# NEURAL NETWORKS

SUMMARY OF THEOREM
PPKE ITK, JANUARY 7, 2017

WRITTEN BY

KOMPORDAY ANDRÁS

2016

# Contents

# Artificial Neurons and the Perceptron Learning

## 1 1. ELŐADÁS

The particularized model of artificial neurons cells consists of three main, and several minor parts, based on its biological counterpart. The three mayor parts are:

- The **input** contains **stimuli** and **synopses**. These input parameters with are mathematically **value** and a **weight**. Inputs with negative weight are called inhibitory, while the ones with positive weight are called excitatory.
- The **nerve cell** is made up of the sum of the inputs, which then goes into a $\phi$ function. This $\phi$ function is monotonically increasing and has a lower bound of -1 and an upper bound of 1. A $b$ input acts as a threshold value for the function.

$$\lim_{n\to-\infty}\phi(n) = -1$$

$$\lim_{n\to\infty}\phi(n) = 1$$

$$\phi(n) = \tanh^{-1}(n)$$

- The **output** is a single $y$ value made of the following equation:

$$y = \phi(\sum_{j=1}^{N}(x_j y_j - b))$$

The activation function of the AN can be chosen from a large set of functions, however, the implementation of the AN we are interested in, namely McCulloch-Pitts neuron uses the $sgn(\overline{w}^T\overline{x})$ function.

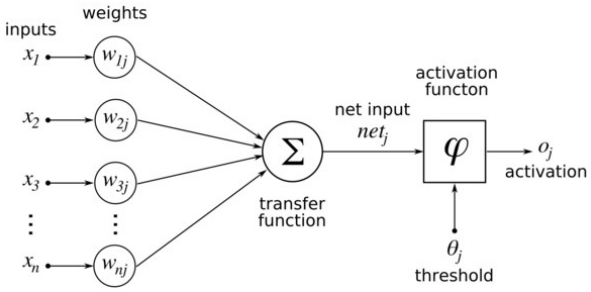


Figure 1.1: The model of AN

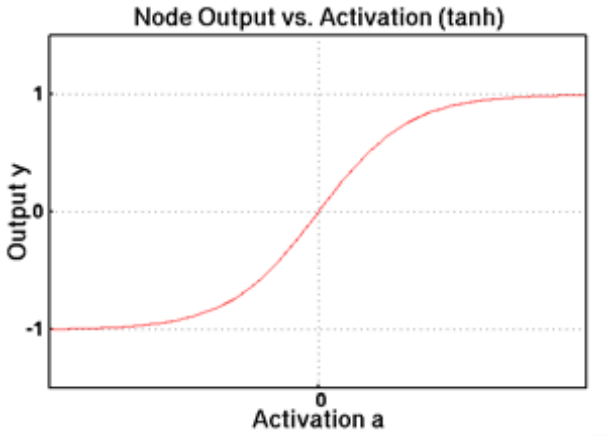


Figure 1.2: The $\phi$ function

## 1.1 THE SEPARATION SURFACE

The following equation is the equation of a hyperplane:

$$\overline{w}^T \overline{x} - b = 0$$

where $\overline{w}$ is the vector of the weights, $\overline{x}$ is the vector of the inputs and $b$ is the bias.

The artificial neurons can implement basic logical functions, depending on the choice of $\overline{w}$. Namely:

- N-dimension logical **AND**: $y = sgn(\sum_{i=1}^{N} x_i - (N - 0.5))$
- N-dimension logical **OR**: $y = sgn(\sum_{i=1}^{N} x_i + (N - 0.5))$

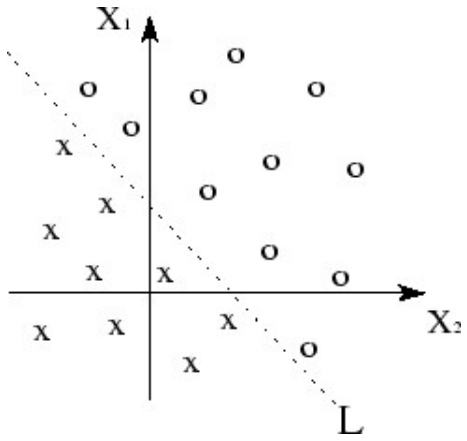


Figure 1.3: 2D example of linearly separable groups

More complex logical functions, such as **XOR** can be implemented by a network of multiple ANs, called *neural networks*.

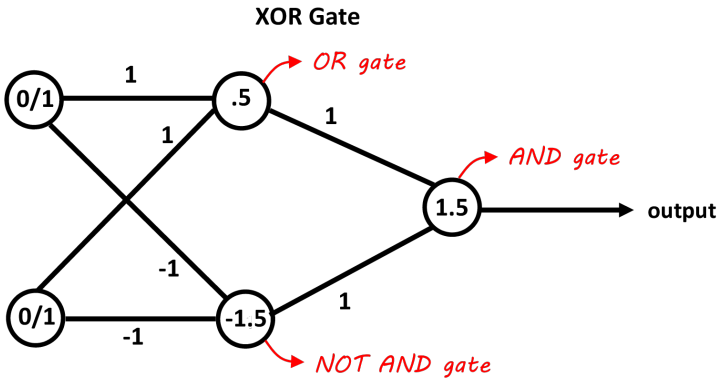


Figure 1.4: A possible implementation of the XOR function.

## 1.2 PATTERN RECOGNITION BY A SINGLE NEURON

### Mathematical model

$\overline{\xi} \in \{\overline{u}, \overline{v}\} \longrightarrow$ add noise $(v) \longrightarrow \overline{\zeta} = \overline{\xi} + \overline{v}$ where $\zeta$ is the observation of $\overline{x}$

**Maximum likelihood decision**

- if $P(\overline{x}|\overline{u} > \overline{x}|\overline{v})$ then we decide $\overline{u}$
- if $P(\overline{x}|\overline{u} < \overline{x}|\overline{v})$ then we decide $\overline{v}$

## 1.3 PATTERN CLASSIFICATION UNDER GAUSSIAN NOISE

In this case $\overline{v}$ is of Gaussian distribution.

$$P_{\overline{v}}(\overline{x}) = \frac{1}{\sqrt[2]{(2\pi)^N det(\frac{1}{K})}} e^{-\frac{1}{2}\overline{x}^T K^{-1} x}$$

$$\overline{w} := \overline{\overline{K}}^{-1}(\overline{u} - \overline{v})$$

$$b := \frac{1}{2}\overline{v}^T \overline{\overline{K}}^{-1} \overline{v} - \frac{1}{2}\overline{u}^T \overline{\overline{K}}^{-1} \overline{u}$$

$$y := sgn(\overline{w}^T \overline{x} - b)$$

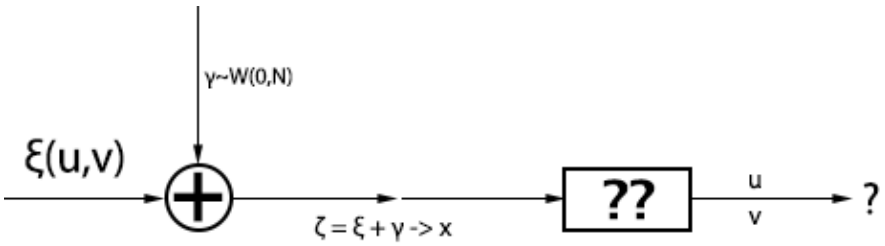


Figure 1.5: classification under Gaussian noise

The figure above shows the process of pattern classification under Gaussian noise, where $\xi(u, v)$ is the original messages which we never know, $\gamma$ is the Gaussian noise and $\zeta$ is the noisy message which we receive.

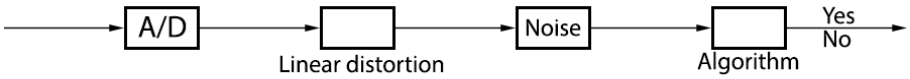


Figure 1.6: The process of basic Yes/No recognition

**Speech recognition**

### 1.4 THE COVER SENTENCE

The Cover sentence defines the minimal number of neurons required to implement a certain logical function depending on the number points with a general position and the dimension of the function.

$$L(P,N) = 2\sum_{i=0}^{N}\binom{i}{P-1}$$

, where $L$ is the minimal number of ANs, $P$ is the points with a general position and $N$ is the dimension.

## 2 2-3.ELŐADÁS

### 2.1 PERCEPTRON LEARNING ALGORITHM

The PLE, alias Rosenblatt algorithm[1] uses recursion to implement the learning process of ANs. In Rosenblatt algorithm training patterns are presented to the network's inputs; the output is computed. Then the connection weights $w_j$ are modified by an amount that is proportional to the product of

- the difference between the actual output, y, and the desired output, d, and
- the input pattern, x.

[1] Learn more about the Rosenblatt algorithm here:
*https://catalogue.pearsoned.ca/assets/hip/us/hip_us_pearsonhighered/samplechapter/*0131471392.*pdf*

The algorithm is as follows:

Initialize the weights and threshold to small random numbers. Present a vector x to the neuron inputs and calculate the output. Update the weights according to:

$$\overline{w}(k+1) = \overline{w}(k) + \zeta l(k)\overline{x}(k)$$

where

1. $l(k) = d(k) - y(k)$
2. $y(k) = sgn(\overline{w}^T(k)(\overline{k}))$ is the error and
3. $0 <= \zeta <= 1$ is the learning parameter

**Example:**

Table 1.1: Example of recursive learning

| **k** | **x(k)** | **d(k)** | **y(k)** | **l(k)** | **w(k-1)** | **w(k)** |
|---|---|---|---|---|---|---|
| 1 | [-1 1 1] | 1 | 1 | 0 | [-2 -1 1] | [-2 -1 0] |
| 2 | [-1 -1 0] | -1 | 1 | -2 | [-2 -1 1] | [0 1 1] |
| 3 | [-1 0 -1] | 1 | -1 | 2 | [0 1 1] | [-2 1 -1] |
| 4 | [-1 1 1] | 1 | 1 | 0 | [-2 1 -1] | [-2 1 -1] |
| 5 | [-1 -1 0] | -1 | 1 | -2 | [-2 1 -1] | [0 3 -1] |

# Neural Networks

## 1 4.ELŐADÁS

### 1.1 ASSOCIATIVE MAPPING (AM)

Associative mapping[1] can be used to recognize bitmap images or other patterns with (usually Hopfield) neural networks. In AM there are a number of stored images with different bit-level representation. The AM algorithm chooses the most similar stored memory item for any given input, containing so-called *clues* (parts of the input pattern similar to some parts of a stored memory item).

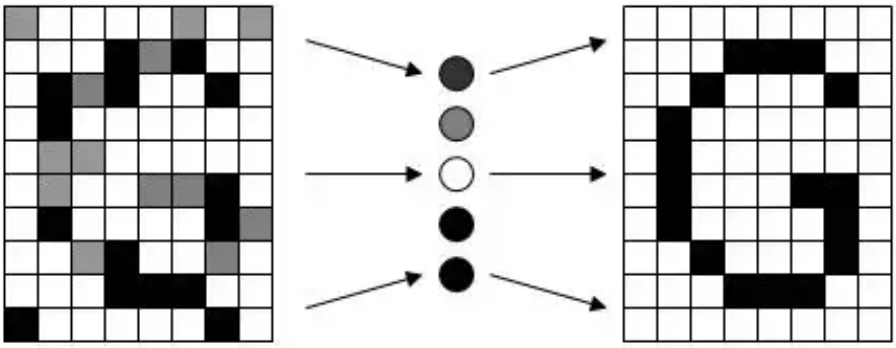

Figure 2.1: Exaample of non-binary associative mapping

[1] Learn more about AM here: *https://page.mi.fu-berlin.de/rojas/neural/chapter/K12.pdf*

**Mathematical representation**

$$\Psi : X = \{0,1\}^N \rightarrow S = \{s^{-\alpha}, \alpha = 1...M\}$$
$$\Psi(\overline{x}) = \overline{s}^{\beta}$$
$$d(\overline{x}, s^{\beta}) < d(\overline{x}, \overline{s}^{\alpha}), \forall \alpha = 1...M, \alpha \neq \beta$$

where

- $\Psi$ is the *observation space*
- $\overline{x}$ is the clue in the observation space
- $S$ is the set of stored memory items, $\overline{s}^{\alpha}$ are the stored memory items and $\overline{s}^{\beta}$ are the clues in the space of stored memory items.
- $M$ is the capacity of the network (number of stored items)
- The stability is the measure of misassociations. The stability decreases as one increases the capacity.

**Static and dynamic paradigm**

The **static paradigm** of Associative Mapping is based on a look-up-table (LUT) of the memory items and the binary vectors associated with them. The deterministic capacity of the static approach is $M \leq N$ where $N$ is $rank(\overline{\overline{W}})$ and $M$ is the number of stored items.

Table 2.1: LUT approach for AM

| x | s |
|---|---|
| $x_1$ | |
| $x_2$ | $s^{\alpha}$ |
| $x_3$ | |
| $x_4$ | |
| $x_5$ | |
| $x_6$ | |
| $x_7$ | $s^{\beta}$ |
| $x_8$ | |
| $x_9$ | |

The **dynamic paradigm** can be based on any recursive algorithm which has fix points and a polinomial convergence time. In this case the algorithm associates by settling in a stable point of the algorithm. Mathematically it can be represented as follows:

$$S := \{\overline{s}^{\alpha}, \alpha = 1..M\}, \exists \phi() : \overline{s}^{\alpha} = \phi(\overline{s}^{\alpha}), \alpha = 1...M$$

The dynamic paradigm can be implemented with a Hopfield Neural Network.

## 1.2 HOPFIELD NEURAL NETWORKS

Hopfield networks are constructed from artificial neurons (see Fig. 1). These artificial neurons have N inputs. With each input $x$ there is a weight $w_i$ associated. They also have an output. The state of the output is maintained, until the neuron is updated[2].

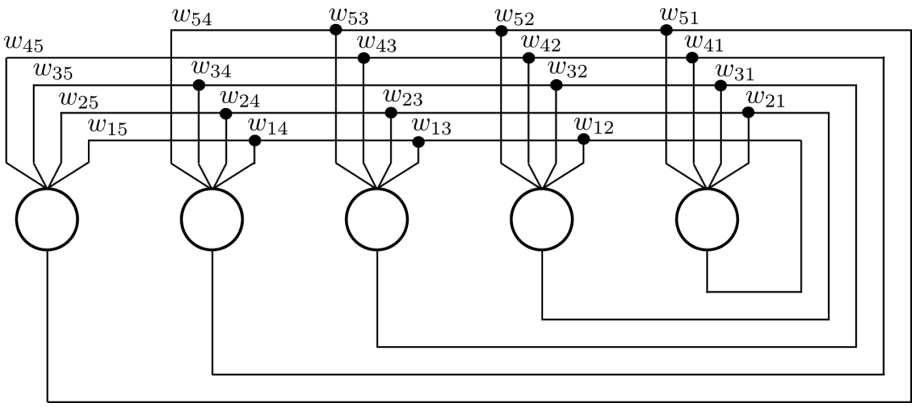


Figure 2.2: The structure of a Hopfield Neural Network

HNNs are always built upon a Lyapunov function. This function decreases in a monotone fashion and is bounded below. The process of computation can be thought as iterating through the function, and finally settling in a local minimum (stable point). A set of starting points, from where the algorithm settles in a specific stable point is called a **basin of attraction**, where the stable points are the **attractors**:

The main applications of AM are, fore example, Image Processing and Character recognition.

[2]http://www.comp.leeds.ac.uk/ai23/reading/Hopfield.pdf

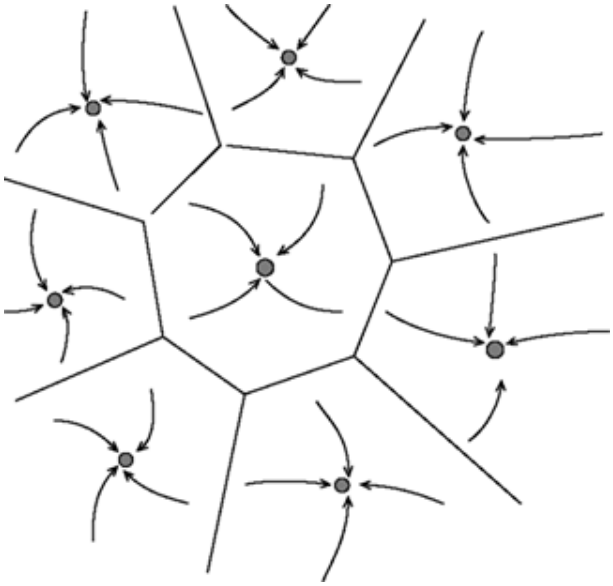

Figure 2.3: Basins of attraction

**State-transition equation**

$$y_l(k+1) = sgn\{\sum_{j=1}^{N} W_{lj} y_j(k) - b_e\}, l = mod_n k$$

where $y(0)$ is the starting state, and

$$y(k+1) = sgn\{\overline{\overline{W}}\overline{y}(k) - \overline{b}\}, \overline{y}(k) \in \{-1,1\}^N$$

**Indexing rule**

- **sync/paralell**: in this case every neuron changes its state at once.
- **async/sequential**: in this case the neurons changes their state one after another. The next neuron-to-update is chosen by a random generator.

**Dynamic deterministic capacity**

The capacity of a HNN can be computed as follows:

$$M < N\frac{N^2}{(N-2)^2}$$

This, unfortunately converges to 1, so we have to find an other approach to the solution.

## 2 5.ELŐADÁS

### 2.1 HEBBIEN LEARNING RULE

From the point of view of artificial neurons and artificial neural networks, Hebb′s principle can be described as a method of determining how to alter the weights between model neurons. The weight between two neurons increases if the two neurons activate simultaneously, and reduces if they activate separately. Nodes that tend to be either both positive or both negative at the same time have strong positive weights, while those that tend to be opposite have strong negative weights[3].

In reality it works as follows: given a network of N nodes and faced with a pattern that we want to store in the network, we chose the values of the weight matrix as follows:

$$\overline{\overline{W}}_{ij} = \frac{1}{N}\sum_{\alpha=1}^{N} s_i^\alpha s_j^\alpha = \frac{1}{N}\sum_{\alpha=1}^{N} \overline{s}^\alpha \overline{s}^{\alpha T}; \quad \forall i,j = 1...N$$

**Statistical approach to capacity**

One approach to increase the capacity of a HNN is to change from orthogonal $s^\alpha$ to quasi-orthogonal ones. In this case all $s^\alpha$ are independent Bernoulli random variables, where $\forall \alpha = 1...M, \quad \forall i = 1...N; \quad P(s_i^\alpha = 1) = P(s_i^\alpha = -1) = 0.5$. Thus, the IT capacity of the static (LUT) approach increases to

$$M \leq \frac{N}{2lgN}$$

and the capacity of the dynamic (HNN) approach increases to

$$M \leq \frac{N}{2\pi}$$

There are some drawbacks in this case. Firstly,

- if $\overline{s}^\alpha$ is stable, then -$\overline{s}^\alpha$ will be stable too, and

[3] $https://en.wikipedia.org/wiki/Hebbian_theory$

| | **static** | **dynamic** |
|---|---|---|
| **orthogonal** | $M \leq N$ | $M \leq \frac{N^2}{(N-2)^2}$ |
| **quasi-orthogonal** | $M \leq \frac{N}{2lgN}$ | $M \leq \frac{N}{2\pi}$ |

Table 2.2: The capacity of Associative Memories

- $\text{sgn}\{\overline{s}^{\alpha}+\overline{s}^{\beta}+\overline{s}^{\gamma}\}$ is stable too.

These cases are called **spurious steady states** and can be dealt with multiple possible solutions. Secondly, we change our samples from orthogonal to quasi-orthogonal ones, it will not be guaranteed if all of our samples will be stable points of the Lyapunov function.

## 3 6.ELŐADÁS

### 3.1 STATISTICAL NEUODYNAMICS

**Micro description of the system**

**Micro state vector:** $\overline{y}(k) \in \{-1,1\}^N$ **Micro dynamics:** $\overline{y}(k+1) = sgn\{\overline{\overline{W}}\overline{y}(k)\} \rightarrow a(k+1) = \phi(a(k))$

**Macro description**

$a(k) = A\overline{v}(\overline{y(k)})$

### 3.2 DIGITAL COMMUNICATION

The basic model of Digital Communication Systems is the model of NNs used to correct messages under Gaussian noise (1.5).

This classification is done by choosing the most possible meaning for each incoming noisy message vector. This can be done as follows:

$$\max_{\overline{y}\in\{-1,1\}^N} P(\overline{y}|\overline{x}) \sim \ldots \sim \min_{\overline{y}\in\{-1,1\}^N} \overline{y}^T\overline{\overline{W}}\overline{y} - 2\overline{b}^T\overline{y}$$

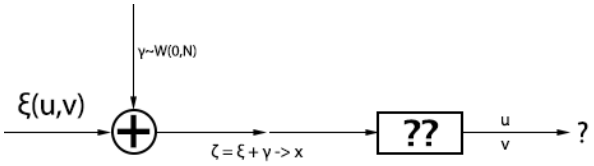


Figure 2.4: Classification under Gaussian noise

This can be done under $O(N^{"}2)$ instead of the $O(2^N)$ time of the traditional approach.

## 3.3 Minimizing a Hopfield network

Computing $locmin_{\overline{y}}\mathcal{L}(\overline{y})$

$$y_l(k+1) = -sgn\sum_{j=1}^{N}\overline{\overline{W}}_{lj}\overline{y}_j(k) - \overline{b}_l$$

With minimizing, the HNN becomes capable of solving quadratically representable problems effectively.

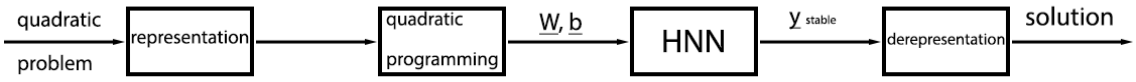


Figure 2.5: The process of solving quadratic problems

## 3.4 Heteroassociative memories

Heteroassociative memories are a type os associative memories, where the HNN works on a transformed space instead of the original input space.

In this case, $dim(\overline{r}^{\alpha}) = N, \quad \alpha = 1...M$ and $dim(\overline{s}^{\alpha}) = N', \quad \alpha = 1...M$.

The $\tau$ transformation must be:

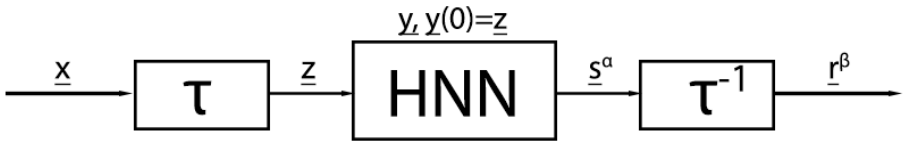


Figure 2.6: Sturcture of heteroassociative memories

- orthogonal
- easily computable
- distance-preserving

and mathematically can be described as follows:

$$\tau : \overline{z} = sgn\{\sum_{\alpha=1}^{M} \phi(\overline{r}^{\alpha T}\overline{x})\overline{s}^{\alpha}\}$$

$$\tau^{-1} : \overline{z} = sgn\{\sum_{\alpha=1}^{M} \phi(\overline{s}^{\alpha T}\overline{x})\overline{r}^{\alpha}\}$$

where $\phi$ can be for example: $\phi(u) = \gamma \times u$ or $\phi(u) = e^{\gamma \times u}$

## 4 7.ELŐADÁS

### 4.1 FEED FORWARD NEURAL NETWORKS

FFNNs consists of multiple neurons sorted to multiple layers. Any input signal is propagated from the input nodes through a set of so-called hidden nodes, and finally to the output nodes. In FFNNs there are no loops or cycles.

**Mathematical representation**

$$\overline{y} = \phi(\sum_{i} w_i^{(n)} \phi(\sum_{j} w_{ij}^{i=1} ... \phi(\sum w_{num}^{(l)} x_m)...))$$

where$\sum w_{num}^{(l)} x_m = \overline{z}$

$$\overline{y} = \mathcal{N}et(\overline{x}, \overline{w})$$

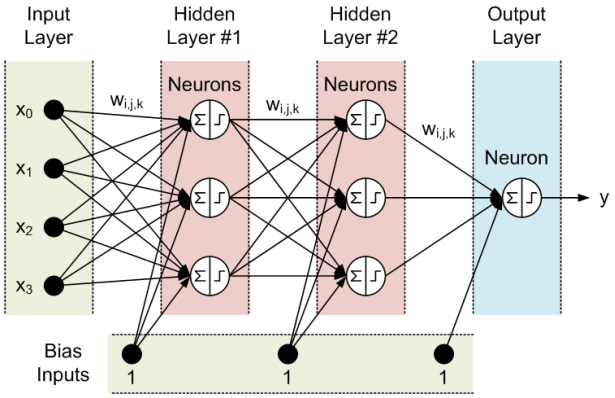


Figure 2.7: Structure of a simple FFNN

1. **Representation capability:** Can any functions be represented by FFNNs?

2. **Learning:** If a function is to be represented, then how to find the weights?

3. **Generalization capability:** What output appears to a previously not known input?

**Representation theory of FFNN**

$$\left\| F(\overline{x} - \mathcal{N}et(\overline{x},\overline{w})) \right\| < \epsilon \Rightarrow F(\overline{x}) \in F; \quad \mathcal{N}et(\overline{x},\overline{w}) \in \mathcal{NN}; \mathcal{NN} \in \mathcal{F}$$

$$if \quad \epsilon > 0; \quad \forall F(\overline{x}) \in \mathcal{F} \qquad \exists \overline{w} : \left\| F(\overline{x} - \mathcal{N}et(\overline{x},\overline{w})) \right\| < \epsilon$$

# 5 8.ELŐADÁS

## 5.1 BLUM-LI CONSTRUCTION

Blum and Li showed, that a 3-layer FFNN can represent any square integrable function. The complexity of the implementation depends on the derivative of the function. The neurons of the network represent the section borders of the discrete resolution of the function.

## 5.2 Learning of the FFNN

### Strong learning

Given $F(x) = d$ desired output.

$$\overline{w}_{opt} : \min_{\overline{w}} \| d - \mathcal{N}et(\overline{x}, \overline{w}) \|^2 \sim \min_{\overline{w}} \frac{1}{|X|} \int_X .. \int (d - \mathcal{N}et(\overline{x}, \overline{w}))^2 dx_1 ... dx_n$$

### Weak learning

$$\tau^{(k)} = \{(\overline{x_k}, \overline{d}_k); k = 1...K\} \rightarrow \overline{w}^{(k)}_{opt} : \min_{\overline{w}} \frac{1}{K} \sum_{k=1}^{K} (d_k - \mathcal{N}et(\overline{x}, \overline{w}))^2 \sim \overline{w}_{opt} : \min_{\overline{w}} \mathbf{E}(d - \mathcal{N}et(\overline{x}, \overline{w}))^2$$

Weak learning is algorithmically trackable, and if counted as a *limes in mean*:

$$\lim_{l \rightarrow \infty} \overline{w}^{(k)}_{opt} = \overline{w}_{opt}$$

## 5.3 Generalization

### Bias-variance dilemma

$$\mathbf{E}(d - \mathcal{N}et(\overline{x}, \overline{w}^{(k)}_{opt}))^2 = \mathbf{E}(d - \mathcal{N}et(\overline{x}, \overline{w}_{opt})^2 + \mathcal{N}et(\overline{x}, \overline{w}_{opt}) - \mathcal{N}et(\overline{x}_k, \overline{w}_{opt})^{(k)})^2$$

thus

$$\mathbf{E}(d - \mathcal{N}et(\overline{x}, \overline{w}_{opt}))^2 + \mathbf{E}(\mathcal{N}et(\overline{x}, \overline{w}_{opt}) - \mathcal{N}et(\overline{x}, \overline{w}^{(k)}_{opt})$$

where the first tag is the **bias** and the second is the **variance**. The point of bias-variance dilemma is that one must chose which one they want to minimize.

## 5.4 Traveling Salesman Problem

The solution to the problem can be computed by calculate a minimal-cost Hamilton path. The path can be represented with a permutation matrix as follows:

$$V_{ij} = \begin{cases} 1, & \text{if we are in the } i\text{th position in the } j\text{th step} \\ 0, & \text{else} \end{cases}$$

For example:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \min_{\overline{V}} \sum_i \sum_j \sum_l V_{il} D_{ij} V_{j(l+1)}$$

# 6 9.ELŐADÁS

## 6.1 ERROR BACK PROPAGATION

### Initialization - Learning Set Representation

$W(0), W_{ij}(0)$ with RNG Batch (offline) or sequential (online).

### Signal forward propagation

$$I_i^{(l)} = \sum_{j=0}^{m_{l-1}} w_{ij}^{(l)} ... y_j^{(l-1)} \quad l = 1...L$$

$$y_i^{(l)} = \phi(I_i)^{(l)} \quad y_j^{(0)} = x_j \quad y_0^{(l)} = -1 \text{ (bias)}$$

### Error back propagation

Local errors

$$l_i^{(l)} = \begin{cases} \phi'(I_i^{(l)})(d_i - y_i^{(L)}) \\ \phi'(I_i^{(l)}) \sum_{j=1}^{m_{l+1}} w_{ij}^{(l+1)} e^{(l+1)} \quad l \neq L \\ \rightarrow \text{the errors in the previous layer, computed from the ones in the next,} \end{cases}$$

### Adjusting the weights

$$w_{ij}^{(l)}(k+1) = w_{ij}^{(l)}(k) + \Delta l_i^{(l)} \times y_j^{(l-1)}$$

**Exit criteria**

- For example: if $R_{emp}(w(k+1)) < C_1$ the algorithm stops
- Or: $R_{val} < C_2$ where $R_{val}$ is a validation function

## 7 10.ELŐADÁS

### 7.1 DESIGN PROCEDURE

Given $\alpha, \epsilon$(QoS). $P(\left|R(\overline{w}_{opt}) - R(\overline{w}_{opt}^{(k)})\right| > \epsilon) \overset{?}{<} \phi(\epsilon, K, W)$ where $\overline{w}_{opt}$ is unattainable and $\overline{w}_{opt}^{(k)}$ is computed.

**Vapnik-Chrmonenkis dimension**

VC theory explains the learning process from a statistical point of view. It us often used for the characterization of classificators. VC: maxM. In a neural network, with M inputs the number of implementable functions is $2^M$.

$$2^M \begin{cases} \overline{x}_1; & \overline{x}_1; & \dots & \overline{x}_M \\ - & - & \dots & - \\ - & - & \dots & + \\ \vdots & \vdots & \ddots & \vdots \\ + & + & \dots & + \end{cases}$$

**Design of FFNNs**

Given $\epsilon, \alpha$ (QoS), K (sze of the learning set), W (number of free parameters)

1. $\alpha = (\frac{2eK}{Vc})e^{-\epsilon^2 K} \rightarrow \overline{w}, K$
2. $\tau^{(k)} = \{(\overline{x}_k, d_k), k = 1...K\}$
3. $\mathcal{N}et(\overline{x}, \overline{w})$ according $\overline{w}$
4. $\overline{w}_{opt}^{(k)} : \min_{\overline{w}} \frac{1}{K} \sum_{k=1}^{K} (d_k - \mathcal{N}et(\overline{x}_k), \overline{w})$

5. $\mathcal{N}et(\overline{x}_k, \overline{x}_{opt}^{(k)}) :\longrightarrow P(\left|R(\overline{w}_{opt}) - R(\overline{w}_{opt}^{(k)})\right| > 2\epsilon < \alpha)$

# Cellular Neural Networks

## 1 11.ELŐADÁS

### 1.1 CELLULAR NEURAL NETWORKS

CNNs are analog neural networks which utilize connections exclusively between neighbouring (local) cells. The network is mostly built up as a 2D array (gird) of these cells. This type of neural networks is mainly used in the field of image processing (for example edge detection), 3D surface analysis or modeling biological functions.

where $g(x_{ij})$ is the DP plot (driving point plot) and $w_{ij}$ is the bias.

CNNs can be characterized via so-called templates. These templates include the two (feed forward and feeback) operators which are present in the characterizing differential equation of the network. This equation is as follows:

$$x'_{ij} = -x_{ij} + \sum_{kl \in S_r(ij)} A_{ij,kl} \times y_{kl} + \sum_{kl \in S_r(ij)} B_{y,kl} \times u_{kl} + z_{ij}$$

Where *i, j* are the indexes of the current cell, *k, l* are the indexes of the neighouring cells in order, $\overline{\overline{A}} and \overline{\overline{B}}$ are 3x3 matrices, $u_{ij}$ are the input, $x_{ij}$ are the state and $y_{ij}$ are the output values of the current cell and $z$ is a threshold. This can be transformed to:

$$x'_{ij} = \underbrace{-x_{ij} + a_{00} y_{ij}}_{g(x_{ij})} + \underbrace{\overline{\overline{A}} \times \overline{\overline{Y}}_{ij} + \overline{\overline{B}} \times V_{ij} + z}_{w_{ij}}$$

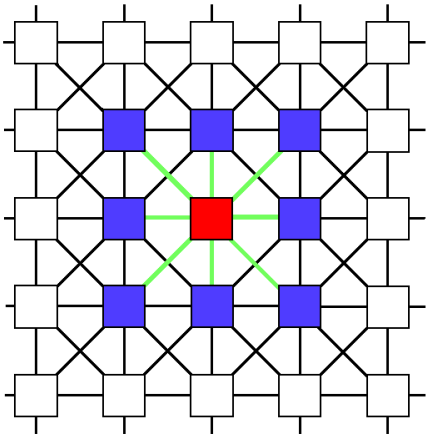

Figure 3.1: Local connections in a CNN

From this computed inner state we are able to calculate the output $y$ by the activation function. The activation function of the cells of the CNN is usually as shown below:

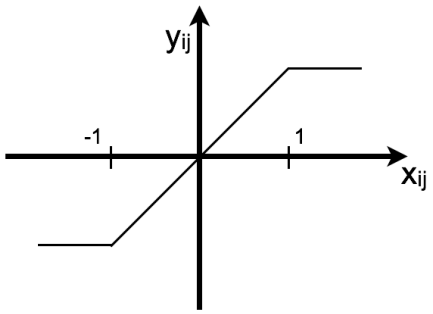


Figure 3.2: The activation function of the cells

In time-and space-invariant CNNs the template used to calculate $y$ is the same regardless, of the location (indexes) of the actual cell. In this case we always can use the same template, defined like the one below:

$$\overline{\overline{A}} = \begin{bmatrix} a_{-1-1} & a_{-10} & a_{-11} \\ a_{0-1} & a_{00} & a_{01} \\ a_{1-1} & a_{10} & a_{11} \end{bmatrix} \qquad \overline{\overline{B}} = \cdots$$

$$\overline{\overline{Y}} = \begin{bmatrix} y_{i-1j-1} & y_{i-1j} & y_{i-1j+1} \\ y_{ij-1} & y_{ij} & y_{ij+1} \\ y_{i+1j-1} & y_{i+1j} & y_{i+1j+1} \end{bmatrix} \qquad \overline{\overline{V}}_{ij} = \cdots$$