Networks*, Vol. 5. pp. 779-787. 1992.
- [Sha02] Shawe-Taylor, J. - Cristianini, N. "On the Generalization of Soft Margin Algorithms", *IEEE Trans. on Information Theory*, Vol. 48. pp. 2721-2735. 2002.
- [Sim00] Simoes, M., Furukawa, C., Mafra, A., Adamowski, J. C., „A Novel Competitive Learning Neural Network Based Acoustic Transmission System for Oil-Well Monitoring”, *IEEE Transactions on Industry Applications*, Vol. 36, No. 2, pp. 484-491. 2000.
- [Sjö95] Sjöberg, J. - Zhang, Q. - Ljung, L. - Benveniste, A. - Deylon, B. - Glorennec, P.-Y. - Hjalmarsson, H. - Juditsky, A. "Nonlinear black-Box Modeling in System Identification: a Unified Overview" *Technical Report*, Linköping University, No: LiTH-ISY-R-1742, 1995.
- [Sma06] Van Der Smagt, P. – Omidvar, O. M. (eds), "Neural Systems for Robotics" Academic Press, 2006.
- [Sol81] Solis, F. J. - Wets, J. B. "Minimization by Random Search Techniques", *Mathematics of Operation Research*, Vol. 6. pp. 19-30. 1981.
- [Sol96] Sollich, P. - Krogh, A. "Learning with ensembles: How overfitting can be useful", *Advances in Neural Information Processing Systems*, Vol. 8, pp. 190-196, The MIT Press, 1996.
- [Spr65] Sprecher, D. A. „On the Structure of Continuous Functions of Several Variables”, *Transactions American Mathematical Society*, Vol. 115, pp. 340-355. 1965.
- [Spe93] Sperduti, A. - Starita, A. "Speed Up Learning and Network Optimization With Extended Back Propagation", *Neural Networks*, Vol. 6. pp. 365-383. 1993.
- [Str98] Strausz, Gy. - Horváth, G. - Pataki, B., „Experiences from the results of neural modelling of an industrial process” *Proc. of Engineering Application of Neural Networks, EANN'98*, Gibraltar pp. 213-220. 1998.
- [Sto78] Stone, M. "Cross-Validation: A Review", *Mathematische Operationsforschung Statistischen*, Vol. 9. pp. 127-140. 1978.
- [Su03] Su, S. F. - Tao T. - Hung, T. H. "Credit assigned CMAC and Its Application to Online Learning Robot Controllers", *IEEE Trans. on Systems Man and Cybernetics- Part B*. Vol. 33, pp. 202-213. 2003.
- [Sum99] Sum, J. - Leung, C. - Young, G. - Kay, W. „On the Kalman-filtering Method in Neural Network Training and Pruning” *IEEE Trans. on Neural Networks*, Vol. 10, pp. 161-166. 1999.
- [Sun00] Sun, Y. "Hopfield Neural Network Based Algorithms for Image Restoration and Reconstruction", *IEEE Trans. on Signal Processing*, Vol. 48, No. 7, July 2000. pp. 2105-2131.
- [Sun05] Sun, Q. - DeJong, G. "Explanation-Augmented SVM: an Approach to Incorporating Knowledge into SVM Learning", *Proc. of the 22nd International Conference on Machine Learning*, Bonn, Germany. pp. 864-871. 2005.
- [Sut98] Sutton, R. S., - Barto, A. G. "Reinforcement Learning: An Introduction", Bradford Book, MIT Press, Cambridge, 1998.
- [Suy99] Suykens, J. - deMoor, B. – Vandewalle, G. E. "The development of the time-delay neural network architecture for speech recognition" *Technical Report*, CMU-CS-88-152, Carnegie-Mellon University, Pittsburgh, PA. 1999.
- [Suy00] Suykens, J. A. K. - Lukas, L. - Vandewalle, J. "Sparse least squares support vector machine classifiers", *ESANN'2000 European Symposium on Artificial Neural Networks*, pp. 37-42. 2000,
- [Suy02a] Suykens, J. A. K. - Gestel, V. T. - De Brabanter, J. - De Moor, B. - Vandewalle, J. "Least Squares Support Vector Machines", World Scientific, 2002.

- [Suy02b] Suykens, J. A. K. De Brabanter, J. - Lukas, L. - Vandewalle, J. "Weighted least squares support vector machines: robustness and sparse approximation", *Neurocomputing*, Vol. 48. No. 1-4, pp. 85-105. 2002.
- [Szök68] Szőkefalvi-Nagy Béla: "Valós függvények és függvénysorok", Tankönyvkiadó, Budapest, 1968.
- [Tak93] Takahashi, Y., Generalization and Approximation Capabilities of Multilayer Networks, *Neural Computation*, Vol. 5, pp. 132-139. 1993.
- [Tak03] Takács G. "Írányítószám-felismerő rendszer", TDK dolgozat, BME VIK. 2003.
- [Tik77] Tikhonov, A. N. - Arsenin, V. Y. "Solutions of Ill-posed Problems", Washington, DC: W. H. Winston, 1997.
- [Tow91] Towell, G. G. "Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction", *Ph.D Thesis*, University of Wisconsin, Madison, USA, 1991.
- [Val84] Valiant, L. G. „A theory of the learnable” *Communications of the ACM*, Vol. 27. pp. 1134-1142, 1984.
- [Val03] Valyon, J. - Horváth, G. „A generalized LS-SVM”, *SYSID'2003*, Rotterdam, pp. 827-832. 2003,
- [Val04] Valyon, J. - Horváth, G., „A Sparse Least Squares Support Vector Machine Classifier”, *Proc. of the International Joint Conference on Neural Networks, IJCNN 2004*, pp. 543-548. 2004.
- [Van92] Van Ooyen, A. "Improving the Convergence of the Back Propagation Algorithm", *Neural Networks*, Vol. 5, pp. 465-471. 1992.
- [Van00] Van Gorp, - J. Schoukens, J. - Pintelon, R. „Learning Neural Networks with Noisy Inputs Using the Errors-in-Variables Approach”, *IEEE Trans. on Neural Networks*, Vol. 11. No. 2, pp. 402-414. 2000.
- [Vap79] Vapnik, W. N. - Tschervonenkis, A. Ja. "Teorie der Zeichenerkennung", Akademie, Berlin, 1979.
- [Vap91] Vapnik, V. Chervonenkis, A. Ya. „The Necessary and Sufficient Conditions for Consistency of the Method of Empirical Risk Minimization”, *Pattern Recognition and Image Analysis*, Vol. 1, pp. 284-305, 1991.
- [Vap95] Vapnik, V. N. "The Nature of Statistical Learning Theory", Springer, New York, 1995.
- [Vap98] Vapnik, V. N. "Statistical Learning Theory", Wiley-Interscience, 1998.
- [Vap99] Vapnik, V. N. "An overview of statistical learning theory" *IEEE Trans. on Neural Networks*, Vol. 10. No. 5, pp. 988-1000. 1999.
- [Vee95] Veenturf, L. P. J. "Analysis and Application of Artificial Neural Networks", Prentice Hall, London, 1995.
- [Vid93] Vidyasagar, M. "Convergence of Higher-Order Neural Networks With Modified Updating", *Proc. of the Intl. Conference on Neural Networks*, Vol.3, pp. 1379-1384. 1993.
- [Vin00] Vincent, P. - Bengio, J. "A Neural support Vector Network Architecture with Adaptive Kernels", *Proc. of the Intl. Joint Conference on Neural Networks, IJCNN 2000*, pp. 187-192. 2000.
- [Vog88] Vogl, T. P. - Manglis, J. K. - Rigler, A. K. - Zink, W. T. - Alkon, D. L. "Accelerating the Convergence of the Back-propagation Method", *Biological Cybernetics*, Vol. 59, pp. 257-263. 1988.
- [Wai89] Waibel, A. "Modular construction of time-delay neural networks for speech recognition", *Neural Computation*, Vol. 1. No. 1, pp. 39-46. 1989.
- [Wal91] Wallace, G. K. "The JPEG Still Picture Compression Standard", *Communication of the ACM*, Vol. 34, No. 4, pp. 30-44. 1991.
- [Wan90] Wan, E. A. "Temporal Backpropagation for FIR Neural Networks", *Proc. of the Intl. Joint Conference on Neural Networks, IJCNN 1990*, Vol. I, pp. 575-580. 1990.

- [Wan93] Wang, Y. "Analog CMOS Implementation of Backward Error Propagation", *Proc. of the Intl. Joint Conference on Neural Networks, IJCNN 1993*, pp. 701-706. 1993.
- [Wan94] Wan, E. A. "Time Series Prediction by Using a Connectionist Network with Internal Delay Lines", *Time Series Prediction: Forecasting the Future and Understanding the Past*, (Eds: Eigend, A.S. - Gershenfeld, N.A) pp. 195-217. Addison Wesley, 1994.
- [Wan96a] Wang, Z. Q. - Schiano J. L. - Ginsberg, M. "Hash Coding in CMAC Neural Networks," *Proc. of the IEEE International Conference on Neural Networks*, Washington, USA. vol. 3. pp. 1698-1703, 1996.
- [Wan96b] Wang, L-Y. - Karhunen, J. "A Unified Neural Bigradient Algorithm for Robust PCA and MCA", *International Journal of Neural Systems*, Vol. 7. No. 1. pp. 53-67. 1996.
- [Wei94] Weigend, A. S. - Gershenfeld, N. A. "Forecasting the Future and Understanding the Past" Vol.15. Santa Fe Institute Studies in the Science of Complexity, Reading, MA. Addison-Wesley, 1994.
- [Wer74] Werbos, P. J. "Beyond Regression: New Tools for Prediction and Analysis in the Behaviour Sciences", *Ph. D Thesis*, Harvard University, Cambridge, MA. 1974.
- [Wer93] Werntges, H. W. "Partitions of Unity Improve Neural Function Approximations", *Proc. of the IEEE International Conference on Neural Networks*, San Francisco, USA. Vol. II. pp. 914-918, 1993.
- [Wer00] Wermter, S. - Sun, R. (eds.), "Hybrid Neural Systems", *Lecture Notes in Artificial Intelligence, LNAI 1778*. Springer, 2000.
- [Wid60] Widrow, B. - Hoff, M. E. "Adaptive Switching Circuits" *IRE WESCON Convention Record*, pp. 96-104. 1960.
- [Wid85] Widrow, B. - Stearns, S. D. "Adaptive Signal Processing", Prentice-Hall, Englewood Cliffs, N. J. 1985.
- [Wid90b] Widrow, B. – Gluck, M. A. "Adaptive Neural Networks", 1990. Annual International Course on Neural Networks, London. 1990.
- [Wil89] Williams, R. J. - Zipser, D. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks", *Neural Computation*, Vol. 1. pp. 270-280. 1989.
- [Won93] Wong, Yiu-fai "CMAC Learning is Governed by a Single Parameter", *Proc. of the IEEE International Conference on Neural Networks, IJCNN 1993*. pp. 1439-1443. 1993.
- [Wu04] Wu, X. - Srihari, R. "Incorporating Prior Knowledge with Weighted Margin Support Vector Machines", *Proc. of the Tenth International Conference on Knowledge Discovery and Data Mining, KDD'04*, Seattle, Washington, USA, pp. 326—333. 2004.
- [Zar00] Zarate, L. E. "A Model for the Simulation of a Cold Rolling Mill, Using Neural Networks and Sensitivity Factors," *VI. Brazilian Symposium on Neural Networks (SBRN'00)*, p. 185-190. 2000.
- [Zee97] Zeevi, A. - Meir, R. - Adler, R. "Time series prediction using mixtures of experts", *Advances in Neural Information Processing Systems 9*, pp. 309-315, MIT Press, 1997.
- [Zha92] Zhang, Q. - Benveniste, A. „Wavelet Networks”, *IEEE Trans. on Neural Networks*, Vol. 3. pp. 889-898, 1992.
- [Zha97] Zhang, Q. „Using Wavelet Networks in Nonparametric Estimation”, *IEEE Trans. on Neural Networks*, Vol. 8. pp. 227-236, 1997.
- [Zha05] Zhang, H. – Wu, Y. – Peng, Q. " Image Restoration Using Hopfield Neural Network Based on Total Variational Model ", *Advances in Neural Networks – ISNN 2005*, Lecture Notes in Computer Science., Springer Verlag, Berlin, pp. 735-740, 2005.
- [Zho97] Zhong, L. - Zhongming Z. - Chongguang, Z. "The Unfavorable Effects of Hash Coding on CMAC Convergence and Compensatory Measure," *IEEE International Conference on Intelligent Processing Systems*, Beijing, China, pp. 419-422, 1997.

- [Xio04] Xiong, H. - Swamy, M. N. S. - Ahmad, M. O. "Learning with the Optimized Data-Dependent Kernel" *Proc. of the 2004 Computer Society Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04)*, pp. 95-98. 2004.
- [Xio05] Xiong, H. - Swamy, M. N. S. - Ahmad, M. O. "Optimizing the Kernel in the Empirical Feature Space", *IEEE Trans. on Neural Networks*, Vol. 16. No.2 pp. 460-474, 2005.
- [Yan00] Yang, S. X. - Meng, M. Q.-H.: An efficient neural network approach to dynamic robot motion planning. *Neural Networks*, Vol.13. No. 2. pp. 143-148. 2000.
- [Yu95] Yu, X. H. - Chen, G. A. - Cheng, S. X. "Dynamic Learning Rate Optimization of the Backpropagation Algorithm", *IEEE Trans. on Neural Networks*, Vol. 6. No. 3. pp. 669-677. 1995.
- [Yu97] Yu, X. H. - Chen, G. A. "Efficient Backpropagation Learning Using Optimal Learning Rate and Momentum", *Neural Networks*, Vol. 10. No. 3. pp. 517-527. 1997.
- [Yu04] Yu, T. - Jan, T. - Debenham, J. - Simoff, S. "Incorporating Prior Domain Knowledge in Machine Learning: A Review", *Proc. of the IEEE Intntl. Conference on Advances in Intelligent Systems Theory and Applications*, Luxembourg, 2004.