Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Szakdolgozat

Megerősítéses tanulás alkalmazása az emberi motoros pályákat alapul véve, mozgásminták tanításához szimulációkban

Szabó Gergely
Molekuláris bionia mérnök BSc

2018

Témavezető:
dr. Horváth András

Alulírott Szabó Gergely, a Pázmány Péter Katolikus Egyetem Információs Technológiai és Bionikai Karának hallgatója kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, és a szakdolgozatban csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen a forrás megadásával megjelöltem. Ezt a szakdolgozatot más szakon még nem nyújtottam be.

_____

Aláírás

Tartalomjegyzék

Kivonat

Szakdolgozatom célja, hogy egy általam megalkotott, newtoni fizikán alapuló környezetben az emberi mozgásmintákhoz hasonló mozgások tanítására alkalmas modellt valósítsak meg. Ezen tanítást neurális háló segítségével hajtottam végre, megerősítéses tanulást alkalmazva.

A modell szokványostól eltérő tulajdonsága, hogy az emberi izomzatról mintázott módon tudja csak saját belső paramétereit változtatni, így a problémákat csak dinamikusan képes megoldani. Másik, ilyen módon mozgó, tanulni képes modellre utalást a szakirodalomban nem találtam, így lehetséges, hogy az általam megalkotott modell bizonyos szinten újítást jelent. Modellem másik alapvető tulajdonsága, hogy tanításához nem használtam fel semmilyen módon valós mozgás modellt, azaz a háló teljesen önmagától tanul, próbálkozás útján, így a valóstól eltérő, szimulált környezetben is vizsgálhatóak általa mozgásminták. További jó tulajdonsága ezen modellnek, hogy tanítása rendkívül gyors, így fejlesztése vagy vizsgálata során nem kell hosszú időt várni egy-egy futtatás között.

Ezen modell azonban teljesítményében még sok szempontból alulmarad más, mozgást tanuló modellekhez képest. A modell jelen állapotában még nem képes a hosszú ideig állva maradás feladatát elvégezni, azonban az emberi mozgásmintákra nagyban emlékeztető mozgások végrehajtására már képes. Későbbiekben célom a modell továbbfejlesztése, hogy képessé váljon hosszú ideig állva maradni, valamint járás tanítását is tervezem.

A háló teljesítményének számszerű bemutatására méréseket végeztem, melyeket a szakdolgozatban részletesen is leírtam.

Abstract

The goal of my thesis is to create a model, which can learn to move similarly like humans, in an environment based on Newtonian mechanics created by me. The learning of the model is performed by a neural network that uses reinforcement learning.

The property of my model, differing from ordinary models, is that it can only modify the inner parameters of itself like human muscles do, so it can solve problems only dynamically. I could not find any similar model in the literature, so it is possible, that my model is an innovation in a certain extent. An other primary property of my model is that it does not use any real movement to help the learning of the model, therefore the model learns probabilistically, so movements can be examined through the model in an environment that differs form real environments. An other great property of the model is that the learning of itself is really fast, therefore during the development or examination of the model it does not take a long time to wait between running tallies.

However the performance of this model is worse in many properties than the performance of other movement learning models. In the current state of my model it can not stand for a long time, but it can realize some movements, which are highly similar to human movements. In the future I am planning towards the improvement of my model, so that it would be able to stand for a long time or even walk.

To present the numerical performance of my model I did some measurements, that were included in details in the thesis.

1 Bevezetés

A feladatom ezen munka során az volt, hogy megerősítéses tanulás alkalmazása segítségével megvalósítsak egy modellt, mely az emberi mozgáshoz hasonló mozgásminták tanulására képes, az emberi motoros pályákat bizonyos szinten modellező neurális háló segítségével. A feladatot pontosabban, részfeladatokra bontva a 3. fejezetben írom le. Ezen mozgásminták közül az egyik legalapvetőbb és legfontosabb az állás képessége, eddigi munkámat ebben a témában végeztem, így ezen dolgozat is az állás tanításának megvalósítására koncentrál.

A modellem megvalósításának a célja, hogy hosszú távon képes legyen az emberi mozgás minél pontosabb szimulálására, az egyszerűsített fizikai környezet ellenére. Ennek elérése érdekében nem csak az emberi mozgás tanulásának folyamatát, hanem a mozgást végrehajtó szervek működését is igyekeztem megismerni és modellezni. Ezen célkitűzés legfontosabb lépése az emberi izomzat modellezése volt, melyről a 2.2. és a 4.3.2. fejezetben írok bővebben. Ezen izommodell használata lehetővé teszi, hogy modellem a jelenleg népszerű mozgás szimulációktól eltérő módon tanuljon. Jó példa erre a mozgás szinulációkban gyakran használt Mujoco modell [1], mely a következő oldalon volt elérhető 2018.11.25-én: `https://www.roboti.us/index.html`. Ezen modell – az összes általam megismert, más által készített mozgás-modellhez hasonlóan – az ízületi pontok forgatásával valósítja meg a mozgást (azaz az ízületi pontok viselkedése aktív folyamat). Ezen modellektől eltérően az általam megvalósított modell (melynek részleteit a 4. fejezetben írom le) dinamikus, izomzat alapú mozgást valósít meg, a modellemben az ízületi pontok elfordulása csupán az izomzat működésének és a környezet hatásainak következménye (azaz passzív folyamat). A robotikában használt eszközök jellemzően ugyancsak ízületek aktív forgatását valósítják meg, így ilyen eszközök esetén a Mujoco-hoz hasonló modellek használata teljesen indokolt, azonban az ilyen eszközök gyakran a statikus jellegű feladatok megvalósításában jók, míg a dinamikus feladatok megvalósításában sokszor gyengén teljesítenek, ahogy ezt a következő cikk is mutatja: [2]. Így nem kizárt, hogy a robotika fejlődésével megjelennek az emberi izomzat működését imitáló eszközök is (melyek potenciálisan a jelenleginél sokkal alkalmasabbá válhatnak dinamikus problémák megoldására), és ilyen eszközöket alkalmazó gépek tanításához az általam megalkotott modellhez hasonló modellek használata válhat indokolttá. Mind e mellett az ilyen jellegű modellek vizsgálata az emberi mozgás jobb megértését is elősegíthetik.

Másik sajátos tulajdonsága a modellemnek, melyet ezen dolgozatban bemutatok, hogy mozgás tanulása nem alapszik valós mozgásminták vizsgálatán. Ezen tulajdonsága gyakran jelentős hátrányt jelent a modell tanításának sikeressége szempontjából. Összevetés képpen, a Berkley egyetem egy csoportja 2018 áprilisában publikálta a következő oldalon elérhető eredményeit: [3]. Ezen cikkben bemutatott modellek rendkívül látványos ered-

ményeket mutatnak, azonban fontos kiemelni, hogy tanításuk minden esetben emberi beavatkozás segítségével történik, vagy valósan (kaszkadőr által) végrehajtott mozgásmintákat igyekszik a modell leutánozni, vagy egy művész által megrajzolt mozgásmintával igyekszik megtenni ugyanezt (például a cikkben bemutatott T-rex vagy sárkány esetén). Ennyire látványos eredményeket egyáltalán nem várok a modellemtől elsősorban azért, mert a modellem (a fentebb bemutatott példától eltérően) teljességgel magától tanul. Azonban ezen tulajdonsága nagy előnyt is jelenthet, mivel potenciálisan bármilyen felépítésű bábut bármilyen fizikai környezetben képes lehet megtanítani bizonyos mozgásokra (például gravitáció vagy súrlódás mentes térben is képes lehet tanulni). Az általam megvalósított modellhez hasonló elven mozgó és tanulni is képes modellt nem találtam a szakirodalomban, így lehetséges, hogy ezen modell bizonyos szinten újítást jelent.

Dolgozatomat három nagy egységre tagoltam. A 2. fejezetben a modell megvalósításához szükséges elméleti hátteret írtam le. A 4. fejezetben a feladat megvalósítását részleteztem. Az 5. fejezetben pedig a modellemen végrehajtott méréseket és a jövőbeli továbbfejlesztési lehetőségeket igyekeztem bemutatni. Ezen nagy egységeken kívül pedig a 3. fejezetben pontosítottam a feladatkiírást, valamint és a 6. fejezetben írtam egy rövid összefoglalást a dolgozatról.

A szakdolgozathoz tartozó kódokat CD-n mellékeltem.

2 Elméleti háttér

2.1. Emberi motoros szabályozás [4]

A motoros pályák az ember idegrendszerében is a mozgás szabályozásának feladatát látják el. Ilyen módon bizonyos szintű vizsgálatuk elengedhetetlen egy humanoid modell mozgásának tanításához.

Sokféleképpen csoportosíthatóak a motoros pályák. Egyik leginkább kézenfekvő megközelítés a mozgás tudatosságának szintje, amelyért felelnek. Ilyen módom megkülönböztethetjük az automatikus reflexeket (például patella reflex[5]), valamint a tudatosan szabályozott mozgásmintákat (például járás). Azonban ezen két kategória nem válik el élesen egymástól. Például a patella reflex, noha automatikusan hajtódik végre, később tudatosul az agyban, hogy megtörtént az esemény. Valamint például az izomtónus is - amely akár tudatosan is szabályozható bizonyos mértékig - hatással lehet a reflex végrehajtódásának módjára. Ugyanez igaz azonban fordítva is. Tekintsük példának most a járást. A járás komplex mozgás, sok különböző idegi folyamat együttes eredménye. Ezen idegi folyamatok között pedig mindenképpen kell lenniük tudatosan szabályozottaknak (hiszen a járás maga is tudatosan szabályozott). Azonban automatikus folyamatok is résztvesznek a sikeres járás megvalósításában. Többek között ilyen az előző példaként említett patella reflex is, amely biztosítja a szükséges izomtónust.

Mindkét fentebb tekintett folyamatban az információ áramlása (legalábbis a folyamat bizonyos részeiben) körkörös. A patella reflex esetében[5] a jel az izomorsótól egy gerincvelői interneuronon keresztül egy gerincvelői motorneuronba jut, amely felelős egy vagy több izom összehúzódásáért. A folyamat körkörösségét az emberi test fizikai egysége biztosítja. Azaz a cél izom összehúzódása idővel kihat annak az izomnak az állapotára is, amelyen az izomorsó található. Ezen időben folytonos körfolyamat segítségével képes a szervezet fenntartani az állva maradáshoz szükséges szöget a combcsont és a lábszárcsont között. A járás is tekinthető egyfajta körfolyamatnak, mivel a motor kortexbe[6] áttételesen érkeznek információk a test pozíciójáról (például a fej függőlegessel bezárt szögéről), majd ezen információk alapján hoz az ember döntést a következő mozdulatról, amelyet motoros pályák valósítanak meg. A kört itt is a mozdulat és annak hatása közötti kapcsolat zárja.

Az emberi motoros szabályozásban az előző két példán kívül még rendkívül sok más folyamat is részt vesz. Ezek közül az egyik legjelentősebb a cerebelláris szabályzás[7][8]. A cerebellum többek között a mozgás finomhangolásáért felelős. Ezt a beérkező jelek gyengítésével vagy erősítésével, majd a kimenet visszacsatolásával éri el. Így egy belső körhálózat is kialakul a tudatos motoros információk szabályozásában, amely lehetővé teszi hogy a mozgás folytonos szabályozása ne csak reaktív, hanem tanult módon proak-

tív legyen. Azaz a motoros rendszer saját múltbeli belső állapota is szabályozó tényező a jelenlegi állapot kialakulásában, nem csak a környezetet aktuálisan meghatározó tényezők.

A motoros rendszer ezen sajátosságai képezik az alapját azon mesterséges intelligenciát megvalósító neurális hálóknak, amelyekkel mozgás tanítását igyekszem elérni. Azaz ezen hálóknak nem csak a környezet jelenlegi állapota biztosítja a bemenetét, hanem a múltbeli állapotok is. Így az emberi cerebellumhoz bizonyos szintű hasonlósággal, ezen hálók is képesek lehetnek időbeli kiterjedésű mintázatok megvalósítására.

2.2. Elasztikus izommodellezés

Az idegrendszer modellezése mellett elengedhetetlen az izom- és csontrendszer modellezése is, az emberéhez hasonló mozgás megvalósítására képes modell elkészítéséhez. Míg a csontrendszer elvi működése egyszerű (a csont alapvető tulajdonsága, hogy közel teljesen szilárd, azaz egy adott csonton felvett pontok egymáshoz képest közel azonos távolságra és azonos szögben lesznek mozgás közben is), az izomzat, vagy akár csak egy izom modellezése is mind a mai napig kutatás tárgya. Jelen pillanatban nem ismert az izomzatot tökéletesen leíró modell, az izomzat modellezéséhez használt eljárásoknak különböző erősségeik és gyengeségeik vannak, vagyis a legjobb eljárást az adott probléma határozza meg.

Egyik leginkább kézenfekvő megoldás az izomzat működését rugó modellekkel szimulálni. [9] Egy rugómozgás során fellépő erőket Hooke törvénye[10] jó közelítéssel modellezi.

$$F = -kx \tag{2.1}$$

Ahol F a rugó által kifejtett erő nagysága (előjelesen, rugó irányú egységvektor szerint), k az adott rugóra jellemző tulajdonság (rugómerevségnek szokták nevezni, ennek reciproka pedig a gyakorlatban gyakrabban használt rugóállandó), x pedig a rugó előjeles elmozdulása az egyensúlyi pontjából. A fenti képletből látható, hogy a rugó végein lévő tömegpontokat gyorsító erő mindig olyan irányba hat, hogy a rugó elérje az egyensúlyi állapotát. És mivel a rugó végpontjainak pozíciója a rugó által kifejtett erő második deriváltjától függ, Newton második törvénye szerint [11], a rugóegyenlet rezgőmozgást ír le. Valamint ha a rugó modellezésénél csillapítást alkalmazunk (azaz az n+1-edik idő-pillanatban a tömegpontok sebessége valamilyen arányosság szerint csak egy bizonyos része az n-edik idő-pillanatbeli sebességeknek), akkor csillapított rezgőmozgást kapunk.

Hagyományos esetben a rugó stabil hossza és rugóállandója a rugóra jellemző konstans. Azonban egy ilyen modell nem lenne alkalmas emberi izmok modellezésére, mivel az izmok által kifejtett erő nem mindig az adott izom hosszától függ. Ennek a problémának a feloldására jó megoldás a változó rugóllandójú modellek alkalmazása [9].

Egy változó rugóállandókat és konstans stabil hosszakat alkalmazó modell már alkalmas az emberi izmok modellezésére bizonyos közelítéssel [9]. Ilyen esetben antagonista izmok alkalmazása szükséges, ami ugyan hasonlít az ember izom- és csontrendszerének

felépítésére, de a modellell komplexitását jelentősen növeli.

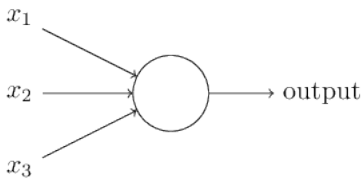
A modell komplexitásának csökkentésére jó megoldást jelent különböző izmok egy izomként történő ábrázolása. Ezzel az egyszerűsítéssel az antagonista izmok használatát is el lehet kerülni, amennyiben a modellben használt rugóknak nemcsak a rugóállandója, de a stabil hossza is változtatható. Ilyen módon egy rugó már képes összehúzást és távolítást is megvalósítani, bizonyos erővel. Vagyis egy változtatható rugóállandókat és stabil rugóhosszakat is megengedő rendszer képes az emberi izomzat leegyszerűsített modellezésére (bizonyos közelítéssel), úgy, hogy két végpont között csupán egy rugó elhelyezése szükséges.

2.3. Neurális hálók alapjai [12]

A neurális hálókat az emberi idegrendszer működésének modellezésére használják. Ezen hálók alkalmasak az emberi idegrendszer bizonyos elemeihez hasonló feladatok elvégzésére is. Például konvolúciós hálózatok alkalmazása rendkívül elterjedt mesterséges látás megvalósításához [13], és az ilyen hálózatok felépítése logikus módon az emberi retina felépítését veszi alapul.

2.3.1. Perceptron

A neurális hálók legalapvetőbb egysége a Frank Rosenblatt [14] által kifejlesztett perceptron, amely a bemenetek súlyozott összeadását, majd valamiféle nemlinearitást valósít meg. A perceptron felépítése a 2.1. ábrán látható.



2.1. ábra. A perceptron sematikus felépítése [12], melynek segítségével egyszerű döntési folyamatokat lehet megvalósítani. Perceptronokból felépített háló segítségével komplex problémák is megoldhatók.

Az egyik legegyszerűbb és leggyakrabban használt nemlinearitás a ReLu vagy Rectifier [15], amely a következő egyszerű összefüggést valósítja meg.

$$output = \begin{cases} 0 & if \sum_j w_j x_j \leq 0 \\ 1 & if \sum_j w_j x_j > 0 \end{cases} \tag{2.2}$$

Azaz, ha a bemenetek összege eléri a küszöbértéket (amely általában 0), akkor a neuron átereszti a bemenetek összegét, míg ha nem éri el, akkor mindenképpen nullát ad vissza. Nemlinearitás alkalmazása azért fontos, mivel a szumma függvények sorozata felírható egyetlen szumma fügvényként is, azaz nemlinearitás nélkül a rendszer egyetlen nagy összeadást valósítana meg, amelynek komplexitása a legtöbb probléma megoldására egyáltalán nem elég. A nemlinearitás alkalmazása azonban azt jelenti, hogy az

összeadások nem vonhatóak egybe, így a perceptronok számának növelésével a rendszer komplexitása egyre növelhető, és így egyre komplexebb problémák is megoldhatóak perceptronok segítségével. A perceptronok nemlinearitásaik segítségével képesek logikai függvények megvalósítására is, így a perceptronokból felépített háló alkalmas összetett logikai döntési sémák leírására.

2.3.2. Neurális hálók felépítése

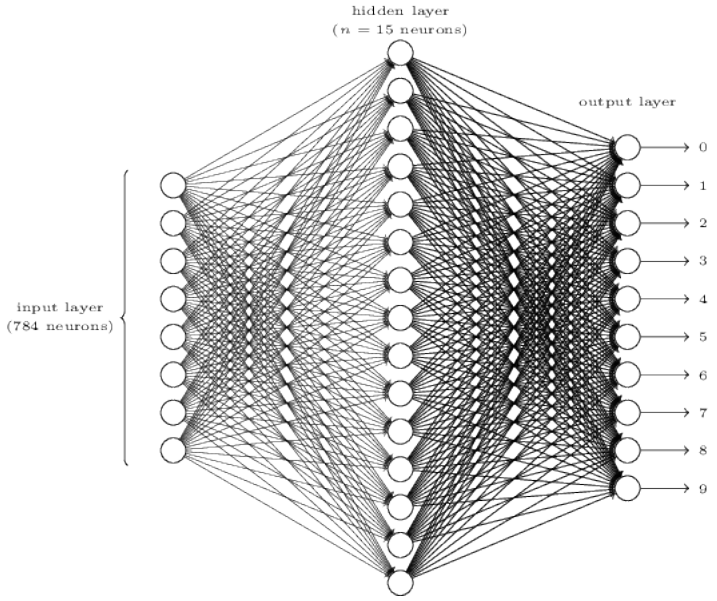
A perceptronokból felépített hálót neurális hálónak nevezzük. Ezen hálónak ki kell elégítenie a matematikai háló definícióját, azaz bármely kételemű részhalmazának létezik legkisebb felső korlátja és legnagyobb alsó korlátja. Vagyis körmentesnek kell lennie, és minden bemenetnek (a bemeneti rétegbe) kapcsolatban kell lennie a kimenetekkel (a kimeneti rétegben), nehogy információvesztés történjen. A 2.2. ábrán egy ilyen háló sematikus rajza látható, amelynek a bementi rétegében 784 neuron található, a kimeneti rétegében pedig 10, a kettő között a kapcsolatot itt egy rejtett réteg biztosítja, 15 neuronnal. Ezen háló úgynevezett fully connected háló, azaz minden neuron kapcsolatban áll az előtte levő réteg összes neuronjával. Ez a fajta felépítés a legalapvetőbb szerkezet a neurális hálók felépítésénél, és a leginkább logikus is, amennyiben nincs egyértelmű szándékunk az adatok áramlásának szabályozásában (például konvolúciós hálók esetén csak a szomszédos mezők információja adódik tovább, ezzel biztosítva az alacsonyabb rétegekben kevésbé komplex, majd egyre komplexebb objektumok felismerését). Fully connected háló használata továbbá nagy perceptron számú hálóknál sem célszerű, mivel itt a futási idő már komoly szempont, és egy fully connected háló általában ilyen téren veszít egy kevesebb kapcsolatot tartalmazó hálóval szemben. Ezen kívül megjelenhet a túltanulás jelensége is (különösen fully connected hálóknál), amelyre jó megoldást jelenthet a dropout [16].

2.3.3. A háló tanítása

Egy, a fentebb leírtak szerint megvalósított háló már képes komplex logikai függvények megvalósítására, azonban még nem képes önmagától tanulni. A háló legegyszerűbben állítható paraméterei a súlyok. Ezek segítségével szabályozható, hogy melyik bemenet melyik neuronra milyen szinten hat. Azaz ezen súlyok változtatásával megvalósítható egy tanítási folyamat.

A tanítás legegyszerűbb módja az evolúciós módszer [17]. Ezen módszernél a háló súlyait véletlenszerűen állítjuk, hasonlóan ahhoz, ahogy a biológiai evolúcióban a véletlen mutációk történnek, ezért is nevezik evolúciós módszernek. Majd ha az eredmény jobb lett, mint a legutóbbi hálóval, akkor megtartjuk a legújabb verziót, ha nem, akkor marad az előző állapot. Ez a módszer, bár elméletileg képes eljutni idővel a tökéletes megoldáshoz, általában nem jó választás. Ugyanis egy ilyen módszerrel tanuló háló tanítási ideje rendkívül hosszú lehet, könnyen annyira hosszúvá válhat, hogy ezen módszer teljességgel használhatatlanná válik.

Az egyik legnépszerűbb megoldás a háló tanítására az úgy nevezett gradient descent optimizer eljárás [12]. Ezen eljárás során az evolúciós eljárástól eltérően nem teljesen



2.2. ábra. Egy neurális háló felépítésének sematikus ábrája [12]. A háló, amelyről ez az ábra készült, képes bizonyos hibaszázalékkal meghatározni, hogy milyen értékű szám képe érkezett a bemenetére.

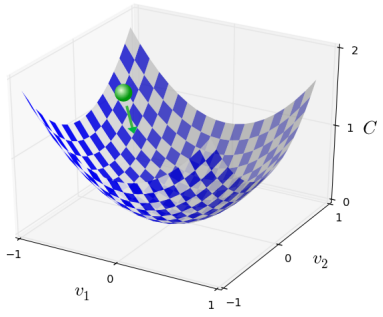
véletlenszerű, hogy mennyire változtatjuk meg a háló súlyait. Hanem e helyett az előző lépés sikeressége határozza ezt meg. Azaz, ha egy változtatás sikeres volt, akkor sok esetben egy hasonló változtatás is hasonló sikert jelent (mivel a problémákat leíró függvények általában viszonylag folytonosak, és nem túl gyakran változtatják deriváltjuk előjelét). Egy ilyen fajta tanítást legkönnyebben a matematikából ismert gradiens segítségével lehet megvalósítani. A gradiens számításának módja a következő.

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T \tag{2.3}$$

Az efféle modellezés hasonlít egy lejtőn leguruló labda fizikai modelljéhez is, gyakran szokták ahhoz hasonlítani. Ezen analógia szerint a sokdimenziós lejtő a háló súlyainak feladat által generált sokdimenziós állapottere, amelynek értéke az akuális megoldás jósága, és amelynek globális minimuma a cél, amelyet szeretnénk elérni. A lejtőn mozgó labda pedig a háló jelenlegi állapotát jelenti. Az analógiát a 2.3. ábra jól mutatja.

Ez a megoldás – különösen sok dimenziós terekben – gyakran jól működik. Problémát jelenthetnek viszont a függvényben megjelenő lokális minimumok, ugyanis ha minden dimenzió szerint lokális minimumról van szó, akkor ezt elérve a tanulás megáll, így a globális minimumot nem érjük el.

Erre a problémára nincs tökéletes megoldás, azonban sok esetben segíthet az úgy nevezett Adam optimizer [18]. Ezen megoldás az előző bekezdésben bemutatott labda analógia fizikájához még hasonlóbb módszert használ. Ez az eljárás ugyanis bevezeti a momentum fogalmát, ami azt jelenti, hogy a gradiens függvényében nem közvetlenül a lépést módosítjuk (ami fizikai analógiában a sebesség lenne), hanem a momentumot, amelytől időben függ a lépésköz (ez fizikai analógiában a gyorsulást jelenti). Ennek a



2.3. ábra. Két dimenziós lejtő és labda analógiája [12] a gradient descent optimizer és az Adam optimizer eljárások működésének bemutatására

megközelítésnek két nagy előnye is van. Egyrészt egy hosszú lejtő mentén (a lejtő általam használt fogalma azt jelenti, hogy a labda által megtett út adott szakaszán végig azonos előjelű a gradiens) nagyon gyorsan "le tud gurulni" a háló állapotát jelképező labda, mivel folyamatosan gyorsul, azaz folyamatosan nő a lépésköz. Másrészt képes lehet áthaladni lokális minimumokon is, ha nem túl nagy azok vonzáskörzete, ahogy egy hegyről leguruló labda is képes kellően megnövekedett sebességével átgurulni egy kisebb dombon.

2.3.4. Veszteség függvény

Egy működő háló megvalósításához az utolsó szükséges lépés a veszteség generálása. A veszteséget egy előre definiált függvénynek kell meghatároznia, és gyakran ennek jó megválasztásán múlik egy háló sikeressége. Ennek az az oka, hogy a veszteséget generáló függvénynek egyetlen számban kell leírnia, hogy mennyire volt jó a háló, és ha a jóság fogalma nem egyezik az emberi gondolkodásban és a matematikai leírásban, akkor a háló, bár matemaikailag lehet, hogy jól tanul, de mégsem a várt eredményt adja. A veszteség függvény megválasztása emiatt nagyban feladatfüggő, és mivel gyakran nincs egyértelmű leképzés az emberi gondolkodás és a matematikai felírás között, sokszor intuitív módon történik. A veszteség függvény generálására épp ezért nincs általánosan elfogadott, mindig működő módszer.

2.4. Reinforcement learning [19]

Az előző fejezetben bemutatott háló már sokféle tanítási folyamatra alkalmas, ugyanakkor mozgás szimulációt még valószínűleg nem képes megtanulni. Ennek az az oka, hogy a teljes modell lefutásának végén generált és visszaterjesztett hiba túl sok információt tartalmaz nem eléggé komplex módon leírva. És ahogy például egy függvény adott szakaszon vett Riemann integráljából [20] sem állíthatóak vissza a függvény értékei, ugyanígy a modell végén generált hiba sem határozza meg, hogy a háló által hozott döntések önmagukban mennyire jók, csupán a teljes futtatás sikerességét méri.

Erre jelenthet megoldást az úgynevezett reinforcement learning (megerősítéses tanu-

lás), ezen belül is az úgynevezett temporal difference learning, rövidebben TD learning. TD learning során a hiba visszaterjesztése a szimuláció futtatása során minden egyes szimulációs lépés végén megtörténik. [21] A veszteség függvény generálásához reinforcement learning esetében jutalom függvényt is felhasználunk. A jutalom függvény generálása is sokféle lehet. Egyik megoldás például, ha minden időpillanatban generálunk jutalom függvényt, a szerint, hogy az adott állapotban mennyire teljesített jól a háló. Ez a megoldás egy jól megválasztott jutalom függvény segítségével már képes lehet megvalósítani jelentős időbeliséggel rendelkező problémákat is.

Ez a fajta háló tanulás esetén a minél nagyobb jutalom elérésére fog törekedni az adott időpillanatban, ami bizonyos esetekben a legjobb megoldástól eltérő megoldásokhoz vezethet. Ugyanis ha a háló rátalál a jutalom függvény egy lokális maximumára, akkor nem biztos, hogy tovább fog próbálkozni más megoldások mentén, ahogy esetleg jelentősen nagyobb jutalmakat is kaphatna.

Erre a problémára jelenthet megoldást a felfedezés-optimalizálás séma. [22] [21] Ezen sémát nemcsak mesterséges intelligenciáknál alkalmazzák, hanem más területeken is, ahol a minél nagyobb haszon elérése a cél, és ahol a lokális maximumok csapdát jelenthetnek. A séma lényege az, hogy a rendszer általában az optimalizálás eljárását követi, ez reinforcement learning esetén például a TD learninget jelenti. Azonban bizonyos időközönként a rendszer megkísérel felfedezni új, az eddigieknél lényegesen eltérő és potenciálisan azoknál jobb megoldásokat is. Ez reinforcement learning esetében megvalósítható például azzal, hogy a hálót rövidebb ideig az előző fejezetben bemutatott evolúciós eljárással futtatjuk, így a rendszer véletlenszerűen rátalálhat az eddigieknél jobb megoldásra is.

TD learning megvalósítására három alapvető megoldási lehetőséget ismerünk.[21] Lehetséges a modell alapú felírás. Ilyen esetben magát a modellt kísérelјük meg közelíteni. Ez azonban csak olyan esetekben működik, amikor a modell tökéletesen felírható változók segítségével. Vagyis ez a megoldás általában nem jó út.

Másik megoldás a jutalom függvény érték szerinti közelítése. Ezt a megoldást mutattam be ezen fejezetben. Ilyen megoldás esetén a háló mindig az adott pillanatban legnagyobb jutalmazást jelentő opciót fogja választani. Azonban az előzőekben is megemlített módszerek használatával (például Adam optimizer alkalmazása, felfedezés-optimalizálás séma használata) elkerülhető bizonyos mértékig a lokális maximumokban ragadás veszélye.

A harmadik megoldási módszer az úgynevezett policy alapú megközelítés, amelyre a következő fejezetben fogok részletesebben kitérni.

2.5. Q learning [23]

A Q learning is egyfajta reinforcement learning, így az előző fejezetben leírtak nagy része igaz lesz erre a megoldásra is. Alapvető eltérés azonban az eddigiekhez képest, hogy Q learning esetén a tanítás alapját nem az állapottér adott pontjában felvett jutalom határozza meg közvetlenül, hanem bizonyos döntési folyamatokhoz rendelünk az adott döntési folyamat jóságát leíró értéket.

Az eljárás megvalósítása legegyszerűbb esetben egy Q table létrehozásával történik. Ebben a táblában reprezentálva van az adott állapotban az összes lehetséges következő lépés jósága. A tábla frissítése pedig a Q függvény segítségével történik, amelyet a Bellmann egyenlet ír le, a következő képpen. [24]

$$Q_{uj}(s,a) = Q(s,a) + \alpha \left[R(s,a) + \gamma max Q'(s',a') - Q(s,a)\right] \tag{2.4}$$

Ez után a Q table segítségével elméletileg bármilyen döntési sémát meg lehet oldani, mégpedig úgy, hogy mindig a legnagyobb Q értékkel rendelkező opciót választjuk a döntési lehetőségek közül.

Ennek a megoldásnak azonban több hibája is van. Egyrészt lehetetlen vele folytonos állapotterű rendszert modellezi, mivel ilyen esetben végtelenül nagy Q table-re lenne szükségünk. Másrészt a valóságban gyakran a véges állapotterű rendszerek is rendkívül sok állapotból állnak (gyakran az eltelt idővel együtt nő az állapottér nagysága), így a Q table ilyen esetben is könnyen túl nagyra nőne. A Q table nagysága elsősorban nem is az általa elfoglalt terület miatt probléma (például memóriában elfoglalt terület), hanem mivel a rendszernek minden egyes lépésben minden Q értéket kell számítania, ami így rendkívül lassú futást eredményezne.

Erre jelent megoldást az úgy nevezett policy gradient módszer. [25] [26] Ez esetben a háló nem az érték-függvényt tanulja meg, amelynek kiszámítása minden állapotra meghatározza a Q táblát, aminek alapján döntést tud hozni a következő lépésre, hanem az adott pillanatbeli állapothoz közvetlenül rendeli hozzá a döntést.

Policy gradient módszer alkalmazásánál a policy maga lehet sztochasztikus vagy determinisztikus is. Determinisztikus esetben a policy az állapottérből egyértelműen meghatározza a döntést. Míg sztochasztikus esetben egy bizonyos valószínűséget ad az adott döntés meghozására. Utóbbinak nagy előnye, hogy nem körmentes, irányított döntési gráf esetén sem fog végtelen ciklusban maradni, mivel bizonyos eséllyel kilép a körből akkor is, ha ez nem a legvalószínűbb döntés.

A policy gradient megoldásnak alapvetően egy nagy hátránya van a Q table használatával szemben. Mégpedig hogy a policy gradient megoldás gyakran konvergál a jutalom függvény egy lokális maximumához a globális optimum helyett. A policy gradient módszer megvalósításához szükséges összefüggések a következőek. [26]

$$Policy\ gradiens : E_\pi \left[\nabla_\theta (log\pi(s,a,\theta)) R(\tau)\right] \tag{2.5}$$

$$Update : \Delta\theta = \alpha * \nabla_\theta (log\pi(s,a,\theta)) R(\tau) \tag{2.6}$$

3 Feladatkiírás

"Vizsgálja meg a szakirodalomban a mozgásirányításhoz használt módszereket eljárásokat"

Megismerkedtem a mozgásminták tanításához leggyakrabban alkalmazott eljárásokkal, melyek alapvetően a megerősítéses tanulás témakörébe tartoznak, ezekről a 2.4. és a 2.5. alfejezetekben írtam részletesebben. Valamint megismerkedtem a legsikeresebb és leggyakrabban alkalmazott mozgás szimulációkkal is, melyekről elsősorban a 1. fejezetben írtam (erre a témára azért nem tértem ki részletesebben, mert ezek a modellemtől jelentősen eltérő alapelveket felhasználó modellek, így felépítésük is nagyban különbözik az általam megvalósított modell jelenlegi állapotától). Ezen kívül megismerkedtem az ember, mozgásminták megvalósításáért felelős szervrendszereivel, melyekről a 2.1. és 2.2. fejezetekben írtam.

"Ismerkedjen meg és mutassa be a megerősítéses tanulás módszertanát"

Megismerkedtem a megerősítéses tanulás alapvető elméleti hátterével, valamint megismerkedtem annak bizonyos megközelítéseivel is. Megismerkedtem a policy gradient, valamint a Q-learning tanulási eljárásokkal. Ezen témákról a 2.4. és a 2.5. alfejezetekben írtam.

"Hozzon létre egy szimulációt, ahol egyszerű mozgásmodellek, fizikai rendszerek vizsgálhatóak"

Megvalósítottam egy szimulációs környezetet, mely a Newtoni fizikán alapul, és melyben mozgásmintákat tanuló neurális hálók könnyen taníthatók, valamint teljesítményük könnyen kiértékelhető. Ezen kívül a vizsgálatok megkönnyítése érdekében létrehoztam egy grafikus megjelenítésre alkalmas egységet, melynek segítségével a futtatások különböző állapotai vizualizálhatóak. Ezen témákról a 4. fejezetben írtam, több alfejezeten keresztül.

"Keressen megfelelő paramétereket és hozzon létre modelleket a szimulációs rendszerben"

Sok különböző paraméterezési lehetőséget megvizsgáltam, valamint megvalósítottam különböző állapotterekben különbözőképpen teljesítő modelleket. Ezen modellekről szintén a 4. fejezetben írtam, több alfejezeten keresztül, valamint a 5.1. fejezetben különböző állapottereken is végeztem méréseket.

"Vizsgálja meg a modellek taníthatóságát különböző módszerek, paraméterek és kezdeti feltételek mellett"

A modellek taníthatóságát a modellek fejlesztése során folyamatosan vizsgáltam, és kipróbáltam különböző kezdeti feltételeket, paramétereket és tanítást megvalósító felírásokat. Ezen vizsgálatok közül néhányat számszerűen is leírtam a 5.1 alfejezetben, valamint általánosságban említettem más vizsgálatokat is a 4. fejezetben.

"Értékelje a kapott eredményeket"

A kapott eredményeket a dolgozat folyamán igyekeztem mindig az adott eredménnyel együtt kiértékelni. Valamint a 6. fejezetben írtam egy rövid összefoglalót, mely tartalmazza az eredmények általános értékelését is.

4 A feladat megvalósítása

4.1. A feladat megvalósításához felhasznált eszközök

A feladat alapvetően két nagy részre volt bontható.

Az első egység a szimulációt megvalósító környezet, melynek alkalmasnak kellett lennie a newtoni fizika modellezésére, valamint képesnek kellett lennie a humán passzív és aktív motoros rendszerek modellezésére is. Ezenkívül közel elengedhetetlen volt a szimuláció grafikus megjelenítése, a tanítás kiértékeléséhez. Valamint olyan megoldást kellett választani, mely képes volt a szimulációt gyorsan futtatni, így nem növelve feleslegesen a tanításhoz szükséges időt.

Első próbálkozásként C++ nyelven írtam meg a teljes első egységet egyetlen kódként, mivel ezt a nyelvet már viszonylag jól ismertem, és megfelelt a kritériumoknak (a nyelv hivatalos referenciája 2018.11.18-án a következő oldalon volt elérhető: `https://en.cppreference.com/w/`). Ezt a megoldást végül funkcionális okokból két külön állományra bontottam, az egyik felelt a szimuláció számítási műveleteinek megvalósításáért, valamint képes lett a rendszer pillanatnyi állapotát txt formátumú fájlban rögzíteni. A másik futtatható állomány felelt a txt formátumú fájl beolvasásáért, valamint képes lett megjeleníteni az abban rögzített állapotokat.

A második egységnek a neurális háló felépítését és a tanítási folyamatokat kellett megvalósítania. Erre a feladatra a TensorFlow keretrendszert választottam, mely mindeddig jó döntésnek bizonyult (a keretrendszer hivatalos referenciája 2018.11.18-án a következő oldalon volt elérhető: `https://www.tensorflow.org/`). A TensorFlow keretrendszert Python2 környezetbe implementálva használtam (a Python nyelv 2018.11.18-án a következő oldalon volt elérhető: `https://www.python.org/doc/`). A futtatásokat a Google Collaboratory fejlesztői környezetén végeztem (2018.11.18-án a következő oldalon volt elérhető: `https://colab.research.google.com/`).

A Python és a C++ kódokat ctypes Python könyvtár segítségével szándékoztam összekötni (a ctypes 2018.11.18-án a következő oldalról volt elérhető: `https://docs.python.org/2/library/ctypes.html`). Azonban ezt több különböző próbálkozással sem sikerült megvalósítanom, mivel a ctypes (számomra egyelőre nem tisztázott okokból) úgy fordította át a C függvényeket (melyeket C++ függvényekből alakítottam át pointerek segítségével), hogy az objektumok inicializálása még sikeres volt, de a belső állapotukat már nem lehetett tagfüggvényeikkel sikeresen változtatni, olyan módon, hogy módosított értékeik a következő lekérdezésnél is megmaradjanak.

A probléma megoldásaként felmerült fájlok használata, ez azonban valószínűleg rendkívül lassította volna a tanítás folyamatát, mivel azokat minden tanítási lépés során meg kellett volna nyitni, írni kellett volna, majd be kellett volna zárni, és ezt meg kellett

volna ismételni, az írás helyett olvasással. Miután ezt az ötletet elvetettem, arra jutottam, hogy a legegyszerűbb megoldás az lesz, ha a C++ kódok legutolsó állapotát átírom Python kóddá.

Így végül az első egység azon részét, mely a szimuláció belső állapotának számításáért volt felelős, Python kódként implementáltam azon kódba, mely később a TensorFlow keretrendszer segítségével volt hivatott megvalósítani a neurális hálót és annak tanítását. Valamint az első egység azon részét, mely a grafikus megjelenítésért volt felelős, ugyancsak átírtam Pythonba, a további nyelvek közti problémák elkerülése érdekében. A grafikus megjelenítéshez Tkinter-t, a Python saját grafikus könyvtárát használtam (2018.11.18-án a következő oldalról volt elérhető: `https://docs.python.org/2/library/tkinter.html`). Mivel a Tkinter könyvtár nem volt elérhető a Google Collaboratory fejlesztői környezetében, ezért a Jupyterlab-ot használtam ezen kód verzióinak futtatásához (a Jupyterlab hivatalos dokumentációja a következő oldalon érhető el: `https://jupyterlab.readthedocs.io/en/stable/`).

4.2. Newtoni fizikát megvalósító környezet

A newtoni fizikát alkalmazó modell megvalósítását a folyamat során végig a masszív és aktív motoros rendszerek modellezésével együtt végeztem, így könnyítve a modell helyességének tesztelését. A környezetet jelen pillanatban csak két dimenziós kitéréssel modelleztem. Ennek első sorban az volt az oka, hogy a három dimenziós megjelenítés lényegesen komplikáltabb feladat, mint a két dimenziós megjelenítés. A későbbiekben van lehetőség a jelenlegi modell átalakítására úgy, hogy három térdimenziót is képes legyen kezelni.

A fizikai rendszer lemodellezése matematikailag a differenciál-egyenlet rendszer megoldásán alapul. Azonban a szimulációt nem előre megoldott differenciál-egyenletek segítségével futtattam. Ennek elsődleges oka az volt, hogy a különböző, viszonylag egyszerű matematikai modellek együtt rendkívül bonyolult rendszert alkotnak. Továbbá ha egy ilyen rendszer felírása és megoldása lehetséges is, az nem bővíthető tetszőlegesen, mivel minden egyes hozzáadott elem után újra kell számítani a differenciál-egyenlet rendszert.

Így a differenciál-egyenletek közvetlen megoldása helyett az iteratív felírást választottam, melyet numerikus módon oldottam meg. Ennek a megoldásnak nagy hátránya, hogy numerikus hibák léphetnek fel. Azonban valós fizikai rendszerek vizsgálata során is előfordulnak különböző típusú hibák. Így egy olyan háló, mely bizonyos szintű hiba esetén is képes megoldani a problémát, potenciálisan még jobb is lehet valós fizikai problémák megoldására, mint egy "steril" környezetben tanított háló.

Ennek tudatában a differenciál-egyenlet rendszer megoldására a legegyszerűbb iteratív megoldást alkalmaztam. Felvettem hat változót minden egyes tömegpontra, melyek páronként felelnek a pozíció (p), sebesség (v) és a gyorsulás (a) pillanatnyi értékének két dimenziós, előjeles tárolásáért. Majd ezen változókat minden iteratív lépésben egymástól függően frissíti a program. Mivel a sebesség a pozíció első időbeli deriváltja, a sebesség pedig a gyorsulás első időbeli deriváltja, ezért a következő iterációs egyenletek írhatók

fel.

$$p(x,y)_{t(n+1)} = p(x,y)_{t(n)} + v(x,y)_{t(n)} \tag{4.1}$$

$$v(x,y)_{t(n+1)} = v(x,y)_{t(n)} + a(x,y)_{t(n)} \tag{4.2}$$

A gyorsulás kiszámítása pedig az adott időpillanatban a tömegpontokra ható erő (F) és a tömegpontok tömege (m) alapján történik.

Ebben a felírásba rendkívül egyszerűen bevezethető egyfajta súrlódási tényező. Ezen súrlódási tényezővel nem a valós súrlódást kívántam modellezni, csupán annak egy torzított imitálására törekedtem. A valóságban a súrlódási és közegellenállási tényezők gyakran függnek a sebességtől és más paraméterektől is. Az általam bevezetett tényező konstans, egyfajta csillapítást jelent. Ha a súrlódási tényező állandó (α), akkor a sebességre felírt iterációs egyenlet a következőképpen módosítható (α itt a megmaradó sebesség arányát jelenti az eredeti sebességhez képest, és α értéke 0 és 1 közötti, azaz sebességvesztés történik).

$$v(x,y)_{t(n+1)} = \alpha \cdot v(x,y)_{t(n)} + a(x,y)_{t(n)} \tag{4.3}$$

Python nyelven ezt a következő kódrészlet írja le.

```
self.x += self.vx
self.y += self.vy
self.vx = self.vx*dampening+self.ax
self.vy = self.vy*dampening+self.ay
```

A rendszer határ-felületeire azonban más egyenletek bevezetése is szükséges. Ezen egyenleteket úgy választottam meg, hogy a talajnál és az oldalfalaknál (azaz ha a tömegpont y vagy x pozíciója átlép egy bizonyos értéket) bizonyos szinten rugalmas ütközést szimuláljon. Mindemellett pedig a numerikus hibák hatásának csillapítása érdekében ha a tömegpont bizonyos x vagy y irányú sebesség alatt közelíti meg a talajt (ez kis sebességet jelent), akkor a sebessége automatikusan nullára csökken. Ez egyben bizonyos szinten modellezi a tapadási súrlódás jelenségét is. A szimuláció bizonyos újabb verzióiban bevezettem (tanítási anomáliák miatt), hogy amennyiben valamelyik tömegpont megközelíti az oldalfalakat, a szimuláció automatikusan leáll. Így ezekben a verziókban a szimuláción belüli inkoherenciák elkerülése végett kikommenteltem az oldalfalakra vonatkozó feltételt. Ezen feltételek Python kódrészletként megvalósítva a következőképpen néznek ki.

```
if self.x<=5 or self.x>=XX-5:
    self.vx = -self.vx*xflexibility
    if self.x<=5:
        self.x = 5
    if self.x>=XX-5:
        self.x = XX-5

if self.y>=YY-flor-7:
```

```
self.vy = floordampening*self.vy
self.vx = floordampening*self.vx
if abs(self.vy)<floorstickyness
    or abs(self.vx)<floorstickyness:
    self.vx=0
    self.vy=0
```

Mindemellett egy általános gravitációs erőteret is be kellett vezetnem, mely függőlegesen hat a talaj irányába. Ezen erőteret nem ábrázoltam két dimenziós potenciáltérként, csupán a tömegpontokra gyakorolt hatását implementáltam. Egy függőleges irányú, g együtthatójú gravitációs erőtér hatása m tömegpont gyorsulására a következőképpen írható fel.

$$a(y)_{t(n+1)} = a(y)_{t(n)} + g \cdot m \tag{4.4}$$

Ennek megvalósítása Python kódrészletként a következőképpen néz ki.

```
self.ay += G*self.m
```

4.3. Passzív és aktív motoros rendszerek modellezése

4.3.1. Csontok modellezése

Mint ahogy azt a 4.2. alfejezetben említettem, a szimulációban tömegpontokat használok, nem számolok kiterjedt testek fizikájával. Ezen tömegpontokat az ízületi pontokon helyeztem el (ízületi pont az, ahol két csont találkozik). Tömegpont a rendszeremben csak ízületi pontokon helyezkedik el, kumulálva ezzel az adott csontvégződésekre ható erőket. A tömegpontok fizikájának megvalósítását (ezzel az ízületek megvalósítását is) a 4.2 fejezetben leírtam.

Az ízületi pontok mellett fontos passzív motoros építőelemek a csontok is. A csontok modellezésénél azt a közelítést használtam, hogy azok tökéletes szilárdak, vagyis hosszuk és alakjuk állandó. A modellemben kizárólag egyenes csontokat alkalmazok, ennél összettebb alakú csontokat több egyenes csontból lehet benne megalkotni (legalább három csont szükséges ehhez, melyek háromszög alakzatban helyezkednek el egymáshoz képest).

A csontokat ízület páronként reprezentáltam, valamint ehhez rendeltem hozzá a csont hosszát. A csontok azon tulajdonságát, hogy hosszuk állandó legyen, folyamatos javítással biztosítottam. Ha egy csont egyik végén az egyik ízület elmozdul, akkor a csont hossza megváltozhat. Ezt a változást javítottam vissza azzal a megoldással, hogy a csont egyik végén lévő ízületet a csont új iránya szerint (melyet a jelenlegi hosszal normált vektorokkal határoztam meg) olyan pozícióba helyeztem át, hogy a csont hossza ismét az inicializáláskor meghatározott hossz legyen. Így ha egy csont egyik végén lévő ízület pozíciója rögzített, akkor a másik végén lévő ízület kizárólag egy körpálya mentén helyezkedhet el (javítás után), és mivel az ízületi pontokra igaz a gravitáció törvénye a 4.2. fejezetben leírtak szerint, ezért egy ilyen összeállítás ingamozgást kell hogy végez-

zen (amennyiben a gravitáción kívül más külső erő nem hat a rendszerre). A csontok viselkedését Python nyelven a következő objektum írja le.

```
class Bone:
  def __init__(self, j1, j2):
        self.j1 = j1
        self.j2 = j2
        self.l = dist(self.j1, self.j2)

  def repair(self):
        d = dist(self.j1, self.j2)
        ex = (self.j2.getx()-self.j1.getx())/d;
        ey = (self.j2.gety()-self.j1.gety())/d;
        px = self.j1.getx()+self.l*ex;
        py = self.j1.gety()+self.l*ey;
        self.j2.repos(px,py);
```

4.3.2. Izmok modellezése

Az izomzatot azért nevezzük aktív motoros szervrendszernek, mert szabályozható a működése (ellentétben a passzív motoros egységekkel, melyek tulajdonságai vagy adottak vagy az aktív motoros rendszer vagy más külső tényezők határozzák meg). Így a csont- és ízületmodelltől eltérően az izommodellnek kell lennie olyan paraméterének, melyet a rendszert szabályozó neurális háló irányít.

Mint azt a 2.2. alfejezetben részletesebben is kifejtettem, az izomzat modellezésére alkalmas a fizikai rugó modell, mégpedig úgy, hogy a rugónak valós esetben nem változtatható paramétereit képes állítani a neurális háló. Azaz a neurális háló arról hoz döntést, hogy az izmok egyensúlyi hossza és rugóállandója miképpen változzon. Az izom viselkedését ezen paraméterek fogják meghatározni, a többi paraméter értéke ezen két paraméter értékétől függ időben (valamint a környezeti tulajdonságoktól, azaz például más izmok hatásaitól).

Az izmokat a csontokhoz hasonlóan ízületi párokként reprezentáltam, valamint ehhez rendelt változóként az izom paramétereit, az egyensúlyi hosszát és az erősségét (nem a potenciális legnagyobb erejét értem erőssége alatt, hanem az aktuális erejét). A csontok viselkedésétől eltérően azonban az izmok viselkedését nem egy javító eljárással biztosítom, hanem egy folytonos függvénnyel, mely az izom mindkét végére ható rugóerőként érvényesül. Ezen erő hatását gyorsítás formájában továbbítom az ízületek felé az ízület objektum egy tagfüggvénye segítségével.

A gyorsítás (a) kiszámítása egy ízületre, normált irányvektorok (i) segítségével történik, melyek kiszámítása minden időpillanatban meghatározza az izom irányát. Ezután a kapott irányvektor szerint meghatározott irányban, az izom erejétől (S), valamint az izom aktuális- (d) és egyensúlyi (l) hosszának különbségéből történik az előjeles gyorsítás meghatározása a következőképpen (c egy tetszőlegesen választott konstans, mely segít S

értékét könnyen értelmezhető nagyságrendben tartani).

$$d = \sqrt{(X_{P1} - X_{P2})^2 + (Y_{P1} - Y_{P2})^2} \tag{4.5}$$

$$i(x, y) = \frac{P1(x, y) - P2(x, y)}{d} \tag{4.6}$$

$$a(x, y) = i(x, y)\frac{d - l}{c}S \tag{4.7}$$

Az izom modell viselkedését a következő Python kódrészletek valósítják meg (c általam választott értéke itt éppen 10000).

```
def dist(j1, j2):
  x1 = j1.getx()
  x2 = j2.getx()
  y1 = j1.gety()
  y2 = j2.gety()
  return math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))

def movement(self):
    d = dist(self.j1, self.j2)
    ex1 = (self.j2.getx()-self.j1.getx())/d
    ey1 = (self.j2.gety()-self.j1.gety())/d
    px1 = (ex1*(d-self.l)/10000*self.strength)
    py1 = (ey1*(d-self.l)/10000*self.strength)

    self.j1.addacc(px1,py1)

    ex2 = (self.j1.getx()-self.j2.getx())/d
    ey2 = (self.j1.gety()-self.j2.gety())/d
    px2 = (ex2*(d-self.l)/10000*self.strength)
    py2 = (ey2*(d-self.l)/10000*self.strength)

    self.j2.addacc(px2,py2)
```

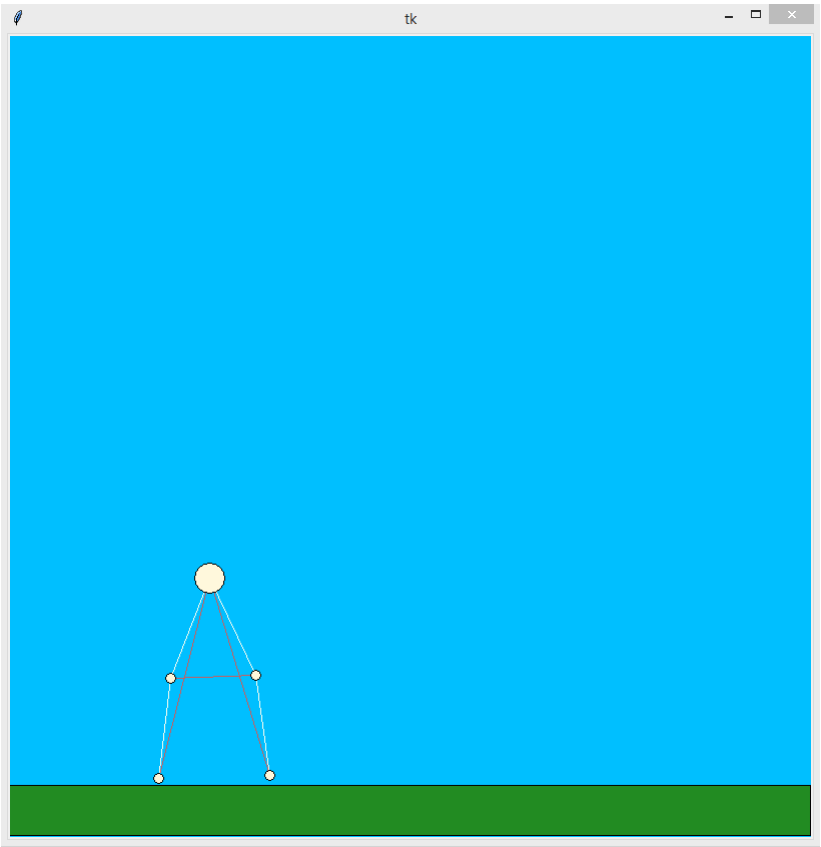
4.4. A grafikus ábrázolás megvalósítása

A háló viselkedésének vizsgálatához elengedhetetlen volt egy kód, mely képes a belső fizikai állapotból generált fájlok megjelenítésére. Ez azért szükséges, mert az ember számára valósan "jó" megoldás, és a matematikailag legjobb megoldás nem feltétlenül egyezik (ahogy ezt a 2.3.4. alfejezetben már leírtam).

A grafikus megjelenítés során igyekeztem csak a legszükségesebb dolgokat ábrázolni, ezzel elősegítve annak érthetőségét. Így a tömegpontok ábrázolásánál némi torzítást is alkalmaztam. A tömegpontokat (azaz az ízületeket) sárgásfehér (fff8dc) körként ábrázoltam, melyeknek sugara arányos a tömegpont tömegével, középpontja pedig a tömegpont

pozíciójával. A csontokat sárgásfehér (ffffff0) vonalakként ábrázoltam, melyek az ízületi pontokat kötik össze. Az izmokat a csontokhoz hasonlóképpen ábrázoltam, rózsaszín (cd5c5c) színben. Ezenkívül a talajt ábrázoltam zöld színben (228b22), valamint a további területeket kék (00bfff) színben. Ezen egységek fizikai viselkedését a 4.2. fejezetben írtam le.

A grafikus ábrázolás megvalósítását a 4.1. ábra mutatja.



4.1. ábra. A modell grafikus ábrázolása az általam megalkotott grafikus egység segítségével. A kép néhány tizedmásodperccel a szimuláció futtatásának indítása után készült.

4.5. Megközelítések a tanítás megvalósításához

A háló megvalósítására, valamit annak tanításának megvalósítására is felmerült több, koncepciójukban is teljesen eltérő megoldás. Legnagyobb szabású változtatások és jövőbeli potenciális változtatások a háló struktúráját, valamint a jutalmazási lehetőségeket érintik. Ezen kívül jelentős konceptuális eltérés a folytonos- vagy véges állapottér használata. Sok szempontból ennek módosítása a legnehezebb, mivel a teljes tanítási mechanizmus újragondolását is jeleni. Ezen kívül bizonyos esetekben célszerű változtatni a környezet felírásán is. Ez a fajta módosítás azonban azt jeleni, hogy nem a tanulás módszere változik meg, hanem maga a feladat. Ami viszont azt jelenti, hogy a különböző környezeti állapotokon futtatott hálókat számszerűen semmiképpen sem célszerű összehasonlítani, mivel egy "egyszerűbb" feladatot megoldó háló valószínűleg sokkal "jobb" megoldást fog adni, mint egy "nehezebb" feladatot megoldó háló. Ennek ellenére

– nyilvánvaló környezeti hiba esetén – be kellett vezetnem ilyen jellegű módosításokat is.

A háló és tanításának modelljét egy nagyon egyszerű, úgy nevezett "cart pole" modell tanítására alkalmas modell alapján inspirálva építettem fel, valamint az ezen feladatot megvalósító példakód egyes részleteit is felhasználtam saját modellem megírása során. Ezen kód a következő linken érhető el: `https://github.com/awjuliani/DeepRL-Agents/blob/master/Vanilla-Policy.ipynb`, valamint ezen oktató jellegű oldal anyagához tartozik: [27].

4.5.1. Háló struktúrája

A háló optimális struktúrájának meghatározására nincs jól bevált módszer. Így ezen probléma megoldása során bizonyos szinten régebbi tapasztalataimra hagyatkoztam, jobbára viszont kísérleti módon igyekeztem a már meglévő hálót fejleszteni.

A háló megvalósítására tett eddigi próbálkozásaim mindegyike fully connected hálót jelentett. Ennek oka az volt, hogy nem láttam struktúrális prioritást a bemeneti adatok és a háló későbbi rétegei között (mint ahogy például egy képelemzési problémában gyakran van környezeti prioritás vagy periódikusság, melyre a hálót eleve fel lehet készíteni, például konvolúciós rétegek alkalmazásával). A későbbiekben viszont tervezek kísérleteket végezni nem csak fully connected rétegeket alkalmazó hálókkal is, ezen lehetőségekről a 5.2. fejezetben írok.

Amiben az általam eddig kipróbált hálók alapvető különbségeket mutattak, azok az alkalmazott nemlinearitások típusai, valamint a háló mélysége és a rétegek szélessége. A nemlinearitások változtatásának az volt az oka, hogy a tanítást eredendően folytonos állapottérben kíséreltem meg (erről a 4.5.2 alfejezetben részletesebben is írok). Ezen állapottérben az izmok paramétereinek állítására lett volna lehetőség folytonos módon, így a kimeneti réteg aktivációs függvényére különböző megoldások is felmerültek (többek között ReLu és Sigmoid is). Az aktivációs függvényekről részletesebben a 2.3. fejezetben írtam. Azonban a tanítás módszerének megválasztása azt eredményezte, hogy a folytonos állapottér jellegű megoldást el kellett vetni. A diszkrét állapottérben pedig döntéshozást egyértelműen a Sigmoid függvény segíti elő, mivel a Sigmoid függvény inflexiós pontjának értékére (azaz 0.5-re) helyezett treshold jól szeparálja az állapotokat. Így a véglegesen alkalmazott hálók mindegyikében a bemeneti- és rejtett rétegek aktivációs függvényeként ReLut, míg a kimeneti réteg aktivációs függvényeként Sigmoidot használtam.

A háló mélységét és a rétegek számát a tanítások során többször is megpróbáltam változtatni. Azonban azt tapasztaltam, hogy a probléma felírásától (azaz a környezet felírásától, a tanítás módszerétől, a jutalom függvény felírásától és még sok más paramétertől is) gyakran függ a szükséges háló mérete. Mind e mellet pedig a rendszer jelenlegi állapotában rendkívül gyorsan tanul (ezt a 5.1.4. alfejezetben méréssel is alátámasztom), azaz a mérések végrehajtása és a háló fejlesztése szempontjából jelen pillanatban nem jelentős tényező a tanításhoz szükséges idő. Így arra jutottam, hogy amíg a háló nem mutat olyan eredményt, mely után már nem tervezem módosítani a probléma felírását, addig a háló méretének optimalizálása számomra nem jó cél (mivel amint változtatom a probléma felírását, a háló méretének optimalizálását is lehet, hogy elölről kell kezdenem).

Ennek eredményeképpen az eddigi tapasztalataimból kiindulva végül egy olyan méretű hálót választottam (háló mérete alatt itt most a neuronok számát értem), melynek mérete (várakozásaim szerint) mindig sokszorosan felülmúlja a szükséges méretet. Azaz ha a háló méretét sokszorosan lecsökkenteném, még akkor sem változna a futtatások átlagos jutalom értéke. Így amennyiben sikerül egy, az előzőeknél jobb felírást megvalósítanom, a háló mérete nem okoz problémát, elfedve ezzel a felírás jóságát.

Az előbbiekben leírt gondolatmenet szerint az elvégzett méréseim során mindig ugyanazt a hálót használtam. Ennek 1 bemeneti rétege van, 10 párhuzamos neuronnal, ReLu aktivációs függvénnyel, 3 rejtett rétege van, 30-70-30 párhuzamos neuronnal, ReLu aktivációs függvénnyel, és 1 kimeneti régete van, 12 párhuzamos neuronnal, Sigmoid aktivációs függvénnyel (melynek kimeneti értékét később, a fizikai környezet futtatása során alakítom át treshold segítségével diszkrét döntési sémává). A háló egymásutáni rétegei mind fully connected rétegek. Ezen háló felépítését a következő kódrészlet valósítja meg.

```
self.state_in= tf.placeholder(shape=[None,s_size],
        dtype=tf.float32)
hidden1 = slim.fully_connected(self.state_in,h1_size,
        biases_initializer=None,activation_fn=tf.nn.relu)
hidden2 = slim.fully_connected(hidden1,h2_size,
        biases_initializer=None,activation_fn=tf.nn.relu)
hidden3 = slim.fully_connected(hidden2,h3_size,
        biases_initializer=None,activation_fn=tf.nn.relu)
self.output = slim.fully_connected(hidden3,a_size,
        activation_fn=tf.nn.sigmoid,biases_initializer=None)
```

4.5.2. Folytonos állapottér problémája, diszkrét állapottér megvalósítása

A valós folyamatok – így a mozgások is, melyeket modellezni igyekszem – folytonos térben valósulnak meg. Így a háló által megvalósított döntési folyamatnál is logikus döntésnek látszik a folytonos állapottér használata. Azaz a háló közvetlenül képes (bizonyos határokon belül) szabályozni a fizikai rendszer állítható paramétereit (azaz az izmok erősségét és egyensúlyi hosszát). Ezen megoldás implementálásával sokat próbálkoztam, azonban végül több különböző megközelítésben is bebizonyosodott, hogy nem lehetséges, mivel az általam alkalmazott tanítási megoldás véges számú policy-ken alapszik (a policy gradient megoldásról a 2.5. fejezetben írtam). Így a folytonos állapottér ötletét egyelőre el kellett, hogy vessem (a későbbiekben tervezek visszatérni ehhez a problémához, és más tanítási módszer alkalmazásával megvalósítani a folytonos állapotteret).

A diszkrét állapottér sem feltétlenül jelent azonban a valóságtól elviekben eltérő megoldást. Ennek az az oka, hogy az idegrendszerben az akciós potenciálok diszkrét információátvitelt jelentenek (a motoros pályák működéséről a 2.1. fejezetben írtam). Ezenkívül pedig azt tapasztaltam, hogy amennyiben a paramétereket megfelelően választom meg, a rugó modellek alkalmazása miatt a diszkrét módon felírt erő változások is közel folytonos mozgást eredményeznek, mivel a rugókon nemcsak a végpontok pozíciójának függvénye,

de azok sebesség-függvénye is folytonos (a rugó modellről részletesebben a 2.2. és a 4.3.2. fejezetekben írtam).

A diszkrét állapotteret úgy alkottam meg, hogy a háló egy döntési sémát határozzon meg a kimenetén. Minden egyes kimeneti változó (összesen 12 van a jelenlegi modellben) egy bináris döntést határoz meg. Az változók első fele (jelen modell szerint első 6) az izmok hosszának változtatását határozza meg páronként, a második fele pedig az izmok erősségének változását határozza meg páronként. A változó párok az adott érték azonos mértékű növelését és csökkentését képesek kiváltani, amennyiben értékük meghaladja a küszöböt. Így 4 lehetséges állapot alakul ki, melyek közül kettő hatása megegyezik.

Az első lehetőség, hogy egyik változó sem éri el a küszöböt, a második pedig, hogy mindkettő eléri. Ezen esetekben az adott paraméter, melyért a változók felelnek nem fog változni (mivel vagy egyáltalán nem változott az értéke vagy pedig ugyanannyit csökkent, mint amennyit nőtt). A második és haramdik lehetőség azt jelenti, hogy csak az egyik változó értéke érte el a küszöböt. Ez esetben attól függően, melyik változóval történt ez meg, az adott paraméter értéke vagy csökkeni vagy nőni fog.

A diszkrét döntési sémát leíró kódrészletek a következőek.

```
def add_length(self, n, b):
    if b>0.5:
        self.l += n
    if self.l <= 0.01:
        self.l = 0.01

def mult_strength(self, n, b):
    if b>0.5:
        self.strength *= n

self.muscle1.add_length(5, m_prob[0])
self.muscle1.add_length(-5, m_prob[1])
self.muscle2.add_length(5, m_prob[2])
self.muscle2.add_length(-5, m_prob[3])
self.muscle3.add_length(5, m_prob[4])
self.muscle3.add_length(-5, m_prob[5])

self.muscle1.mult_strength(1.25,m_prob[6])
self.muscle1.mult_strength(0.8,m_prob[7])
self.muscle2.mult_strength(1.25,m_prob[8])
self.muscle2.mult_strength(0.8,m_prob[9])
self.muscle3.mult_strength(1.25,m_prob[10])
self.muscle3.mult_strength(0.8,m_prob[11])
```

4.5.3. Környezet hatása a háló teljesítményére

A modell megvalósítása során a környezetet elsősorban a nem megfelelő feladatkiírás miatt kellett módosítanom (a háló a legtöbb ilyen esetben jól tanult matematikailag, viszont az általa megvalósított mozgás eltért a várt eredménytől). Ezen folyamat során eddig három jelentősebb változtatást hajtottam végre a környezeti paraméterekre, valamint egy paraméterrel kíséreltem meg jelentősebb változtatásokat, azonban ezt egyelőre a kiindulási állapotra állítottam vissza.

Az első jelentős módosítást a szimuláció leállási feltételén kellett végrehajtanom. A szimuláció leállása jelen pillanatban két komponenstől függ egymástól függetlenül (azaz ha az egyik teljesül, akkor leáll a szimuláció). Ez a két komponens a rendszer "energiája" (nem igazi energia, csupán a pozíciók különbségének összege minden lépésben), mely egy fix értékről indul, és minden lépésben csökken az elmozdulások szerint. Az energiának az a szerepe, hogy egy tökéletes megoldás se fusson örökké, valamint egyben kényszeríti a hálót (amennyiben sikerült már megoldania a feladatot, azaz állva maradnia), hogy egyre finomabb, folytonosabb mozdulatokkal oldja meg azt. A másik leállási feltétel pedig bizonyos ízületek pozíciója (például amennyiben a "fej", azaz a középső ízület földet ér, a modell futása leáll). Ezen feltételeket a modell jelenlegi verziójában a következő kódrészletek írják le.

```
def okay(self):
    if self.joint3.y>YY-flor-20:
        return False
    if self.joint2.y>YY-flor-10:
        return False
    if self.joint4.y>YY-flor-10:
        return False

    if self.joint1.x>XX or self.joint1.x<0 or self.joint2.x>XX
        or self.joint2.x<0 or self.joint3.x>XX
        or self.joint3.x<0 or self.joint4.x>XX
        or self.joint4.x<0 or self.joint5.x>XX
        or self.joint5.x<0:
        return False
    if self.aviable_energy <=0:
        return False
    return True
```

```
self.energy += abs(self.prev_x-self.x)
    + abs(self.prev_y-self.y)
    self.prev_x=self.x
    self.prev_y=self.y
```

```
self.sum_energy = self.joint1.energy+self.joint2.energy
```

```
    +self.joint3.energy+self.joint4.energy+self.joint5.energy
self.aviable_energy -= self.sum_energy
```

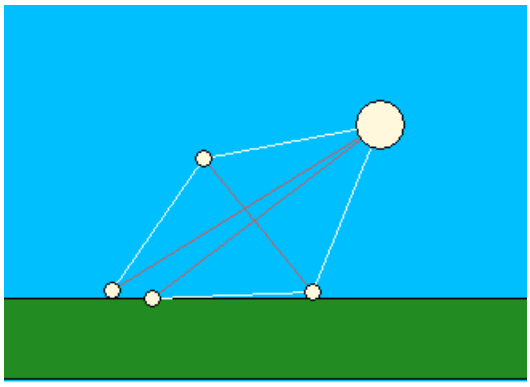
Az első tanítások során még megengedett volt a modellnek, hogy leérjenek a "térdei" (azaz sorban a 2. és a 4. ízületek). Ennek azonban az lett az eredménye, hogy a modell egy "könnyebb utat" választott. Vagyis "féltérdre ereszkedett", így képes volt a rendszer energiájának elfogyásáig is megállni. Ez persze ha csak az időt nézzük, amit képes volt eltölteni a modell a leállásig, akkor jobb teljesítményt nyújtott, mint a modell mostani legjobb verziója, de ezen megoldások nyilvánvalóan nem egy ember számára megfelelő állást írnak le, így a feltételeket úgy módosítottam, hogy a modell 2. és 4. izülete sem érhet le, valamint egyéb hibák elkerülése végett azt is kizártam, hogy a bábu vízszintes irányban kirepüljön a képernyőről (az előző kód már ezt az állapotot írja le).

Mindemellett végrehajtottam egy mérést ugyanilyen feltételek mellett, a háló jelenleg használt jutalmazási függvényével (hogy összehasonlíthatóak legyenek a mérési eredmények), és ezen mérés azt mutatja, hogy az ilyen feltételek mellett futó modell matematikailag sem adott jobb eredményt a jelenlegi legjobb modellnél, csupán könnyen fennakadt a "letérdelés" jelentette lokális minimumon. Ezen mérés részleteit a 5.1.3. alfejezetben írom le.

A "letérdelést" megengedő rendszerben a háló egy lehetséges viselkedését a következő oldalon elérhető videó mutatja (a felvételeket, melyeket bemutatok a továbbiakban, én készítettem, a videókra mutató linkek 2018.11.24-én lettek generálva, valamint ekkor elérhetőek voltak itt a videók):

`http://users.itk.ppke.hu/~szage11/BSC%20szakdolgozat%20felv%c3%a9telek/Mozgas0.avi`

Ezen felvétel egy részletét mutatja a 4.2. ábra.



4.2. ábra. A modell egy megoldása, amennyiben megengedett, hogy leérjen a bábu "térde". Ez a megoldás, bár matematikailag jó eredményt mutat, nem az általam kívánt módon oldja meg a feladatot, így előfordulását a továbbiakban ki kellett küszöbölnöm.

A második szükséges változtatás igazából csak egy hibajavítás volt. A modell viselkedéséből sikerült észrevennem, hogy a környezet felírása során vízszintes irányba nem határoztam meg tapadási súrlódást, csupán függőleges irányba tettem ezt meg. Vagyis a modellnek "végtelenül csúszós" talajon kellett így megállnia, amit jelen pillanatban még túl nehéz feladatnak gondolok a háló számára. A későbbiekben (amennyiben sike-

rül a hálónak megtanulnia állni, tapadással rendelkező talajon) még úgy tervezem, hogy visszatérek erre a nehezítettebb feladatra is, mindenképpen érdekes lenne kipróbálni, hogy megvalósítható-e az egyensúlyozás tökéletesen csúszós talajon is.

Az harmadik szükséges módosítást a bábu kiindulási pozícióján hajtottam végre. A hálót jó ideig olyan állapotból indítottam, hogy a szimuláció egy viszonylag nagy eséssel indult a bábu számára. Ezen esést gyakran sikerült a modellnek "szépen" lereagálnia, azonban még így is az esés következtében lényegesen gyorsabban "esett el" a bábu, mint miután a "lábait" (azaz az 1. és 2. ízületet) a földről indítottam. Így ezen módosítás a háló jelenlegi legjobb állapotában megtalálható, azonban ezen problémához a későbbiekben még mindenképpen szeretnék visszatérni. Ugyanis az esés után néha jelentősen eltérő mozgásmintákat tanult meg a háló, az esés nélkülihez képest. Azaz egy, a két problémát véletlenszerűen váltva tanuló modell potenciálisan jobb eredményt érhet el, mint az eddigi modelljeim bármelyike.

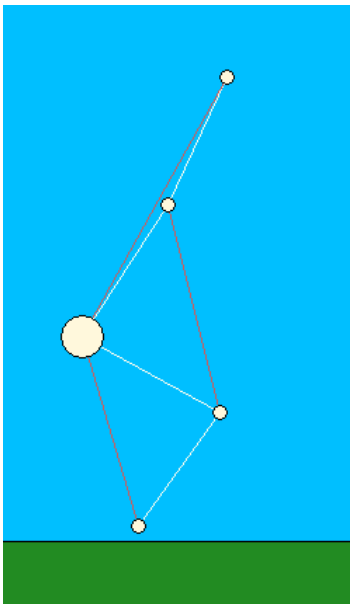
A harmadik módosítás előtti állapotból induló hálók (véleményem szerint) érdekesebb megoldásait mutatják a következő címeken elérhető felvételek.

`http://users.itk.ppke.hu/~szage11/BSC%20szakdolgozat%20felv%c3%a9telek/Mozgas1.avi`

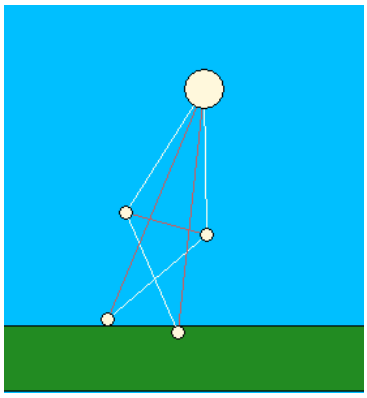
`http://users.itk.ppke.hu/~szage11/BSC%20szakdolgozat%20felv%c3%a9telek/Mozgas2.avi`

`http://users.itk.ppke.hu/~szage11/BSC%20szakdolgozat%20felv%c3%a9telek/Mozgas4.avi`

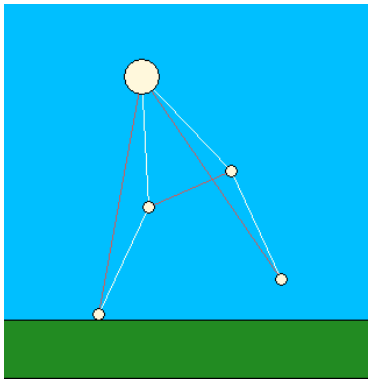
Ugyancsak ezen videók részleteit mutatják a következő ábrák, a linkekkel megegyező sorrendben.



4.3. ábra. A modell egy megoldása, amennyiben a bábu jelentősen a talaj fölül indul. Ezen megoldásban a bábu rendkívül furcsa mozgást végez, az egyik lábával támaszkodik a talajon, míg másik lábával a feje fölött igyekszik megtartani egyensúlyát, sikertelenül. A folyamatot a videó (melynek elérhetőségét fentebb közöltem, és amelyből ez a kép is származik) jobban mutatja.



4.4. ábra. A modell egy megoldása, amennyiben a bábu jelentősen a talaj fölül indul. A modell ez esetben meglehetősen dinamikus módon kísérli meg az állva maradást, így az ilyen jellegű megoldások a későbbiekben még jó kiindulási alapot jelenthetnek például járás tanításához.



4.5. ábra. A modell egy megoldása, amennyiben a bábu jelentősen a talaj fölül indul. Ezen megoldás viszonylag statikus, így az állás tanulásának egy jó lehetséges közelítése.

4.5.4. Jutalmazási lehetőségek

A háló jutalmazásának meghatározása az a folyamat, mely során a feladat paraméterezése történik a háló számára, mivel innentől kezdve a háló a jutalom függvény szerint fog tanulni. Amennyiben a jutalom függvény jól megválasztott, úgy a matematikailag jobb az esetek nagy részében a valóságban is jobb megoldást ad. A jutalmazásról (vagy a veszteség-függvényről) általánosságban a 2.3., 2.4. és 2.5. fejezetekben írtam. A jutalom függvényt azonban intuitívan kell megválasztani, felírására nincs jól bevált módszer, mivel a jutalom függvény is nagyban feladatfüggő. Ennek megfelelően a jutalom függvényen hajtottam végre a legtöbb változtatást a tanítást módosító paraméterek közül. Az alfejezet további részében így csak a jelentősebb, koncepció szintű változtatásokat írom le (a számértékeket változtatókat nem).

A jutalom függvény első megközelítése az volt általam, hogy az állás alatt a felegyenesedett egyensúlyozást értem. A felegyenesedés pedig itt azt jelenti, hogy az állást tanuló modell "feje" minél magasabban legyen. Így első megközelítésben a jutalom függvény a földtől való távolsággal volt egyenlő. Ez a jutalom függvény azonban meglehetősen furcsa megoldásokhoz vezetett, mivel a modell gyakran feláldozta saját stabilitását annak

érdekében, hogy minél magasabban legyen a "feje".

Így a jutalom függvény következő verziója az előző jutalom függvény módosítása lett. Itt már nem az volt a cél, hogy a modell minél magasabban tartsa a "fejét", hanem hogy egy fix magassághoz minél közelebb tartsa meg azt L1 norma szerint. Vagyis az itt generált jutalom annál kisebb volt, minél messzebb helyezkedett el aktuálisan a modell "feje" a fix értéktől. A fix érték legegyszerűbb és számomra leginkább logikus megválasztása az a pozíció volt, ahonnan a modell "feje" indult a szimuláció kezdetén. Így azonban a jutalom értéke negatív lett volna, vagyis azt hozzá kellett adnom egy olyan általam választott konstanshoz, melynek értéke mindig nagyobb, mint a legnagyobb negatív érték, amit hozzáadok (azaz az eredmény mindig pozitív lesz). Az így megvalósított jutalom függvény az előzőhöz hasonlóan lineárisan csökken, ahogy a "fej" közeledik a talajhoz, viszont ugyancsak elkezd csökkenni, ha a "fej" magasabbra emelkedik, mint a saját kiindulási pozíciója. Ezzel a jutalom függvénnyel sokkal jobb eredményeket sikerült elérni, mint az előzővel. Azonban a modell gyakran még mindig furcsa megoldásokkal oldotta meg a feladatot. Egy ilyen megoldást mutat a 4.3. ábra. Ezen kívül a háló csak nagyon rossz arányban tudott elérni olyan megoldást, ahol legalább rövidebb ideig képes volt egyensúlyozni.

Erre jelentett részben megoldást az L2 norma bevezetése, az L1 normához hasonlóan felírva. Itt azonban a nagy konstans értékét, melyből kivonom a norma eredményét sokkal nagyobbra kellett megválasztanom, így a kisebb elmozdulások a kiindulási ponttól szinte egyáltalán nem jelentettek különbséget, viszont egy nagyobb elmozdulás már jelentősen rontott a jutalom értékén. Ez azt eredményezte, hogy a modell "feje" minél inkább vesztett a magasságából, annál jobban törekedett a háló ezen hiba kijavítására. Így a tanítás már sokkal gyakrabban jár bizonyos mértékű sikerrel. Ezen megoldás esetén azonban már be kellett vezetnem egy konstanst, mellyel leosztom a jutalom értékét, hogy az ne érje el a milliós vagy akár milliárdos értékeket, vagyis számomra is könnyen értelmezhető maradjon.

Az előző megoldásnál valamennyivel még jobb eredményt mutatott az általam jelenleg is alkalmazott jutalom függvény. Ezen jutalom függvény az előzőhöz hasonlóan L2 normát alkalmaz, azonban nemcsak y, hanem x irányban is. Vagyis a "fej" kiindulási pozíciójának és jelenlegi pozíciójának x és y irányban vett L2 normáit adja össze, majd ezen összeget vonja ki az fentebb leírt nagy konstansból. A két L2 normán belüli értékeket külön konstansokkal is leosztom, a nagyságrendek illesztése végett, valamint ezen konstansokkal szabályozható a két dimenzió súlya a rendszerben (jelen állapotukban a két konstans megegyezik, de tettem kísérletet a külön változtatásukra is). Ezzel a fajta norma felírással potenciálisan lehetőség nyílik járás tanítására is, mivel a hálótól "elvárt" x pozíció értéke folyamatosan módosítható, azonban ilyen jellegű kísérleteim eddig egyáltalán nem jártak sikerrel. A jutalom meghatározását a jelenlegi kódban a következő kódrészlet valósítja meg.

```
self.reward = (100000 -
pow((self.joint3.y-self.desired_y)/30, 2)-
pow((self.joint3.x-self.desired_x)/30, 2))/1000
```

5 Kiértékelés

5.1. Mérési eredmények

Ezen fejezet további részében a modellemen általam végrehajtott méréseket mutatom be a teljesség igénye nélkül. A modell tervezése során a továbbiakban leírt méréseknél jóval több rövid kiértékelést is elvégeztem, azonban csak az itt is leírt méréseket végeztem el szükségesen nagy adathalmazon. Valamint az itt leírt méréseim eredményei egymással összehasonlíthatóak, mivel a jutalom függvényt nem változtattam meg a különböző mérések között (azaz a jutalom függvény értéke egy általános, a hálók jóságát egzakt módon leíró értéket is jelent ezen mérések során).

5.1.1. Többszöri tanítások lefolyása egy adott hálón

Az első méréssel a tanítások lefutását kívántam bemutatni. A mérés során 26-szor futtattam le ugyanazt a hálót, véletlenül választott, eltérő kezdeti paraméterekkel. Minden futtatás 200 tanítási iterációt jelentett a háló számára. Minden ötödik iterációs lépésnél rögzítettem a háló jutalomfüggvényének aktuális értékét, úgy, hogy az első értéket az első futtatás során mentettem. Ez minden hálóra 40 értéket jelent így, melyek 5 iterációnként követik egymást. A hálónak ugyancsak minden 5 iterációs lépésben megengedett, hogy tanuljon, így valósítottam meg a felfedezés-optimalizálás sémát, melyről a 2.4. alfejezetben írtam bővebben. A modell ezen paraméterek mellett minden szempontból megegyezik azon hálóval, melyet a 4. fejezetben jelenlegi állapotként bemutattam.

A mérési eredményeket az 5.2. ábra mutatja, oszlopokként ábrázolva a különböző futtatásokat, és sorokként ábrázolva az adott adott iteráció szerinti jutalom értékeket. Valamint ezen eredményeket az 5.1. ábra mutatja grafikon formában.

Az 5.2. ábra értékeit a következő címre is feltöltöttem, a jobb átláthatóság érdekében.
`http://users.itk.ppke.hu/~szage11/BSC%20szakdolgozat%20felv%C3%A9telek/5.2-es_abra.PNG`

Ezen eredmények alapján jól látszik, hogy a tanítás az esetek döntő többségében már 100 iterációs lépés alatt lezajlik, viszont bizonyos esetekben még 200 iterációs lépés sem elegendő, hogy a háló megállapodjon egy adott állapotban. Továbbá látható, hogy a háló állapotának jutalom értéke nincs egyenes arányban a végállapot beli jutalom értékével, sőt, gyakran a legjobb eredményeket elérő futtatások kezdeti jutalom értéke kimondottan alacsony más futtatásokkal összehasonlítva. Ez arra enged következtetni, hogy amennyiben a kezdeti paraméterek "túl jók", akkor az optimalizálás könnyen elakad egy lokális minimumban, míg "gyengébb" kezdeti értékek mellett az Adam Optimizer kellő momentumot képes összegyűjteni, így potenciálisan képes áthaladni bizonyos lokális

minimumokon.

Mindemellett az is látszik, hogy a 26 futtatásból mindössze 12 hálónak sikerült 10000-et meghaladó jutalom értéket elérnie a tanítás végére (a 10000-es jutalom értéket elérő hálókra gondolom azt, a jelenlegi paraméterek szerint, hogy sikerült elérniük legalább már minimális haladást az állás megtanulása felé). Ez arra enged következtetni, hogy az esetek több, mint felében a kezdeti paraméterek annyira "rossz" állapotból indultak, hogy a háló képtelen volt bármiféle megoldást találni a probléma megoldására.

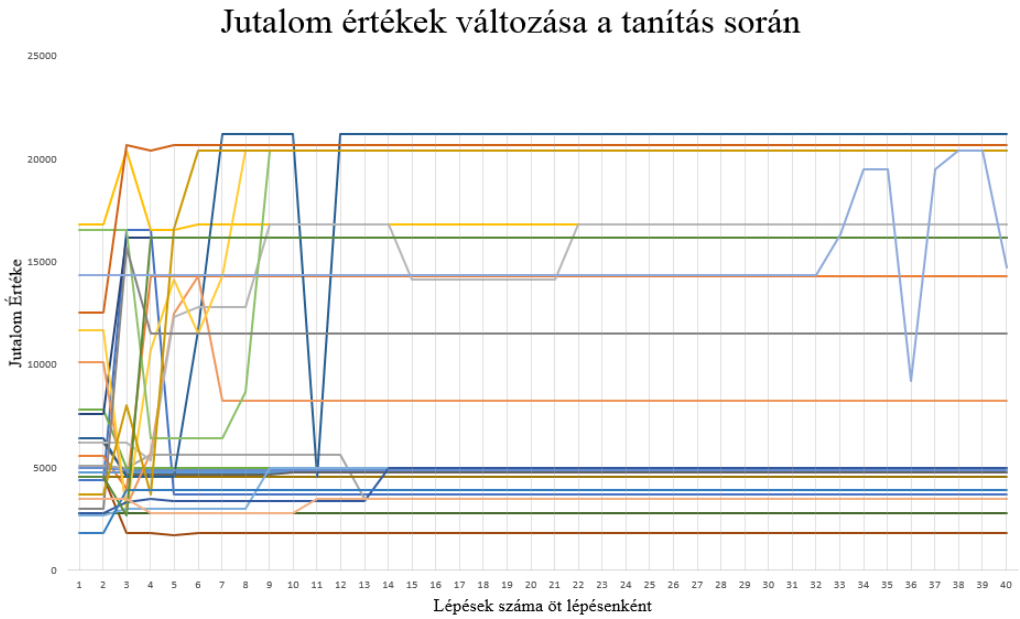
Egy ilyen "rossz" kezdeti állapotból induló háló megoldását mutatja be a következő linken elérhető felvétel. A futtatás értékeit az 5.2. ábra 8. oszlopa mutatja.

`http://users.itk.ppke.hu/~szage11/BSC%20szakdolgozat%20felv%c3%a9telek/Mozgas5.avi`

Összehasonlításképpen két "jó" megoldást mutat be a következő linkeken elérhető két felvétel, melyek értékeit az 5.2. ábra 10. és 15. oszlopa mutatja.

`http://users.itk.ppke.hu/~szage11/BSC%20szakdolgozat%20felv%c3%a9telek/Mozgas6.avi`

`http://users.itk.ppke.hu/~szage11/BSC%20szakdolgozat%20felv%c3%a9telek/Mozgas7.avi`



5.1. ábra. Jutalom értékek változása 26 különböző futtatás során, futtatásonként 200 iterációs lépéssel, egy azon háló esetén. A mérési pontokat 5 iterációs lépésenként vettem fel, így 40 mérési pont segítségével ábrázoltam a tanulás folyamatát minden egyes futtatásra.

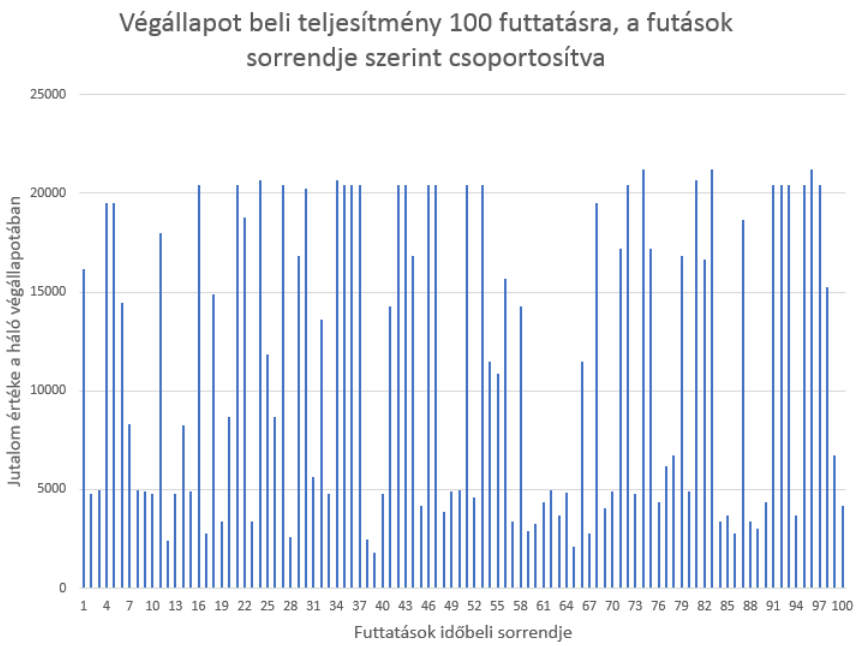
5.1.2. A háló százalékos taníthatósága kezdeti állapot szerint

Az előző mérés (5.1.1. alfejezet) a jutalom-függvények ábrázolására, és azok lefolyásának vizsgálatára megfelelt, azonban a mérések száma ahhoz kevésnek bizonyult, hogy a háló tanításainak eloszlását felírjam. Így az előző mérést megismételtem némi módosításokkal.

4365.38	5539.503	5063.069	16823.98	4758.72	7805.904	7619.26	4567.166	6202.426	4567.258	6413.44	2792.898	4958.304	10118.27	6209.321	11663.91	2692.385	16524.74	2792.898	12532.37	2989.866	3682.274	1798.442	4563.366	14361.15	3484.554
4365.375	5539.503	5063.069	16823.98	4758.72	7805.904	7619.26	4567.166	6202.426	4567.258	6413.44	2792.898	4958.304	10118.27	6209.321	11663.91	2692.385	16524.74	2792.898	12532.37	2989.866	3682.274	1798.442	4563.366	14361.15	3484.554
16553.91	3976.047	4955.237	20392.29	4758.72	4958.965	16148.29	1798.451	4660.588	4567.258	4567.802	2792.898	4860.981	3091.621	6214.701	2889.169	2988.224	16524.74	3287.167	20678.2	15694.02	8017.064	3881.213	2692.383	14361.15	3484.554
16553.91	14286.46	5636.547	16553.91	4758.72	4958.801	16148.29	1798.451	4660.588	4567.258	4567.802	2792.898	4860.981	5732.985	5347.269	10693.86	2988.224	6407.467	3484.554	20388.19	11489.68	3682.274	3881.213	16148.29	14361.15	2792.189
3684.414	14286.46	5636.547	16553.91	4758.72	4958.801	16148.29	1698.717	4660.588	4567.258	4567.802	2792.898	4860.981	12486.19	12339.91	14134.64	2988.224	6407.467	3385.705	20678.2	11489.68	16590.24	3881.213	16148.37	14361.15	2792.189
3684.414	14286.46	5636.547	16823.98	4758.72	4958.801	16148.29	1798.451	4660.588	4567.258	11680.52	2792.898	4860.981	14286.46	12783.53	11489.68	2988.224	6407.467	3385.705	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	2792.189
3684.414	14286.46	5636.547	16823.98	4758.72	4958.801	16148.29	1798.451	4660.588	4567.258	21204.5	2792.898	4860.981	8268.462	12783.53	14286.46	2988.224	6407.467	3385.705	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	2792.189
3684.414	14286.46	5636.547	16823.98	4758.72	4958.801	16148.29	1798.451	4660.588	4567.258	21204.5	2792.898	4860.981	8268.462	12783.53	20392.29	2988.29	8664.53	3385.705	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	2792.189
3684.414	14286.46	5636.547	16823.98	4758.72	4958.801	16148.29	1798.451	4660.588	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	3385.705	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	2792.189
3684.414	14286.46	5636.547	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	3385.705	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	2792.189
3684.414	14286.46	5636.547	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	4567.156	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	3385.705	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	5636.547	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	3385.705	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	3385.705	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	14135.02	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	14135.02	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	14135.02	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	14135.02	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	14135.02	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	14135.02	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	14135.02	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14361.15	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	16329.63	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	19480.31	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	19480.31	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	9194.674	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	19480.31	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	20392.29	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	20392.29	3484.554
3684.414	14286.46	3487.674	16823.98	4758.72	4958.801	16148.29	1798.451	4759.534	4567.258	21204.5	2792.898	4860.981	8268.462	16823.98	20392.29	4955.436	20392.29	4955.237	20678.2	11489.68	20392.29	3881.213	16148.29	14726.41	3484.554

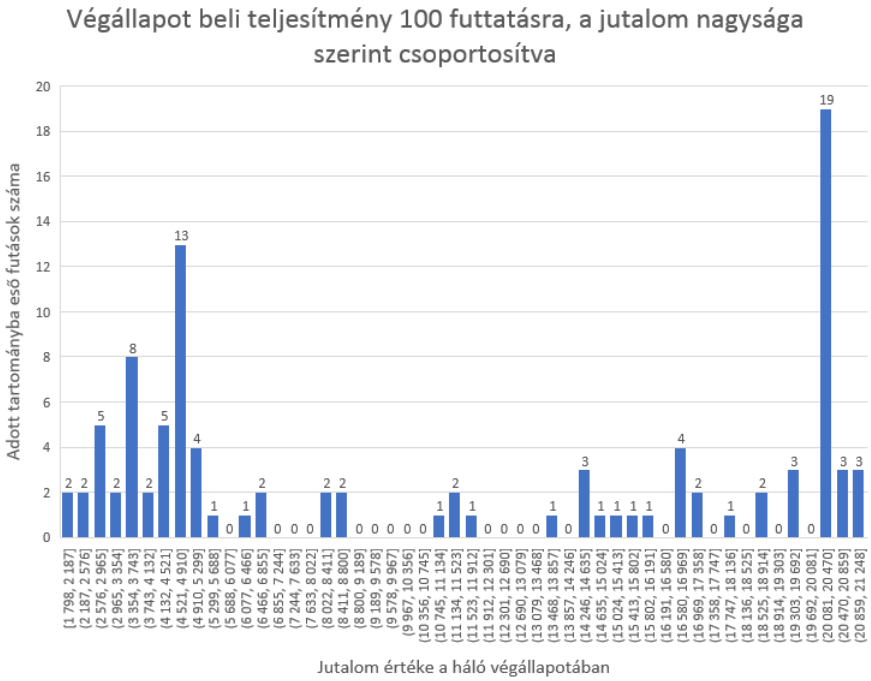
5.2. ábra. Jutalom értékek változása 26 különböző futtatás során, futtatásonként 200 iterációs lépéssel, egy azon háló esetén. Ezen értéktáblázatot azért közöltem, mert a 5.1.1. ábrán nem minden érték vehető ki pontosan, a táblázatban szereplő értékek megegyeznek azokkal, melyeknek alapján a 5.1.1. ábrát készítettem

Ezen mérés során 100 különböző futtatást hajtottam végre, és csak a futtatások végső állapotában rögzítettem a jutalom értékeket. A modell paraméterei az előző méréssel megegyezőek voltak. A mérések értékeit időbeli sorrendjük szerint az 5.3. ábra mutatja. Ezen ábra alapján azt feltételezem, hogy a mérések értékei függetlenek időbeli eloszlásuktól. Az 5.4. ábra azonban már sokkal több értelmezhető információval szolgál.



5.3. ábra. A futtatások sorrendje szerint csoportosított jutalom értékek, egy adott hálóra, 100 futtatás esetén, egyazon háló használatával. Az ábra azt mutatja, hogy a jutalom értéke nem függ a futtatások időbeli eloszlásától (ezt pontosabb mérés esetleg cáfolhatja).

Az 5.4. ábrán a jutalom nagysága szerint csoportosítottam a méréseket, és az adott



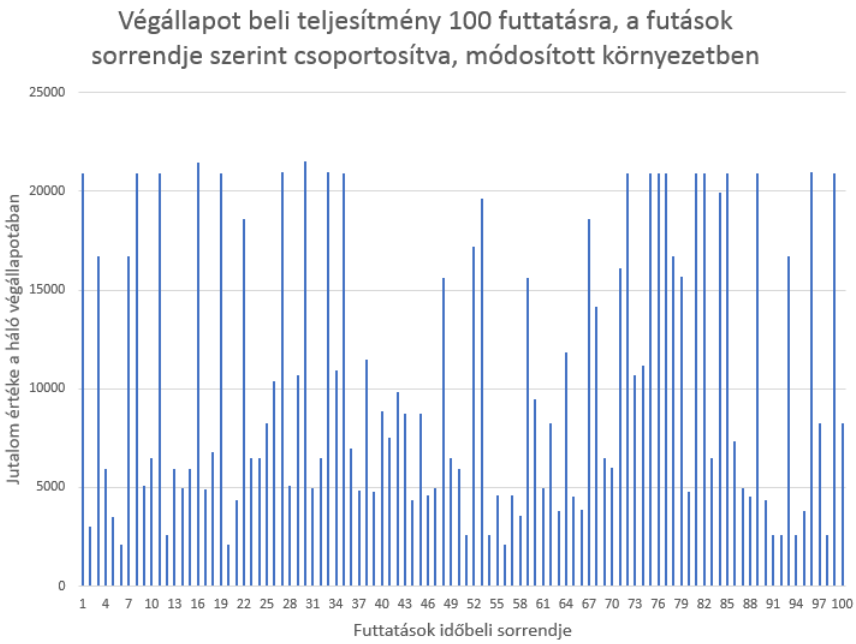
5.4. ábra. A mért értékek csoportosítása, jutalom értékek szerint felvett intervallumokon, 100 futtatás esetén, egyazon háló használatával. Ezen ábra ugyanazon értékek alapján készült, mint az 5.3. ábra. Az ábrán jól láthatóan elkülönülnek a sikeres tanítások jutalom értékei a sikertelen tanítások értékeitől, valamint megfigyelhetőek kisebb csoportosulások is.

(jutalom szerinti) intervallumba eső mérések számát ábrázoltam. Ezen ábrán jól látszik, hogy a jutalom értékek különböző klaszterekbe esnek. Első közelítésként két nagyon jelentős, könnyen elkülöníthető klaszter figyelhető meg. Ezen klaszterek jelentik a számomra irreleváns és releváns tanítások halmazait. Azonban a releváns futtatások halmazában is megfigyelhetőek további kisebb csoportosulások, melyekről azt feltételezem, hogy részben a különböző típusú megoldásokat is jelentik. Ennek vizsgálatára további, pontosabb méréseket tervezek végezni a jövőben.

5.1.3. Környezeti módosítás hatása a háló teljesítményére

A harmadik mérés, melyet elvégeztem, az előző (5.1.2. alfejezetben leírt) méréssel azonos mérési séma szerint történt. Az eltérés a két mérés között a modell környezeti paraméterei jelentik. Azaz a két mérés összehasonlítása jól szemlélteti egy bizonyos környezeti paraméter változtatásának hatását, mivel minden más paraméter szerint a két mérés megegyezett.

Az általam módosított környezeti paraméter itt azt jelentette, hogy ezen mérés során a bábu "térdei", azaz 2. és 4. ízülete leérhetett a földre, hasonlóan ahhoz, ahogy a 4.5.3. alfejezetben ezt a problémát leírtam. Várakozásaim szerint ennek azt kellett volna eredményeznie, hogy megjelenik egy, az eddigieknél matematikailag jelentősen jobb teljesítményt mutató klaszter (a "térdelő" megoldások klasztere), azonban ez nem történt meg. A mérés eredményeit, az előző méréshez hasonlóan az 5.5. és az 5.6 ábra mutatja.



5.5. ábra. A futtatások sorrendje szerint csoportosított jutalom értékek, egy adott hálóra, 100 futtatás esetén, egyazon háló használatával, módosított környezetben. Az ábra azt mutatja, hogy a jutalom értéke módosított környezetben sem függ a futtatások időbeli eloszlásától (ezt pontosabb mérés esetleg cáfolhatja).

A mérés különös eredményének okára alapvetően két elképzelésem született. Az egyik teóriám az, hogy a jutalom függvény két dimenziós, L2 beli felírása (ahogy azt a 4.5.4. fejezetben leírtam) már azt jelenti, hogy a "térdelő" megoldások szignifikánsan rosszabb eredményt jelentenek matematikailag is, így már eleve nem fordulnak elő ilyen jellegű hibák, vagyis a környezet változtatása nem volt hatással a modell viselkedésére.

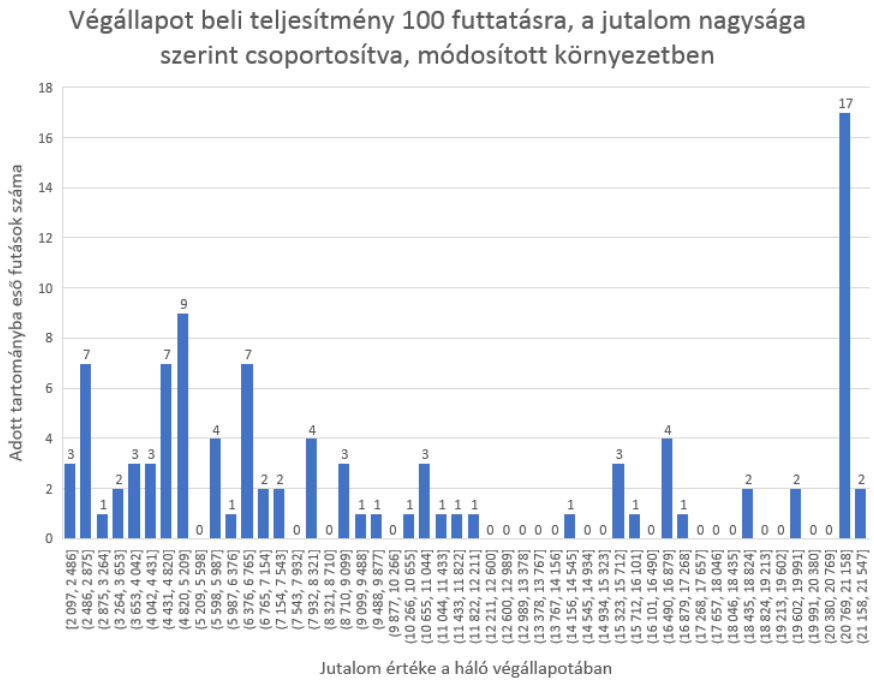
A másik teóriám az, hogy a "térdelő" megoldások jutalom szempontjából nem jobbak, mint az "álló" megoldások, csupán egy másfajta megoldást jelentenek hasonló mennyiségű jutalom megszerzésére. Ez a teória indokolhatja a kisebb csoportosulások elhelyezkedésében történt változásokat (azonban ezen változásokat a véletlen is okozhatta a viszonylag kis számú részeredmény miatt).

A jövőben azt tervezem, hogy minkét teóriát megvizsgálom jóval pontosabb mérésekkel (ezres vagy tízezres vizsgálati számú mérésekkel), ahol a mérések eredményét a grafikus megjelenítő segítségével is vizsgálom, azonban ez rendkívül sok időt vett volna igénybe, így egyelőre ilyen jellegű vizsgálatokat nem végeztem.

5.1.4. Tanítási idő mérése egy adott hálón

Az általam megalkotott modell egyik legjobb tulajdonsága a rendkívül gyors kiértékelhetősége, mivel a tanításhoz szükséges idő a másodperces nagyságrendben mozog. Ezen állításomat az alábbi méréssel kívánom alátámasztani.

Ezen mérés során 95 futtatást végeztem, az 5.1.2. alfejezetben leírt méréssel megegyező paraméterekkel. Azonban ezen mérés során nem csak a hálók jutalmait, hanem a tanításukhoz szükséges időt is mértem (a Python saját időmérő eszköze segítségével).



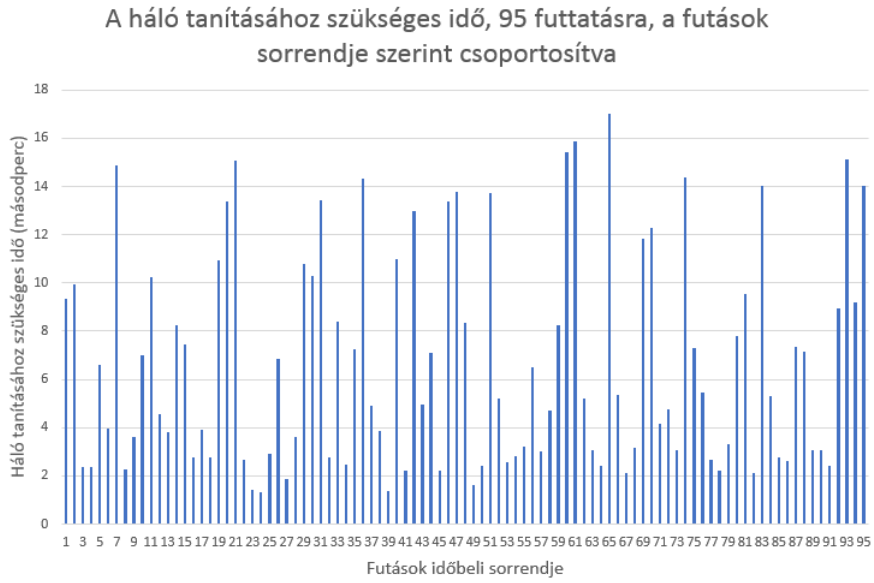
5.6. ábra. A mért értékek csoportosítása, jutalom értékek szerint felvett intervallumokon, 100 futtatás esetén, egyazon háló használatával, módosított környezetben. Ezen ábra ugyanazon értékek alapján készült, mint az 5.5. ábra. Az ábrán az 5.4. ábrához hasonlóan megfigyelhető a jó és rossz tanítások elkülönülése, valamint az 5.4. ábrához hasonlóan megfigyelhetők a kisebb csoportosulások is, de eltérő pozíciókban. Ezt okozhatja a módosított környezet, de lehet véletlen is.

A tanításhoz szükséges idő eloszlását az 5.7. és az 5.8. ábrákon jelenítettem meg, a tanítások sorrendje szerint felírva valamint tanításhoz szükséges idő szerint klaszerezve.

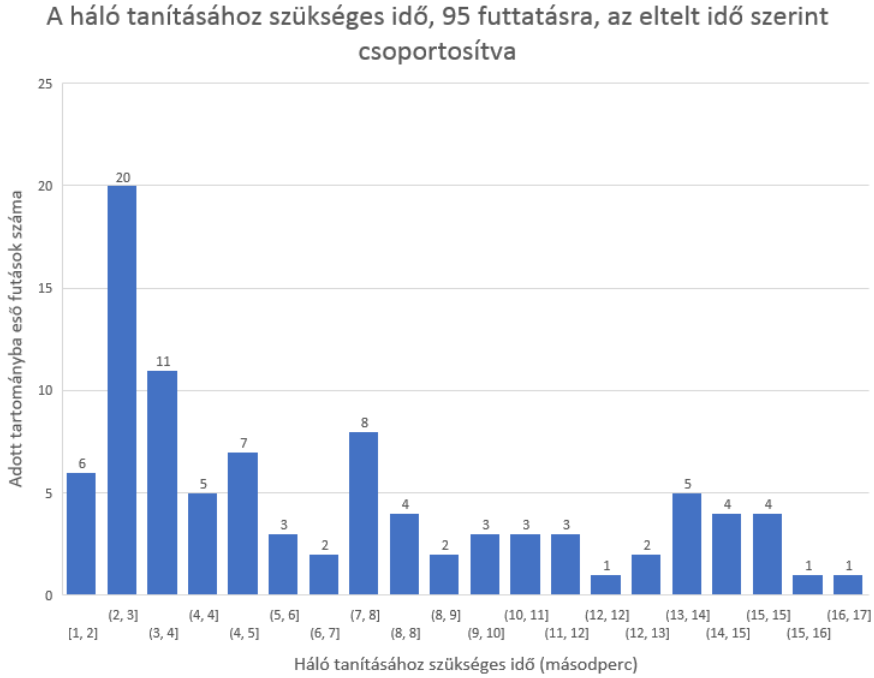
Ezen ábrák azt mutatják, hogy a tanításhoz szükséges idő (a jutalom értékekhez hasonlóan) szintén független a tanítások sorrendiségétől. Azonban a tanításhoz szükséges idők érték szerinti eloszlása jelentősen eltérő klaszterezettséget mutat a jutalom értékek klaszterezettségéhez képest. Ez arra enged következtetni, hogy a tanításhoz szükséges idő és a jutalom értékek között nincs egy egyértelmű, egyenes arányosságot mutató leképzés.

Ezen teóriámat ezen a fentebbi mérési értékeknek, az eddig általam használt ábrázolási módoktól eltérő ábrázolásával kívántam alátámasztani. Ezen ábrázolási mód során a jutalom értékek függvényében jelenítettem meg a tanításhoz szükséges időt (a tengelyek felcserélése hasonlóan jó ábrázolást mutatott volna, így tetszőleges módon választottam ezt a fajta felírást). Az ilyen módon ábrázolt értékeket az 5.9. ábrán jelenítettem meg (az értékekre illesztettem egy ötöd fokú polinomot is, a könnyebb értelmezhetőség érdekében).

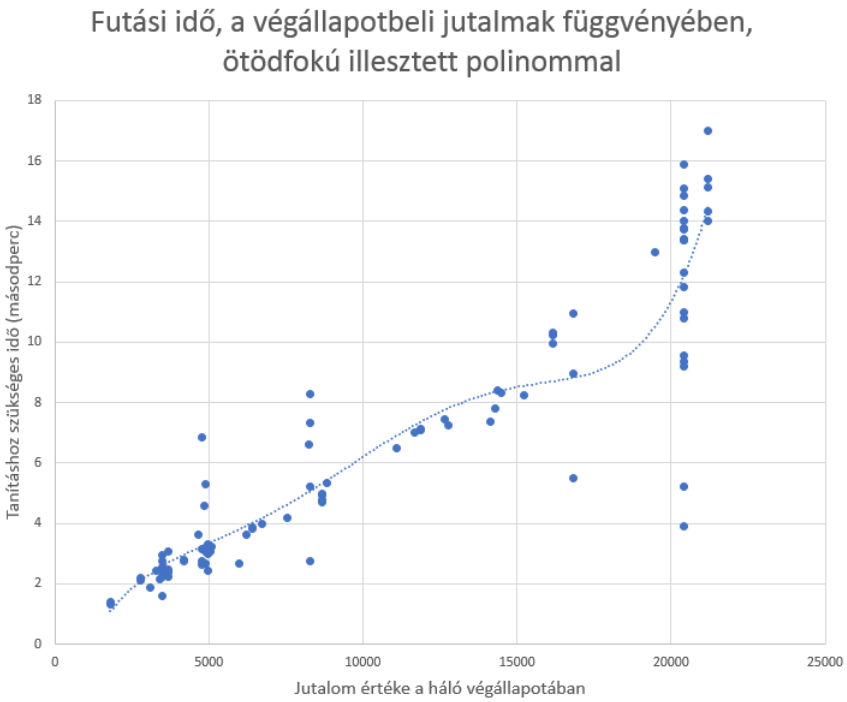
Ezen ábrán jól látszik, hogy bár a tanításhoz szükséges idő az esetek többségében nő a jutalom értékének növekedésében, ez nem mindig igaz. Látható például az ábrán két eset, ahol a tanítási idő viszonylag alacsonyan marad annak ellenére, hogy a jutalom értéke második legmagasabb értékek között van. Az ilyen és ehhez hasonló eseteket a jövőben még szeretném mélyebbre hatóan is vizsgálni, mivel potenciálisan segítséget nyújthatnak a háló inicializálásának jelenleginél jobb megválasztásához.



5.7. ábra. A futtatások sorrendje szerint csoportosított tanítási idő értékek, egy adott hálóra. Az ábra azt mutatja, hogy a tanításhoz szükséges idők értékei nem függnek a tanítások időbeli eloszlásától (ezt pontosabb mérés esetleg cáfolhatja). Jól látszik az ábrán, hogy a tanításhoz szükséges idő másodperces nagyságrenden belül mozog ezen modell esetén, azaz a tanítás rendkívül gyors.



5.8. ábra. A tanításhoz szükséges idők értékeinek csoportosítása idő-intervallumok szerint, egy adott hálóra. Ezen ábra összehasonlítása az 5.4. ábrával jól mutatja, hogy a tanításhoz szükséges idő és a teljesítmény eloszlása nem feltétlenül hasonló.



5.9. ábra. A tanításhoz szükséges idő értékeinek csoportosítása a jutalom értékek függvényében egy adott hálóra. Ezen ábra azt mutatja, hogy a tanításhoz szükséges idő nem feltétlenül egyenesen arányos a háló teljesítményével. Az ábra könnyebb értelmezhetőségének érdekében a pontokra ötödfokú polinomot illesztettem

5.2. Jövőbeli lehetőségek

A fentebb leírt méréseken kívül más, nagyobb szabású kísérletek végrehajtását is tervezem. Ezen kísérletek jelentősebb módosítások lennének a jelenlegi modellen, érinthetik a háló struktúráját, a tanítás eljárását, és talán a környezeti paramétereket is (a környezeti paraméterek módosítását elkerülnél, amennyiben ez lehetséges). A módosítások célja elsősorban a háló teljesítményének növelése lenne, de cél lehet még a háló optimalizálása különböző szempontok szerint.

5.2.1. Jelenlegi háló teljesítményének növelése

A jövőbeli lehetőségek közül ez a célkitűzés talán a leginkább aktuális. A modell teljesítményének növelése számomra jelen állapotban elsősorban azon idő növelését célozza meg, ameddig a modell képes állva maradni. Ezen probléma megoldására több, koncepciójában eltérő elképzelésem is van.

Egyik lehetséges megoldás a háló értékeinek véletlenszerű, kis mértékű elállítása, tanítás közben. Ez a módosítás a lokális minimumokból való kijutását javíthatja a rendszernek, ugyanakkor szignifikánsan le is ronthatja a háló teljesítményét.

Az előbbihez némiképpen hasonló megoldás lehet a felfedezés-optimalizálás arány változtatása (melyről a 2.4. alfejezetben írtam részletesebben). Ezen módosítás esetén valószínűleg a felfedezésre szánt lépések számának növelése vezethet jobb eredményre, mivel így a háló két optimalizálási lépés között sokkal több időt tölt el a potenciális

megoldások bejárásával, ezzel elősegítve a lokális minimumokból való kijutást.

Lehetőség van a környezeti paraméterek véletlenszerű változtatására is az iterációs lépések között, így a háló egyszerre több feladatot kell, hogy elvégezzen, vagyis kisebb az esélye a túltanulásnak. Ez természetesen azt is eredményezheti, hogy a háló sok esetben képtelen az egyik vagy másik, esetleg mindkét feladatot elvégezni, ezzel jelentősen rontva a tanítások sikerességének arányát.

Jó megoldást jelenthet esetleg a legjobb teljesítményt jelentő hálók végállapotát használni egy új futtatás inicializálásaként, mivel így a még viszonylag nagy lépésközzel induló Adam optimizer (melyről a 2.4. alfejezetben írtam) képes a lokális minimumból kihozni a rendszert (az inicializálásként használt tanítás valószínűleg egy lokális minimumban akadt el). 2.4 Ezen lehetőségeken kívül még rendkívül sok megoldás elképzelhető a háló tanítására, viszont a közeljövőben én valószínűleg a fentebb leírtakat kísérelem meg implementálni. A háló teljesítményén ezeken a megoldásokon kívül valószínűleg rendkívül sokat javítanának a valamilyen módon emberi tapasztalatokkal támogatott tanítási eljárások (például ha minden ízületre beállítanék egy olyan optimális értéket, ami szerintem a probléma megoldásához vezetne, és a jutalom függvényt ennek arányában írnám fel). Azonban ezen eljárásokat mindenképpen szeretném elkerülni, mivel a jelenlegi modell egyik legjobb tulajdonsága, hogy abszolút "magától" tanul, anélkül, hogy "látta volna, hogyan kell megoldani a feladatot", így ezen modell potenciálisan a valós (például a Földön uralkodó) fizikai körülményektől jelentősen eltérő helyzetben is helytállhat.

5.2.2. Háló optimalizálása erőforrások és futásidő szerint

Optimalizálásra minden háló esetében van lehetőség, azonban a 4.5.1. alfejezetben leírt okokból eddig nem ezt éreztem a legfontosabb feladatnak. A közeljövőben továbbra sem tervezek a háló optimalizálására vonatkozó módosításokat megvalósítani, mivel a jelenlegi rendszer teljesítményének növelését fontosabb problémának tartom.

A háló optimalizálására azonban – ha sikerül a modellt megtanítani állni, járni vagy egyéb mozgásmintákat megvalósítani – mindenképpen szeretnék időt szakítani. A háló optimalizálása jelentheti a neuronok számának csökkentését, vagy akár hardwares gyorsítást is (például GPU használatát).

A háló optimalizálásának problémáján eddig még nem gondolkodtam sokat, valamint ilyen témájú szakirodalmat is kéne még olvasnom, mielőtt hasonló jellegű problémákat meg tudnék oldani, így erről a témáról itt nem írok részletesebben.

5.2.3. Más felépítésű hálók alkalmazása

Még a háló szerkezetének felírása előtt felmerült bennem több megoldási módszer, ahogy az emberi motoros pályák működéséhez még inkább hasonlító hálót fel lehetne írni. Ezen megoldások mögött a motiváció az, hogy az emberi motoros pályák valóban képesek megoldani az állás problémáját (sőt sokkal bonyolultabb mozgáskoordinációkat is), így a jelenleginél pontosabb modellezésük a jelenleginél potenciálisan lényegesen jobb eredmények eléréséhez is vezethet.

A legtöbb ilyen jellegű elgondolásom már gondolatban megbukott (nagy részük matematikai problémába ütközött, és szinte biztosan taníthatatlan lett volna), azonban egy modell megvalósításának gondolatát mind a mai napig nem vetettem el.

Ezen megoldás egy külön háló felépítését jelentené, mely a jelenlegi háló kimenetét és saját maga előző (vagy régebbi) iterációs lépésbeli kimenetét fogadná bemeneteként. Ezzel az emberi kisagy viselkedését tervezem nagyon leegyszerűsített módon modellezni, mivel a kisagyban is találhatóak körkörösen visszacsatolt sejtcsoportok, ahogy ezt a 2.1. fejezetben részletesebben is leírtam. Egy ilyen modell potenciálisan képes lenne időben periodikus folyamatok felismerésére, és ezen információ felhasználására, a bemeneti jel javítása érdekében.

5.2.4. Tanítási eljárás leváltása

A háló tanítását jelenleg végző eljárásnak több jelentős hátránya is van. Ezek közül talán az egyik leginkább problémás az az a tulajdonsága, hogy (legalábbis eddigi tapasztalataim szerint) rendkívül könnyen konvergál az állapottérben található lokális minimumokhoz.

Ezen probléma megoldására a 5.2.1. alfejezetben már írtam megoldásokat. Ezen megoldások azonban inkább kisebb módosítások lennének. Amennyiben ezen megoldási módszerek nem vezetnek jelentős sikerhez, úgy a hálót tanító eljárás teljes kicserélése is lehet egyfajta megoldás. Ezen megoldást természetesen szeretném elkerülni, amennyiben lehetséges, valamint ha úgy látszik, muszáj alkalmaznom, akkor is mindenképpen maradnék a megerősítéses tanulás módszeréhez tartozó eljárásoknál.

Egyik lehetséges jó út a Q learning alkalmazása (melyről a 2.5. fejezetben írtam). Azonban ezen módszernek talán a jelenleginél is súlyosabb hibái vannak, így használata egyáltalán nem biztos, hogy sikerhez vezetne.

Ezen kívül vannak más megerősítéses tanulást megvalósító eljárások, melyeket a jövőben tervezek megismerni a hozzájuk tartozó szakirodalomból, így ezekről itt nem tudok írni.

5.2.5. Járás tanítása

Járás tanítása jelen pillanatban nem tartozik a közeljövőben megvalósítandó céljaim közé, azonban amennyiben sikerül megtanítani a jelenlegi modellt hosszú ideig állni (képessé válik akármeddig állva maradni, amennyiben nem fogyhat el a rendszer energiája), valószínűleg a járás tanítása válik következő nagy szabású célommá.

Járás tanulására akár egy állásra képes modell is alkalmas lehet. Legegyszerűbb megközelítésként a jutalom függvény kisebb módosítása is mozgásra bírhatja a modellt, ehhez mindössze az szükséges, hogy a modell fejének elvárt pozíciója vízszintes irányban elmozduljon (ezt a lehetőséget a 4.5.4. fejezetben már említettem). Potenciálisan egy ilyen egyszerű módosítás is hozhat nem várt, jó eredményt, azonban valószínűbbnek tartom, hogy az állás tanítása után (amennyiben azt sikerül megvalósítanom) a járás tanítása egy még sokkal nagyobb kihívást jelentő feladat lesz.

6 Összefoglalás

Összegezve az általam eddig elért eredményeket, a kiírt feladatokat sikeresen teljesítettem. Azaz elkészítettem egy, a Newtoni fizikán alapuló modellt, mely képes az emberi mozgásmintákhoz hasonló mozgások megtanulására. Az állás tanítását a modell jelen állapotában még nem mondanám sikeresnek, mivel a modell még nem képes néhány másodpercnél hosszabb ideig megállni, ugyanakkor már sikerült elérnem, hogy dinamikus módon egyensúlyozzon rövidebb ideig (ezen mozdulatok némelyike hasonlóságot mutat az emberi járáshoz vagy más mozdulatokhoz).

A modell, melyet megvalósítottam így sok szempontból még alulmarad a hasonló, ismert modellekhez képest, ugyanakkor bizonyos paramétereiben újszerű módon közelíti meg a mozgás tanításának problémáját, így jövőbeli továbbfejlesztése nem várt eredményeket is hozhat.

Ezen okból mindenképpen szeretném a modell fejlesztését folytatni a továbbiakban is. Valamint amennyiben a jövőben sikerül a jelenlegi környezetben általam megfelelőnek ítélt eredményeket elérnem (azaz a bábu képes hosszú ideig állva maradni), úgy módosított (nehezebb) környezetben is tervezek kísérleteket végezni. Az állás tanításán túl pedig mindenképpen szeretnék próbálkozni majd a járás tanításának feladatával is, melyet ugyanezen modell használatával szeretnék megvalósítani.

Bibliography

[1] E. Todorov, T. Erez, and Y. Tassa, „Mujoco: A physics engine for model-based control", in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, IEEE, 2012, pp. 5026–5033.

[2] *DARPA Robotics Challenge: A Compilation of Robots Falling Down*, https://spectrum.ieee.org/automaton/robotics/humanoids/darpa-robotics-challenge-robots-falling, Accessed: 2018-11-25.

[3] *Towards a Virtual Stuntman*, https://bair.berkeley.edu/blog/2018/04/10/virtual-stuntman/, Accessed: 2018-11-25.

[4] D. A. Rosenbaum, *Human motor control*. Academic press, 2009.

[5] F. L. Golla and S. Antonovitch, „The Relation of Muscular Tonus and the Patellar Reflex to Mental Work", *Journal of Mental Science*, vol. 75, no. 309, pp. 234–241, 1929. DOI: 10.1192/bjp.75.309.234.

[6] J. N. Sanes and J. P. Donoghue, „Plasticity and primary motor cortex", *Annual review of neuroscience*, vol. 23, no. 1, pp. 393–415, 2000.

[7] R. L. Buckner, F. M. Krienen, A. Castellanos, J. C. Diaz, and B. T. Yeo, „The organization of the human cerebellum estimated by intrinsic functional connectivity", *American Journal of Physiology-Heart and Circulatory Physiology*, 2011.

[8] J. C. Eccles, *The cerebellum as a neuronal machine*. Springer Science & Business Media, 2013.

[9] T. Reich, S. Lindstedt, P. LaStayo, and D. Pierotti, „Is the spring quality of muscle plastic?", *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, vol. 278, no. 6, R1661–R1666, 2000.

[10] A. E. Moyer, „Robert Hooke's Ambiguous Presentation of" Hooke's Law"", *Isis*, vol. 68, no. 2, pp. 266–275, 1977.

[11] J. Gundlach, S. Schlamminger, C. Spitzer, K.-Y. Choi, B. Woodahl, J. Coy, and E. Fischbach, „Laboratory test of Newton's second law for small accelerations", *Physical review letters*, vol. 98, no. 15, p. 150 801, 2007.

[12] M. Nielsen, „Using neural nets to recognize handwritten digits", *Neural Networks and Deep Learning*, 2015.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, „Rethinking the inception architecture for computer vision", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[14] F. Rosenblatt, „The perceptron: a probabilistic model for information storage and organization in the brain.", *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[15] K. He, X. Zhang, S. Ren, and J. Sun, „Delving deep into rectifiers: Surpassing human-level performance on imagenet classification", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, „Dropout: a simple way to prevent neural networks from overfitting", *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, „Differential evolution training algorithm for feed-forward neural networks", *Neural Processing Letters*, vol. 17, no. 1, pp. 93–105, 2003.

[18] D. P. Kingma and J. Ba, „Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[19] G. Tesauro, „Temporal difference learning and TD-Gammon", *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[20] R. Henstock, *The general theory of integration*. Clarendon Press Oxford, 1991.

[21] *An introduction to Reinforcement Learning*, `https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419`, Accessed: 2018-11-18.

[22] S. W. Wilson *et al.*, „Explore/exploit strategies in autonomy", in *Proc. of the Fourth International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, vol. 4, 1996, pp. 325–332.

[23] C. J. Watkins and P. Dayan, „Q-learning", *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[24] *Diving deeper into Reinforcement Learning with Q-Learning*, `https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe`, Accessed: 2018-11-18.

[25] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, „Policy gradient methods for reinforcement learning with function approximation", in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[26] *An introduction to Policy Gradients with Cartpole and Doom*, `https://medium.freecodecamp.org/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f`, Accessed: 2018-11-18.

[27] *Simple Reinforcement Learning with Tensorflow: Part 2 - Policy-based Agents*, `https://medium.com/@awjuliani/super-simple-reinforcement-learning-tutorial-part-2-ded33892c724`, Accessed: 2018-11-23.