Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Távközlés és Médiainformatika Tanszék

Mesterséges neurális hálók modelljének javítása valószínűségen alapuló módszerrel

TDK DOLGOZAT

Készítette

Havlik Marcell

Konzulens

Barta Gergő

BUDAPEST, 2016

Tartalomjegyzék

Összefoglaló

Egy új módszert kutatok gradiens alapú gépi tanuló rendszerek modellillesztésének éles környezetben történő alkalmazásának javítására. Amire azért van szükség, mert a gépi tanuló rendszerek esetenként több millió paramétert használnak egy-egy rendszer modellezésére, amely miatt képesek olyan összefüggéseket megtalálni, amelyek csak az adott adathalmazon érvényesek, azaz az általános összefüggések helyett a tanító adathalmaz sajátosságaira illeszteni. Ez hosszú távon a neurális hálózat teljesítményének és hasznosíthatóságának radikális csökkenéséhez vezet.

Az általam bemutatott stratégia egy lehetséges módszert ad a hibás paraméterek automatizált szűrésére heurisztika segítségével. A modell optimalizálására használt technika, automatizált módon képes a neurális hálókban létező rossz összefüggések meghatározására, majd generációs módon a modell szűkítésére, ilyen módon a modellillesztés hatékonyságának növeléséhez és gyorsításához vezet.

A kutatott módszer végeredményben lehetőséget ad egy ismeretlen összefüggéshalmaz feltérképezésére azáltal, hogy a túlparaméterezett modellt leszűkíti a hibásnak értékelt paraméterek kiszűrésével, amelyre jelenleg kevés effektív módszer létezik. A folyamat végeredményeként egy olyan összefüggéshalmazt kapunk, amely ugyan olyan jól illeszkedik a tanulásra szánt adatsorhoz, azonban kevesebb paraméterrel valósítja meg azt. Ennek megfelelően az új modell védett a túltanulás („overfitting) ellen és jobb eredményt ér el éles adathalmazon.

Abstract

During my thesis, I develop a new method, that is able to filter models used by the gradient based machine learning algorithm. It makes the model more effective by keeping only the functions, those converge to the ideal model and dropping the other correlations which only fulfill the rules of the actual dataset. It is useful for example when the model has more than millions parameters and we can't develop futher because of the capacity limits.

It is a generation based algorithm that observes the learning process and it makes evaluations about the parameters whether they belong to the model or not. This is a possible solution to automatize the regularisation process of the model.

It is possible to use it in research, with the purpose of revealing new patterns in an unknown dataset. By filtering the model it opens up a new chance to improve searches to find more pattern in the dataset.

This method is an initiation of fully automatizing machine learning process. Automatizing the model generation and filtering process could open up the world for computers to help us understand much more coherences in the world than today.

1. Bevezetés

A dolgozat középpontjában a mesterséges neurális hálók valós körülmények között történő alkalmazásának javítása áll. Ezt a modell effektívebb illeszkedésével érem el. Ugyanis a gépi tanulás során gyakori probléma, hogy az elért eredmény nem működik éles környezetben és ezt ad-hoc módon kell manuálisan javítanunk. A módszer egy lehetséges módszert ad a gépi tanuló algoritmusban használt modell eredményeinek automatizált javítására, hogy az laboratóriumi körülmények mellett végzett tesztelés után is megfelelően jó eredményt produkáljon a termék éles környezetben történő alkalmazásakor.

Korunkban exponenciális gyorsulást tapasztalhatunk a szellemi területeken történő automatizálás eredményeként. Ami az egyre növekvő mennyiségű szenzornak és a gépi tanuló algoritmus alkalmazásának köszönhető. Egyre több információt rögzítünk a körülöttünk lévő világról, amely a gép számára is feldolgozható formában jelenik meg azonban emberi képességeinket már meghaladja. Az így létrejött adathalmaz segítségével az emberiség számára meghatározó új megállapításokra tehetünk szert mind a fizika, biológiai, pénzügyi folyamatok, egészségügy és még számos más területen. Ezeknek a lehetőségek feltárására és fejlesztésére jött létre egy új ágazat az adatbányászat.

Adatbányászat az adatokban rejlő információk kinyerésével foglalkozik. Ez 2016-ban gépek segítségével félautomatikus módon történik, leggyakrabban gradiens alapú módszerek segítségével. A terület azért vált fontossá, mert üzleti potenciál is rejlik benne, ugyanis az emberi elemzéseknél sokkal nagyobb adathalmazokon bonyolultabb összefüggések felfedezésére ad lehetőséget, melyek felhasználhatók számtalan új technológiai vívmány megalkotásában, illetve pontosabb becslések vagy optimálisabb döntések meghozatalában.

1.1 Téma értelmezése

A cím „Mesterséges neurális hálók modelljének javítása valószínűségen alapuló módszerrel”. Kifejtve, célom; egy olyan módszer kifejlesztése, amely képes hatékony modellillesztésre. Egy olyan modell architektúra megalkotására, amely kizárja a nem megfelelő paramétereket a rendszerből. Nem megfelelő paraméter alatt, a túltanulást okozó paramétert kell érteni. A jobb megértés érdekében tisztázom, hogy mit nevezek, modellnek az értekezés során. A modell, egy paraméterekkel és műveletekkel leírt összefüggéshalmaz, amely differenciálható egyenleteket határoz meg, ezáltal leírja a bemenetek hatását a kimenetre. Egy egyszerű példa a modellre: P(x,y,z) jelöli a test elhelyezkedését a térben, és ismert a V(t) sebesség függvény és D(t) függvény mely egy egységvektor, amely minden időpillanatban megadja az aktuális irányt, ekkor a V és D függvények segítségével képesek vagyunk leírni a P(x,y,z) függvényt, ebben a konkrét esetben $\int V(t)\, D(t)\, dt$ képlettel. Ekkor a $P(t) = P(t - dt) + V(t)\, D(t)$ lesz a rendszert leíró modell.

1.2 Felhasználhatóság, indokoltság

Mesterséges neurális hálók használata során gyakori probléma, hogy több millió paraméterrel rendelkező parciális egyenletrendszerrel bíró optimalizációs problémára keresünk megoldást. Egy ilyen kutatás során nincs megfelelő figyelem a rendszer túlparaméterezésének kezelésére. Az általam fejlesztett módszer egy automatizált megközelítést ad erre a problémára.

A túlparaméterezés ellen történő kezdeményezéseknek számos hasznuk van. Amennyiben képesek vagyunk, a paraméterek számát csökkenteni, kevesebb paraméterrel kell számolni, amely ennek megfelelően kevesebb memóriát és számításikapacitást vesz igénybe melynek következtében az időigény nagymértékben csökkenthető. Backpropagation és forwardpropagation algoritmus[1] használatát feltételezve neurális hálózatokon ez lineárisan kevesebb memória és számításikapacitást jelent. A paraméterszám csökkentés esetenként 10 millió paraméter helyett 1 000 000, 100 000 vagy akár még kevesebb paraméterre szűkítheti a rendszer méretét. Ez nagymértékben függ a kiindulási modellünk felépítésétől és az adathalmaztól, amelyet vizsgálunk.

A probléma orvoslása indokolt, és amennyiben létezik rá jó módszer, egy új távlatot nyithat meg a neurális hálók alkalmazásán belül. A fókusz, annak a legszűkebb modell felépítése, amely tartalmazza a megfelelő paramétereket de nem tartalmazza a rendszer általános leírására nem alkalmas paramétereket. A tudományos diákköri munkám során ezt nevezem el effektív modellnek.

Az összefüggéshalmaz paramétereinek csökkentésével egy eddig létező fejlődési irányban is új lehetőségek jelenhetnek meg. Amelyek azért nem voltak elérhetőek, mert az aktuális modellünk mérete elérte a számítási kapacitásunk végső határait, például 2014-ben 400 millió paraméterrel definiált modellt használtak klasszifikációs probléma megoldására[2].

1.3 Rövid összefoglaló

A bemutatott módszer egy működő prototípus a tanulási folyamat automatizált gyorsítására, optimálisabb paraméterhasználatára, a neurális hálók lehetőségeinek bővítésére és használhatóságának növelésére éles környezetben.

A 2. fejezetben kifejtem a dolgozat megértéséhez szükséges szakmai fogalmakat, felhasznált technológiákat és a módszeremhez hasonló stratégiákat. A 3. fejezetben egy átfogó képet szeretnék adni az algoritmus elvéről, architektúrájáról és működéséről. Kitérek a rendszerben felhasznált eszközökre, illetve a bevezetett metódusokra. 4.-dik fejezetben részletezem a megvalósítást, magyarázatot adok az algoritmus működésére. Végül az 5.-dik fejezetben, mérésekkel bizonyítva bemutatom a kiemelkedő eredményeket, amelyet az algoritmussal értem el, kitérve az egyes eredmények részletes magyarázatára.

Utolsó előtti pontban, a 6. fejezetben, értékelem az algoritmust, leírom saját véleményem és az algoritmussal szerzett érdemeket. Majd legvégül a tovább fejlesztési lehetőségekre írok érdemleges opciókat.

2. Irodalom

A tanuló rendszerek rengeteg algoritmust használnak. A dolgozatom megértéséhez ezen algoritmusok működésének pontos ismerete nem szükséges. Alábbiakban megpróbáltam összegyűjteni azokat az információkat, amelyek azonban jelentősek lehetnek az eljárás értelmezéséhez.

2.1 Háttérismeretek

2.1.1 Gradiens alapú tanuló rendszerek[3]

Ide sorolhatunk minden olyan tanulási folyamatot, amely a paraméterek rendszerben felvett gradiensének értékének felhasználásával csökkenti a költségfüggvény értékét. A tanulandó adathalmaz és az aktuális paraméterállapotokkal a modell hibáját kiszámítva a költségfüggvény értéke kiszámolásra kerül. Majd ezt a hibát csökkentjük azáltal, hogy minden paraméterre képesek vagyunk megmondani, hogy a paraméter értékét megváltoztatva az milyen irányú elmozdulást hoz létre a modell kimenetén, ami gyakorlatilag a paraméter adott bemenetének az elsőfokú derivált kiszámításával kapható meg. Következésképpen elmondható, hogy a gradiens alapú rendszerek mohó módon változtatják a paramétereiket olyan irányba, hogy csökkentsék a költségfüggvény értékét, tehát egy optimalizációs algoritmusról van szó. A korunkban ismert leggyorsabb algoritmusok melyeket a hiba és gradiens kiszámítására használunk neurális hálókon a *forwardpropagation (2.1.3)* és a *backpropagation (2.1.4).*

2.1.2 Mesterséges neurális hálók

A neurális hálózat egy „neuronokból” és összeköttetések hálójából álló, irányított gráfként ábrázolható struktúra. A rendszerben minden neuron egy művelet, amely a bemenet függvényében egy kimenetet produkál. A kimenet értékét meghatározza a bemenet és a bemenetek függvényében definiált művelet[4][5].

Végletekig egyszerűsítve, a neurális háló egy olyan műveletet leíró rendszer, amely bizonyos bemenetekre egzakt kimenet(et/eket) produkál.

A kimenet függvényében megkülönböztetünk klasszifikációs vagy approximációs neurális hálózatot. Az értekezésemben nem jelent számottevő különbséget, mivel mind a kettő esetében ugyanúgy alkalmazható az általam kifejlesztett algoritmus. Ha nagyon figyelmesen vizsgáljuk, akkor az egyetlen különbség, hogy klasszifikáció esetén nagyobb eséllyel megjelenhet a „*vanishing gradiens*” és az „*exploding gradiens*” probléma, azonban ezekre nem térek ki külön.

2.1.3 Forwardpropagation

A bemenetek és paraméterek függvényében a rendszer kimenetének meghatározása, mely alapján kiszámítható a hiba. Triviális módon, a bemeneteket egyesével végig

terjesztjük az egyes műveleteken (neuronokon), és így hierarchikusan felépítve eljutunk a kimenethez, amely a bemenetek függvényében a rendszer válaszát adja meg.

2.1.4 Backpropagation

Egész pontosan „backward propagation of error”. A hiba visszafelé terjesztése a rendszerben. A hiba minimalizálásáért van rá szükség. Az algoritmus segítségével minden paraméterre kiszámítható, hogy azoknak milyen irányú és mértékű elmozdulásra van szükségük a rendszer aktuális állapotában, hogy a kimeneti hiba csökkenjen. Ennek algoritmikus működése; a gradiens segítségével minden neuron kimenetének a hibája és a neuron által végzett művelet megadja a neuron bemenetének szükséges változtatásának irányát és értékét, amit változtatni kell a paraméterein, hogy a neuron csökkentse a hibát a rákövetkező neuronokra nézve. A kimenet felől a bemenetek irányába rekurzív módon végig játszható az algoritmus, így kapjuk meg a paraméterek változtatásához szükséges értéket.

A memória igénye kétszerese a forward propagationnak, azonban a paraméterek számával így lineárisan skálázódik.

2.1.5 Gradiens alapú optimalizációs algoritmusok

A rendszer paramétereinek változtatása szintén nem triviális feladat, hiszen csak a gradiensek értékével való elmozdulások támogatása nagyságrendekkel több iterációt igényelhetnek, mint egy szabályozás által meghatározott paraméteridentifikáció. Ezeknek az algoritmusoknak segítségével gyorsabban tudjuk meghatározni a megfelelő paramétereket. Ilyen az *Adam momentum, Adagrad, Nesterov,* vagy *Adadelta* [15], de még sok másik algoritmus létezik, a felsoroltak méltán a legfelkapottabbak korunkban. Ezeket implementáltam a publikációik alapján azonban a terjedelemkorlátozásra való tekintettel nem térek ki részletesebben.

2.1.6 Regularizáció

A neurális hálók esetében, gépi tanulásban használt regularizáció alatt szabályozást értünk, ami a túltanulás és hibás tanulás mértékének csökkentésére próbál megoldást találni. Ez több módon lehetséges, mint például a paraméterek számának csökkentése, a paraméterek értékének korlátozása, vagy a paraméterek „*jelentőségének*” csökkentése. A módszer, melyet bemutatok az értekezésem során, szintén egy regularizáció, amely a paraméterek számának csökkentését tűzi ki célul.

2.2 Hasonló munkák

Az általam kutatott eljárás a gépi tanulás területén még újnak számít. Nem találtam olyan megközelítést, amely teljes mértékben egy kategóriába sorolható vele, ami azt jelenti, hogy nem találtam olyan algoritmust, amely az összefüggések számát próbálja csökkenteni, későbbiekben kifejtem a különbségeket.

Egy távolabbi kapcsolódási ponton azonban lehet vizsgálni a hasonlóságot, a regularizációs algoritmusokkal. Ugyanis ezek szintén egyfajta regularizációt hoznak be a modellbe. Alábbiakban ismertetem ezeket a regularizációs megoldásokat.

2.2.1 L2 regularizáció

A legegyszerűbb regularizációs formula. A paraméterek nullától való eltérésének értékének négyzetével „bünteti" a paramétereket, ami a költségfüggvényben jelenik meg. Minden súly paraméter ω esetén hozzáadunk a kimenethez egy $\frac{1}{2}\lambda\omega^2$ ahol λ a regularizáció mértéke. Az $\frac{1}{2}$ konstans osztás oka, hogy a regularizáció deriváltja $\lambda\omega$ és nem $2\lambda\omega$. Az L2 regularizációnak egy intuitív megközelítése, a kiugró paraméterértékek büntetése.[8][9]

A regularizációnak van hatása a költségfüggvényre, ezért kis eséllyel új lokális minimumok keletkezhetnek. Egy mai publikáció szerint amennyiben ez történik meg, akkor ezek a lokális minimumok közel vannak a globális minimum értékéhez. [10]

2.2.2 L1 regularizáció

Hasonlóan az L2-regularizációhoz, a súlyok büntetve vannak a rendszerben viszont most lineáris módon $\lambda\,|\omega|$. Lehetőség van a kettő regularizáció kombinálására, $\lambda_1\,|\omega| + \frac{1}{2}\,\lambda_2\,\omega^2$ képlettel a költségfüggvényben ezt Elastic net regularizációnak is nevezik[9].

Az L1 regularizációnak egy komoly hátránya, hogy a súlyok nullázására törekszik, amely hatására ritka mátrixok jönnek létre, amelyek többet nem mozdulnak el a 0 állapotból, még ha nem 0 az optimális sem. A nulla súlyok hatására az új rendszer csak részhalmaza az eredetinek. Például érzéketlenné válik a rendszerben zajos bemenetekre[9] ezen kívül fellép a vanishing gradient probléma, ami a közel nullás szorzások egymásutánja miatt elnyeli a gradienst így stagnálni fog a modell.

Másik negatív tulajdonsága, hogy a regularizáció hatással van a költségfüggvényre. Így L2-höz hasonlóan kis eséllyel elakadhat a minimumkeresés lokális minimumokban.

2.2.3 Maximális normalizációs megkötések

Másik regularizációs formula a paraméterek szélső határértékekkel történő korlátozása és gradient descent módosítása, hogy valóban ne lépjenek túl a határértékeken. Gyakorlatban legtöbbször úgy néz ki, mint egy normális paraméterfrissítés. Iterációk vannak, és minden iteráció után végzünk egy levágást, hogy a súlyparaméterek valóban $c_{alsó} < \omega < c_{felső}$ korlátok között legyenek. Vannak feljegyzések, hogy ezzel a módszerrel olykor javítottak az eredményeken. Ez feltételezhetőleg annak köszönhető, hogy a rendszer nem képes „felrobbanni" mert nagy tanulási ráta esetén is korlátok közé van kényszerítve. [9]

Nagy hátránya, hogy a paraméterek határértékeit az embernek kell meghatároznia, amelyről legtöbb esetben nincs megfelelő ismeretünk így fennáll a veszélye annak, hogy

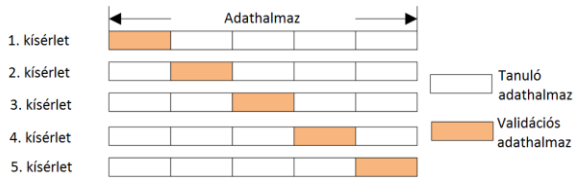
olyan helyzetbe hozzuk a tanuló rendszert, amely egy jelentősen rosszabb lokális minimumban akad el, mint az optimum.

2.3 Technológiák

2.3.1 Keresztvalidáció

A tanító adathalmazon történő részleges tanítás és validációnak folyamatos körbeforgó változtatásán alapuló modell validációs technika. Nem kimondottan regularizációs technika mert lényege, alapvetően a tanulásra felhasznált adathalmaz kibővítése a validációs adathalmazzal.

A kibővítés a validációs adathalmaz felcserélése a tanulásra használt adathalmaz egy részhalmazával történik. Itt több különböző módszert nevez meg az irodalom, megkülönböztető jegyük a tanulásra használt adathalmaz és a validációs adathalmaz változtatásának a módja. Legelterjedtebb módszer, az adathalmaz 5 részre osztása, amelyből 4 „train” és 1 „validation” adathalmazt választ ki, majd a tanulás során ezt folyamatosan cserélgetjük. A felcserélés költségének függvényében érdemes meghatározni, milyen sűrűen végzünk cserét. [12]



A módszer erőssége, hogy a modell sokkal nehezebben tanul rá egyes modellek egyediségére, mivel folyamatosan változik a tanulásra használt adathalmaz. Konklúzió, a rendszer általános összefüggéseit nagyobb eséllyel fogja megtalálni az algoritmus.

Számomra azért van jelentősége a „cross validation”-nek, mert egy kis módosítással az én algoritmusomon is lehet használni.

Korlátja az algoritmusnak, hogy rekurziót tartalmazó neurális hálók esetében nem használható, ami azt jelenti, hogy parciális egyenleteket tartalmazó összefüggéshalmazok esetén nem használható. Véleményem szerint ahhoz, hogy ez megvalósítható legyen az RNN hálózatok adatsoraival is, ahhoz az adathalmaz felosztását szigorítani kell, de egyelőre ilyen megoldás létezésére nem találtam publikációt[11].

2.3.2 TensorFlow

A TensorFlow[13]**Hiba! A hivatkozási forrás nem található.** Google által fejlesztett adatfolyamokon alapuló *„state of the art”* gépi tanuló algoritmusokat tartalmazó nyílt forráskódú szimbolikus programozást megvalósító függvénykönyvtár. Ezt használom fel, mert rengeteg algoritmus előre implementált módon megtalálható benne, és **jól skálázódnak** több magos processzoron, több videókártyán vagy akár elosztott rendszerek környezetében is.

Jelenleg rendelkezésre álló TensorFlow 0.11, támogatott „interface”-k:

- Python API (ezt használom)

    - C++ backend a háttérben
    - Interpretált programozást tesz lehetővé
    - Dinamikus RNN-t is képes kezelni
- C++ API
    - Modell készítés még nem támogatott

Az elérhető függvénykönyvtárak közül az egyik leggyorsabb fordítási idővel rendelkezik. Linux/Mac támogatás, emellett Android OS-en is használható.

Jelenleg leggyorsabban fejlődő keretrendszer.

Használja:

- *CUDnn*: nVidia által fejlesztett NN függvénykönyvtár.

Konkurens keretrendszerek:

- Theano
- Caffe
- Torch
- Baidu's Warp-CTC
- Microsoft CNTK (Computational Network Toolkit)
- Microsoft CNTK (Cognitive Toolkit) (2016. 10. 25.)

A Google által közzétett publikációk hetes késéssel bekerülnek a könyvtár függvényei közé, amelyek nem ritkán áttörések a gépi tanulás területén (például: WaveNet [14]).

3. Az elvégzett munkáról

Ebben a fejezetben szeretnék egy átfogó képet adni az algoritmusról, a jelenlegi alternatív megoldásról, amit ma végzünk el helyette manuálisan és az eljárás technikai hátteréről, indokokkal együtt. Majd az implementáció igénye nélkül bemutatni, hogyan épül fel és fog működni a bemutatott módszer.

3.1 Algoritmus

Az eljárás, túltanulás ellen nyújt védelmet mesterséges neurális hálókon. Túltanulásnak nevezzük azt a folyamatot, amikor a rendszer olyan összefüggéseket talál meg az adathalmazokon, amelyek csak az adott vizsgált adathalmaz sajátos elrendezésében jelennek meg, és nem érvényes a vizsgált rendszerre általános formában.

Az algoritmus működése azon alapszik, hogy a rendszer folyamatosan analizálja a validáción és a tanító adathalmazon elért hibát és ezt visszacsatoláson keresztül felhasználja a paraméterek „hasznosság” értékének meghatározására. Az ötlet lényege, hogy a validáción és tanulóadatsoron elért hiba és a paraméterek változásának értékének függvényében kiválasztjuk azokat a paramétereket, amelyek nagyobb valószínűséggel vesznek részt, vagy jobban közelítik az ideális modellt, amely az általános összefüggéseket tartalmazza a megfigyelt rendszerről. Hogy miért a legvalószínűbb paraméterek kerülnek kiválasztásra a 3.6-os pontban van kifejtve. Teljes biztonsággal a legjobb paraméterek kiválasztására nem ismert jól skálázódó megvalósítás, mert nem ismerjük a pontos összefüggéshalmazt és exponenciálisan sok összefüggés létezhet egy adott adathalmaz esetében. Ezért az általam bemutatott módszer egy heurisztika a legjobb paraméterek megtartására, mely működését leírom és alátámasztom az algoritmus validálásához szükséges mérési eredményekkel.

3.2 Jelenlegi eljárás a gép taníttatására

Amennyiben az adatok tiszták és zajmentesek, ami azt jelenti, hogy nem igényel műveletet a kiugró hibás mérési adatok szűrése az algoritmusba táplálás előtt. Akkor a gépi tanulásban használt modell taníttatása, egy bevett sablon szerint félautomatikus módon történik:

1. Elkészítünk egy modellt, minél több lehetséges összefüggéssel a kimeneti adatok függvényében.
2. A tanulásra szánt adatsoron betanítjuk a modellt.
3. Validáción ellenőrizzük a modell eredményességét
   - X - Nem kielégítő eredményt elérve, kezdjük az 1-es lépéstől.
   - ✓ - Elvárt eredményt elérve 4. pont.
4. Kiválasztjuk a validációs adathalmazon legjobban teljesítő modellt.

5. Éles környezetben való felhasználás.

A megfelelő modell kiválasztásánál figyelembe kell vennünk több szempontot. Ha a validáción elért hiba és a tesztadatsoron elért hiba nagyon különbözik, akkor a modell éles környezetben sem feltétlen az elvártaknak megfelelően fog működni, mert olyan összefüggéseket tanul meg a tesztadatsoron a rendszer, amelyeket az adatok adott sorrendben történő fennállása hozott létre. Ilyen esetek hatása a rendszerre nézve nem kiszámítható pontatlanságot eredményezhetnek éles bemenetek esetén. Ekkor érdemes a tanulásra szánt adathalmaz mennyiségét növelni vagy a paraméterek számát csökkenteni.

Másik probléma a validációs adathalmaz megtanulásából eredeztethető. Az első problémát elkerülve addig kísérletezünk újabb és újabb modellek felállításával, amíg találunk egy olyan összefüggéshalmazt, amely a validáción is jó eredményt produkál. Ekkor előfordulhat az, hogy az összefüggéshalmaz, amelyet felírtunk véletlenül illeszkedik a validációs adatsorra is. Nem ritka, hogy a validációs adatsorra jelentősen jobban illeszkedik a modell. ennek egy lehetséges elkerülési módja, ha van lehetőségünk a validációs adathalmaz növelésére. Illetve Andrew Ng ennek a problémának az elkerülésére olyan lehetőséget mutat[17], hogy egy második validációs adatsort használjunk, amely alapján kiválasztjuk mely modellek a legjobbak a validációs adathalmazon. Ezt a lépés a fentebb felírt 3-mas és 4-es eset közé kell beiktatni. A modellt javítjuk kísérletezés segítségével úgy, hogy: Új összefüggéseket írunk fel a modellben, ha javul az illesztés a validáción, akkor az új összefüggéseket elfogadjuk, ha romlik, akkor kivesszük belőle. Következő kísérlet esetén megint újabb összefüggéseket írunk, fel majd veszünk ki ez így megy, amíg megfelelően jó eredményeket érünk el mind a két adathalmazon.

Fentebb leírtak alapján érthető miért történik egy indirekt tanulás a validációs adathalmazra. Ezért válhat szükségesnek például egy harmadik adatsor a végső döntés meghozatalához.

3.3 Problémák

Az általam használt kutatásokban olyan korlátokba ütköztem, ahol hiába adtam meg nagyobb modellt akár algoritmikusan, az így legenerált modell, amely több 10 ezer paraméterből állt erősen hajlamos volt a túltanulásra, és míg a tanító adatokon olykor 60%-os illesztést értem el a validáción már csak 51-53%-ot ért el igaz-hamis döntési probléma esetében. Ezért jött az ötlet, hogy szükséges lenne a tanulási folyamat során figyelemmel kísérni, hogy mikor indul el a tanulás a túltanulás irányába, illetve valamilyen módszert meghatározni azon paraméterekről, amelyek nem az általános összefüggéseket írják le.

A tervezés részletes leírása, a választott megoldások indoklása

3.4 Elméleti kiegészítés

Logaritmikus hiba: [22]. A hiba értékelésére számszerűsített mérce, klasszifikációs problémák esetében használható jól. Előnye, hogy jól párhuzamosítható a számítása:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\mathrm{p}_i) + (1 - y_i) \log(1 - \mathrm{p}_i))$$

Hiperparaméter: olyan paraméter, amely nem automatizált módon kerül meghatározásra. Ezekkel finomhangolni lehet a rendszert az adott problémára specifikusan.

A modellben szereplő paramétereknek nem csak értékük lehet. Az optimalizációs algoritmusból adódóan vannak segédváltozók az optimalizáció elérése érdekében, ilyen például a „lendület", „gradiens", vagy az én általam felhasznált „jóság" alparaméter. Fontos megjegyezni, hogy a modell mérete, függ a gradiens alapú optimalizációs algoritmustól. Ez memóriát tekintve lehet jelentős.

Az általam végzett kutatás jelentős kapcsolatban van a függvények elemzésével. Az általam vizsgált modellek kategorizálása céljából hasznos lenne kitérni erre a témára, azonban nem találtam megfelelő területet, amely az általam elvárt szempontok szerint jellemezni tudná a modellek felépítését. Mivel ez nem tartozik a kitűzött feladatom közé, és túl tág téma ráadásul nem találtam megfelelő publikációt, ezért a teljesség igénye nélkül mérésekkel próbálom jellemezni az algoritmust konkrét függvények esetében. A tudományos értekezésem során a modelleket megpróbálom minél átfogóbban jellemezni és rögzíteni.

3.5 Tervezési döntések

Az 3.1 fejezetben ismertetett módszer alapján a tanulás gyakorlatilag két számérték vizsgálata alapján történik. Ha javult a validáció akkor elfogadjuk a javításokat, ha nem akkor nem fogadjuk el. Az erőssége, a lépcsőzetes építkezésnek, hogy tisztán látszik az újonnan behozott összefüggés hatása, és jelentősen kevesebb interferencia lép fel.

Olyan esetben, amikor egy hatalmas generált adathalmaz akartunk leírni több 100 ezer paraméterrel, akkor a „hibás" paraméterek kiszűrése ellehetetlenedik. Ez két okra vezethető vissza:

- Túl sok paraméter esetén, nincs lehetőségünk egyesével eltávolítani/visszarakni és ilyen módon automatizált tesztelést megvalósítani (vagy nem egyesével végezni, mert akkor lehetőség volna valamilyen bináris keresés segítségével ezt javítani de akkor is már jelentős az interferencia).
- A redundancia, előfordulhat, hogy más-más futtatásokban ugyan olyan rossz eredményt kapunk, pedig az adott paraméter kiszűrése egy jó döntés lenne az ideális modellhez mérten. Ez azért van, mert egyes paraméterek redundáns módon képesek leírni ugyan azokat az összefüggéseket. Erre példa: a *gaussian node*, mely képes kijelölni a bemenetek alapján egy teret, így bekategorizálni egy részét a térnek. Ilyen függvények léte miatt a paraméterek változtatásával több is képes ugyan azt a helyet kijelölni, ezért

nem észlelhető, ha egy paraméter tényleg rossz mivel más helyettesítette a hatását a tanulás során.

Ekkor jöhet számításba az általam kidolgozott algoritmus. Nem az utolsó két érték alapján történik a becslés, hanem a tanulás minden egyes iterációjánál megállapítunk egy új jellemzőjét a rendszernek, amely a „*tanulás jó irányba történő elmozdulás fogalma*”. Az algoritmus lehetőséget ad automatizáltan vizsgálni a paraméterek helyességét egy heurisztikán keresztül. Itt is rengeteg interferencia lép fel, azonban a paraméterek súlyfüggvényéből adódóan nagyobb valószínűséggel fogja a rossz paramétereket kiszűrni mint a jókat.

3.5.1 Jó irányba történő elmozdulás

A rendelkezésre álló adataink minden iteráció során:

- paraméterek
    - értékei
    - gradiensei
    - elmozdulása (optimalizációs algoritmus miatt az elmozdulás nem feltétlen a gradienssel egyezik meg.)
    - egyéb (gradiens alapú optimalizációs algoritmus segéd paraméterei, modell felépítése)
- tanuló és validációs adathalmazon elért hiba
- kiindulási értékek

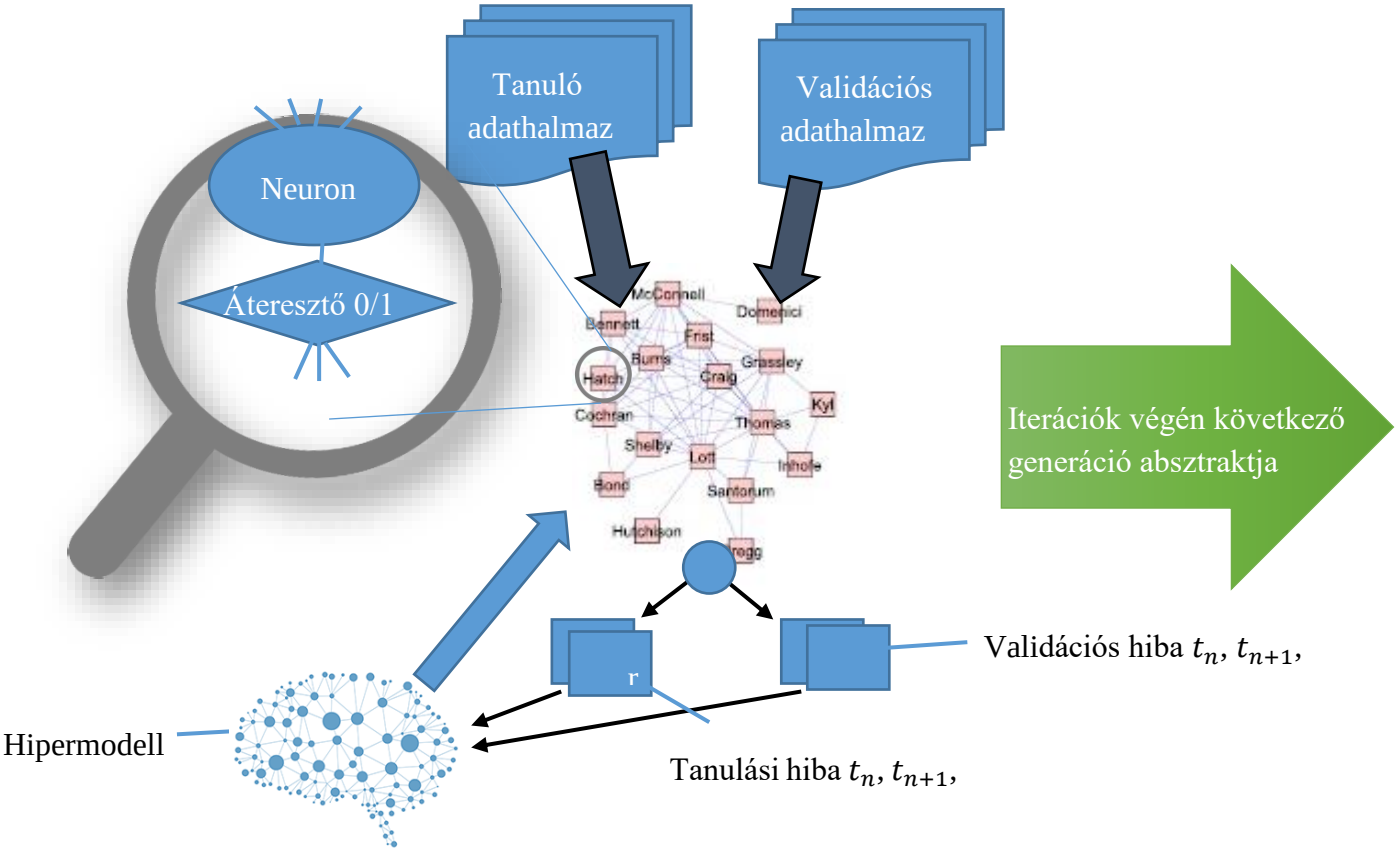
A kérdés, ezek alapján hogyan lehetne mérni a jó irányba történő iterációkat.

Az ötletem azon alapszik, hogy a teszt adatsoron és a validáción történő javulás meghatározza, hogy az aktuális változás az iteráció során mennyire effektív (pontosabban mennyire általános összefüggés tanulását jelenti). Ez alapján meg lehet határozni, egy kettő paraméteres függvény segítségével a „jó tanulás” mértékét. A kidolgozás 3.7 részben kifejtem milyen elvárások vannak a függvénnyel kapcsolatban.

3.6 Rendszer algoritmikus működése

Generációs alapú, tehát az algoritmusom nem avatkozik bele a tanulás folyamatába csak külső megfigyelő. Feljegyezi a tanulás eredményességét és a hibás tanulást eredményező paramétereket minden iterációban, majd ez alapján hozza létre az új modellt, amely a következő generációs modellt jelenti.

Grafikusan ábrázolva. A nyilakkal jelölőm, hogy milyen irányba halad az adat. Egyes ponton mit csinál azt szövegesen fejtem ki. (Vastag nyíl nagy adathalmazt, vékony nyíl egy-egy értéket jelképez.)



3.1. ábra: Az algoritmus adatfolyamának vizualizálása

A 3.1.-es ábra alapján elkülönített főbb egységek:

- Bemeneti adathalmaz (tanuló és validációs adathalmaz)
- Modell
  - Neuronok + áteresztő 0,1 érték
- Kimenet (tanuló, validációs hiba érték)
- HiperModell

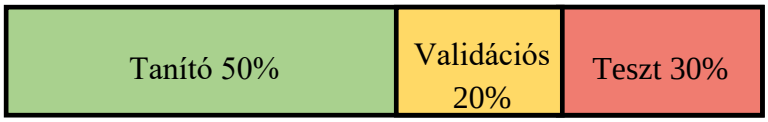
– Új generáció absztraktja (futtatás kimenete, mely a modell egyes paramétereinek „jóságát” tartalmazza)

Az, hogy egyes pontok mit jelentenek, alábbiakban fejtem ki. Implementációs részletek a részletes kidolgozás fejezetnél 3.7 fejezet lesznek kibontva.

3.6.1 Bemeneti adathalmaz

Az adathalmazokat három részre osztom. Ezek a tanításra használt („*train*”) adatsor, a validációra használt („validation”) adatsor és a tesztelésre használt („test”) adatsor. Mind a háromnak más szerepe van.

A 3.1.-es ábrán felül található meg a tanító és validációs adatsor, ez a kettő adatsor kerül felhasználásra az algoritmus taníttatásában. A felosztások a következők:



3.2. ábra: az adatsorok felosztása

A tanulásra szánt adatsorral történik a paraméterek meghatározása. A validációs adatsorral történik a taníttatás effektivitásának ellenőrzése. A teszt adatsor jelenti a végleges validációt, ez adja meg az algoritmus teljesítményét éles környezetben. Ha nincs nagy eltérés a teszt és a tanulásra szánt adatsor hibája között, akkor nincs túltanulás és éles adatsorokon is jól fog teljesíteni a rendszer.

3.6.2 A modell felépítése

A neurális háló minden változója ki/be kapcsolható módon egy konstans mögé van kapcsolva. Így szabályozható, hogy melyik része legyen aktív a modellnek. Az általam aktivációs rétegnek nevezett alparaméterek azt eredményezik, hogy a modell mérete nő a paramétereinek számával. A modell neuronszáma kétszerese az eredeti modellnek (memóriaigénye nem kétszeres), de ez egy konstans 1 vagy 0 szorzást jelent minden neuron után.

Ehelyett lehetne egy optimálisabb megoldást használni, ahol az 1-gyel történő szorzásokat elhagyjuk, és a 0-val szorzott elhalt ágakat kiszedjük a gráfból. De ez most egy prototípus, a nyelvi elemek sem teszik lehetővé a modell futásidejű szerkesztését, és nem hozna más változást az eredményekben, mint a kód gyorsabb futását.

3.6.3 A modell kimenete

Minden iterációnak van egy hibaértéke. Itt meg van különböztetve tanító és validációs adatsor. Attól függően, hogy mikor melyik adathalmazt táplálom be, a modellbe, az eredmények más-más változóban vannak kimentve.

3.6.4 Hipermodell

Az így kiszámolt értékek szerint egy utófeldolgozó rész, a validáció és tanító adatsoron elért hibaérték alapján kiszámol egy delta változást, a tanító és validációs adatsoron elért eredményeken. Ezt használom bemenetként azon a modellen, amely egy heurisztikát ad a tanulás jóságára. Ezt nevezem hipermodellnek az ábrán. A „jóság érték" alatt azt értem, hogy mennyire általános összefüggések megtanulása történt az aktuális iteráció során.

A hipermodell visszaad egy „jóság" értéket, amely alapján a gradiens alapú optimalizációs algoritmus egy összefüggés segítségével frissíti a paraméterekhez rögzített egyedi „jóság" változójának értékét. Egész pontosan én így határoztam meg:

$$J[P_i] += \left| \frac{\Delta P_i}{|P_i| + 10^{-1}} \right| H(t, v)$$

3.1. képlet: A „jóság" alparaméterének számítása

Itt a paraméterekhez rögzített jóság változók halmaza alkotja a következő generáció absztrakt modellét. H(t,v)-vel jelöltem a jóság faktorát, ami a jó összefüggések tanulásának mértékét adja meg a bemeneti tanuló és validációs hiba változásának függvényében. A P_i az adott paraméter értéke. $J[P_i]$ a P_i paraméter jóságát tároló változó.

3.6.5 Új generáció absztraktja

Miután megtörtént az összes iteráció. (Beállítható, hogy mennyi ideig fusson az algoritmus az egyes generációk között.) A futtatás után kapunk egy jóságtényezőt minden paraméterre nézve, ezt nevezem a modell absztraktjának. Ez után a feltételnek megfelelő jóságtényezővel rendelkező paramétereket aktiválom, amelyek nem felelnek meg azokat deaktiválom a következő generáció modelljében.

Egy hosszú futtatás után egy új generációs szűrt modellt kapunk, amelynek egyes összefüggései vagy paraméterei ki/be vannak kapcsolva. Ezek az azonosíthatóság kedvéért 0. 1. 2. 3… generációs modellként jelölöm.

Az algoritmus a fentebb ismertetett eljárást követi, és így a heurisztika segítségével javulást ér el a kezdeti modellen.

Feltételezve, hogy a heurisztika tényleg jól határozza meg az egyes paraméterek jóság tényezőjét az algoritmusról belátható:

- Képes automatizált módon nagyságrendekkel nagyobb modelleket megszűrni.
    - Kisebb memória igényt
    - Gyorsul az algoritmust (gyorsabb iterációk)
    - A tanulási folyamat hatásosabb (nagyobb lépésekben csökken a hiba)
    - A modell tovább fejlesztésének lehetősége. Szűkítés után újabb összefüggéseket lehet felírni.

- Jobb illesztés elérése éles adathalmazon (az adott adathalmazra specifikus összefüggések kiszűrésével)
- Nincs hatással a tanulás költségfüggvényére.

3.7 A módszer részletes kidolgozása

Alábbiakban kitérek a rendszer elemeinek felépítésében alkalmazott konkrét függvényekre. A 3.6.2-es fejezetben, a modell bemeneti adathalmaza és kimeneti értékek részletesen ki lettek fejtve, ezekre nem térek ki részletesebben.

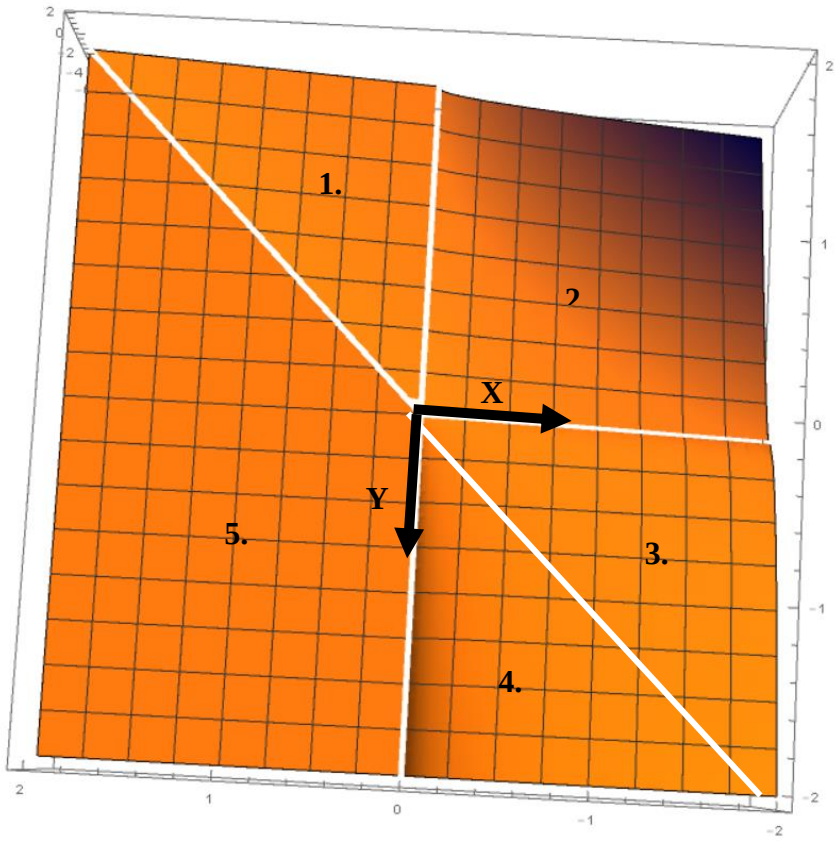
3.7.1 Alkalmazott hipermodell

A hipermodell egy összefüggés halmaz, amelynek bemenetei a validációs és a tanító adatsor hibájának az előző iterációhoz történő eltérése. A két változó függvényében meghatároz egy olyan leképezést, amely képes megállapítani, hogy az utóbb futott iterációs lépés a tanító adathalmaz specifikus felépítését tanulja-e meg vagy általánosan a validáción megjelenő összefüggés megtalálásáról van szó.

Ehhez felvettem egy kétdimenziós (két bemenettel rendelkező) függvényt. Az így definiált függvénynek skálafüggetlennek kell lennie. Ez azt jelenti, hogy a helyes irányba történő elmozdulás függvényének alakja nem változhat a bemenetek nagyságrendjétől. Azért fontos, mert a tanuló és validációs adatsoron a hiba változása nagyságrendileg rendkívül nagy eltérést mutathat (10^2-tól, 10^{-10}-ig). Ez azt jelenti, hogy például el kell kerülni: azt, hogy a túl kicsi bemenetet négyzetre emeljük, hiszen ebben az esetben eltűnhet az érték (például: 10^{-8}esetén $\rightarrow$ 10^{-16} értéket vehet fel).

Ezt az optimális modellt több koncepció és kísérletezés során határoztam meg. A döntéseimet ebben a fejezetben lentebb részletesen ki fogom fejteni. (A munkám során többféle függvényt teszteltem, ezek alapján hoztam meg a végső döntést. Az alternatív függvényeket itt most nem ismertetem.)

A világosabb magyarázat érdekében első körben szeretnék egy 2D-s függvényt felvázolni, amelyen bemutatom a jellegzetesen elkülönülő részeket.



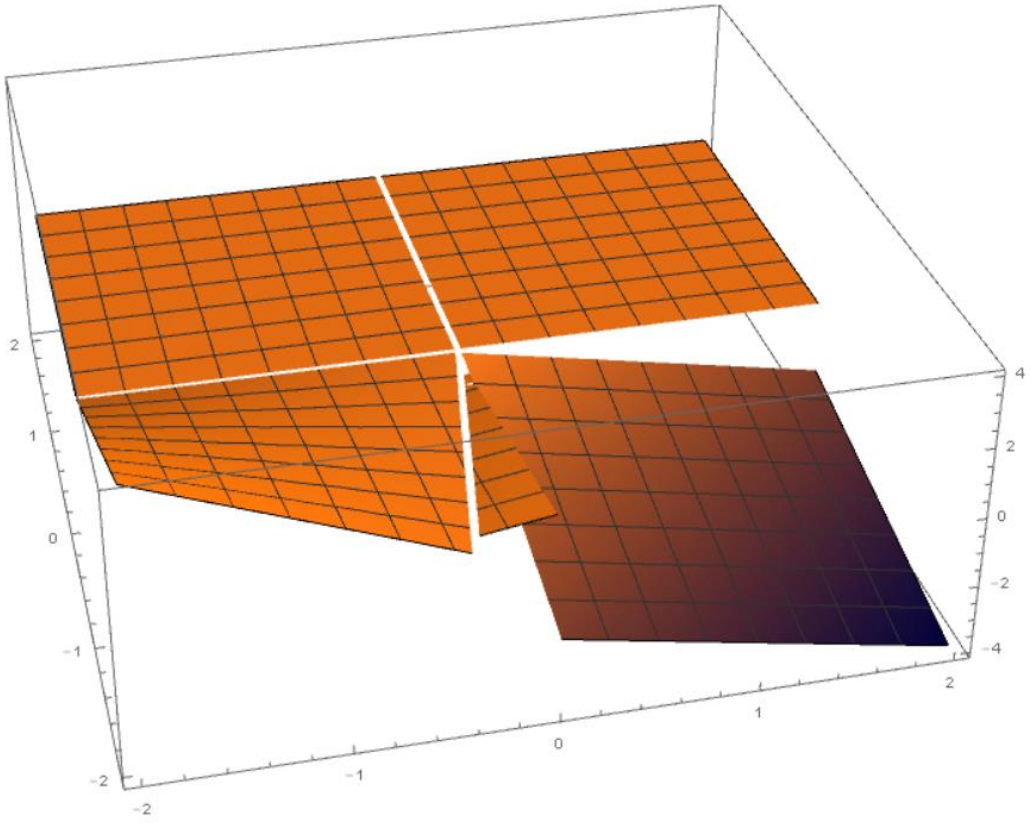
3.2. ábra: Hipermodellben használt függvény területeinek felosztása.

Az ábrán „X” tengellyel van jelölve a tanító adathalmazon történő illesztés hibaértékének változása. Az „Y” tengelyen a validációs adathalmaz hibájának elmozdulása van jelölve. A nyilak irányába csökken az adott adathalmazon végzett illesztés hibája.

Validáció és tanító adathalmaz függvényében felosztottam 5 különböző területre a síkot, mert ezek más-más eljárás szerint lesznek értékelve a tanulás során. Az 1. síknyolcad azt az esetet fedi le, amikor a validáción jobban nőtt a hiba, mint a tanulásra szánt adathalmazon. A 2. terület, amikor *„rossz tanulás”* történt, mert ekkor a hiba nőtt a validációs adathalmazon, de javulás volt a tanulásra szánt halmazon. A 3. és 4. síkrészek valójában a *„jó tanulás”* esetei. Azért lettek megkülönböztetve, mert nem ekvivalens a kettő teljes mértékben. Mivel az az eset, amikor a tanulásra szánt teszt adatsoron jobb eredményt értünk el, mint a validáción az egy alulértékeltebb információ, hiszen ekkor előfordulhat, hogy nem minden összefüggés volt helyes tanulás, amely általánosan igaz volna a megfigyelt rendszerre. Ellenben pozitívabb értékelést érdemel az a variáció amikor a tanulás során a validáción nagyobb javulás figyelhető meg, mint a tanuló algoritmuson. Utolsó megkülönböztetett állapot az az eset, amikor negatív irányú tanulás jött létre, azonban a validáción kisebb mértékű a hiba növekedése, vagy nem nőtt a hiba.

A fenti felbontást figyelembe véve az alapján, hogy a validáció és a tanító algoritmus hibája hogyan változik, meghatározhatunk egy valószínűségen alapuló értékelő függvényt. A koncepció tehát, hogy nagyobb valószínűséggel legyen a rossz paraméter, kiszűrve mint a jó.

Első hipermodellem függvénye:



3.3. ábra: Első „jóság” értékelő függvény

Képletesen:

$$H_1(t,v) = \begin{cases} t - v, & t < 0 \ \wedge v \ > 0 \\ 0, & t > 0 \ \vee v > 0 \\ (v - t)\ 0.8, & 0.2\ t - v < 0 \\ v\ t, & t <= 0 \vee v <= 0 \end{cases}$$

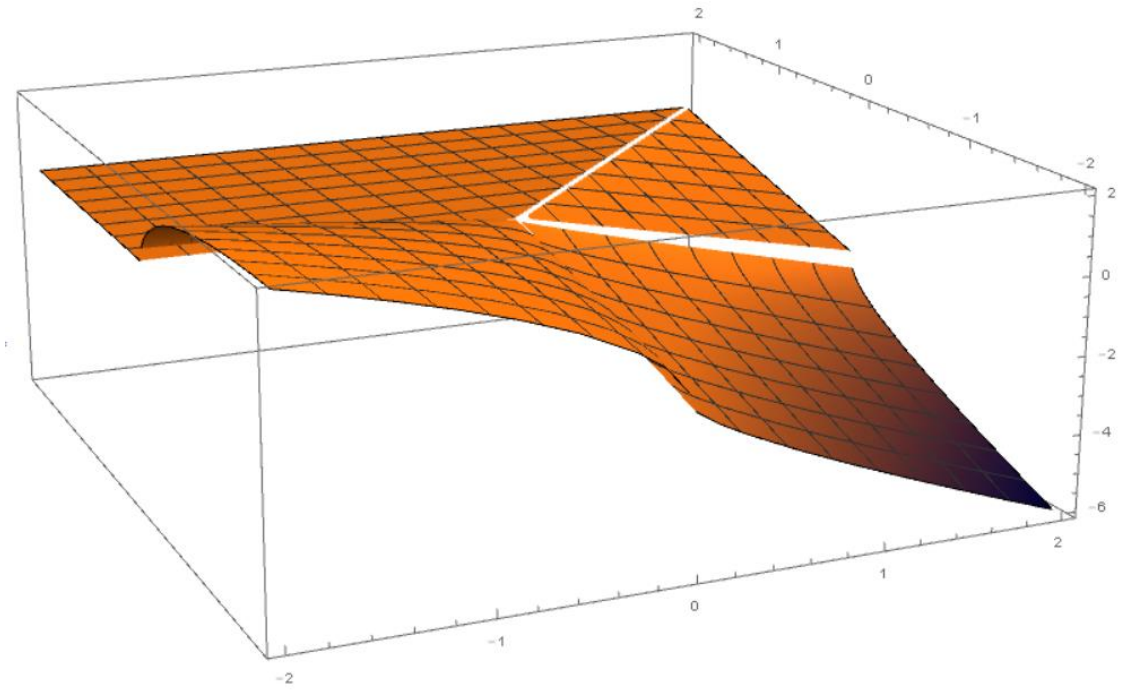
3.2. képlet: 3.3-as ábrához tartozó „jóság” értékelő függvény

A t és v a tanító és validáló adathalmazon elért hibának a változásával egyezik meg.

Az első modell függvényéről az olvasható le, hogy azt az esetet, amikor rossz tanulás történik, abból nem vonok le következtetéseket. Oka, hogy rossz tanulás esetén a gradiens alapú paraméterkeresés túllő a célon, ekkor a validáción történő semmilyen változást nem tartok releváns mutatónak. A 2-es síknegyedben egyszerűen büntetem a hibás tanulást. A 3-as és 4-es síknegyedben annak függvényében, hogy mennyire hasonló irányba történt a tanulás annál pozitívabbnak értékelem a tanulást. Az ábrán látható egy apró „töredék”, amely azt az esetet fedi le, amikor a tanulás a tanító adathalmazon eredményes, de a validáción szinte alig észrevehető a változás. Feltételezem, hogy a tanulás ekkor olyan összefüggéseket tanul meg, amelyeknek fele megjelenik és egy része nem jelenik meg a validációs adatsoron, az ilyen esetet negatívan próbálom értékelni.

Ezzel az első modellel kevesebb mérést végeztem. A mérések azt mutatták, hogy a rendszer képes paraméterek kiszűrésére. Hamar tovább lett fejlesztve. Tesztelésként lefuttattam, mi történik, ha mindent ellentétesen értékelek, ekkor nem mutatta ezt a konzisztens javulást az algoritmus.

Hosszabb fejlesztés eredményeként végül létrehoztam egy „modernebb” modellt. Ebben egy új szempontot is szem előtt tartottam, ami a függvény elsőfokú folytonosságát jelentette. Az így kapott modellt nevezem 2. hipermodellnek a dokumentum során:



3.4. ábra: Folytonos skálafüggetlen „jóság” értékelő hipermodell

Képletesen:

$$H_2(t,v) = \begin{cases} \sqrt[2]{\text{abs}[v\ t]}\ 2 + t, & t < 0\ and\ v > 0 \\ t - \sqrt[3]{\text{abs}[v\ v\ t]} - v, & t < 0\ and\ v >= 0 \\ 0, & t >= 0\ or\ v < 0 \\ t - v, & t >= 0\ and\ v >= t \end{cases}$$

3.3. képlet: 3.4. ábrához tartozó javított hipermodell

Magyarázat: a kijelölt 1-es területnél, rossz tanulásnak neveztem azt, amikor a tanuló algoritmuson jobban nőtt a hiba, mint a validáción. Minél nagyobb az eltérés a tanuló és validáló adathalmaz hibája között annál rosszabbnak értékelem a tanulást. A legrosszabb esetnek azt tekinthetjük, amikor a tanuló algoritmuson csökken a hiba ellenben a validáción növekszik. Jó esetnek azt tekintem, amikor a tanuló adatsoron és a validáción is hasonlóan jó eredményt érünk el. Említésre érdemes, hogy amennyiben a tanulás a tanuló adathalmazon sikeres és a validáción nem történik javulás, azt negatív módon értékelem. Ennek oka, hogy ilyenkor a tanuló adathalmazon egy-egy összefüggés jó,

azonban ugyan ilyen mértékben nem jó néhány másik összefüggés, hiszen együttes hatásuk a validáció hibáját tekintve 0.

Az alábbiakban a 2-es modellel fogok méréseket végezni, mert ezt találtam eredményesebbnek. Azért nem végzek több mérést több hipermodellel, mert a tudományos diákköri munkám célja egy olyan módszer bemutatása, amely eredményes paraméterek szűrésében, nem pedig az algoritmus egy részegységének optimalizálása.

3.7.2 Új generáció absztraktja

Az új generáció létrehozásához adott a régi modell minden paraméteréhez hozzárendelt jóságérték, „az új generáció absztraktját”. Ez határozza meg, mely paramétereket kell megtartani a következő generációban. Ehhez használunk egy szűrőfeltételt.

3.7.2.1 Szűrőfeltétel

Az újabb generáció meghatározásakor, egy újabb problémába ütközünk. Ez a következő generáció modellének áteresztő neuronjainak értékének meghatározása, a „jóság” értékeik alapján. Minden neuronra meg kell határozni, hogy az a neuron benne maradjon-e a modellben vagy nem, tehát az áteresztő neuron az adott paraméterhez konstans 1-re vagy 0-ra vegyen fel a következő modellben. Ehhez meg kell szabni egy feltételt.

A feladat nehézsége, hogy az általunk definiált hipermodelltől erősen függ, hogy milyen feltétellel tudjuk meghatározni a rossz paramétert a jótól. A „jóság” minden paraméterhez hozzárendelt érték felvehet pozitív vagy negatív értéket, az én esetemben ez $[-10^2, 10^{-2}]$ terjedő szám, de sok alkalommal minden paraméter jósága csak a negatív értéktartományban helyezkedik el, vagy hosszabb állítgatás után csak a pozitív értéktartományban vesz fel értéket. Az első tesztelt feltétel, hogy minden paraméternek, melynek a $jóságtényezője > 0$, bent tartjuk, a többit kinullázzuk tehát nem bizonyult megfelelőnek.

Kezdetekben célom volt, hogy a jóság értékek felvétele meghatározzák a jóságot olyan módon, hogy a jó paraméter pozitív, a negatív paraméter negatív értékkel szerepeljen. Ez sok kísérletezgetés után nem tűnt egyszerű feladatnak. Ezek után tovább gondolva, arra a következtetésre jutottam, teljesen felesleges a paraméterek ilyen fokú rögzítése, egyszerűbb lenne, ha a paraméterek egymáshoz viszonyított értékét vizsgálnám. Bevezettem a jóságtényezők átlagát. Az „i”-edik változóhoz tartozó jóság értéket „j_i”-nek elnevezve, a képlet a következőképp néz ki:

Ha: $\frac{(\sum_{i=0}^{n} j_i)}{n} > 0$ akkor:

$$C_a = \begin{cases} 1, & \left(\frac{\sum_{i=0}^{n} j_i}{n} - 10^{-9}\right) 0.4 < j_a \\ 0, & \left(\frac{\sum_{i=0}^{n} j_i}{n} - 10^{-9}\right) 0.4 \geq j_a \end{cases}$$

Ha: $\frac{(\sum_{i=0}^{n} j_i)}{n} < 0$ akkor:

$$C_a = \begin{cases} 1, & \left(\frac{\sum_{i=0}^{n} j_i}{n} - 10^{-9}\right) 1.5 < j_a \\ 0, & \left(\frac{\sum_{i=0}^{n} j_i}{n} - 10^{-9}\right) 1.5 \geq j_a \end{cases}$$

3.4 képlet: Következő generáció áteresztő 0/1 kapu értékeinek maghatározása

C_a jelöli az „a”-adik paraméterhez tartozó szűrőfüggvény értékét.

„*n*” jelöli a szűrt paraméterek számát a modellben.

Döntések magyarázata: Azért lett ketté bontva a függvény mert így lehetőség van módosítani a kiszűrt paraméterek számát. Ilyen módon két különböző stratégia alkalmazására került sor a szorzó tényezők segítségével. Abban az esetben, ha az átlag pozitív, akkor csökkentettem a kiszűrt paraméterek számát, ha az átlag negatív akkor szorzással még nagyobb negatív számot kapunk, amely szintén csökkenti a kiszűrt paraméterek számát. Ez azért is előnyös, mert így a kiugró értékeket szűrjük ki, amelyek az eljárás szerint a legvalószínűbben overfitting tulajdonsággal bíró paraméterek.

A szűrésnél figyelembe kell venni, az összefüggést, amelyben a paraméterek szerepelnek. Az én esetemben olyan függvényeket alkalmaztam, amelyek teljesítik a kizáráshoz szükséges követelményeket. Ezt részletesebben a felhasznált függvények pontnál 4.3.

3.8 Algoritmus működésének elméleti igazolása

Az elméleti gondolatmenet gyakorlati úton mérésekkel alá lesz támasztva az 5-ös fejezetben.

Az elméleti bizonyításhoz két dolgot kell belátnunk.

1. Az algoritmus generikus úton képes csökkenteni az overfitting jelenségét, így javítani a modell eredményeit azonos adathalmazon ugyanazon modell segítségével.
2. Az algoritmus jobb eredményeket ér el, mint ugyanazon az adathalmazon az algoritmus nélkül futtatott rendszer a túltanulás problémájának fennállásának esetén.

3.8.1 Első állítás

3.8.1.1 Igazságának jelentése

Az állítás teljesülése esetén, a rendszer valóban képes kiszűrni a jó paramétereket a rendszerben, ezzel csökkenteni a tanuló és test adatsor hibája közötti hézagot. Amely az algoritmus céljának teljesítését jelenti.

3.8.1.2 Bizonyítása

A tanulásra szánt adatsor meghatároz „n” egyeneltet. Mivel a paraméterek száma $p > n$ ezért előfordulhat, hogy a rendszer csak úgy igazítja a megfelelő paramétert, hogy az a tanulásra szánt adathalmaz kimeneteihez való illeszkedésen javítson. Az ilyen paraméteridentifikáció, ha helyes akkor az azt jelenti, hogy a validáción is javulni fog az illesztés, ha nem helyes, akkor a validációs adathalmazon nem fog azonos mértékben javulni a tanulás, mint a tanulásra szánt adathalmazon. Így detektáltunk egy olyan összefüggést, amely csak a tanuló adathalmaz egyedi állásán volt helyes, ami nincs a rendszerre igaz általános összefüggésekre.

Azért lesz valószínűség alapú a paraméter szűrésünk, mert a rendszer paraméterei között interferencia lép fel, és csak értékelni tudjuk, hogy a rendszer paramétereinek együttes változása milyen változást hozott létre. A valószínűsége pedig azért lesz nagyobb a rossz paraméterek kiszűrésének, mert a rossz paraméterek elmozdulásaik során rossz irányú változást hoznak létre, míg a jók minden elmozdulásukkal jó irányú változást hoznak létre. Így, ha minden iteráció során ellenőrizzük, hogy melyik paraméterek milyen változást hoztak létre az adott elmozdulásuk értéke mellett akkor egy olyan súlyozását kapjuk a rendszernek, ahol a rossz paraméterek minden elmozdulásuk esetén egy kicsivel több büntetést kapnak. Feltehetőleg ez az arány egy exponenciális skálával egyezik meg. Az a paraméter, amely minden elmozdulásával jó iterációt hoz létre, átlagosan jobb értékelést kell kapjon a nagy számok törvénye alapján, mint annak a paraméternek, amely minden elmozdulásával rosszabbat hoz létre.

Erre van egy valós életben működő azonban kicsit bagatell példa. Ahol egy játékos egyéni képesség alapján történő „rankingolása” a csapatos teljesítmény alapján történik. Az egyéni játékos értékelése, egész pontosan akkor kerül feljebb, ha a csapata nyer. Amelyhez, ha átlagosan ő mindig nagyobb teljesítményt rak hozzá, mint a csapattársai akkor nagyobb valószínűséggel fog nyerni egy olyan csapat ellen amelyik a csapatának átlagos képességével rendelkezik.

3.8.2 Második állítás

Az algoritmusnak ennek az állítás teljesítése nem célja, mivel nem egy jobb módszer felkutatása a cél, hanem egy effektív módszer mutatása a paraméterek számának csökkentésére az adott modellben anélkül, hogy a rendszer tanulása romoljon a validációs adathalmazon.

3.8.2.1 Igazságának valódi jelentése

Az állítás, hogy a módszer valóban eredményesebben teljesít-e túltanulásos környezetben, mint az algoritmus alkalmazását mellőző rendszerek.

3.8.2.2 Bizonyítása

Ez a következő két eset összehasonlítását jelenti. Amikor „n” mennyiségű tanuló adathalmazunk van és „m” db validációs adathalmazunk van. Az algoritmusomban a kettő aránya $5:2$ a 3.6.1-es fejezetben látható. Másik amikor „$n+m$” mennyiségű

tanuló adathalmazunk van. Túltanulás lép fel az adatok mennyisége és paraméterek számából adódóan. Feltételezve, hogy a bemeneti adatok függetlenek, maximálisan $n + m$ paraméter konkrét értékét képesek meghatározni. Túl nagy modell használata esetén, ahol a paraméterek száma $p > n + m$ a rendszer túltanulásra lesz hajlamos.

Az algoritmusom nélkül a gradiens alapú tanulás végez egy illesztést, ami illeszkedni fog a bevitt adatokhoz, még az is lehet, hogy jobb illeszkedést fog mutatni, mint az általam készített algoritmus. Azonban ezeket az eredményeket csak a tanulásra szánt adathalmazon produkálja, a validációs adathalmazon ez már nem mondható el. A rendszer az összes paramétert fel fogja használni a tanulásra, hogy így javítsa az eredményt, pedig több paraméter hibás összefüggéseket ír le. A túl sok paraméter miatt teszt adatsoron nem definiált mértékű hiba jöhet létre, a modell és az adathalmaz függvényében.

Az én algoritmusom ezzel ellentétben, végez egy tanítást, mint azt a normális algoritmus végezné, azonban ezt „*n*” adatsoron. Ezáltal létrejön egy overfitting, amit az algoritmusom látni fog a fennmaradó „*m*” adatsoron. A következő generációban látva a túltanulás mértékét a rendszer elhagy paramétereket, csökkentve a paraméterek számát már akár első generációban majdnem egy nagyságrenddel. Ezek után a következő tanítás esetén p<n ahol amennyiben az n paramétere nem teljesen független egyenletrendszereket alkot akkor még létezhet lehetőség overfittingre az újabb tanulás során. Ez, a következő generációban még tovább csökkenthet.

A szűrés során csak arról az esetről beszéltünk, amikor csak rossz paramétereket szűrtünk ki. De ez egy valószínűségen alapuló módszer. Előfordulhat, hogy olyan paramétereket is kiszűrünk amelyek az ideális összefüggésben vannak benne. Az algoritmus ebben az esetben nem feltétlenül, de rosszabbul is teljesíthet. Az egész azon alapszik, hogy az jóság értékelésére és a paraméter iterációnkénti jóságértékelése szerint, jobban bünteti a rossz paramétereket, mint a jókat. Ennek a feltételnek a bizonyítására a 3.8.1.2 végén található példa a bizonyíték, és az 5 fejezet mérései.

4. Megvalósítás

A megvalósítás során több mellékes programot felhasználtam, de az algoritmus a Google által készített TensorFlow python API-ján alapszik. Célom volt, hogy a kutatás és algoritmus validálás minél több ponton automatizálva legyen, ezzel csökkentve az emberi hiba lehetőségét.

4.1 Algoritmus követelmény

A mérések szükséges tulajdonsága hogy reprodukálhatóak legyenek. A random részeket rögzített „seed”-del kellett legenerálni. Ilyen a tanuláshoz használt modellek paramétereinek kezdőértéke, a bemenetek legenerálása, a legenerált adathalmaz kimeneteihez használt paraméterek rögzítése. De jelentős szerepe van a megfelelő összehasonlítás szempontjából, hogy az újra meg újra elindított generációk szintén ugyan abból a kezdeti modell által beállított paraméterértékekből induljanak.

Az algoritmus csak akkor működik megfelelően, ha a tanításra használt adatsor befér a memóriába és a tanítás csak ezzel történik. Jelentős szempont, hogy a tanulás ne ingadozzon. Azért nem engedhető meg ingadozás, mert az azt jelentené, hogy a validáció és tanító adatsor hibájának értékének változása nem azt árulná el, hogy az adott adatsort milyen módon tanulja meg az adathalmaz, hanem hogy kevés a felhasznált adat, vagy divergál a rendszer egyik paramétere.

Be kell állítani a megfelelő hiperparamétereket az optimalizációs algoritmushoz és learning rate értéket, hogy a tanulás monoton javuló folyamat legyen. Ebben a cikkben található egy összefoglaló az ideális tanulás megkeresésére [9].

Első iterációk során az optimalizációs algoritmus hajlamos ingadozásra. Ezért a túltanulás méréséhez szükséges egy korlát meghatározása, amely megadja, hogy mikortól történjen a paraméterek „jóság” értékének feljegyzése.

4.2 Gradiens alapú optimalizációs algoritmus

A feladat során megpróbáltam javítani az előre definiált, illetve publikációkban elérhető optimalizációs algoritmusokon azáltal, hogy többet egymásba építettem.

Felhasználtam az RPROP algoritmust[16]. Itt találtam egy lehetséges hibát a keretrendszerben ugyanis nem volt megfelelően rögzítve a műveleti sorrend, amely folytán az egyes végrehajtásoknál a gráf egyes értékei előbb lettek kiolvasva mint, hogy azok frissültek volna, aminek köszönhetően nem voltak determinisztikusak a futtatásaim. *(Ezt nem küldtem el pull requestbe a google fejlesztőinek, mert nem voltam biztos benne, hogy ez okozta a hibát, és nem volt időm kísérletezni vele.)*

Az RPROP-hoz hozzáillesztettem az *adam momentum* algoritmust[18]. Amely az egy irányba történő iterálás lépésszámát csökkenti. Az algoritmus a szó szoros értelmében a

momentumot implementálja a gépi tanulás területén, ahol a gradiens a gyorsulás vektornak feleltethető meg.

A két optimalizációs algoritmus mellé, a state-of-the-art algoritmust is beillesztettem, amely a *nesterov* algoritmus[19]. Ez tovább javulást ért el a tanulás sebességén.

Fenn állt az esélye, hogy az algoritmusok működésükből adódóan interferenciák miatt rosszabbul működnek együtt. Azonban mérések során nem ezt tapasztaltam. Mindig gyorsabb és gyorsabb tanulást értem el az újabb algoritmus bevonásával.

4.3 Felhasznált függvények

Függvényekkel írhatóak fel az összefüggések. Az egész algoritmus ezen összefüggések paramétereinek szűrésével foglalkozik. A szűrés a paraméter hatásának nullázását jelenti a rendszerre nézve. Ahhoz, hogy egy paraméter hatása a kimenetre nézve 0 legyen nem minden esetben megfelelő a paramétereinek értékének 0-ra csökkentése. Az $f(0) = 0$ feltétel teljesülése esetén egyértelmű a „neuron” kimenetének 0-ra változtatása. Ezek a következők:

- Összeadás: $f(x) = p_1 + x$
- Szorzás: $f(x) = p_1\, x$
- Tanh: $f(x) = \tanh(x)$
- „SoftSign”: $f(x) = \frac{x}{1+|x|}$

Nem triviális függvények:

- Osztás: $f(x) = \frac{1}{x}$
- Logaritmus függvény: $f(x) = \log(x)$
- Logisztikus regresszió: $f(x) = \frac{1}{1+e^{-x}}$
- Gauss aktivációs függvény: $f(x) = e^{-x^2}$

A nem triviális függvények megkülönböztetett kezelésére van lehetőség az algoritmusomon belül, azonban ilyen esetekre, egyelőre nem készítem fel a rendszert. (Megkülönböztető elnevezés segítségével lehetne külön kezelni a speciális műveleteket végző neuronokat, hogy bővítsük a lehetséges függvények alkalmazását.)

4.4 Eredmények loggolása

A folyamat során létrejövő hibaértékeket, „.csv” állományba mentettem ki. Így mentem le minden futtatás eredményeit. Könnyedén vizualizálni lehet az adatokat grafikonon.

Loggolom a futtatás konfigurációját egy külön fájlban. Itt megtalálható, az felhasznált optimalizációs algoritmus neve és hiperparaméterei, „learning rate”, Felhasznált adathalmaz, mettől-meddig (sorok száma), validációra felhasznált adatok aránya,

felhasznált modell, generációk paramétereinek száma, generációkban használt iterációk száma, hipermodell leírása, a tanulás „jóságára” használt modell, és hogy az algoritmus mikortól kezdi el megfigyelni a tanulás milyenségét. Gyakorlatilag ezek a jellemzők részlegesen leírják az algoritmust, amelyből reprodukálható az eredmény.

Az adatok, amelyek nem kerülnek kimentésre, csak futás közbeni loggolásra:

- pontosan mely paraméterek kerülnek kikizárásra
- megmaradt paraméterek számossága.
- a paraméterekhez kötődő jóságértékek

4.5 Algoritmus korlátai

Azt tapasztaltam, hogy az algoritmus konzisztensen eredményes működéséhez fontos a következő peremfeltételek teljesülése:

- Az „overfitting” jelenléte. Ezt csökkenti az algoritmus, ha ilyen nincs akkor az azt jelenti, hogy a modell minden összefüggése megfelelően illeszkedik a tanító és teszt adatsorhoz. Ez akkor fordul elő ha sok a paraméterünk.
- A 4.3 fejezetnek megfelelő függvényeket használ a rendszer.
- A tanítás nem batch alapú.
- Neurális hálóban nincs visszacsatolás

Természetesen a peremfeltételek nem teljesülése esetén is okozhat pozitív csalódást az algoritmus. Azonban ezekben az esetekben nem teszteltem vagy elméleti úton nem állítható, hogy nem minden esetben hoz javulást a rendszer.

5. Mérések

Az algoritmus működésének bizonyításához több összehasonlítást végezek. Felállítok egy követelmény rendszert, aminek meg kell felelnie az algoritmusnak.

5.1 Követelmény

Meg kell vizsgálni az alábbi pontokat.

- A generációk közötti javulás detektálása. Ideális és torzított adatsoron.
- A validációs adathalmazra történő rátanulást ellenőrizni. Teszt és validációs adatsor hibája közötti különbségeket megnézni.
- Megegyező futási idő alatt melyik vezet jobb eredményre.
- „Végtelenített” futtatás.
- Letesztelni numer.ai által biztosított kódolt tőzsdei adatokon az algoritmus működését.

A feladatom során meg kell határoznom olyan teszt adathalmazt, amelyen végzem a méréseket. A mérések során nélkülözhetetlen a reprodukálhatóság. Ennek értelmében a feladathoz tartozik az adathalmazok meghatározása és csoportosítása, majd az adatok specifikációjából adódó egyediségekből következtetések leszűrése melyek megmagyarázzák az adott adathalmazon elért eredményeket.

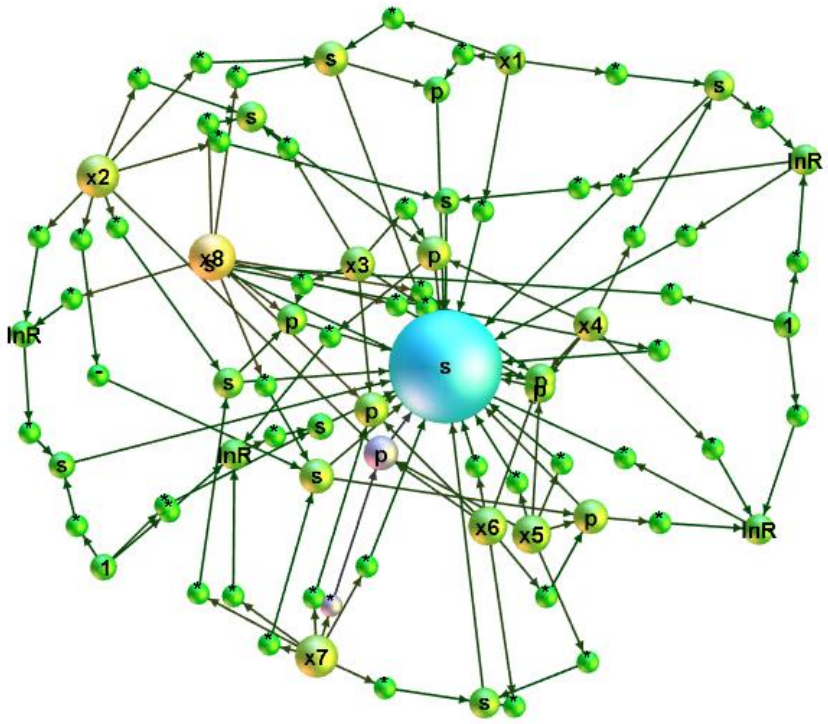
5.2 Adathalmazok

Különböző adathalmazokat vizsgáltam, végeredményben arra a következtetésre jutottam, hogy az algoritmus validálásához, szükséges tudnom a kiindulási modellt ahhoz, hogy tudjam ellenőrizni az algoritmus fejlődését.

A mesterséges adatsor 8 bemeneti és egy kimeneti csatornából áll. Gyakorlatilag a rendszer megfelel 8 szenzor folyamatos leolvasásának, amely egy általam adott modell alapján egy kimenetet ad végeredményül.

Készítettem 3 különböző modellt.

1. Ehhez a Gephi[20] gráf szerkeztő programot használtam. Vizualizálva a complex_3:



5.1. ábra: mintaadat generálásához használt complex_3 megnevezésű modell

2. Ezek összefüggéseket írnak le, a bemenetek függvényében. Ezeket az összefüggéseket kiexportáltam. A legkisebb model a következőképpen nézett ki. (S_0 a kimenet.)

G_n : {Gerjesztő inputok}

P_i : {Paraméterek}

Algoritmus által használt model amelynek a paramétereit határozta meg:

$$S_0[t] := S_1[t] + G_2[t]\ (p_8\ G_1[t]) + S_2[t] + G_4[t]\ (p_{11}\ G_2[t]) + S_3[t] + \frac{p_{20}}{1 + e^{p_{21}\ (p_{23}\ S_4[t])\ p_{22}\ p_{26}}} +$$
$$p_{27}\ G_5[t] + p_{28}\ S_4[t] + p_{29}\ G_4[t] + p_{30}\ G_3[t] + p_{31}\ G_7[t] + p_{32}\ G_1[t] + p_{33}\ G_7[t] +$$
$$G_0[t]\ (p_{34}\ G_3[t]) + S_5[t] + G_7[t]\ (p_{43}\ G_6[t]) + G_0[t]\ (p_{44}\ G_2[t]) + (p_{45}\ G_3[t])\ (p_{46}\ G_2[t])$$

$$S_1[t] := \frac{p_0}{1 + e^{p_1\ (p_4\ G_0[t])\ p_2\ p_5\ p_3\ (p_6\ G_1[t])}} + p_7\ 1$$

$$S_2[t] := p_9\ G_3[t] + p_{10}\ G_0[t]$$

$$S_3[t] := \frac{p_{12}}{1 + e^{p_{13}\ p_{16}\ p_{14}\ (p_{17}\ G_0[t])\ p_{15}\ (p_{18}\ G_2[t])}} + p_{19}\ 1$$

$$S_4[t] := p_{24}\ G_5[t] + p_{25}\ G_6[t]$$

$$S_5[t] := \frac{p_{35}}{1 + e^{p_{36}\ (p_{39}\ G_5[t])\ p_{37}\ (p_{40}\ G_0[t])\ p_{38}\ p_{41}}} + p_{42}\ 1$$

5.2. ábra: us modell egyenletrendszerekkel leírva.

3. A kigenerált összefüggéshalmaz rögzített „seed”-del történő paraméter initializációját elvégeztem.

4. Generáltam rögzített „seed”-del bemeneteket.
5. A legenerált összefüggéseken a bemeneteken végig terjesztettem, így kiszámoltam a modell kimenetét.
6. Ezek után rögzített „seed”-del csináltam torzított adatsorokat. Az eredeti adathalmazra először fehér zajjal eltolva, majd második esetben normális eloszlásnak megfelelő multiplikációval és eltolással.

A fenti lépések alapján, 3*3 mesterségesen előállított adatsort generáltam. 1 torzítatlan és 2 különböző zajjal torzítottam az adatsorokat.

Adatsor jellemzője	Normál	Fehér zaj	Standard eloszlású szorzás és eltolás
us(mikro)	us1.csv	us1.white_noise.csv	us1.noise.csv
simple	simplex_3.csv	simplex_3.white_noise.csv	simplex_3.noise.csv
complex	complex_3.csv	complex_3.white_noise.csv	complex_3.noise.csv

5.1. táblázat: méréseknél felhasznált adatsorok

A méréseim a fenti lépések alapján reprodukálhatóak legyenek.

Felhasznált éles adatsor a numer.ai [21] honlapján adott stock market adathalmaz, 57771 sornyi, 21 különböző „feature”-rel rendelkező adat. Egy klasszifikációs meghatározást vár kimenetként. Rendkívül nehéz jó eredményt elérni rajta, illetve tudni kell, hogy az eredmények közel 54-55%-os pontosság körül mozognak. Ami azt jelenti, hogy alig ismerjük az ideális összefüggéseket. Tözsdei, ismeretlen 10^{11} mennyiségű összefüggés is lehetséges a rendszer leírására, teljesen kiszámíthatatlan, legjobb becslések 56%-ot érik el.

5.3 Tanulásra használt modellek

3 adatsorhoz mindegyikhez definiáltam egy modellt. Mindegyikre igaz, hogy amelyik adatsorhoz készült annak legenerálásához szükséges modellt magába foglalják. Oka, hogy a tanulásnak legyen lehetősége megtanulni az adatsort. A valóságos helyzethez hasonlóan amikor nincs meg a megfelelő mennyiségű adat, vagy zaj van a rendszerben, akkor hogyan lehet ezt a túltanulást vagy hibás tanulást megszüntetni.

Ezek alapján a modellek felsorolás szintjén:

- *us_1 modell*: A legkisebb modell. Összesen 4205 paraméter segítségével írja le.
- *simplex_3 modell*: 45037 paraméter használ a modell a futtatásához.
- *complex_3 modell*: 36795 paraméteres modellt használ a futtatáshoz. Ez kevesebb mint a simplex_3 modell, azonban kezdetekben 6.2 millió

paraméterre jött ki a modell mérete, amit túlcsökkentettem. Ettől függetlenül más összefüggéseket ír le, mint a simplex_3 modell.

Az alábbiakban ezekkel a megnevezésekkel fogok hivatkozni a modellekre.

5.4 Futtatási sebességek

A kezdeti tesztmodellek iterációi több másodpercesek voltak. Így kisebb modellek tesztelése is több órába telt, ezen csökkentenem kellett.

A számítógépem paraméterei alapján, a CPU 250 GFlops, a GPU 1314 GFlops számolásra képes. A CPU-ról GPU-ra történő áttérés után 4x-es „speedup” volt tapasztalható.

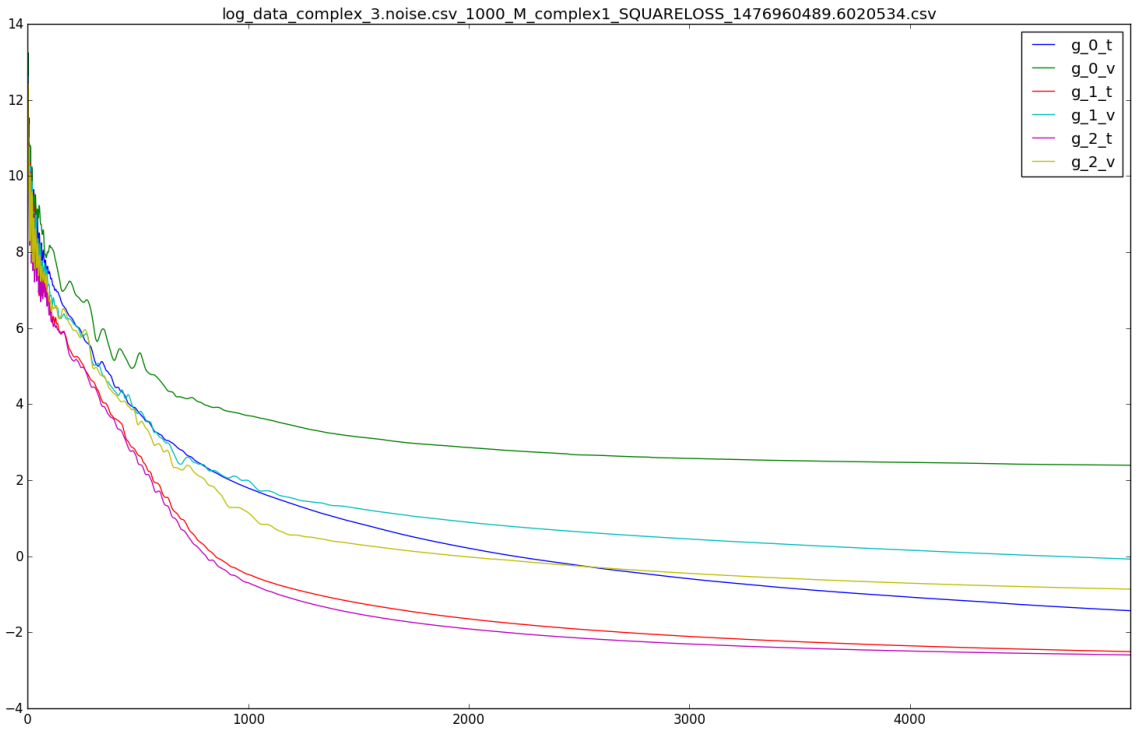
Csökkentettem a feldolgozandó adatmennyiséget és a modellek méretét is 1 nagyságrenddel, így elértem, a 0.2 másodperces iterációs időt. Ez modelltől függően ingadozó volt, de így már lehetőség volt több tesztet is lefuttatni 1 nap alatt.

5.5 Követelmények mérésekkel történő igazolása

A 9 adatsoron megvizsgáltam a következő eredményeket. Mindegyikkel hasonló eredményeket kaptam. A teljesség igénye nélkül egy-egy példán mutatom be az algoritmus teljesítményét.

5.5.1 A generációs fejlődés bizonyítása

Az algoritmus lényege, hogy a generációk során csökken a túltanulás mértéke. Ez úgy lehet megfigyelni, hogy a generációk során a validációs adathalmazon elért hiba csökken.



5.3. ábra: generációnkénti logaritmikus hiba csökkenése a validációs adathalmazon.

A tanulás a *complex_3* modell segítségével történt a complex_3.noise.csv adatokon.

A fenti ábrán a g_0_v, g_1_v, és a g_2_v görbék mutatják a generációk validáción történő hibáját. A „g” a generációt, „t” és a „v” tanuló és validációs adatsort rövidítik. A logaritmikus hiba ábrázolásán jól látható, hogy **a validáción mért hiba nagyságrendekkel kisebb**. A tanulásra szánt adathalmaz hibáját jól közelítik validációs adathalmazon elért hibák a generációk során.

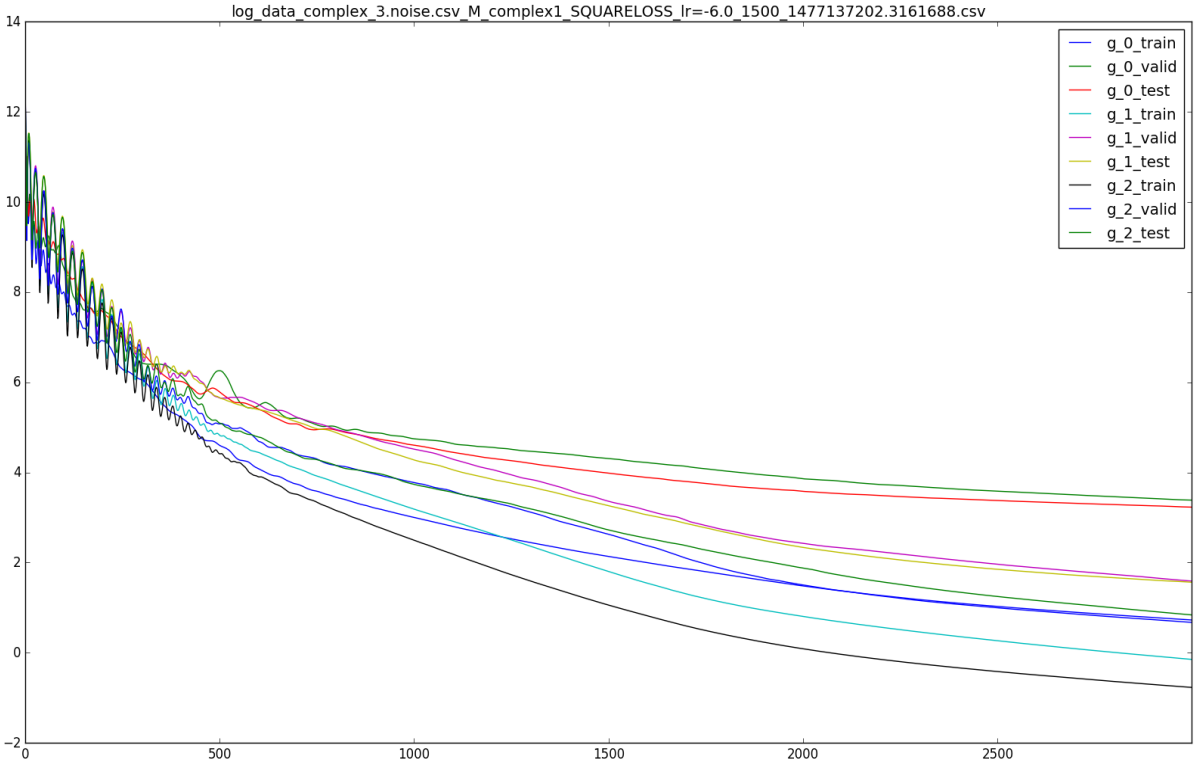
Megfigyelhető, hogy a tanulás gyorsabban játszódik le a szűrt modellekkel.

Ez és a többi nem dokumentált mérésem alapján elmondható, hogy az algoritmus a generációk során képes effektíven csökkenti a túltanulás mértékét.

5.5.2 A validációs és teszt adathalmaz ellenőrzése

Ahhoz, hogy elegendő legyen csak a validációs adathalmazt vizsgálni a projekt során szükséges megvizsgálni, hogy valóban nem történik rátanulás a generációk során erre az adathalmazra. Ez egyben azt is bizonyítja, hogy az újabb generációban kizárt összefüggések nem validáció specifikus összefüggések volnának.

Erre szintén több futtatást végeztem:



5.4. ábra: a validáció és a teszt adatsor hibája közel megegyezik

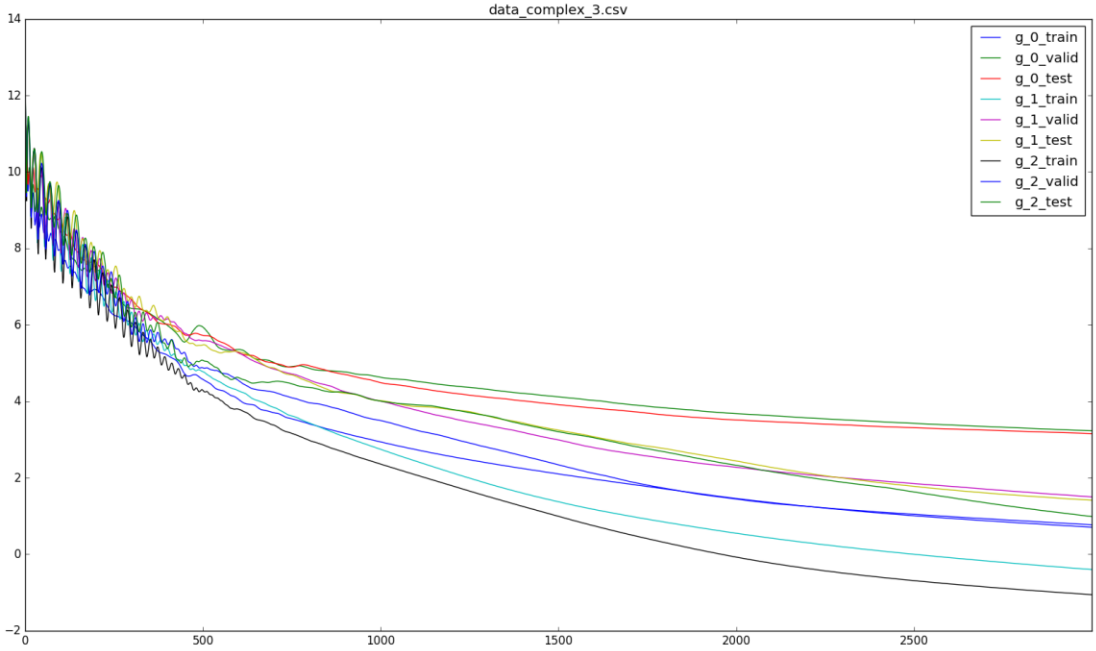
Amellett, hogy látszik az algoritmus működése, látható, hogy apróbb eltéréseknél több nem jelenik meg a validációs és teszt adatsor hibája között, sőt néhol a teszt adatsoron ér el jobb eredményt az újabb generáció. Ez azt jelenti, hogy nem történik a validációra rátanulás az adatsor oldaláról. Tehát a validációs adatsor valóban érvényes validálási értékkel bír, akár csak a teszt adatsor.

Ami még megfigyelhető, hogy vannak apró eltérések a két adatsor hibája között. Azonban ez a két adathalmaz megkülönböztethetőségéből eredeztethetően léphet fel.

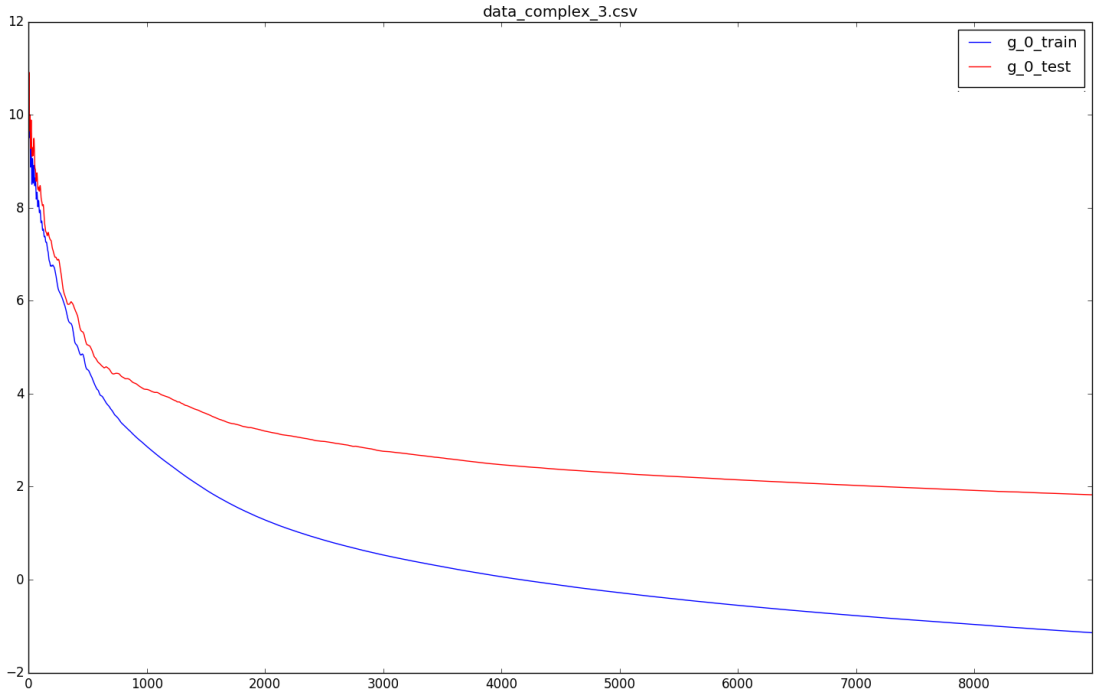
5.5.3 Normál és saját algoritmus összemérése

Megegyező ideig történő futtatás esetén melyik algoritmus mutat jobb eredményt. Amennyiben ez is teljesül akkor elmondható, hogy az algoritmus, amit itt prototípusként kutattam megállja a helyét rendes algoritmusok mellett is, amennyiben létezik túltanulás.

A két ekvivalens ideig történő futtatás eredménye:



5.5. ábra: saját algoritmusom által mért futtatás



5.6. ábra: 9000 iterációs futtatása a normális algoritmusnak

Összehasonlítás. A két algoritmus más-más tényezőkben teljesít jobban, ezek megmagyarázhatók az algoritmusok működésével.

A normális gradiens alapú optimalizációs algoritmus több adaton végzi a gradiens alapú optimalizációt. Ez megmagyarázza miért teljesít jobban a 3000-dik iterációban az

én algoritmusom 0-dik generációjának 3000-dik iterációnál. Ez természetesen nem probléma, mert a 0-dik generáció esetén még nem avatkozott bele az algoritmusom, a következő generációkban pedig nagyságrendekkel jobb eredményt produkál az algoritmusom.

Ami ennél a mérésnél a fontos, az a végeredmény, amely:
Saját algoritmus 3000 iterációnál:

Adatsorok:	tanító	teszt
Gen_0:	2.02,	23.39
Gen_1:	0.66	4.09
Gen_2:	0.34	2.67

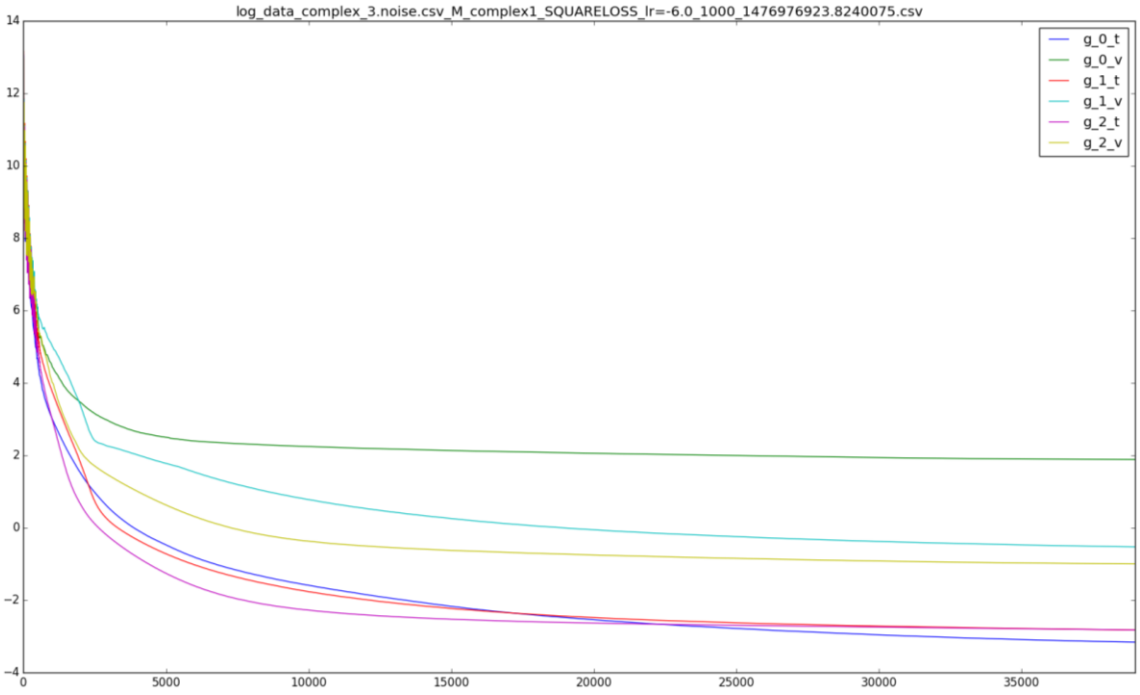
Normális algoritmus 9000 iterációnál:

Futtatás:	0.31	6.20

Az adatok alapján levonhatjuk azt, hogy nagyobb adathalmazon több iterációval jobb eredményt ér el az gradiens alapú algoritmus az előzőnél, azonban még ez is jelentősen rosszabb, mint a modell szűrésével elért eredmény, amely már az első generáció kapunk a kutatás során kitalált algoritmusnál.

5.5.4 A modell végteletekig történő futtatásának tesztelése

Érdekes mérési pont lehet annak megállapítása, hogy amennyiben végletekig futtajuk az algoritmust továbbra is azt tapasztaljuk-e, hogy a tanuló adatsoron is jobb eredményt kapunk-e a generációk során, mert ez egy ideális adathalmaz esetén ellent mond a logikának. A validáción természetesen ez nem kérdés, az 5.5.3 fejezetben láthattuk, hogy a validáció megragad egy korai értéknél a túltanulás miatt. A mérés:



5.7. ábra: az algoritmus 38000 iterációs futása

Jól látható, hogy a tanuló adatsorra a 0-dik generáció tanul rá legjobban. Ez várható volt, mert a 0-dik generációs modellben van a legtöbb paraméter, hiszen a rákövetkező generációkban a rossznak gondolt paraméterek kitiltásra kerülnek. Ez alapján egy kérdés merül még fel. Ez azt jelentené, hogy jó paramétert is kitiltottunk, vagy mi a magyarázat arra, hogy nem ugyanahhoz az illesztéshez konvergál az algoritmus. Ezt nehéz megmagyarázni, mivel ekkor már közbe jön a floating point pontosság. Több mérésre lenne szükség, hogy meg tudjuk állapítani a hiba pontos okát, sajnos van egy olyan magyarázat, hogy a hipermodell nem kezeli megfelelően az előjelváltást és ezért megvan rá az esély, hogy kitilthat jó paramétert is, de hasonlóan valószínű okból számtalan létezik.

Összefoglalva, az algoritmus továbbra is jól teljesít, láthatóan gyorsabban eléri azt a pontot, amely alá már nem képes menni. észrevettünk egy „hibát" amelyre későbbiekben nagyobb figyelmet kell szentelni.

5.5.5 Éles adatsoron történő tesztelés

Legutolsó mérési pont. Tőzsdei adatokon történő tesztelés.

A numer.ai [19] honlapján adott stock market adathalmaz, 57771 sornyi, 21 különböző „feature"-rel rendelkező adat. Egy klasszifikációs probléma. Rendkívül nehéz jó eredményt elérni rajta, az eredmények közel 54-55%-os pontosság körül mozognak. Ami azt jelenti, hogy alig ismerjük az ideális összefüggéseket. Ismeretlen a modell amely a tözsdei kimenetet írja le, a lehetséges összefüggések száma végtelen. Legjobban működő algoritmusok a világon 56%-ot érik el, amely 0.688 körüli logaritmikus hiba értéknek felel meg 2016 májusában. Az én méréseimben is logaritmikus hibát fogok használni.

5.5.6 Az első modell

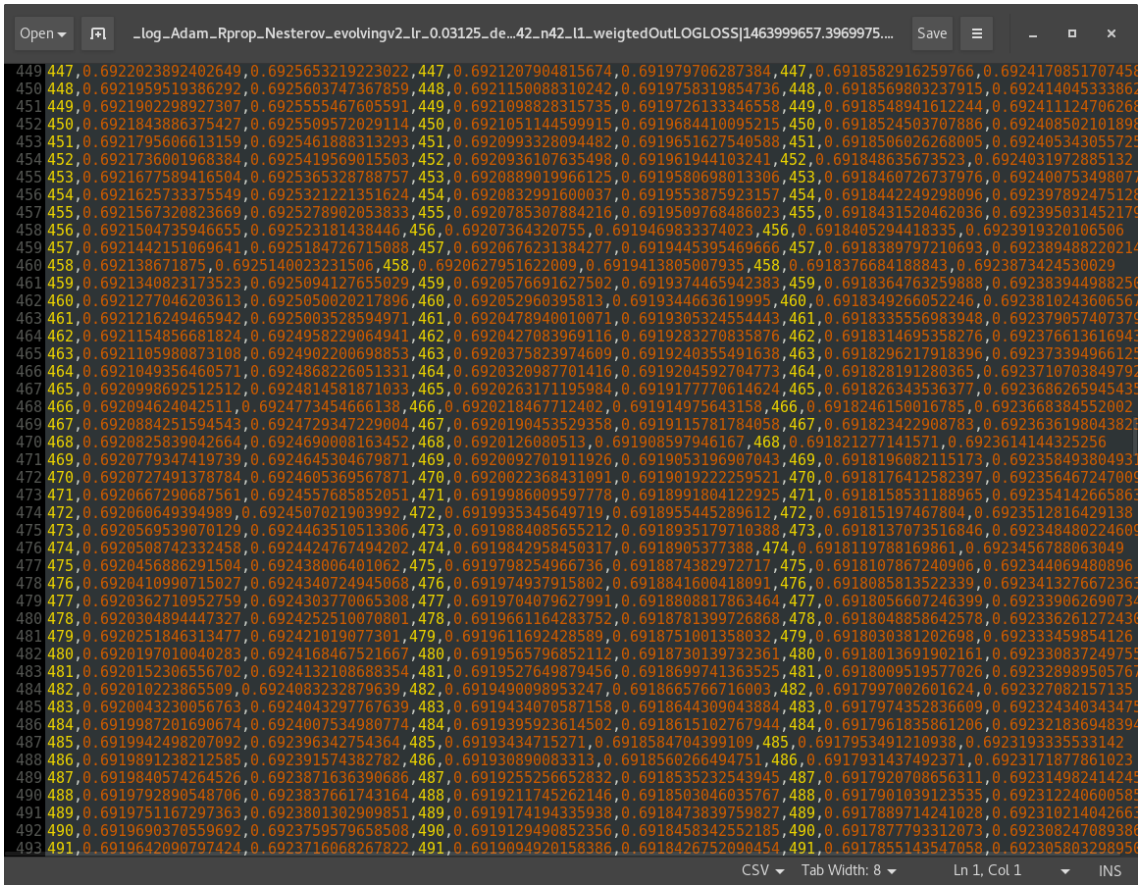
l1 kódot kapta. A modell felépítése a rengeteg adat miatt rendkívül kicsi:

$$modell(\bar{X}) = \bar{X}\,\bar{w} + b$$

5.2. képlet: numer.ai első verziós modell vektorokkal leírva

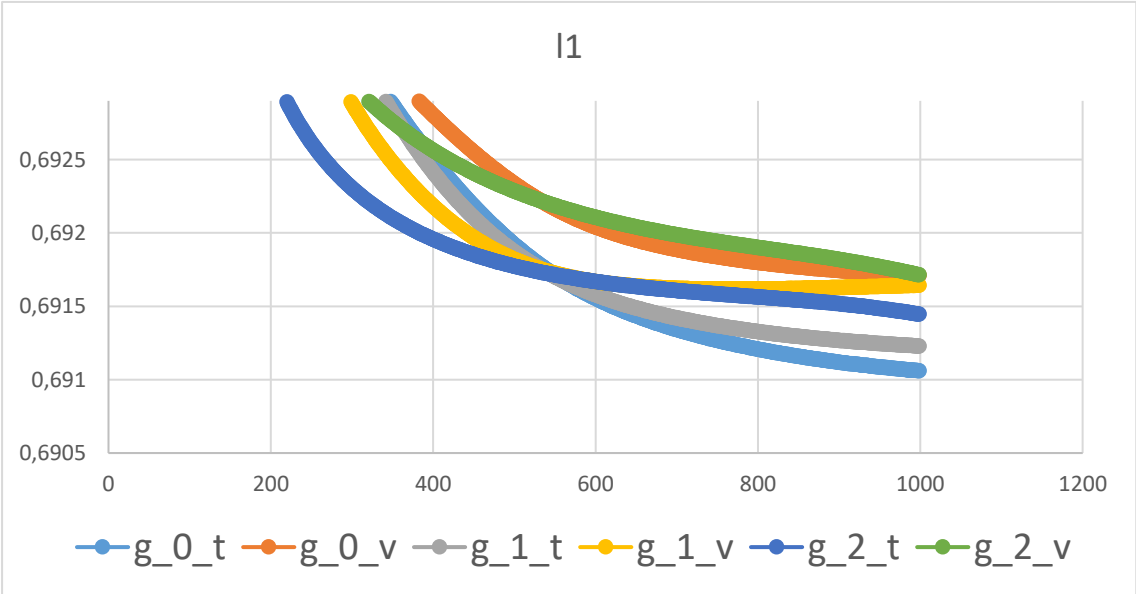
Egy skalárral történő eltolással és a 21 bemenet súlyozva rá van kötve egy kimeneti logisztikus regresszióra.

Egy példafuttatás során ilyen outputokat kapok:



5.8. ábra, csv fájl log

Fenti ábrán látható, 3 generációnak tanuló és validáló adathalmazon elért LOGLOSS értékei iterációnkénti bontásban. Grafikonon ábrázolva az 1000 iterációt. (Kezdeti 200 iteráció levágtam mert rendkívül nagy hibával indult.):



5.9. ábra. csv loggolás ábrázolása

Látható, hogy itt első ránézésre nem kaptunk minden tekintetben jó eredményt. Az első és talán legfontosabb észrevételünk, hogy a modellünk minden generációnként egyre rosszabb eredményeket ér a tanulásra szánt adathalmazon.

Adatsorok	gen_0	gen_1	gen_2
Tanító	0.69106	0.69122	0.69144
Validáló	0.69170	0.69164	0.69171

5.2. táblázat: numer.ai-s teszt futtatások eredményei

Azonban azt láthatjuk, hogy második generációban a validáción jobb eredményt értünk el, kisebb a két adatsor hiba differenciálja is. A harmadik generációban nem működött megfelelően az algoritmus, ennek több oka is lehet. A tanulás is gyorsabban ment végbe.

A szűrés a 3-dik generáció után így néz ki:

```
I tensorflow/core/kernels/logging_ops.cc:79] wOut_u [15][6][1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 0 0]
I tensorflow/core/kernels/logging_ops.cc:79] bias_l [1][0][1]
I tensorflow/core/kernels/logging_ops.cc:79] weig_l [17][4][1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1]
```

5.10. ábra, paraméterek szűrésének loggolása

Itt azt láthatjuk, hogy a 3-dik generációnál 15 paraméter meg van tartva, és 6 ki van zárva a kimenet súlyozásából. A p_{wi} 17/4, ami azt jelenti, hogy 17 paramétere meg van tartva a 21-ből: $p_{wi}\ f_i$ Az egész el van tolva egy változóval („bias”-szal).

Elmondható, hogy jelen pillanatban kicsi mátrixról van szó, és nincs olyan sok túltanulás, hogy annak kiszűrése nagyobb javulást érjen el a validációs adathalmazon.

5.5.7 A második modell:

l1_g1 kódot kapta:

A modell felépítése a következő, egy lineáris modell a 21 „feature”-én értelmezve, eltolás, súlyozás, tanh node-dal a kimenetre kötve. Amihez gauss aktivációs függvény hozzá van kötve. f_x-szel jelölve az adat egy „feature”-jét a következőként írható le:

$$e^{-(p_{wi}\, f_i + p_{bi})^2}$$

Ez azt jelenti, hogy nem csak lineáris összefüggések vannak a modellben. Az alkalmazott modell rendkívül kicsi, de nagyobb modellek lassúsága és rossz teljesítményének köszönhetően ez bizonyult a legjobb választásnak.

A futtatás után a következő méréseket kaptam:

Adatsorok	gen_0	gen_1	gen_2
Tanító	0.69154	0.69033	0.69028
Validáló	0.69307	0.69140	0.69148
Numer.ai éles	0.69295	0.69287	0.69293

5.3. táblázat: number.ai-s éles adatsorokon történő eredmények.

Megjelent a táblázatban a „numer.ai éles”, ami egy olyan adatsort jelent, amelyhez csak a bemeneteket ismerjük és nekünk kell legenerálni a kimenetet. Feltöltve ezt a kimenetet a numer.ai szervezői egy értéket küldenek vissza, amely a logaritmikus hiba az adatsoron.

Egyértelmű javulás látható a tanulási és validálási szakaszon. Azonban valami hiba miatt nem sikerül javulni az éles adathalmazon. Már az első érték is rendkívül gyenge. Feltételezhetően a gauss aktivációs függvények nem elegendően jó megoldás. Ezektől a tényektől függetlenül a verseny eredményeken javulás ment végbe.

5.5.8 Éles adatsor tesztelésének konklúziója

A rendszer be lett üzemelve, hogy képes legyen feldolgozni éles adatsort. Bár a peremfeltételek nem teljesülnek az algoritmus használatára, a futtatásokat elvégeztem. A peremfeltételek teljesítésére nem volt lehetőségem mert kicsi a számítási kapacitásom. Ettől függetlenül az első generációnál javulás volt mérhető az éles adathalmazon, amely kis javulást hozott az eredményekben, lásd 6.2 fejezet.

Ezeket az éles adathalmazon történő futtatásokat nem az aktuális legjobb verzióval futtatam azóta több javítás történt az algoritmuson, például a hipermodellen, amely miatt érdemes lehet újra futtatni ezeket a méréseket. De idő hiánya miatt ebben a tudományos munkában, ez már nem kerül bemutatásra.

6. A megtervezett műszaki alkotás értékelése

Azt gondolom egy nagyon jó kezdeményezés arra, hogy automatizált módon javítsunk a mesterséges neurális hálók tanulási folyamatán a modell szűkítés segítségével. Az algoritmus egy új regularizációs módszert mutat, amely nincs hatással az illesztés folyamatára. Egyik legnagyobb erőssége a többi regularizációs módszerrel szemben, hogy külső megfigyelőként von le következtetéseket a modellről és az újabb generációk generálása közben végez regularizációt a paraméterek számán.

A felépített rendszer rendkívül komplex. A hipermodellek és feltételek rengeteg rugalmasságot biztosítanak, külön publikációs cikket lehet írni ennek optimális meghatározása. Az elért eredmények a tanító és validáló adathalmazokon megfeleltek az elvártaknak.

Összességében elmondható, hogy a tanító és validáló adathalmazon elért eredmények a módszer működését sikeresnek mutatták.

6.1 Kritikai elemzése

Numer.ai adatsorán a random tree módszerek célravezetőbbek. Amelyek közül egyik legeredményesebb az XGBoost [23] jelenleg. Azonban nem ilyen módszerre van kifejlesztve ez az algoritmus. Külön diplomamunka lenne kifejleszteni egy algoritmust, amely az ilyen rendszerek esetében is megfelelő hatékonysággal működik.

6.2 Nemzetközi versenyeredmény

Az algoritmussal sikerült javítani a modellen, és így a numer.ai honlapon elérhető tőzsdepiaci adatok megbecslésének nemzetközi versenyén 8-dik helyezést elérni.

6.1. ábra: numer.ai nemzetközi ranglista, részeredmény

7. Továbbfejlesztési lehetőségek

A dolgozat implementációja folyamán több lehetséges bővítési lehetőség merült fel, amelyek előre vihetik az algoritmust, hogy az letisztult állapotában bekerüljön az elfogadott regularizációs módszerek közé. Ezeket összegyűjtöttem.

Az algoritmus előreviteléhez, illetve továbbfejlesztésére számos hasznos opció létezik.

- Fel kellene használni a rendszert éles környezetben, ezáltal csökkenteni a paraméterek számát, gyorsítani az alkalmazott algoritmust és végső soron csökkenteni a tanulásra szükséges adathalmazt a modell szabadságfokainak (paramétereinek) csökkentésével.
- Éles környezetben történő használat esetén, szükséges a rendszer átgondolása más típusú modellek esetére is. Ez főként a függvénykészlet bővítését jelenti, illetve a megfelelő függvények „nullába kapcsolásának” automatizálását.
- A gráf szűrése helyett lehetne alkalmazni, az egyes részek teljes mértékű eltávolítását a modellből. Így memóriát lehetne spórolni. Ehhez előnyös, *sparse mátrix*okat támogató rendszerbe átvinni az algoritmust.
- Praktikus lenne kifejleszteni, egy egyszerű verzió kezelőt, amellyel lehetne kódolni a kiindulási gráfot majd annak szűrésével létrehozott gráfot. Ezek után lehetne több modellt építőelemként kezelni. Evolúciós algoritmusok irány.
- Fentebb kifejtettem milyen számos előnye lehet az algoritmus felhasználásának, hozzájárulna az algoritmus validálásához.
- Célom a vércukor predikciós kutatásomban felhasználni az algoritmust.
- Angol publikációban leközölni.

8. Fejezet
Köszönetnyilvánítások

Köszönet a TMIT tanszéknek és Barta Gergőnek akik támogattak abban, hogy ez a kutatás végig vihető legyen diplomatervként.

Köszönet illeti Havlik Tamást, aki hasonló szintén gépi tanulás témában kutatott és az egyes algoritmusok megértésének segítségével lehetővé tette, hogy *state of the art* algoritmusok is implementálásra kerüljenek a diplomatervemben.

Ezen kívül a numer.ai alapítójának Richard Craib-nek aki kiírta gépi tanulással foglalkozó személyek számára ezt a versenyt és biztosította az adatsort ha az indirekt módon történt is. A verseny segítségével tanultam meg milyen fontos a validáció, amely ihletet adott ennek az új módszernek a kidolgozására.

Irodalomjegyzék

[1] *Deep Learning in Neural Networks: An Overview*, J¨urgen Schmidhuber, The Swiss AI Lab IDSIA, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, University of Lugano & SUPSI, Galleria 2, 6928 Manno-Lugano, Switzerland, Október 2014

[2] *The wonderful and terrifying implications of computers that can learn*, Jeremy Howard Augusztus 2015

[3] Gradient Based Optimization Methods, Antony Jameson

[4] *Fundamentals of neural networks. Architectures, algorithms, and applications,* Laurene Fausettw

[5] *Wikipedia: Artificaial neural networks,* https://en.wikipedia.org/wiki/Artificial_neural_network (megnézve, 2016.05.02)

[6] *Wikipedia: Vanishing gradient problem* https://en.wikipedia.org/wiki/Vanishing_gradient_problem (megnézve, 2016.05.22)

[7] *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov

[8] *Regularization and variable selection via the elastic net*, Hui Zou and Trevor Hastie

[9] *Must Know Tips/Tricks in Deep Neural Networks,* http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html, by Xiu-Shen Wei (megnézve, 2016.04.12)

[10] *Local Minima and Plateaus in Multilayer Neural Networks,* Kenji Fukumizu and Shun-ichi Amari.

[11] *Cross validation in recurrent neural network,* http://www.mathworks.com/matlabcentral/answers/225901-cross-validation-in-recurrent-neural-network (megnézve, 2016.08.19)

[12] *Wikipedia: Cross validation*, https://en.wikipedia.org/wiki/Cross-validation_(statistics) (megnézve, 2016.05.06)

[13] *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems* (Preliminary White Paper, November 9, 2015)

[14] *WaveNet: A Generative Model for Raw Audio,* Aaron van den Oord, Sander Dieleman, Heiga Zen,Karen Simonyan, Oriol Vinyals, Alex Graves,Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu

[15] *An overview of gradient descent optimization algorithms*, http://sebastianruder.com/optimizing-gradient-descent/ (megnézve, 2016.08.17)

[16] *A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm*, Martin Reidmiller, Heinrich Braun

[17] https://www.coursera.org/learn/machine-learning (megnézve, 2016.05.20)

[18] *Adam: A method for Stochastic Optimization, Diederik P. Kingma*, Jimmy Lei Ba

[19] *Nesterov's Accelerated Gradient Descent*, https://blogs.princeton.edu/imabandit/2013/04/01/acceleratedgradientdescent/ (megnézve, 2016.08.18)

[20] *Gephi*, https://gephi.org/ (megnézve, 2016.10.25)

[21] *Numer.ai competition*, https://numer.ai/ (megnézve, 2016.05.20)

[22] *Logarithmic Loss*, https://www.kaggle.com/wiki/LogarithmicLoss (megnézve, 2016.04.01)

[23] *XGBoost: A Scalable Tree Boosting System*, Tianqi Chen, Carlos Guestrin