



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Mesterséges neurális hálózatok fejlesztése TensorFlow alapon

SZAKDOLGOZAT

Készítette
Kémény Károly

Belső Konzulens
dr. Strausz György

Külső Konzulens
Daróczy Bálint

2016. december 7.

Tartalomjegyzék

Kivonat	4
Abstract	5
Bevezető	6
0.1. Motiváció	6
0.2. A gépi tanulás igen rövid történelme	7
1. A Tensorflow[11] ökoszisztéma áttekintése	9
1.1. A Tensorflow könyvtár.	9
1.2. A Tensorflow modellek monitorozása és hibamentesítése	10
1.3. A Tensorflow futás közben	10
2. A Neurális hálózatokkal való képosztályozás lehetőségeinek áttekintése	12
2.1. Az algoritmusok kiértékelése	12
2.1.1. Az adathalmazok	12
2.1.2. Az adathalmazokon értelmezett tipikus metrikák	13
2.2. A Legelterjedtebb neurális hálózatok képfeldolgozáshoz	14
2.2.1. A többrétegű perceptron (MLP)	14
2.2.2. A Korlátozott Boltzmann Gép[20]	17
2.2.3. State of the art MLP hálózatok	21
2.2.4. konvolúciós hálózatok	23
2.3. További érdekes irányok a neurális képfeldolgozásban	26
3. Hálózatok impelmentálása és elemzése tensorflowban	29
3.1. A baseline osztályozók	29
3.2. A saját magam által kialakított fejlesztőkörnyezet	29
3.3. A kísérletek alatt használt optimizátorok	30
3.4. A saját implementációk bemutatása	30
3.4.1. A hálózatok monitorozása	30
3.4.2. A logisztikus regresszió	31
3.4.3. A többrétegű perceptron	31
3.4.4. Az RBM hálózat	32
3.4.5. A DBN struktúra	34

3.4.6. Hibrid modellek	34
3.4.7. A Konvolúciós modell	35
3.4.8. A konvolúciós modell skálázása	36
4. A mérési eredményeim összegzése	39
4.1. A keretrendszer általános teljesítménye	40
4.1.1. Rendszer szintű mérések	40
4.1.2. A GPU gyorsítás mérése	42
4.2. A baseline eredmények ismertetése	43
4.3. Az RBM struktúra optimalizálása	43
4.4. Az MLP-vel elvégzett mérésem eredményei	44
4.4.1. Az MNIST mérések	44
4.4.2. A CIFAR-10 mérések	45
4.4.3. Az előtanítás eredményeinek összefoglalása	46
4.5. A konvolúciós hálózatokkal elvégzett mérésem eredményei	47
4.6. A konvolúciós és az MLP hálózatok erőforrás igényének az összehasonlítása	49
4.7. A hibrid architektúrák eredményei	50
4.8. A CNN és az SVM+RBM összehasonlítása	53
4.9. A három megközelítés végső összehasonlítása	54
5. Az eredmények összefoglaló értékelése	55
Köszönetnyilvánítás	57
Irodalomjegyzék	61

HALLGATÓI NYILATKOZAT

Alulírott *Kemény Károly*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettettem, egyértelműen, a forrás megadásával megjelöltetem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervezek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2016. december 7.

Kemény Károly
hallgató

Kivonat

A szakdolgozatomnak két súlypontja volt, az egyik hogy elmélyítsem a tudásomat a mesterséges neurális architektúrák területén elméleti síkon, a másik hogy elsajátítsam a Tensorflow - ami egy elosztott, gráf alapú numerikus könyvtár a Google-től - ökoszisztémájának kezelését az előbb említett hálózatok programozásán keresztül. Mivel ez mindenféle keret nélkül hatalmas témakör lenne, ezért a dolgozat fókuszául a képosztályozást választottam.

A szakdolgozat először is arra a kérdésre próbál meg röviden választ adni hogy miért kell még ma is effektív tudással rendelkeznie az embernek a hálók működéséről ahhoz hogy valóban komplex rendszerekben használni tudja őket, még akkor is ha az eddig elkészült architektúrák nagy része sokszor előre elkészítve, sőt akár bizonyos feladatokra előre tanítva is elérhető. Ezután egy rövid történelmi összefoglalóval teszem kontextusba a dolgozat témakörét.

A dolgozat irodalom kutatással foglalkozó részében a Tensorflow ökoszisztémáját, és a mesterséges neurális hálózatokkal való képosztályozás elméletét tekintem át, számos nagyobb témakört érintve. Ezek sorrendben az előtanulás nélküli többrétegű perceptronok, az előtanított többrétegű perceptronok, a konvolúciós hálózatok, és a hibrid architektúrák. A hibrid architektúrákon belül kitérek két megközelítésre is. Az első amikor egy generatív architektúrát kombinálunk egy teljesen külön álló diszkriminatív osztályozával, esetbenben egy korlátozott boltzmann gépet egy szupport vektor géppel. A második amikor maga a neurális architektúra hibrid, tehát egyben modellben ötvözi a generatív és a diszkriminatív megközelítést. Ez a hibrid korlátozott boltzmann gépek megközelítése ami egy igen új fejlemény. Ezek után ismertetem a terület egy-két számomra izgalmas és új fejleményét.

A kísérleti részben az előbbiekben leírt elméleti részből kísérletezem számos architektúrával az MNIST és a CIFAR-10 adathalmazokon. Itt inkább előtére helyezem az egyes architektúrákat, és a hálózatok futás idejű karakterszitikáit, és kevésbé fókuszálok az olyan témaköröket mint a bemeneti adatok előfeldolgozása, a tanítás korai megállítása vagy a hiperparaméter térben való effektív keresés és ezek kombinációinak a (kereszt)-validációja. Az előbb említett pontok természetesen egy valós ipari környezetben nagy jelentőséggel bírnának, de egyenként is hatalmas témakörök amiknek a teljes körbejárása elvonná a figyelmet a dolgozat fő céljától, nevezetesen a különböző modellekkel való kísérletezéstől.

A dolgozatot az előbbi fejezetekből levont tanulságokkal zárom.

Abstract

I emphasised two main aspects in my B.Sc thesis. On one hand, I have wanted to deepen my theoretical knowledge in the area of artificial neural networks. On the other, I have wanted to master Tensorflow - a computational graph based distributed numerical framework from Google - through the programming of neural networks. However, it would have been an enormous topic to cover without any constraints, so I have chosen image classification as my topic of focus.

First, I explain the motivation behind my work. The reason why I think that it is still relevant to have profound knowledge of neural architectures if someone wants to use them in a complex application, even though there are a lot of architectures out there ready-made or even pretrained for a specific task.

In the literature research section I have glanced through the ecosystem of Tensorflow and the theory behind neural image classification, covering the most important topics I believe. These were as follows: multilayer perceptrons with and without pretraining, convolutional networks and the hybrid architectures. I have written briefly over two approaches regarding the hybrid solutions. The first one is when one has separately a generative model and a discriminative classifier, in my case this would be a restricted boltzmann machine for the former and a support vector machine for the latter. The second one is when these two are fusioned into one model, which is fascinating to me. I have found two such approaches, the hybrid restricted boltzmann machine (Larochelle 2008), and its stacked version the stacked boltzmann expert network (Alexander G. 2015). In the last section of my research I have highlighted some newer, intriguing advancements in the field.

In the measurements section of my thesis I did plenty of experimentation on some of the models that I have mentioned before, for evaluation I have used the MNIST and the CIFAR-10 datasets. I had to make a decisive decision which parts of a full experimental setting to neglect. I have chosen to pay little to no attention to the preprocessing of data, optimal early stopping algorithms for training, effective search and evaluation in the hyperparameter space. These could make good candidates for a thesis on their own, but they would have gravely drawn the attention from my main goal which was to experiment on a wide range of models.

I conclude my thesis with a brief summarization of what I have learned from the previous sections.

Bevezető

0.1. Motiváció

A neurális hálózat A neurális hálózat egy számítási modell amelyet számos, algoritmikusan nehéz problémára sikeresen lehet alkalmazni. A fő alkalmazási területeik: osztályozási feladatok, regressziós feladatok, dimenzió csökkentés és jellemző kiemelés. Ezt a modellt sikeresen alkalmazták már az ipar számos területén, a teljesség igénye nélkül:

1. *képfelismerés*: Ezen a területen talán a legsokrétűbb a felhasználásuk az egyszerű OCR (Optical Character Recognition - Optikai Karakter Felismerő) rendszerektől kezdve egészen a rákos daganatok detektálásáig elterjedtek.
2. *idősr előrejelzés*: Komplex idősr előrejelzésnél is előszeretettel használják őket, amikor a sok változót és azoknak összefüggését bonyolultságuknál fogva már nem lehet klasszikus statisztikai módszerekkel megragadni.
3. *beszédszintetizálás*: A 2016 szeptemberében publikált WaveNet struktúra 50%-ot volt képes javítani Mean Opinion Scoreban az eddigi state-of-the-art beszédszintetizáló rendszereken. Ezt a javulást nyelvfüggetlenül, angolban és mandarin kínában is képes volt tartani. [35]
4. *szabályozó algoritmusok*: Szintén 2016-ban a DeepMind AI alkalmazása a Google egy adatközpontjának a hűtés vezérlésében 40%-os esést eredményezett az erre fordított kiadásokban.

Ezek véleményem szerint mind kifejezetten izgalmas eredmények, tisztán látszik hogy a terület él, fejlődik és napról napra formálja a jelfeldolgozásról alkotott képünket.

Jelen Dolgozat célja A dolgozat fő csapásirányra egy áttekintő kép alkotása a neurális hálózatok felhasználásáról képosztályozási problémák esetében. A dolgozatomban arra törekszem hogy a neurális hálózatok gyakorlati alkalmazását ezen az alterületen keresztül ismerjem meg. A szakdolgozat egy hosszabb projekt része, ahol egy Tegra mobil chipen kell majd képosztályozó algoritmusokat alkalmazni ami egy robotrepülőgépen fog valós idejű döntéseket hozni. A dolgozat írása alatt értékelődik ki hogy a projektben a Tensorflow reális alternatíva lesz-e a további munkához, illetve hogy melyik képosztályzó eljárás felelne meg legjobban a céljainknak.

A feladat indokoltsága Az olvasó jogosan teheti fel magában a kérdést hogy ha ezek a hálózatok már léteznek, sőt sok esetben konfiguráció nélkül "out of the box" jelleggel használhatóak, akkor mi ad létjogosultságot egy ilyen bevezető jellegű szakdolgozatnak? Nos, habár az előző pont igaz, mégis manapság minél inkább érdemes tisztában lenni ezeknek az eszközöknek a képességeivel, és mind fontosabb a korlátaival. Ha valaki egy saját alkalmazásban szeretné őket használni, akkor érdemes tudni hogy:

1. Az adott alkalmazáshoz milyen háló típusok használhatóak.
2. Van-e esetleg már előre tanított modell a feladatunkhoz.
3. Ha nincs akkor mennyi idő lenne betanítani egyet.
4. Mennyi memóriát és számítást igényelnek az egyes modellek. (Ez erősen például függ a modell paraméter terének nagyságától)

A dolgozat kontextusa Ahhoz hogy kontextusba helyezem jelen munkámat a bevezető további részében szeretném egy rövid áttekintést adni a gépi tanulás történelméről.

0.2. A gépi tanulás igen rövid történelme

1. Az első elismerten tanuló gépet Arthur Samuelnek tulajdonítják 1955-ben, ez a konstrukció képes volt megtanulni dámajátékot játszani. Samuel algoritmusai heurisztikus keresési memóriát alkalmaztak annak érdekében hogy a saját tapasztalataikból tanuljanak. A hetvenes évek közepére ez a rendszer már képes volt emberi játékosok legyőzésére is.
2. Következő fontos pontként Frank Rosenblatt Perceptronját emelném ki, ez volt az első neurális hálózat, 1958-ban alkotta meg Rosenblatt az amerikai haditengerészet "US office of Naval Research" laboratóriumában. Már ezt is vizuális minták felismerésére alkották meg eredetileg. [37]
3. A hetvenes éveket csak úgy emlegetik hogy a mesterséges intelligencia tele, miután Marvin Minksy 1969-ben rámutatott a Perceptron korlátaira az emberek elvesztették az érdeklődésüket a terület iránt. [33] 1985-ben egy forradalmi újítás, a hibavisszaterjesztéses algoritmus (backpropagation algorithm [38]) törte meg a csendet és keltette fel az emberek érdeklődését újfent ezen számítási struktúrák iránt.
4. A kilencvenes években a neurális hálózatok újra kikerültek a középpontból, mert a statisztikusok által alkotott szupport vektor gépek (továbbiakban SVM) lényegesen jobb teljesímenyt tudtak elérni, kevesebb tanítással mint a kor neurális hálózatai. Ezek is hálózati modellek voltak, de egy sokkal rigidebb, matematikailag sokkal jobban alátámasztott megközelítéssel. [13]
5. A neurális hálózatok következő virágkorát napjainkban éljük, ennek egyik fő tényező a fejlett neurális struktúrák felfedezése, illetve az hogy a grafikus egységek és a

számítási fürtök fejlődésének köszönhetően eddig elképzelhetetlen számítási kapacitás áll rendelkezésünkre a hálózataink tanítására. Ezzel szemben a kernel gépeken alapuló modellek nem tudtak a megnövekedett teljesítményt kihasználva a neurális hálózatokhoz hasonló pontosság növekedést elérni sok - az emberek számára igen fontos - területen, mint a hang és képjelek feldolgozása és szintetizálása. Ezért habár számos problémára az SVM még mindig jobb megoldást ad a neurális architektúráknál, a neurális hálózatoknak napjainkban sokkal nagyobb érdeklődés övez. A manapság a neurális hálózatok jelen vannak az élet minden területén ahol szükségünk van mintázatok intelligens felismerésére, lehet a tárgya hang, kép vagy akár szöveges dokumentumok. Google translate, Apple Siri vagy akár a Tesla Mobil önvezető technológiája, csak hogy pár példát szemelvénnyezek a számtalan közül.

A dolgozat felépítése A dolgozatomat az alábbi építem fel:

1. A tensorflow mint neurális rendszerek kutatására, fejlesztésére és éles üzembe helyezésére alkalmas platform bemutatása.
2. A tárgyterület irodalmának áttekintése, szemlézve a következő struktúrákat:
 - (a) Egyszerű többrétegű perceptron.
 - (b) Korlátozott boltzmann gépek, és variánsai
 - (c) Konvolúciós Neurális hálózatok
 - (d) A friss fejlemények a neurális képosztályozás területén.
3. A saját fejlesztésem bemutatása, amely egy két egyszerűbb struktúra implementálása és vizsgálata a fent bemutatottak közül, a tensorflow könyvtárral.
4. A mérési eredményeim kiértékelése, tanulságok levonása.

1. fejezet

A Tensorflow[11] ökoszisztéma áttekintése

1.1. A Tensorflow könyvtár.

A technológiai választás indoklása A modern szoftvermérnöki munka szerves része a rendelkezésre álló eszközök garmadájából a legmegfelelőbb kiválasztása. Ez a kínálat mértéke, az információ elszórtsága és ellentmondásossága miatt koránt sem egy egyszerű feladat. A címből talán úgy sejlik hogy a tensorflow a munkámhoz már egy előre eldöntött választás volt, de ez a megállapítás koránt sem lenne helytálló. A szakdolgozatom nulladik lépéseként számos - a neurális hálózatok fejlesztését támogató - könyvtárat vettet szemügyre: Caffee, Chainer, CNTK, Matlab, Tensorflow, Thenao, Torch. A következőkben szeretném bemutatni az általam választott eszköz a Tensorflow felépítését, és ezzel mintegy implicit módon megindokolni hogy szerintem miért ez az egyik legmegfelelőbb eszköz a neurális hálózatokkal való munkához.

Bevezető Munkám során a Tensorflow nevű könyvtárral dolgoztam. A Tensorflow egy elosztott, számítási gráf alapú numerikus könyvtár. A Goolge Deep Mind kutatócsoport szakemberei hozták létre azzal a céllal hogy saját modelljeiket fejlesszék és értékeljék ki benne. A könyvtár a DistBelief nevű keretrendszer egyenesági leszámazottjának tekinthető. A DistBelief csendben meghúzódva, de ott dolgozik a fejlett világ majdnem minden emberének az élete mögött, lévén hogy az Alphabet cégcsoport (A Google feldarabolásának utódvállalait összefogó holding) több mint 50 csapata adaptálta, és vértezte fel általa intelligenciával alkalmazását. Pár ismertebbet kiemelve: Google Search, Adwords, Google Maps, StreetView, Youtube, és természetesen a Google Translate. Miután évek tapasztalata gyülemlet fel az első generációs könyvtárak használata során, úgy érezték itt az idő hogy - szakítva a technológiai teherrel amit az első generáció hibái miatt magukkal hordoztak - létrehozzák a következő generációs gépi tanulás rendszerüket, ez lett a Tensorflow aminek a fő célja skálázható, elosztott gépi tanulási algoritmusok (főképp neurális hálózatok) fejlesztése[11].

A Tensorflow alapgondolata. Mint már említettem a Tensorflow könyvtárban az ember a modelljeit egy adatfolyam-szerű számítási gráfként definiálhatja. Ennek a megközelítésnek az a nagy előnye hogy nagyon jó skálázódási tulajdonságokkal rendelkezik. Miután a gráf csomópontjai az egyes számítások, ezek adott esetben hatékony módon szétoszthatók különböző eszközök, vagy akár egész gépek között szerverparkokban. A könyvtár ezen tulajdonságának még egy hosszabb részt fogunk később szentelni. A létrejött gráfra mint egy alaprajzra érdemes gondolni, amit aztán az egyes munkamenetek (session-ök) példányosítanak. Ezek a munkamenetek inicializálják a változókat, és a munkamenet képes a gráf egyes csomópontjait lefuttatni, amik igény vezérelt módon minden a bemenetükre kapcsolódó csúcsot lefuttatnak amíg el nem érnek egy bemenetig vagy egy kívülről betáplált változóig. Miután a csomópont sikeresen lefutott a kimenetét a csatolt nyelv egy változójaként adja vissza, például egy Python vagy C++ tömbként. Fontos megjegyezni hogy futás közben a gráffal nem lehet érintkezni, az egy atomi egységként fut le, hogy minél jobban ki lehessen optimalizálni a számításokat.

1.2. A Tensorflow modellek monitorozása és hibamentesítése

A Tensorboard Mivel a modern gépi tanulás modellek hihetetlen összetettséggel bírnak, és nagyon sok mozgó alkatrészük van, ezért természetesen felmerül az igény hogy egy ilyen újszerű keretrendszerben ipari erősségű monitorozó és hibakereső funkciók kerüljenek bele. Ezeket az elvárásokat a Tensorflow esetében a mellékelt Tensorboard alrendszer teljesíti, amelyet munkám során én is extenzíven használtam. Ebből kifolyólag most nem is bocsájtanám bővebb tárgyalásra, hanem majd az önálló munka szekcióban ismertetnémm.

1.3. A Tensorflow futás közben

A Tensorflow serving kiszolgáló rendszer Miután a kutatólaborokból kikerülték az új gépi tanulás modellek nem elég őket csak publikálni, a cégek komoly hasznót remélnek tőlük. A Google is kijelentette hogy "Information Retrieval first company" helyett ők most már egy "AI first company". Ez természetesen egy hozzá illő infrastruktúra nélkül elképzelhetetlen. Ezért is hozták létre a Tensorflowhoz a Tensorflow Servinget, ami egy flexibilis, magas rendelkezésre állású kiszolgálórendszer. A rendszer lehetővé teszi új architektúrák kipróbálását, üzembe helyezését és A/B tesztelését, miközben egy stabil, verziózott API-t biztosít a kliensek számára. Természetesen ez mit sem érne ha nem skálázódna gond nélkül, ezért egyszerűen integrálható a Kubernetes[4] névre hallgató docker alapú cluster kezelő rendszerrel.

A tensorflow skálázodása Érdemes belegondolni hogy az adatközpont TCP (DCTCP) vagy az Infiniband kapcsolatok adott esetben akár több GB/s sebességet érhetnek el, ugyanakkor a mátrix szorzás - ami a gépi tanuló algoritmusoknak egy kardinális eleme - számításigénye négyzetes ordót igényel. Tehát sokkal jobban megéri részgráfokat a csomópontok között szétosztani és inkább a hálózati többlettel kalkulálni, mint hogy egyetlen gépre bízzuk a feladatokat. A másik végletben viszont egy másik elvárás helyezkedik el. Miután

mondjuk egy osztályozási feladatnál a modellünket megtanítottuk felismerni valamit, tegyük fel hogy például egy, a látássérült embereknek készített alkalmazásban felismerni az forgalomjelző lámpákat és azok állapotát, természetes lenne az igény hogy ezt a rászoruló magával tudja vinni. A háló ugyanaz, a súlyokat már megtanultuk, de most az egész modellünket egy mobil eszközön kell futtatni. Ez az eszköz az inferenciát könnyedén bírná, csak a tanulást nem tudtuk volna kivitelezni rajta. Mérnökként logikus az igény hogy ehhez ne kelljen mégegyszer lefejleszteni a modell-t, így időt és pénzt spórolva. Erre lehetőség van a keretrendszerrel.

Mobil környezet A Tensorflow támogatja a mobil eszközön való futást, az előbbi szcenárió a könyvtárral egy teljesen járható úttá válik. A modell-t asztali számítógépen fejlesztem, adatparkon tanítom, és egy mobil eszközön futtam, mindezt jelentősebb kód újraírás nélkül. Jó példája ennek a felhasználási módnak a Google Translate, legújabb, nagy felhajtást elérő verziója. Ez az alkalmazás a nyelvi modelleket a Google irdatlan infrastruktúráján tanul folyamatosan, miközben az inferenciát a telefonkészüléken futtatják lokálisan. Itt igazándiból két neurális háló is szerepet játszik, az egyik a szöveget ismeri fel a képen, a másik pedig az effektív fordítást végzi.

TF-Slim[9] és Keras[3] A Tensorflow alap API-ja nagyon alacsonyszintű fejlesztést tesz lehetővé, ami jó ha az ember valami gyökeresen új operációt szeretne megvalósítani, viszont nem megfelelő prototipizálásra. Erre jelent meg a TF-Slim és a Keras könyvtár amik az alap Tensorflowra illeszkedve adnak magas szintű utasításkészletet. Én a szakdolgozatomban a TF-Slim könyvtárat használtam sokat.

2. fejezet

A Neurális hálózatokkal való képosztályozás lehetőségeinek áttekintése

A kutatómunka jelentős részét tette ki a dolgozatomnak, mivel az évek folyamán nagyon sok sikeres és sikertelen kísérlet született annak érdekében hogy hogyan lehetne neurális hálózatokkal képeket osztályozni. Talán mondhatjuk hogy ezek a hálózatok a legsikeresebb képosztályozó eljárások, de korántsem triviális hogy mik az előnyeik, hátrányaik és az egyes típusok milyen komplexitású jelekkel képesek megbirkózni. Először szeretném bemutatni az adathalmazokat amiken ezeket az algoritmusokat kiértékelik, majd eljutni az többrétegű perceptronuktól a konvolúciós hálózatokig, végül egy-két újabb trend ismertetésével zárnai a fejezetet.

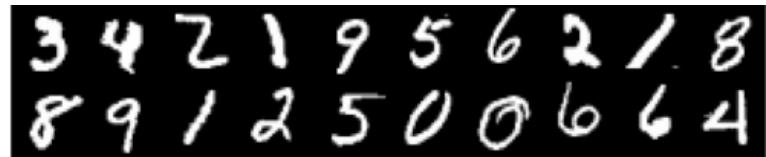
2.1. Az algoritmusok kiértékelése

2.1.1. Az adathalmazok

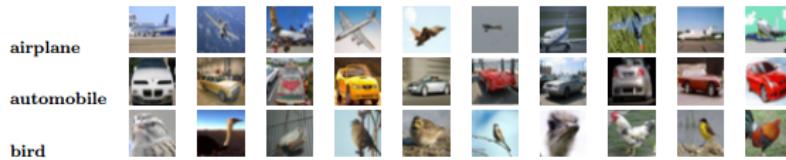
Bevezetés Mint a legtöbb kutatási területnek, ennek is vannak jól ismert "benchmark" adathalmazai, amelyek viszonyítási alapként lehetőséget biztosítanak az egymástól eltérő algoritmusok egymáshoz képesti kiértékelésére. Mivel ezekre az adathalmazokra sokat fogok hivatkozni, ezért szeretném őket egy-egy bekezdésben bemutatni.

MNIST[5] Az MNIST adatbázis fekete-fehér, 28x28 pixelre normalizált írott számjegyeket tartalmaz nullától kilencig, azaz tíz osztállyal rendelkezik. Az adatbázis 60'000 annotált tanító képet és 10'000 annotált teszt képet tartalmaz. Ez a legismertebb adathalmaz. Az adathalmazból a 2.1. ábra mutat egy-két példát.

CIFAR-10[1] A CIFAR-10 egy jóval összetettebb adathalmaz, 60'000 annotált 32x32 pixeles, színes képet tartalmaz. A képek 10 osztállyra vannak felosztva, osztályonként 6000 képpel. Az adathalmazból 50'000 kép van tanításra, és 10'000 tesztelésre fenntartva. Az adathalmazból a 2.2. ábra mutat egy-két példát.



2.1. ábra. Példa az MNIST adathalmaz képeire. Forrás: <http://knowm.org/wp-content/uploads/Screen-Shot-2015-08-14-at-2.44.57-PM.png>.



2.2. ábra. Példa a CIFAR-10/100 adathalmaz képeire. Forrás:<http://www.cs.toronto.edu/~kriz/cifar.html>

CIFAR-100[1] A CIFAR-100 felépítése megegyezik a CIFAR-10el, de osztályrendszere az előbbinél lényegesen összetettebb, 100 osztályt tartalmaz és minden osztályhoz 600 kép tartozik, ezen felül még 20 általánosabb osztályba is be vannak sorolva a képek, hogy a hálózat általánosításáról következtetéseket lehessen levonni. Például az halak szuperosztályhoz tartozik a rája, lazac, stb. Példának úgynégy a CIFAR-10 mintája, a 2.2. ábra tekinthető.

IMAGENET[2] Az IMAGENET a világ legnagyobb képgyűjteménye, a WordNet lexikális adatbázis szinoníma halmazai szerint vannak annotálva a képek. Jelenlegi statiszkái:

- 14'197'112 annotált kép
- 21'841 nem üres szinoníma halmaz
- 1'034'908 kép objektumaihoz van még határoló doboz annotáció is
- 1'000 szinoníma halmazhoz tartozik SIFT jellemzőkkel ellátott kép
- 1'200'000 kép van SIFT jellemzőkkel ellátva.

Látható hogy az előző három adathalmazt az IMAGENET már csak pusztta méreteivel is messze túlszárnyalja. Ezt mondhatjuk az etalon benchmarknak. Az évente megrendezett, a gépi látás "olimpiájának" számító ILSVRC(ImageNet Large Scale Visual Recognition Challenge) is ezen az adatsokaságon szokott megrendezésre kerülni, jellemzően 4 kategóriában: objektum lokalizáció, objektum detekció, helyszín felismerés (pl tengerpart, hegyek), helyszín megértés. A legutóbbi nem takar kevesebbet mint egy kép szemantikus részekre való felosztása, például út, ég, ember vagy ágy. A 2.3. ábra ebből az adathalmazból mutat pár példát.

2.1.2. Az adathalmazokon értelmezett tipikus metrikák

MNIST és CIFAR Az MNIST és a CIFAR-10/100 Adathalmazok esetében mindig az egyszerű pontosság értéket nézzük, tehát az eltalált képek számát osztva a hibásan osztályozott képet számával.



2.3. ábra. Példa az IMAGENET adathalmaz képeire. Forrás: <http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2014/12/imagenet.jpg>.

IMAGENET Mivel az IMAGENET egy ennyire bonyolult adathalmaz, itt top 1 és top 5 hibát is szoktak nézni. A top 5 hiba ahol az számít sikeres találatnak ha a helyes címkét a háló 5 legnagyobb valószínűséggel bíró tippjében benne van.

2.2. A Legelterjedtebb neurális hálózatok képfeldolgozáshoz

Bevezetés Az évek során számos neurális hálózattal kísérleteztek a kutatók annak érdekében hogy rájöjjönek melyek képesek legjobban megtanulni a képeken előforduló szabályosságokat, és ez alapján osztályozni őket. Ezekből szeretném a fő állomásokat kiemelni, és leírni hogy mik voltak a hiányosságok a meglévő architektúrákban amik új struktúrák létrehozását motiválták. Az áttekintésben nem ejtek szót a minden hálózat típusra érvényes általános, különféle reguralizációs eljárásokról, mint a súlyok felejtése[34] vagy a dropout metódus[42]. Csak a hálózatok felépítésének és tanításának architektúrálás különbségeit veszem górcső alá.

2.2.1. A többrétegű perceptron (MLP)

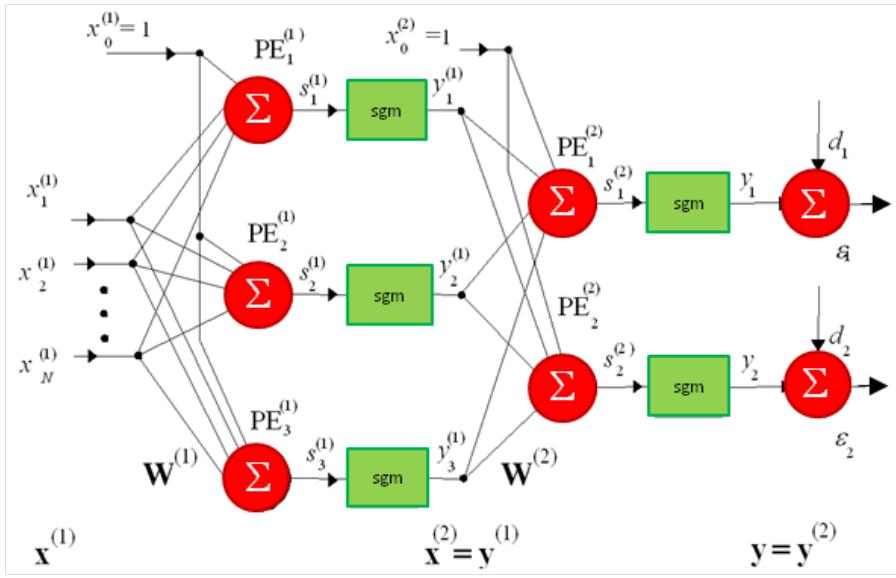
Az MLP előzményei és megalkotása

Előzmények Miután Rosenblatt megalkotta az első perceptron struktúrát a kései ötvenes években[37], a kutatók elkezdtek azon gondolkodni hogy hogyan lehetne ezeket a neuronokat összerendezni úgy, hogy együtt tanuljanak, és komplex regressziók, illetve osztályozási feladatok elvégzésére legyenek képesek. De ezek a kutatások sokáig igen meddőek voltak.

Backpropagation Az áttörés 1986-ban jött, amikor Geoffrey E. Hinton kollégáival sikeresen alkalmazta a hibavisszaterjesztéses algoritmust[38] az MLP súlyainak megváltoztatására a négyzetes hiba minimalizálásának érdekében. Az algoritmus lényege hogy a hibát a hálózatban a deriválás lánc szabályának segítségével terjesztjük vissza. Az algoritmust helymegtakarítás érdekében részletesebben nem ismertetem, az érdeklődők a bekezdés elején referált cikkben további részleteket találhatnak. Az algoritmus pszeudokód összefoglalását a 2.4.-ábrán látható hálózat tanításához a 2.1.-lista mutatja.

Az MLP teljesítménye képosztályozási feladatokra

Aktivitás a területen. Gondolhatnánk hogy ezt a témát már rögtön elfejezték a kutatók, de mivel az MLP egy igen egyszerű struktúra, ezért folyik még néhány kutatás hogy a



2.4. ábra. Egy kétrétegű MLP hálózat. Forrás:<https://mialmanach.mit.bme.hu/neuralis/ch04s01>

2.1. lista. A backpropagation algoritmus pseudokódja

```

inicializáljuk a háló súlyait (általában 0-1 közé eső véletlen számok)
do
    forEach tanító példa legyen tp
        jósolt_címke = háló-kiemelt(háló, tp)
        valódi_címke = tanító_címke(ex)
        hiba számítás f(jósolt_címke - valódi_címke) minden kimeneti egységen
        ΔW(2) kiszámítása
        ΔW(1) kiszámítása
        a hálózat súlyainak frissítése
    until Az összes bemenet sikeresen van osztályozva,
          vagy más megállási kritériumot el nem értünk
    return a hálózatot
  
```

határait megtalálják.

MNIST Az MLP teljesítménye képosztályozási feladatok tekintetében igen szerény a többi hálózathoz képest, de az MNIST adathalmazzal még ez is egész jól megbirkózik, néhány figyelemre méltóbb eredményt a 4.6. táblázat foglal össze. A többi adathalmazon a naiv MLP nem hoz értékelhető eredményt, ennek az okait mindenkor megvizsgáljuk.

2.1. táblázat. Az MLP teljesítménye az MNIST adathalmazon

Rétegek száma	neuron struktúra	Teszt szet hiba százalék
2-réteg	300-10	4.7
2-réteg	800-10	1.6

Az MLP hálózatoknál jelentős javulást hozott amikor a hibafüggvényt négyzetes középértékről(2.1) kereszt-entrópiára(2.2) cseréltek ki. Ma már osztályozási feladatoknál szinte csak a kereszt-entrópia hibafüggvényt használjuk.

$$L = - \sum_{i=1}^n y_i \log(\hat{y}_i) \quad (2.1)$$

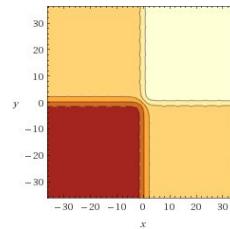
$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (2.2)$$

Ahol y_i az eredeti címke és \hat{y}_i a jóolt címke.

MLP hiányosságai

A tanulás jellege A felügyelt tanulás kapzsi módon a hibafüggvényt a súlyok gradiensének irányába optimalizálja, ezzel az a probléma hogy a hibafelület egy MLP esetében többé nem konvex mint egy egyszerű neuron esetében. A bonyolult hibafüggvény következtében lokális minimumok alakulnak ki. Sok esetben egy jó lokális minimumot sem érünk el naiv tanítással, a globális minimum elérésének az esélye pedig statisztikailag nulla. Ezt a jelenséget hivatott szemléltetni az alább egyszerű függvény $\sigma^2(\sigma(x) + \sigma(y))$ kontúr diagramja (2.5. ábra). Ez habár nem közvetlenül egy hibafüggvény, de azt a tulajdonságot jól szemlélteti, hogy több fennsík alakul ki, ahol a derivált 0 lesz.

A struktúra kialakítása Az MLP annyira általános struktúrát használ, hogy lényegében semmilyen a priori tudást nem használunk fel a hálózat tanításakor. Ez azt eredményezi hogy hatalmas kapacitás kell a képekben megjelenő bonyolult struktúrák megtanulásához, és a hálózat különböző részei kényetlenek rendre ugyanazt megtanulni. Sajnos az előbbi cél csak a hálózat növelésével érhető el, az MLP paraméter tere viszont nagyon rosszul skálázódik. Ha veszünk egy 6 rétegű hálózatot aminek a rétegei rendre 2500-2000-1500-1000-500-10 neuronból állnak, és az MNIST esetén 784 elemű bemeneti vektorral rendelkezik, akkor optimalizálandó paraméter tér mérete annak folytán hogy minden réteg teljesen össze van kapcsolva már: $784 * 2500 + 2500 * 2000 + 2000 * 1500 + 1500 * 1000 + 1000 * 500 + 500 * 10 = 11965000$, ami már nyilvánvalónan hatalmas. Ez is tanítható sikeresen, de ahhoz már a továbbiakban bemutott kisegítő neurális struktúrák szükségesek.



2.5. ábra. Példa egy komponált szigmoid függvényre.

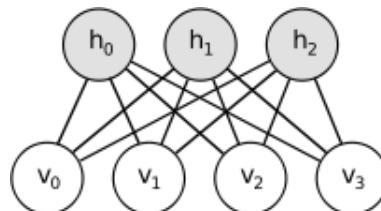
konkluzió Jól látszik hogy az MLP hálózatok ilyen naív formájukban nem igazán felelnek meg az elvárásainknak, ezt a későbbiekben bemutatott mérései eredményeim meg is erősítik.

A továbbiakban bemutatok egy kisegítő neurális struktúrát, amivel sikeresen inicializálták az MLP súlyait hogy jobb eredményeket érjenek el.

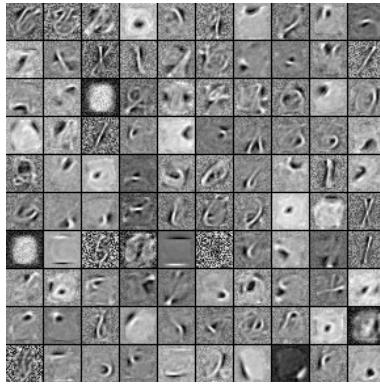
2.2.2. A Korlátozott Boltzmann Gép[20]

Bevezetés A korlátozott boltzmann gép (továbbiakban RBM - Restricted Boltzmann Machine, 2.6. ábra) egy sztochasztikus generatív neurális számítási modell. Működése az eddig tárgyalt MLP-től györekesen eltér, funkciója a bemenet jellemzőinek (featureinek) a megtanulása, és nem az egyes minták helyes osztályozása. Erre mutat példát a 2.7.-ábra. Az intuitív jelentősége abban rejlik hogy feltehetjük hogy a naív MLP a kapzsi tanulása miatt nem képes megtanulni a minták valódi reprezentációját, de ha az MLP súlyait úgy tudnánk inicializálni, hogy a mintákban lévő szabályosságokat már eleve ismerje, akkor ebből könnyebben meg tudná tanulni hogy melyik jellemző mely osztályt azonsítja. Ezt felügyelt tanulás előtti fázist nem felügyelt előtanulásnak hívjuk. Szokás még erre a célra Autoencoder hálózatokat használni, illetve mélyebb hálókra az RBM és az autoencoder többrétegű megfelelőit a mély hiedelem hálózatokat[40], és a stacked autoencodereket[44]. Habár a legújabb eredmények szerint az Autoencoder hasonlóan jó eredményre vezet, és kevésbé bonyolult ezért gyakorlatban az ajánlott, én mégis az RBM-et választottam érdekes struktúrája miatt. Hugo Larochelle et al. publikált egy hosszabb, több módszert tartalmazó cikket az előtanulásról a mélyebben érdeklődőknek "Exploring Strategies for Training Deep Neural Networks" [29] címmel.

Az RBM struktúrája Az RBM mint említettem egy sztochasztikus generatív számítási modell, amelyben fontos hogy az egyes neuronok egy páros gráfot alkotnak (2.6. ábra). A generatív modell egy régi, statisztikából származó fogalom ami azt foglalja magában hogy a model képes megfigyelhető adatpontokat véletlenszerűen generálni. Az RBM egyik legtriviálisabb mérőszáma a rekonstrukciós hiba azt méri hogy ha egy adatpontot a háló bemenetére teszek, akkor azt milyen részletesen tudja visszagenerálni. Tehát az adatpont az a háló tanulási terében egy stabil pontnak számít-e. Ez a hiba mérték nem jó az RBM általánosító képességének mérésére, mégis sokan használják praktikus egyszerűsége miatt. Akit bővebben érdekel a téma a "A practical guide to training restricted boltzmann machines"[20] referenciában talál bőséges irodalmat az RBM tanítását illetően. Itt csak az alapokra szorítkozok.



2.6. ábra. Egy RBM hálózat. Forrás: http://deeplearning.net/tutorial/_images/rbm.png



2.7. ábra. Egy RBM által megtanult filterek az MNIST adathalmazra. Forrás:
http://www.pyimagesearch.com/wp-content/uploads/2014/06/rbm_filters.png

Az RBM tanítása Ebben a bekezdésben belemegyek kicsit az RBM tanításának részleteibe, mert az implementáció tárgyalásánál fontos lesz. Az RBM azért érdekes megközelítés a többi hálózathoz képest, mert probabilisztikus alapokon nyugszik. Az úgynevezett energia alapú hálózatok felfoghatóak úgy, hogy a hálózat minden konfigurációjához tartozik egy $p(x)$ valószínűség, hogy mekkora valószínűsséggel tartózkodik a háló az adott konfigurációban. Azt szeretnénk elérni hogy az alacsony energiájú konfigurációknak nagy legyen a valószínűsége. Ez formalizálva a következőképpen néz ki:

$$p(x) = \frac{e^{(-E(x))}}{Z} = \sum_h \frac{e^{(-E(x,h))}}{Z} \quad (2.3)$$

$$Z = \sum_x e^{-E(x)} \quad (2.4)$$

Ahol az E az energiafüggvényt, Z pedig a partíciós függvényt jelenti. A fizikában jártasabb olvasók megfigyelhetik hogy ez a valószínűségi függvény megfelel a termodinamikában használt Boltzmann eloszlás valószínűségi függvényének. Az eredeti boltzmann gépet egy fizikus alkotta meg, pont erre az analógiára építve[10]. A cél az volt, hogy a Hopfield hálózatok gyengeségeit kiküszöbölje, a (2.3) képletet felhasználva megalkothatjuk a hibafüggvényünket, amely a negatív logaritmikus valószínűségi függvény (negative log likelihood function) lesz:

$$\mathcal{L}(\theta, x) = \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log(p(x^{(i)})) \quad (2.5)$$

Ahol a \mathcal{D} az adatpontok halmaza, N az adatpontok száma, θ pedig a paraméter tér.

Definiáljuk a szintén a termodinamikából származó szabad energia függvényt:

$$\mathcal{F} = -\log \sum_h e^{-E(x,h)} \quad (2.6)$$

Ahol x a megfigyelhető neuronok halmaza, h pedig a rejtett neuronok halmaza.

Ezzel újradefiniálhatjuk a valószínűségi függvényt:

$$p(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \quad \text{ahol} \quad Z = \sum_x e^{-\mathcal{F}(x)} \quad (2.7)$$

Ami megengedi hogy a következő gradienst írhassuk fel:

$$-\frac{\partial \log p(x)}{\partial \theta} \approx \frac{\partial \mathcal{F}(x)}{\partial \theta} - \frac{1}{|\mathcal{N}|} \sum_{\hat{x} \in \mathcal{N}} \frac{\partial \hat{x}}{\partial \theta} \quad (2.8)$$

Ahol \mathcal{N} az úgynevezett negatív minták halmaza, ezek a modell által generált minták amiket lehetőség szerint a $p(x)$ eloszlásból szeretnénk generálni, erre egy MCMC (Markov Chain Monte Carlo) metódust használunk aminek Gibbs mintavételezés a neve. Ezt az eljárást lentebb fejtem ki.

Ha az előbbi függvényt deriváljuk, akkor megkapjuk a súlyváltozók update függvényét:

$$-\frac{\partial \log p(v)}{\partial W_{ij}} = E_v[p(h_i|v) * v_j] - v_j^{(i)} * \text{sigm}(W_i * v^{(i)} + c_i) \quad (2.9)$$

$$-\frac{\partial \log p(v)}{\partial c_i} = E_v[p(h_i|v)] - \text{sigm}(W_i * v^{(i)}) \quad (2.10)$$

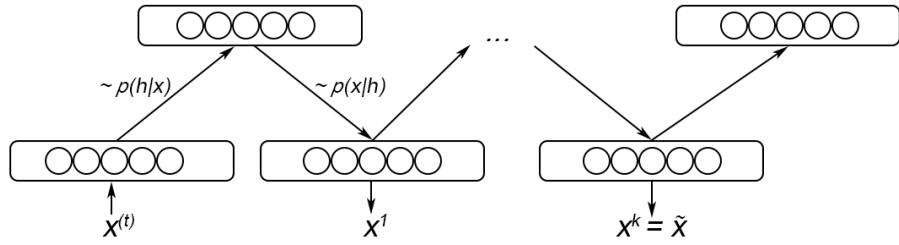
$$-\frac{\partial \log p(v)}{\partial b_j} = E_v[p(h_i|v) * v_j] - v_j^{(i)} \quad (2.11)$$

Ahol W a súly paraméterek mátrixa, v pedig a megfigyelhető neuronok halmaza. Ebből látható hogy az egyes deriváltak kiszámításához szükségünk van a rejttet neuronok valószínűségére a bemeneti neuronok állapotától függően. Ezért szükséges hogy a boltzmann gép korlátozott legyen, tehát egy páros gráf formáját vegye fel, mert így az egyes neuronokhoz tartozó valószínűségek párhuzamosítva számolhatóak, mivel függetlenek egymástól. Erre egy gyors eljárás a kontrasztív divergencia algoritmus amelynek a valószínűségi mintavételét a 2.8. ábra szemlélteti. A Gibbs mintavételezéshez tartozó markov lánc lépéseiinek a képleteit a (2.12) és a (2.13) képlet írja le.

$$h^{n+1} \sim \text{sigm}(W^T v^{(n)} + c) \quad (2.12)$$

$$v^{n+1} \sim \text{sigm}(W^T h^{(n)} + b) \quad (2.13)$$

Ahol c és b az eltolás súlyvektorokat jelzik. Ezekre a képletekre fogok majd a dolgozat későbbi részeiben hivatkozni, amikor az általam megírt RBM struktúrát optimalizálom. Az RBM hálózatok részleteiről részletes leírást a <http://deeplearning.net/tutorial/rbm.html> linken találnak az érdeklődők.



2.8. ábra. A gibbs mintavételezési eljárás. Forrás: <http://recognize-speech.com/images/nicolas/Gibbs.png>

A DBN[40] A DBN (Deep Boltzmann Machine - Mély Boltzmann Gép) a legegyszerűbb formájában egymásra helyezett RBM rétegek sorozata, előnye hogy a generatív modell képes hierarchikus jellemzőkiemelésre.

Az RBM és DBN előtanulás Az előbbiekben bemutatott tanítással megcsinálhatjuk azt, hogy egy MLP egyes rétegeit rétegenként előtanítjuk. Így a felügyelt tanítás elején az MLP súlyai már rögtön a bemenet tulajdonságaira lesznek rászabva.

Az RBM és DBN előtanulás eredményei A hálók előtanításával jelentős teljesítmény javulást értek el a kutatók nagy paraméter térrrel rendelkező hálózatok esetében. Ez azt bizonyítja hogy valóban a jellemző reprezentációhoz közel helyezkedik el egy nagyon optimális lokális minimum. Az 2.2. táblázat szemlélteti az eredményeit az MNIST adathalmazon. Látszik hogy ennek segítségével jóval nagyobb hálók képezhetőek ki és lényeges jobb eredményre vezetnek.

2.2. táblázat. Az MLP teljesítménye az MNIST adathalmazon RBM előtanulást használva. Forrás: [5]

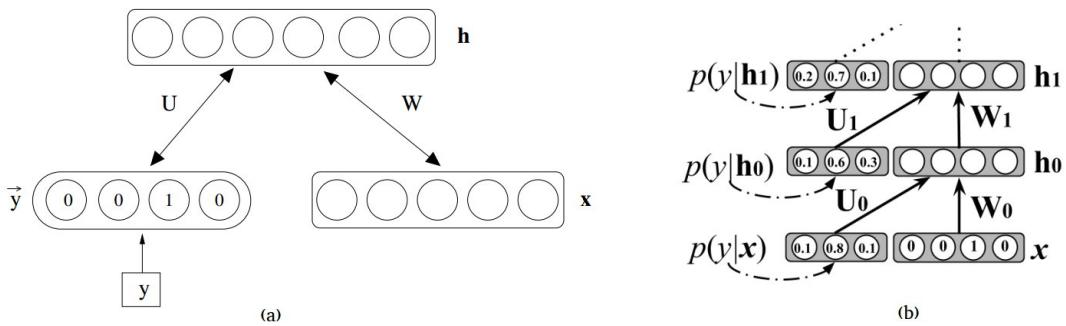
Rétegek száma	neuron struktúra	Teszt szet hiba százalék
2-réteg	800-10	0.7
5-réteg	2500-2000-1500-1000-500-10	0.35

További módszerek az RBM használatára osztályozáshoz Az előtanulásban nem merül ki az RBM jellegű hálózatok potenciálja, ha osztályozási feladatokról van szó, egy pár további lehetőséget szeretnék még érintőlegesen megemlíteni.

- Egy másik, lehetőség szerint az egyes osztályoknak egy RBM-et hozunk létre, és egy Softmax modell-t tanítunk a szabad energia függvényükön. A partíciós függvényt ebben az esetben a softmax paraméterei approximálják.[20]
- Egy harmadik lehetőség hogy két külön látható réteg csoportot tartunk, egyet a kép adatoknak, egy másikat pedig a tanító címkeknek, ebben az esetben az osztályhoz tartozási valószínűséget a teszt vektor, és a címkek szabad energiájának eloszlásában határozzunk meg. Tehát minél alacsonyabb lesz a szabad energia egy címkelvel, annál

valószínűbb hogy a teszt vektor abba az osztályba tartozik, itt végülis tanulásnál a címkék és a mintapontok együttes valószínűségi sűrűségfüggvényét becsüljük.[20]

- Egy újabb fejlemény Hugo Larochelle Hibrid RBM [28] struktúrája, amit közvetlen az RBM-en belül kombinálja a generatív és a diszkriminatív modellek előnyeit, és egyszerre tud tanulni on-line jelleggel címkézett és címkézetlen adatokból egyaránt. A struktúrát a 2.9. ábra szemlélteti.
- A Larochelle féle struktúra továbbfejlesztését szintén a 2.9. ábra szemlélteti. Ez az úgynevezett "Stacked Boltzmann Experts Network"[36]. Itt minden szint ad egy valószínűséget a tanító minta osztályzására, és ezeknek az átlagolásával születik meg a végleges predikció. Ezek a hálózatok azért végtelenül izgalmasak, mert habár ha sok a tanító minta, akkor ugyanolyan a teljesítménye mint a klasszikus módszereknek, de ha kevés címkézet adat áll rendelkezésre, akkor lényegesen jobb a teljesítménye mint ahogyan az a 2.10. ábra is mutatja. Érdemes megemlíteni hogy a publikációban újságcikkekben, és nem képeken tesztelték.



2.9. ábra. A (a) Larochelle féle hibrid RBM [28] és (b) annak a többszintű továbbfejlesztése az SBEN[36] struktúra.

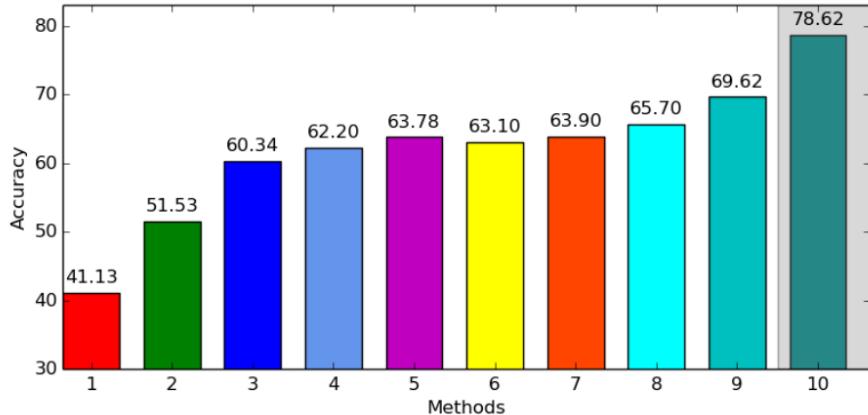
	Error	Precision	Recall	F1-Score
<i>NB-EM</i>	0.369 ± 0.039	0.684 ± 0.022	0.680 ± 0.028	0.625 ± 0.043
<i>MaxEnt-ST</i>	0.402 ± 0.026	0.623 ± 0.025	0.593 ± 0.015	0.583 ± 0.020
<i>SVM-ST</i>	0.342 ± 0.020	0.663 ± 0.010	0.665 ± 0.014	0.644 ± 0.015
<i>HRBM</i>	0.252 ± 0.023	0.740 ± 0.019	0.765 ± 0.016	0.741 ± 0.021
<i>3-Rect</i>	0.328 ± 0.020	0.673 ± 0.017	0.680 ± 0.021	0.654 ± 0.023
<i>3-SBEN,BU</i>	0.239 ± 0.015	0.754 ± 0.014	0.780 ± 0.016	0.754 ± 0.015
<i>3-SBEN,BUTD</i>	0.210 ± 0.011	0.786 ± 0.009	0.784 ± 0.014	0.777 ± 0.012

2.10. ábra. Az SBEN[36] és más, klasszikus struktúrák eredményei a WEBKB adathalmazon, úgy hogy csak a tanító adatok 1%-a volt felcímkézve (8 példa osztályonként). A mérés teljes riportját az SBEN publikációban[36] lehet megtalálni.

2.2.3. State of the art MLP hálózatok

További kutatások Az MLP hálózatokat a mai napig nagy érdeklődés övezi egyszerű struktúrájuk miatt. Látszik hogy az MNIST adathalmazon már nem nagyon van hova

javítani az MLP-k teljesítményét, viszont mint azt pár paragrafussal előbb megemlítem a paraméter tér csökkentésében és komplexebb adathalmazokhoz még van fejleszteni való ezeken a struktúrákon. Egy igen friss publikáció amelynek címe "How far can we go without convolution: Improving fully-connected networks"[31] arra mutat rá hogy hogyan lehet az MLP hálózatok paraméter terét úgy csökkenteni hogy a sigmoid rétegek közé kis méretű lineáris rétegeket teszünk be, példának okáért legyen a két réteg 1500-2000 neuron, akkor a teljes paraméter terük mérete $1500 * 2000 = 3000000$, de ha közé teszünk egy 500 neuronos lineáris rétege, akkor ez lecsökken $150 * 500 + 2000 * 500 = 1075000$ paraméterre, ami igen szignifikáns redukciót jelent a hálózat komplexításában. Ez a redukció oly mértékű hogy a fentebb említett publikációban vizsgált legnagyobb struktúra paraméter terét 112 millióról 2.5 millióra csökkentették, összehasonlítás képpen egy modern konvolúciós hálónak 3.5 millió paramétere van. Látható hogy ezzel sikerült a kutatóknak megoldania a többrétegű perceptron gépek egyik legnagyobb problémáját, a mértéktelenül burjánzó paraméter teret. A cikk az eredményeit a CIFAR-10 2.11. ábra szemlélteti. Az előbb említett táblázat nagyon jól összefoglalja az MLP-k teljesítményének a fejlődését a CIFAR-10es adathalmazt használva.



2.11. ábra. Az MLP fejlődése a CIFAR-10 adahalmazon. (1) Logisztikus regresszió fehérített adatokon; (2) Tiszta backpropagation egy 782-10000-10 méretű hálózaton; (3) Tiszta backpropagation egy 782-10000-10000-10 méretű hálózaton. (4) Egy 10000-10000-10es hálózaton, RBM előtanítással, az utolsó réteg logisztikus regresszió; (5) Egyrétegű 10000 neuronos hálózat logisztikus regressziós kimenettel, RBM előtanítással; (6) "Fastfood FFT" model (7) Zerobias autoencoder hálózat 4000 rejttet neuronnal és logisztikus regressziós kimenettel; (8) 782-4000-1000-4000-10 Z-Lin hálózat; (9) 782-4000-1000-4000-1000-4000-1000-4000-10 Z-Lin hálózat dropoutokkal; (10) Ugyanaz mint a (8), csak adat augmentációval (ami itt a szerzőknél a Higgs adathalmazon való előtanítást jelenti). Az (1)-(5) eredmények Krizhevsky és Hinton 2009-es publikációjából származnak. A legutolsó azért szürkített, mert adat augmentációt használ. Forrás: [31]

konkluzió Ezzel a végére értem az MLP-vel való képosztályozás lehetőségeinek. Lászik hogy igen nagy eredmény javulást hoztak az új kutatások. A paraméter teret 112 millióról 2.5 millióra csökkentették, és a CIFAR-10en a Hinton féle eredeti hálózathoz képest 37%-ot

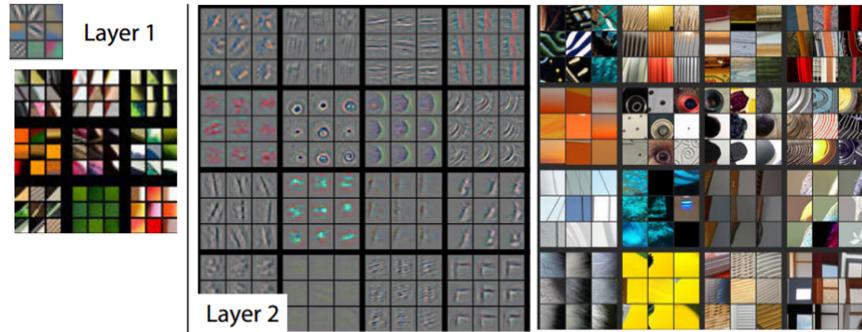
javítottak a háló osztályozó képességén. De ez még mindig kevés a következőleg bemutatott struktúrákhoz képest.

2.2.4. konvolúciós hálózatok

Bevezetés A szakdolgozatom harmadik nagy részét a konvolúciós hálózatok [30] teszik ki. Miután az irodalomkutatásom közben rá kellett jönnöm hogy az általam tanult klasszikus MLP struktúrák nem képesek a komplex jelek, mint például az MNIST adathalmaznál összetettebb képek kielégítő megtanulására, így kénytelen voltam új irányok után nézni. Így találtam meg a konvolúciós architektúrákat. Ezek korunk legjobban teljesítő architektúrái számos adattípuson, ennek oka hogy a jelek nagy része amit fel szeretnénk dolgozni az emberi érzékszervek által érzékelt jelek - például képek - amik jelentős transzlációs invarianciával rendelkeznek, és ezek a hálózatok ezt a priori tudást beleépítik az architektúrába, így lényegesen kevesebb paramétert kell megtanulnunk mint egy *elméleti síkon* hasonló teljesítményű MLP-nél. Természetesen tudjuk Cybenko (1989)[18] és Kurt Hornik (1991)[21] publikációi után hogy egy kétrétegű MLP-vel tetszőleges függvényt képes approximálni, de ehhez annyi neuron kellene a komplex jelek esetében mint a képek hogy praktikusan nem kivitelezhetőek ezek a hálózatok. A rétegek számának növelésével és hálózati kényszerek bevezetésével ez a paraméter tér jelentősen csökkenhető.

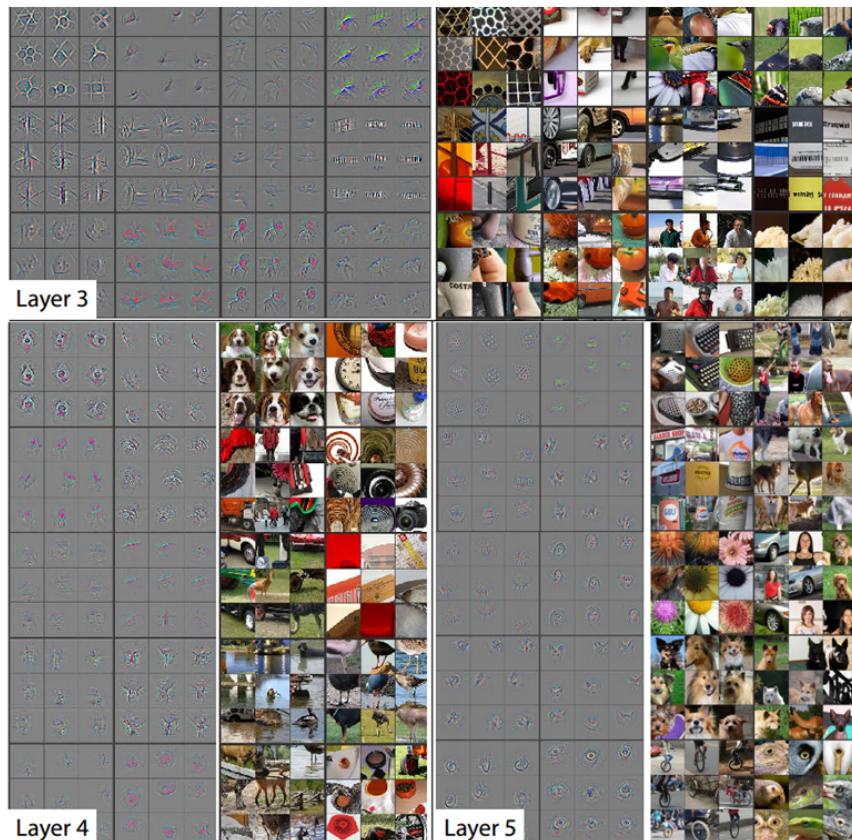
Az intuició A konvolúciós architektúra teljes mértékben biológiaiag inspirált, a macska vizuális kortextének a feltérképezésénél találtak hasonló kapcsolatokat az állat agyában[23] és ennek a mintájára építették fel a mesterséges hálózatot. Alapötlete hogy magába a hálózati struktúrába foglaljuk bele a jel transzlációs invarianciáját. A modellt eleinte képek feldolgozására alkották meg, és az előbbi mondat itt is szemléltethető a legitimitávban. Tegyük fel hogy van egy képünk amin vagy egy objektum, akkor ha fel kell ismerni hogy a kép az adott objektumnak a jellemzőit tartalmazza-e akkor nekünk adott esetben ugyan annyi információval szolgál ha ez a jellemző (mondjuk egy sarok) a bal vagy a jobb oldalon van a képen. Természetesen ezek a lokális struktúrák a rétegekkel felfele egyre globálisabbak lesznek, az egyre magasabb szinteken pedig több jellemzőből komponált összetett jellemzők jelennek meg. És a hálózat az összetett jellemzők jelenlétéből következtet a kép osztályára. Ezt hivatott szemléltetni a 2.12. ábra és 2.13. ábra amely egy konvolúciós háló egyes rétegeinek a szűrőit mutatja be, és hogy milyen képelemek aktiválták őket a leginkább. Az első sikeres alkalmazása ennek a modellnek a LeNet[30] volt 1990-ben, amelyet irányítószámok, karakterek és hasonló dolgok felismerésére használtak, de a modell sokáig nem kapott nagy érdeklődést.

Matematikai interpretációja Tegyük fel hogy van egy $32 \times 32 \times 3$ -as képünk. Ahhoz hogy egy teljes rétegbe kapcsoljuk bele mondjuk 1024 neuronnal ki kellene lapítanunk, és egy $32 \times 32 \times 3 \times 1024 = 3'145'728$ paraméterünk lenne az első rétegen. De tegyük fel hogy a legkisebb jellemző amit érzékelni akarunk az egy 3×3 méretű patchen van a képen, viszont akárhol lehet, akkor ha az első rétegen 32 jellemzőt szeretnénk érzékelni, akkor csak $3 \times 3 \times 32 = 288$ paraméterre lesz szükségünk az első rétegen, ami jelentős redukció.



Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labeled Layer 2, we have representations of the 16 different filters (on the left)

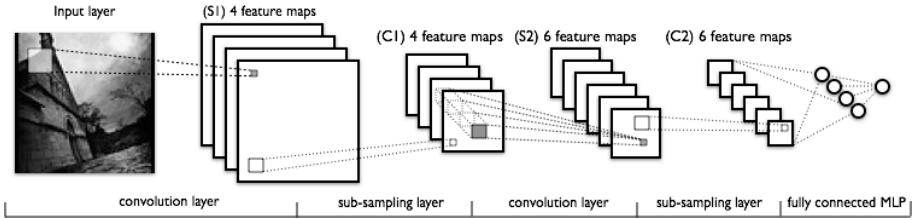
2.12. ábra. Az első két szint szűrői egy konvolúciós hálózatban. Forrás:
"Visualizing and Understanding Convolutional Neural Networks"
[46]



Visualizations of Layers 3, 4, and 5

2.13. ábra. Az felsőbb rétegek szűrői egy konvolúciós hálózatban. Forrás:
"Visualizing and Understanding Convolutional Neural Networks"
[46]

Ezután a következő réteg ehhez a 288 neuronhoz fog kapcsolódni, és ha ott 64 neuron lesz akkor $3 * 3 * 64 = 567$ neuron kell majd, amik az eredeti képből viszont már egy 5×5 -ös szeletet fognak indirekt módon lefedni. Ezt mutatja be egy klasszikus architektúra a leNet[30] a 2.14. ábrán.



2.14. ábra. Az felsőbb rétegek szűrői egy konvolúciós hálózatban. Forrás: "Visualizing and Understanding Convolutional Neural Networks" [46]

Az IMAGENET 2012-ben az AlexNet nevű konvolúciós hálózat amelyet Alex Krizhevsky, Ilya Sutskever és Geoffrey Hinton alkottak fölénnyesen megnyerte a 2012-es ILSVRC versenyen. A háló top 5 hibája (az olvasó konzultáljon az adathalmazokat bemutató résszel a metrika leírásáért.) 16% volt, míg a második helyezett ami egy SVM-eket használó modell volt 26%-os hibát produkált. Ez a 10%-os különbség az egékbe emelte a konvolúciós hálózatok népszerűségét, és hivatalosan is elhozta a neurális képfeldolgozás korát. Innentől kezdve minden évben konvolúciós hálózatok nyerték meg az ILSVRC-t. A 2.3. táblázat bemutatja az egyes évek eredményeit a hálót néhány paraméterével együtt. Összehasonlításképpen, egy átlagos ember teljesítménye az adathalmazon 5-10 hibaszázalék körül mozog[39].

2.3. táblázat. Az ILSVRC győztesei

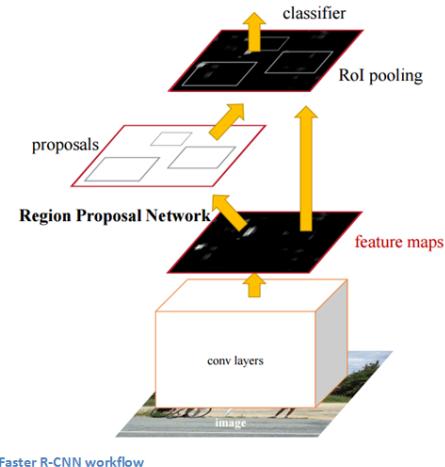
Év	A struktúra neve	Tanítás ideje	Paraméter tér mérete	Top 5 hiba százalék
2012	AlexNet [27]	két GTX 580 GPU-n 5-6 nap	60 millió	16%
2013	ZF Net [46]	egy GTX 580 GPU-n 12 nap	60 millió	11.2%
2014	GoogLeNet [43]	"néhány high end GPU-n egy héten belül"	4 millió	6.7%
2015	Microsoft ResNet [19]	8 GPU-s gépen 2-3 héttig	N\A	3.6%

Értékelés Látható hogy a legújabb konvolúciós architektúrák már az emberi pontossághoz nagyon közel állnak[39]. Ráadásul a kezdeti naiv megközelítést követően a paraméter tér is drasztikus csökkenésnek indult. Manapság ha valaki képosztályozási feladatot szeretne végezni neurális hálózatokkal, akkor egy ilyen előre elkészített architektúrát fog használni. Sajnos az is jól nyomon követhető hogy a hálózatok fejlődésével a szükséges hardware kapacitás is meredek emelkedésnek indult. Ha az ember egy konvolúciós architektúrát egy saját adathalmazra szeretne megtanítani akkor komoly infrastruktúrával kell rendelkeznie hozzá. Éppen itt domborodik ki a Tensorflownak a dolgozat elején említett előnye, hogy miután pythonban specifikáltuk a struktúrát azt képesek vagyunk minden erőfeszítés nélkül egy 8 GPU-s fürtre szétterjeszteni és tanítani, majd utána a paraméter teret lementve akár egy mobilon a megtanított hálót újra betölteni és akár valós idejű inferenciát futtatni. A paraméter tér ha 16 bites floatokkal számol az ember akkor 4 millió paraméternél kb 8 megabyte lesz, ami még egy igen kezelhető mennyiség.

2.3. További érdekes irányok a neurális képfeldolgozásban

Itt, az irodalom kutatásom tárgyalásának végén elértünk arra a pontra ahol a neurális képfeldolgozás már bevett módszereinek a nagyobb állomásait áttekintettük. Ebben a fejezetben röviden fel szeretném villantani a neurális képfeldolgozás néhány újabb és érdekesebb irányát.

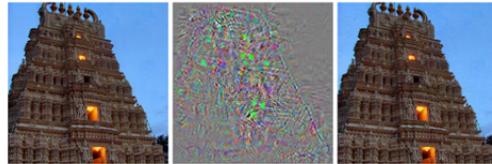
Régió alapú konvolúciós hálózatok[16] Sokan azt mondják hogy ez a publikáció csokor (R-CNN, Fast R-CNN, Faster R-CNN) hosszú idők óta az egyik legfontosabb amit új neurális architektúrákról olvashatott az ember. Eddig meg tudtuk mondani egy hálóval hogy tartalmaz-e a kép valamelyen objektumot. Az R-CNN hálózatok már az objektum pontos helyét is megmondják a képen, ami egy minőségbeli ugrást jelent. Az 2.15. ábra szemlélteti a módszert. A módszer lényege hogy a feladat két neurális hálózatra van faktorizálva amik tandemben dolgoznak, az egyik egy osztály agnosztikus objektum detektor, míg a másik egy osztályozó hálózat.



2.15. ábra. A Faster R-CNN munkafolyamata

Generatív adverziális hálózatok[17] A LeCunn, a konvolúciós hálók megalkotója szerint ez az utóbbi 10 év legérdekesebb ötlete a területen. A lényeg hogy két hálózatot tanítunk egyszerre, egy generatív és egy diszkriminatív modell-t. A diszkriminatív modell dolga eldönteni egy képről hogy valódi-e vagy generált, a generatívé pedig hogy olyan képeket tudjon generálni amivel átveri a másik modell-t, ezért hívják adverziális hálózatnak. Az egész súlya abban rejlik hogy így a diszkriminatív hálózatnak meg kell tanulni az adat egy nagyon jó reprezentációját hogy képes legyen dönten, ezzel mintegy nem felügyelt módon a legfontosabb jellemzőket kiemelni a képből. A generátor a végére pedig képes lesz valósághű képeket "álmodni". A 2.16.-ábra mutat egy tipikus tanító példát.

Képleírások generálása[25] A nem olyan távoli múltban nagyon sok érdekes publikáció jelent meg olyan neurális struktúrákról amelyek képek leírására alkalmasak. Lényegében egy CNN és egy RNN[32] hálózat működik együtt, és fantasztikus dolgokra képesek. Tovább



2.16. ábra. Jobbra: Eredeti kép, középen: Pertubációk, balra: Pertubált kép. A jobb oldalit helyesen, a bal-t pedig hibásan osztályozná egy CNN.

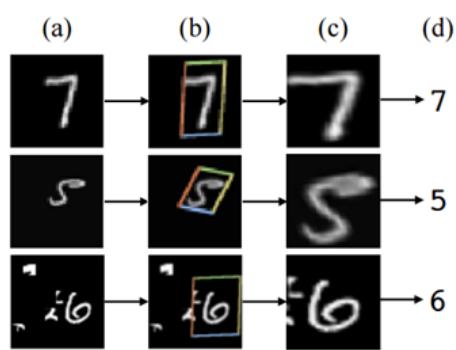
nem is taglalnám, mert viszonylag sok előismeretet igényel a téma, az érdeklődők a megfelelő publikációt megtalálják a "Deep visual-semantic alignments for generating image descriptions"[25] publikációban. A 2.17. mutatja az eredményt.



2.17. ábra. Egy képleíró hálózat által generált annotációk.

Térbeli transzformációs hálózatok (STN [24]) A fejlesztés jelentősége abban rejlik hogy eddig mindig vagy a háló struktúrájába kellett belekódolni ha valamilyen variancia ellen védeni szerettük volna a hálózatot, vagy pedig az adathalmazt kellett úgy augmentálni hogy a háló jól általánosítson. Az előbbire egy jó példa a CNN hálózatok max-pooling rétege, az utóbbira pedig az hogy mondjuk minden képet elforgatva is beadunk a háló tanításakor, hogy invariáns legyen rotációra a megtanult modell. Az STN hálózatok mint egy modulként kapcsolhatóak az egyes hálózatok elé, és ezeket a problémákat megfelelő tanítás után automatikusan megoldják. Jól látható hogy a neurális fejlesztés a monolitikus hálózatokból szintén elkezdett a modulokból felépülő paradigma felé menni. Az 2.18. ábra mutat a hálózat működésére egy példát.

Irodalomkutatás összefoglalása Ezzel az irodalom kutatásom végére értem. Úgy vélem hogy a neurális képosztályozás legtöbb fontos architektúrájára kitértem és kaptam egy átfogó képet arról hogy hol tart a terület, valamint hogy milyen módszerekkel érdemes nekiállni egy ilyen problémának. A továbbiakban szeretném bemutatni a saját munkámat, hogy mely hálózatokat implementáltam, mértem le a saját rendszeremen, és ebből milyen tanulságokat vontam le.



2.18. ábra. Egy térbeli transzformációs hálózat lépései.

3. fejezet

Hálózatok impelmentálása és elemzése tensorflowban

A következő részben szeretném bemutatni saját munkámat és mérési eredményemet. Az előző részben bemutatott hálózatfajtából megvalósítottam néhány példányt Tensorflowban és méréseket végeztem rajtuk az MNIST és a CIFAR-10 adathalmazon. A munkám célja az volt hogy leteszteljem a Tensorflow lehetőségeit kísérleti hálózatok kifejlesztésére és monitorozására. A fejezet a következő részekre tagolható:

1. A baseline metódusok bemutatása.
2. A fejlesztési környezet bemutatása.
3. Az általam használt optimizátorok ismertetése.
4. Saját fejlesztések bemutatása.

3.1. A baseline osztályozók

Természetesen nem lehet méréseket végezni referencia adatok nélkül, ezért a CIFAR-10 és az MNIST adathalmazon is lefuttattam két ismert, minden nehézség nélkül használható osztályozó algoritmust. Az egyik a Logisztikus regresszió[22] volt, a másik pedig az SVM[13]. Mindkettő bemenetére közvetlen a kép pixeljeit tettem.

3.2. A saját magam által kialakított fejlesztőkörnyezet

A Tensorflow ökoszisztemája nagyon jó pontja az érett monitorozási lehetőségek[8], viszont nem triviális egy olyan struktúra kialakítása ahol az ember nagy hatékonysággal dolgozhat. Hosszas kísérletezés után a következő munkafolyamatot találtam a legjobbnak:

- *Gyors prototipizálás:* Erre a célra a Jupyter notebookokba írt TF-Slim magas szintű API-t találtam a legjobbnak, így nagyon gyorsan le lehet akármilyen ötletet tesztelni és kiértékelni.

- *Stabil modellek fejlesztése:* Ha egy modell túljutott a pár soros méreten, vagy tényleg egy nagyobb rendszer részeként szeretnénk használni akkor azt érdemesnek találtam osztályba foglalni. A fejlesztéshez PyCharm IDE-t használtam, mert lényegesen gyorsabban lehet vele haladni komplex python kódnál mint a notebookokkal.
- *A modellek kiértékelése:* A modellek kiértékelésére úgy gondolom hogy a Tensorflow-val érkező Tensorboard a legalkalmasabb, mint majd látni fogja az olvasó a modelljeim elemzésénél hogy ez az eszköz lehetővé teszi komplex struktúrák elemzését egészen a tanulástól az igényelt rendszer erőforrásokig.
- *A modellből kinyert adatok részletes elemzése:* Erre a célra az klasszikus tudományos csomagok használatát találtam a legjobbnak jupyter notebookban alkalmazva, mert így egy helyen van a modell futtatása, a mérési eredmények és azt elemző kód.
- *Egzotikusabb modellek kivitelezése:* Ha az ember olyan modelleket szeretne kódolni amik túlnyúlnak a klasszikus struktúrákon, akkor kénytelen lenyűlni a Tensorflow eredeti programozási absztrakciójához, mint ahogyan azt az RBM esetében látni fogjuk. Ez az alacsony API hatalmas szabadságot ad a programozónak, de iszonyatosan bőszavú (verbose).

3.3. A kísérletek alatt használt optimizátorok

Nagyon sok optimizátor létezik a numerikus optimalizálási feladatok megoldására, mivel a dolgozatom fókusza nem ezeknek a bemutatása, ezért csak futólag szeretném itt megemlíteni az általam használt változatokat. Én két optimizátorral kísérleteztem, a klasszikus *sztochasztikus mini-batch* és az *adaptív momentum becslési*[26] módszerrel. Az első előnye hogy kevesebbet kell számolni, de vagy konstans tanulási rátát használ, vagy kívülről kell valamilyen lehűléses algoritmussal kontrollálni ezt, ami nehezebbé teszi a használatát. A második egy adaptív módszer, ahol az optimizátorba bele van építve a tanulási ráta beállítása, így dinamikusan tudja lekezelni a hibafelület topolójának a változását. Az algoritmusnak két hiperparamétere van, melyre a szerzők által megadott ajánlott értékek általában jó eredményre vezetnek. A különböző optimizátorok összehasonlításáról az érdeklődők <http://sebastianruder.com/optimizing-gradient-descent> ezen a honlapon olvashatnak egy nagyon jó összefoglalást.

3.4. A saját implementációk bemutatása

3.4.1. A hálózatok monitorozása

A Tensorflow érett API-t kínál a hálózat paramétereinek követésére tanulás közben, de nem ment le automatikusan minden egyes lefutáskor minden változó értékét, mivel ez egy modern hálózatnál több millió értéket is jelenthet. A VGG konvoluciós háló például 140 millió paramétert tartalmaz. A hálózat változóiról általanosságban a következő értékeket mentettem le: minimum érték, maximum érték, közép érték és a standardizált szórásukat. Annak érdekében hogy ezeket ne kelljen minden változóra kiadni, ezért a 3.1.-es függvényt alkalmaztam.

3.1. lista. A változók mentése

```
def variable_summaries(name, var):
    """Attach a lot of summaries to a Tensor."""
    with tf.name_scope('summaries'):
        mean = tf.reduce_mean(var)
        tf.scalar_summary('mean/' + name, mean)
        with tf.name_scope('stdev'):
            stddev = tf.sqrt(tf.reduce_sum(tf.square(var - mean)))
        tf.scalar_summary('stdev/' + name, stddev)
        tf.scalar_summary('max/' + name, tf.reduce_max(var))
        tf.scalar_summary('min/' + name, tf.reduce_min(var))
        tf.histogram_summary(name, var)
```

3.4.2. A logiszikus regresszió

Az első modellt implementáltam Tensorflowban az a logisztikus regresszió neurális megközelítése volt. A célja az volt hogy összehasonlítsam a Tensorflow erőforrás igényét egy klasszikus python machine learning csomag, a Scikit-learn igényeivel.

3.4.3. A többrétegű perceptron

A többrétegű perceptronnal való kísérletezést a TF-Slim API nagyon megkönnyíti, minden nehézség nélkül a rétegeket egymás után tenni, ha pedig egyedi elemet szeretne az ember definiálni akkor arra is lehetőség van. Hasonló architektúrákat teszteltem az MNIST és a CIFAR-10 adathalmazon is, ami nagyon jól megfogta a két adathalmaz közötti különbséget. A 3.2.-listázás egy ilyen háló definícióját szemlélteti.

3.2. lista. Egy MLP definíciója

```
def fully_connected(batch_data, batch_labels):
    with slim.arg_scope([slim.fully_connected],
                        activation_fn=tf.nn.relu,
                        weights_initializer=tf.truncated_normal_initializer(0.0, 0.01),
                        weights_regularizer=slim.l2_regularizer(0.0005)):
        # First Layer
        x = slim.fully_connected(batch_data, 400, scope='fc/fc_1')
        variable_summaries('fc/fc_1', x)

        # Second Layer
        x = slim.fully_connected(x, 1024, scope='fc/fc_2')
        variable_summaries('fc/fc_2', x)

        # Third Layer
        last_layer = slim.fully_connected(x, 10, activation_fn=None, scope='fc/fc_3')
        variable_summaries('fc/fc_3', x)
        predictions = tf.nn.softmax(x)

    slim.losses.softmax_cross_entropy(last_layer, batch_labels)
    total_loss = slim.losses.get_total_loss()
    tf.scalar_summary('losses/total_loss', total_loss)

    optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
    return optimizer, predictions
```

Mint látható nagyon könnyen állíthatóak a reguralizációs tagok, megadható többféle optimalizáló és hibafüggvény is, ami igazán könnyűvé tette a kísérletezést.

Az MLP-ben használt elemi alkotóelemeim:

- *fc*: A teljesen kapcsolt réteg, minden neuron, minden előbbi réteg neuronjával össze

van kötve, ennek a definícióját a (3.1)-képlet mutatja be, ahol l a réteg sorszáma, és "Activation" egy tetszőleges aktivációs függvény.

- *reLu*: A rektifikált lineáris egység (reLu) egy aktivációs függvény, ezeket a teljesen kapcsolt réteg után kapcsoljuk, azért hogy nem-linearitásokat vigyünk a rendszerbe. így növelve a leképző képességét. A reLu definíciója a (3.2) képleten látható.
- *sgm*: A sigmoid aktivációs függvény amely szintén egy aktivációs függvény mint a relu, de nehezebb a deriváltját számolni, és érzékenyebb az adatok normalizáltságára. Az sigmoid definíciója (3.3) képleten látszik.

$$X^{(l)} = W * Activation(X^{(l-1)}) \quad (3.1)$$

$$reLu(x) = \max(0, x) \quad (3.2)$$

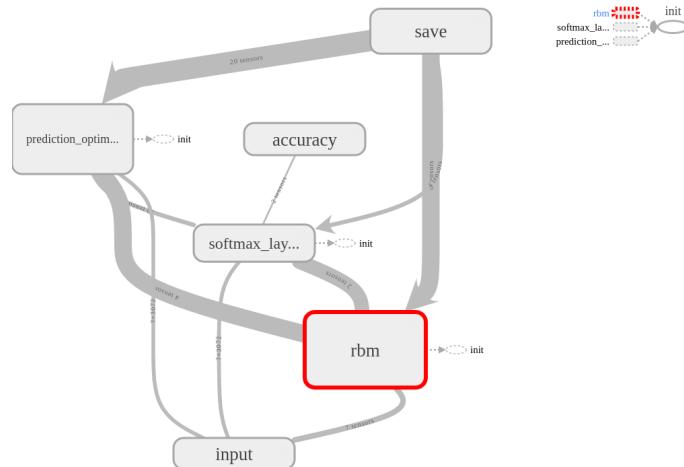
$$sgm(x) = \frac{1}{1 + e^x} \quad (3.3)$$

3.4.4. Az RBM hálózat

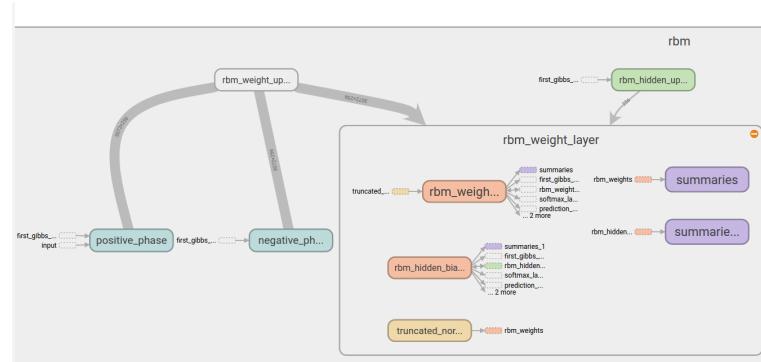
Mint mondtam a TF-Slim nagyon kényelmes volt addig amíg olyan struktúrákkal dolgoztam amik könnyen definiálhatóak. Viszont csak többrétegű perceptronok és konvolúciós hálózatokat lehet benne egyelőre alkotni. Az korlátozott boltzmann gép implementálásához le kellett mennem a Tensorflow alacsony szintű interfácéhez amiben a hálózat implementálása több mint egy hét volt. Viszont ezalatt értettem meg igazán hogy hogyan működik egyrészt a könyvtár, másrészt pedig az RBM-ek. Az RBM struktúra legtrükkösebb része a kontrasztív divergencia volt, mivel az egy valószínűségi döntés, de az egyes batchek futása közben nem tudok közvetlenül a gráfban belül véletlen számokat generálni változtatható mennyiségben. Azért nem lehet egy adott méretben generálni, mert a batch méret változik, és minden számnak kellett generálni. A megoldás amit alkalmaztam az az volt hogy numpy-ban legeneráltam minden tanításnál egy akkora mátrixot véletlen számokból mint amekkora a batch mérete volt és ezt egy placeholderon keresztül injektáltam bele a hálóba. Majd a bináris 0-1 döntést a következőképpen szimuláltam:

1. Kivontam a valószínűségi mátrixot a véletlenszerűen generált számokból.
2. Vettem az előjelét a keletkezett mátrix elemeinek.
3. Átvezettem egy relu rétegen, aminek a kimenete az előjeltől függően stabil 0 vagy 1 lett.

A tanítás A tanítást kétféle képpen végeztem el, egyrészről definiáltam a szabad energia függvényt és a keretrendszerrel számoltattam ki a (2.8) függvényből és különböző beépített optimalizátorokkal teszteltem, sztochasztikus gradiens (SGD) és adaptív gradiens (ADAM) optimizátorral. Másrészt közvetlen kiszámoltam a Gibbs sampling utáni értékekből (2.9), (2.10) és (2.11) függvényeket és egyszerűen csak hozzáadtam őket a súlyvektorhoz. A 3.1. ábra a hálózat madártávlati struktúráját szemlélteti. Illetve a kísérletezés egy másik dimenziója az volt hogy egyes esetekben megengedtem az RBM felé kapcsolt regresszornak hogy az előre megtanult súlyokat változtassa (ezt hívjuk fine-tuning-nak), más esetben pedig nem. Könnyen kivehető hogy egy softmax réteg is hozzá van csatolva az RBM magjához, miután felügyelet nélkül tanítom az RBM-et az a réteg végzi az osztályzást. A 3.2. ábra pedig az RBM belső struktúráját mutatja. Itt látszik igazán hogy a Tensorboard grafikonjai milyen kifinomult vizualizációt tesznek lehetővé. A tanítás optimalizálását a [20]-ben leírtak alapján végeztem, de az osztályozásban meg hoztak érzékelhető javulást, ezért ezeket a mérések nél nem fejtem ki részletesen.



3.1. ábra



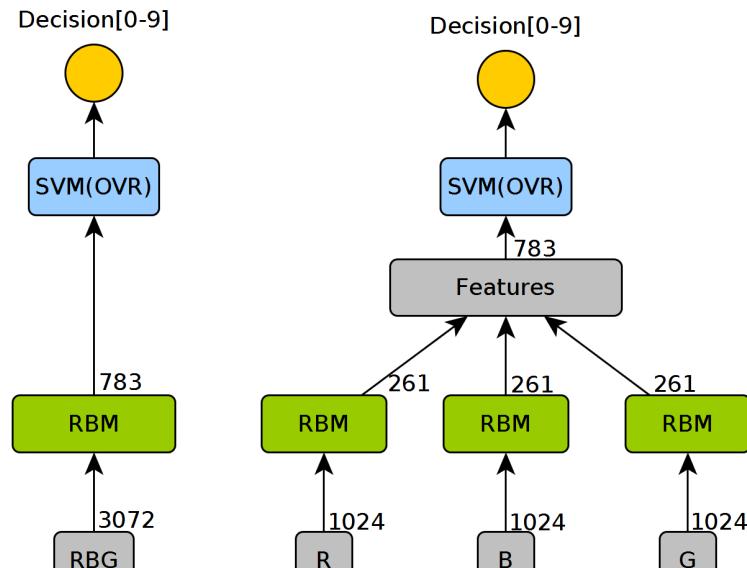
3.2. ábra

3.4.5. A DBN struktúra

Miután az RBM struktúrát megalkottam természetesen le szerettem volna tesztelni többrétegű előre tanított hálózatok teljesítményét is. Erre nem a saját RBM hálózatomat használtam, hanem egy külső könyvtárat, ami viszont hibás volt úgyhogy helyenként meg kellett foltoznom. Itt jött jól az a tudás amit az RBM programozásánál szereztem, mert magabiztosan mozogtam a pythonban írt Tensorflow kódban.

3.4.6. Hibrid modellek

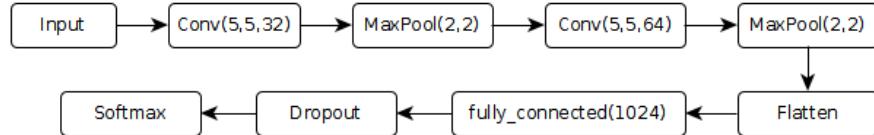
A kísérletezésem során elkezdtem olyan architektúrákkal foglalkozni ahol a kimenet nem a neurális hálózat része, mint például az RBM osztályozók általam tesztelt variánsánál, hanem a kimeneti aktivációk valószínűségi eloszlását adtam be mintánként egy SVM-nek. Így jelentős dimenzió csökkenést értünk el, pl 784-ről 64-re az MNIST esetében, ami kifejezetten meggyorsította az SVM tanítását, anélkül hogy a nyers pixeladatokon tanított SVM-hez képest romlott volna a modell predikciós képessége. Az általam használt teszt gépen a CIFAR-10es adathalmaz nyers pixeljeit nagyon hosszú idő alatt tudtam megtanítani egy SVM-nek. Ezért letömörítettem az az 3072 dimenziós CIFAR-10 adathalmazt egy 783 dimenziós térbe RBM-ek segítségével ami mindenkor még jellemző kiemelést is végzett és ott tanítottam rajta a szupport vektor gépet, remélve hogy jó osztályozási eredményeket kapok. Ezt két megközelítésben próbáltam meg, az egyik az volt amikor egyetlen RBM-em volt, 3072 bemeneti és 783 kimeneti neuronnal, a másik az volt amikor 3 RBM-et tanítottam meg, minden színcsatornára egyet, és utána ezeket egy tömbbe összefűzve adtam át az SVM-nek. A két megközelítést a 3.3. ábra szemlélteti.



3.3. ábra

3.4.7. A Konvolúciós modell

Több konvolúciós modellt kipróbáltam, a méréseim egyik fő iranya az volt hogy ugyanazt a modellt probálom ki az MNIST és a CIFAR-10 adathalmazon, és megnéznem hogy melyik modell hogyan reagál az adat megnövekedett komplexitására. Több hálózatot kipróbáltam, az alapmodell felépítését a 3.4. ábra mutatja. Arra is irányultak kísérleteim hogy plusz teljesen összekötött rétegek hozzáadása, esetleg a konvolúciós rétegek más elrendezése milyen eredményre juttat.



3.4. ábra

Az egyes operációkat szeretném megmagyarázni röviden amiket a konvolúciós modelljeimben használtam:

- *conv(magasság, szélesség, mélység)*: Kétdimenziós, diszkrét konvolúció. Ahol a magasság és a szélesség adják meg a képfolt nagyságát. A mélység pedig hogy hány neuron legyen az adott rétegen. A kétdimenziós konvolúció definícióját a (3.4)-képlet mutatja be.
- *maxpool(magasság, szélesség)* Egy alulmintavételezési operátor a térbeli dimenziókban (magasság, szélesség). Ezzel csökkentjük drasztikusan a jellemzőterek nagyságát és adunk transzlációs invarianciát a rendszerhez. Mindig a legerősebb jelet viszi keresztül a pooling foltból.
- *fc(neuronok száma)*: A teljesen kapcsolt rétegeket a hálózat végére kapcsoljuk. Amíg a konvolúciós rétegek egy magas absztraktióval rendelkező, viszonylag transzláció invariáns jellemzőteret nyújtanak a képről, addig a hálózat végén lévő teljesen kapcsolt rétegek teszik lehetővé a nemlineáris leképzéseket ebben a jellemzőtéren. Az itteni teljesen kapcsolt réteget és a hozzá tartozó nemlinearitásokat is a (3.2), (3.3) írja le.
- *lrn*: Local Response Normalization, az alexNet[27] publikációból kiderül, hogy ez a lokális normalizálási eljárás statisztikailag szignifikáns javulást képest hozni a hálózat osztályozó képességében, mert csökkenti a szaturációt a relu rétegekben. Ezt a matematikai struktúrát a (3.5) képlet szemlélteti. Ez is egy valós biológiai jelenség - a laterális inhibíció - által bevezetett technika.
- *flatten*: Az első teljesen kapcsolt, és az utolsó konvolúciós réteg között ki kell lapítani a 3 dimenziós konvolúciós struktúrát egy vektorba hogy össze lehessen kapcsolni a következő réteggel. Ezt az operációt hívjuk angolul flattennek.

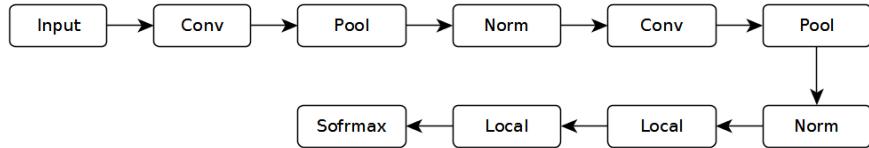
$$h_{ij}^k = \text{activation}((W^k * x)_{ij} + b_k) \quad ahol \quad * \text{ konvolúciós operátor.} \quad (3.4)$$

$$b_{x,y}^i = a_{x,y}^i / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^j)^2 \right)^\beta \quad (3.5)$$

Ahol $a_{x,y}^i$ -nél az i -edik kernel-t alkalmazzuk az (x, y) beli pozícióra, $b_{x,y}^i$ a normalizáció utáni eredmény, n a vizsgált pozíció szomszédos kernel térképei, N az összes kernel a rétegen, α , β és k pedig további hiperparaméterek.

3.4.8. A konvolúciós modell skálázása

Indoklás Az előbbiekben bemutatott konvolúciós modell, és amikkel általánosságban a szakdolgozat keretén belül kísérleteztem könnyedén taníthatóak pár perc/óra alatt egy korszerű CPU-n. Viszont ha az ember nagyobb modelleket szeretne tanítani akkor ez már nem egy reális alternatíva. Ezekre az esetekre egy nagyobb hálózattal kísérleteztem a CIFAR-10 adathalmazon amit az alábbi instrukciók alapján készítettem el[7], aminek a felépítését a 3.5. ábra mutatja.

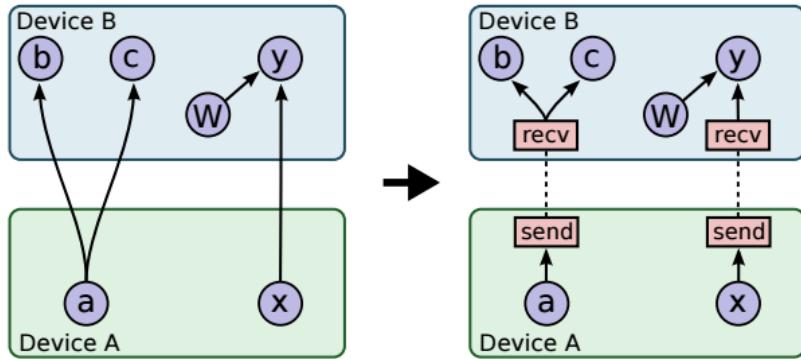


3.5. ábra

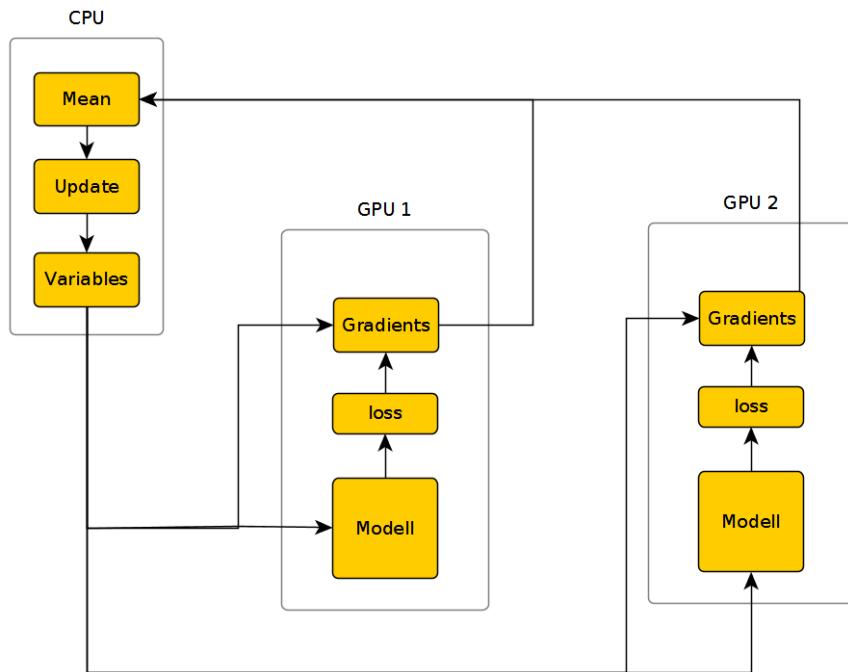
Használt eszközök Ezt a hálózatot teszteltem CPU-n illetve a laptopom NVIDIA 740M típusú videokártyáján, ezen felül a Tensorflow oldalán találtam adatokat egy NVIDIA Tesla K40c-n futtatott scientific accelerator cardon elért eredményekről is ugyanezen az adathalmazon. A program erősen párhuzamosított volt, és a felépítése lehetővé tette volna tetszőleges számú GPU-ra való kiterjesztését, ennek a program felépítésének a részleteit szeretném a következő bekezdésben megosztani.

A számítás elosztása A Tensorflow képes a számításokat automatikusan elosztani az eszközök között olyan módon hogy heurisztikusan megkeresi hogy mely operációk mentén érdemes szétvágni a számítási gráfot, majd azokhoz az élekhez berak küldő és fogadó csomópontokat ahogyan a 3.6. ábra szemlélteti. Ez sok esetben teljesen megfelel az elvárásainknak, ha nem szeretnénk sokat vesződni a hálózat elosztásával, vagy amúgys is túl nagy a hálónk, és nem férne el gradiens számítással együtt rendesen egyetlen eszközön. De ha kisebb a háló és több eszközünk van akkor felmerülhet a lehetőség hogy esetleg jobban megérné a gráfot egy az egyben lemásolni az egyes eszközökre, és a súlyokat a fő memóriában tartani, majd az egyes párhuzamos futások után itt szinkronizálni a lefutásokat. Én ezt a megközelítést teszteltem le, aminek a neve torony párhuzamos tanítás, ezt a 3.7. ábra mutatja.

Adatok dinamikus felolvasása Természetesen az a kérdés is felmerülhet az emberben hogy ez nagyon jó hogy így el tudjuk osztani a számításokat, de az is probléma lehet ha



3.6. ábra. Forrás: [11]



3.7. ábra. Forrás:[7]

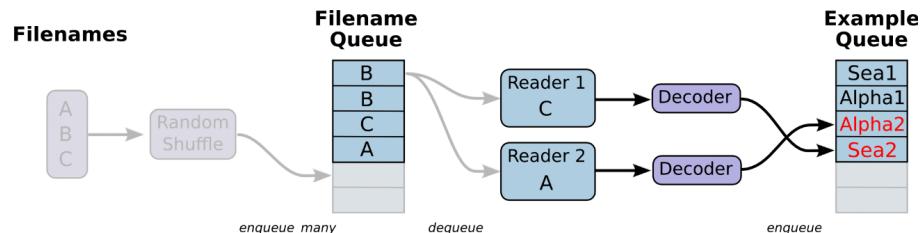
a tanuló adatok nem férnek be a memóriába, akkor nem tudjuk rendesen tanítani őket, kivéve ha valamilyen bonyolult felolvasási kódot írunk hozzá. Hál istennek ezt nem kell megírnunk, hanem be van építve a keretrendszerbe, a Tensorflow erre direkt felolvasási pipelineokat állít a programozó rendelkezésére ennek a felépítését szemlélteti a 3.8. ábra. Egy ilyen pipelinenak a következő paramétereit vannak:

- Bemeneti file nevek.
- A felolvasásra és előfeldolgozásra használni kívánt szálak száma.
- A megállási kritérium
- A pipeline mérete, hogy hány még meg nem tanított kép legyen mindenig a pipelineban.

Ezek után a keretrendszer gondoskodik arról hogy a szálak mind stratifikáltan, külön fileokból olvassanak fel, illetve ha példányosítunk egy koordinátor objektumot és beregisztráljuk hozzá a pipeline-t akkor a szálak közötti hibakezelésről is gondoskodik, hogy egy szál hibája esetén ne álljon le hibával az egész tanítási folyamat. Ezen felül definiálhatunk még előfeldolgozási lépéseket minden képhez, az általam használt implementációban például egyszerre 20 feldolgozatlan képet tart a memóriában, ezeket először közelítőleg fehéríti (ez is beépített funkció), majd ezt az adathalmazt négyeszeri a következő képpen:

- A képek véletlenszerű tükrözése.
- A kép világosságába való véletlenszerű zaj bevezetése.
- A kép kontrasztjának véletlenszerű torzítása.

Ez jelentősen növeli a háló általánosító képességét. Majd ezeket az adatokat beadja egy új sorba, ami véletlenszerű batcheket generál a 16 szál képeiből, és megadhatjuk hogy hány modell vegye kis és dolgozza fel őket egyszerre. Esetben a torony-párhuzamos elrendezésben ez a GPU-k számától függ.



3.8. ábra. Forrás:[6]

4. fejezet

A mérési eredményeim összegzése

Ebben a fejezetben szeretném bemutatni az általam elkészített hálózatok futásának mérési eredményeit. Ezek aspektusai:

1. A keretrendszer általános teljesítménye.
2. A baseline eredmények bemutatása.
3. Az MLP hálózattal elért eredmények és az ezekből levont tanulságok.
4. A konvolúciós hálózatokkal elért eredmények és ennek összegzése.
5. A hibrid hálózatokkal elért eredmények.

Fontosnak tartom megemlíteni a következő problémákat amelyekről nem volt szó a szakdolgozatomban, ezzel is jelezve hogy a terület rendkívül változatos matematikai kihívásokat állít:

- A hiperparaméterek végletekig menő tuningolása grid search-el.
- A hiperparaméter halmazok összehasonlítása, akár kereszt validációval, és egy végeleges hiperparaméter beállítás kiválasztása a teszt halmazon való kiértékeléshez.
- A lépésszámok pontos kíkísérletezése, és korai leállási feltételek megfogalmazása és implementálása.
- Az adatok átfogó előfeldolgozása. minden adatot normalizáltam, de nem végeztem több előfeldolgozást, pl fehérítést.[15]
- Az adathalmaz bővítése. Ide tartoznak a képek véletlenszerű tükrözése, kivágása, rotációi, egyes értékek disztorciói (pl kontraszt).

Tehát a modelljeimet empirikus paraméterekkel, a nyers adatokon teszteltem. Mindeközben tisztában voltam az összes architektúra elméleti teljesítményének a határával, ez is egy érdekes tanulság volt, hogy ilyen "naiv" megközelítéssel milyen közel tudok jutni egy-egy kutatás eredményéhez.

A gép specifikáció amin általánosságban teszteltem (eltekintve a GPU skálázás résztől) a következők:

- Memória: 8 Gigabyte DDR4
- CPU: Intel Core i7-4702MQ CPU @ 2.20GHz x 8

A gépben elhelyezkedő grafikus kártyát nem használtam, az Nvidia hibás eszköz meghajtó programja miatt. Egy GPU-n mért teljesítmény karakterisztikák eltérhetnek, főleg egy olyan hatékony implementáció mellett mint a cuDNN, az Nvidia által fejlesztett mély neurális rendszerekhez használt könyvtár, ami nagyon hatékony implementációkat tartalmaz a Neurális hálózatok legtöbb operációjára. A mértékek szemléltetéséhez a 4.1. ábra szemlélteti. Érdemes hozzátenni hogy egyetlen Tesla 40M ára most kb 1,3 millió forint. Amíg a CPU-s teszben két Intel Xeon E5-2699v3 szerepelt, jelenlegi darab ára hozzávetőlegesen 1 millió forint. Tehát a GPU-s megoldásnál hozzávetőleges 2.6-szor drágább volt csak a megoldás magját képező számítási felszerelés.



4.1. ábra. Az alexNet tanítása két CPU-s intel xeon e5-2699v3 szerveren, és 4 GPU-ból álló Tesla M40 griden. Forrás: <http://images.nvidia.com/content/products/hpc/fasterperformance-than-cpu.png>.

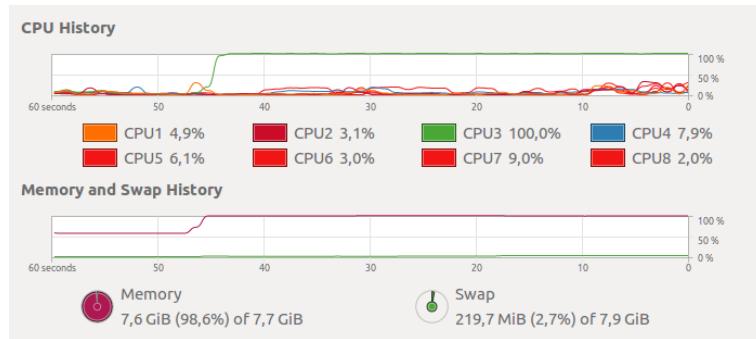
4.1. A keretrendszer általános teljesítménye

Ebben a részben azt vizsgáltam hogy a tensorflow hogyan bánik az erőforrásokkal. Ennek érdekében több dolgot tettem:

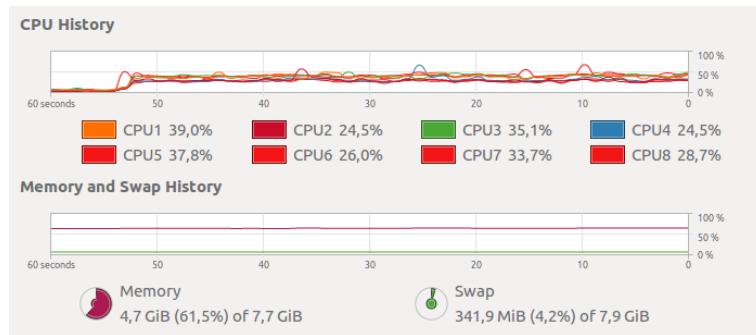
- Egyszerű rendszerszintű méréseket futtattam.
- A Tensorboardon keresztül vizsgáltam a hálók teljesítmény és erőforrás karakterisztikáját.
- Egy konvolúciós hálót kiskáláztam a CPU-ról GPU-ra, majd összevetettem egy HPC GPU teljesítményével ugyanazon az adathalmazon.

4.1.1. Rendszer szintű mérések.

Az első kérdés ami felmerült bennem, az az volt hogy egy egyszerű logisztikus regresszió esetén hogyan viszonyul a tensorflow erőforrás igénye a Python de facto gépi tanulás könyvtárához, a Sci-kit learnhez. Ezt a CIFAR-10 adathalmazon teszteltem, mivel az MNIST számításigénye túl kicsi lett volna ehhez a teszthez. A két mérés erőforrás karakterisztikáját a 4.2. ábra és a 4.3. ábra mutatja, ahol az előbbi a Scikit-learn-ben elindított logisztikus regresszió, az utóbbi pedig a Tensorflowban megvalósított közelítő Logisztikus Regresszió.



4.2. ábra. A Scikit-learnben idított logisztikus regresszió erőforrás igénye.



4.3. ábra. A Tensorflowban idított logisztikus regresszió erőforrás igénye.

Mint látható a scikit-learnben elindított futtatás a teszt gép összes memóriáját felemészette, illetve a magok kihasználása is nagyon rossz volt, minden össze egyetlen magot használt ki. Ezzel szemben a Tensorflowban approximált modell nagyon egyenletes magkihasználtság mellett minimális memória igénnyel oldotta meg a feladatot. Természetesen nem állítom hogy a két feladat identikus volt, mivel más algoritmusokkal jutottak el a végeredményig. További kutatásaimból kiderült hogy ez a Scikit-learn alapértelmezett optimalizációs algoritmusának köszönhető amit a liblinear könyvtár valósít meg. Amikor ezt kicseréltem SAG megoldóra, akkor a Scikit memória kihasználása javult, de még mindig konvergencia problémák léptek fel. Az SAG egy kifejezetten új fejlemény a numerikus optimalizáció területén, amit egy 2013-as publikáció mutat be "Minimizing Finite Sums with Stochastic Average Gradient"[41]. Ezen felül még megpróbáltam az adathalmazt a Newton konjugált-gradiens módszerrel megtanulni, de ez sikertelen lett az erőforrások hiánya miatt. Ami érthető, hiszen a Newton-cg optimizátor gyors konvergenciát ígér, de még a sima liblinear megoldónál is nagyobb erőforrás igényel.

Ez a mérési sorozat úgy gondolom hogy egyértelműen rávilágít a numerikus optimalizáció kiemelkedő fontosságára a gépi tanulás applikációkban. Az adathalmaz amin teszteltem a metódusokat nem volt nagy, a CIFAR-10 minden össze 180 megabyte. Amennyiben ezeket a méréseket az akadémia szerverein végeztem volna el, akkor gond nélkül taníthattam volna akármelyik módszerrel. Viszont így, hogy a tesztrendszer is igen korlátozott jól megvilágította a nagy adathalmazokon való tanulás egyik nagy problémáját. Habár a mostani szereink már hatalmas erőforrásokkal bírnak, a klasszikus módszereink amik folyamatosan az egész adathalmazzal való interakciót igénylik, mint mondjuk az eredeti

logisztikus regresszió vagy SVM tanító algoritmusok mégsem lesznek használhatóak egyszerűen az adathalmaz pusztá nagysága miatt. Ezért is fontos hogy a mostani kutatások nagyrésze a numerikus optimalizáció terén a sztochasztikus módszerekre koncentrál amik véletlenszerűen összeválogatott kis tanító halmazokkal approximálják a teljes adathalmaz tulajdonságait.

4.1.2. A GPU gyorsítás mérése

A mérést a tesztgép GPU-ján végeztem el, illetve ezt összevetettem egy nagyon erős scientific acceleratoron futtatot méréssel ugyan azon az adathalmazon, ugyanazzal az általam használt CNN architektúrával[7].

A hálózat rétegei a következőképp néztek ki: *conv2d(5,5,32) -> maxpool(2,2) -> lrn -> conv2d(5,5,64) -> lrn -> maxpool(2,2) -> flatten -> fullyconnected(1024) -> dropout(0.5) -> fullyconnected(1024) -> softmax.*

A két eszköz paraméterei:

- Nvidia GeForce GT 740M: 384 Cuda mag, 2GB GDDR5 memória. Ez egy teljesen átlagos laptop GPU, leginkább a játékos réteget célozta meg 3 éve, de CUDA 3.0 képes.
- Nvidia Tesla K40c: 2880 Cuda mag, 12GB GPU Accelerator memória. Ez egy high performance computing(HPC) gyorsító(accelerator), itt a GPU már csak az architektúrára utal, de nincsen rajta grafikus pipeline. Az egyetlen dolga hogy nagyon gyorsan tud számolni. Big Data analitikára, és nagy méretű tudományos számításokhoz lett tervezve. A 4.1. ábrán található Tesla 40M ennek a családnak a legújabb tagja.

Arról nincsen információm hogy a Tesla K40c-t milyen CPU vezérelte, illetve hogy mennyi RAM volt a tesztgépükben. Az eredményemet a 4.1. táblázat mutatja. A mérték amit használtam azt mutatja hogy a CIFAR-10 adathalmazon hány képet tudok másodpercenként feldolgozni tanítás közben a fentebb bemutatott hálózaton.

4.1. táblázat. A *Tensorflow* számítási teljesítménye különböző eszközökkel. A mértékegység hogy a CIFAR-10 adathalmazon hány képet tud másodpercenként feldolgozni tanítás közben a 3.5. ábrán mutatott hálózaton. Az első iteráció ki van hagyva mint minimum, amikor még a queue-t tölti fel és nem stabil a teljesítménye. A számot 0 tizedes helyértékre kerekítettem.

Eszköz	Minimum képszám másodpercenként	Átlagos képszám másodpercenként	Maximum képszám másodpercenként
Intel Core i7-4702MQ	174	241	350
Nvidia GeForce GT 740M	351	453	489
Nvidia Tesla K40c	392	470	599

A táblázatból a következő tanulságokat vontam le:

- A CPU sokkal jobban ingadozik, ez nyilvánvaló, mert amellett hogy számolnia kell a hálózatot minden mást is el kell látni a számítógép működésével kapcsolatban. Például a sor feltöltése és a képek előfeldolgozása. Ezt a GPU estében is 16 CPU szál csinálja, ezért ezzel nem kell foglalkoznia a grafikus egységnek.

- Éppen ezért a GPU megoldások sokkal stabilabb teljesítményt adtak.
- Érdekes módon nem volt akkor különbség ebben az esetben egy HPC GPU és egy Commodity GPU között. Szerintem túl kicsi volt a feladat ahhoz hogy érdemben kihasználja a HPC GPU teljesítményét, de azért itt a peak teljesítmények eltérésén itt is látszik hogy mennyivel erősebb mint a mezei GPU.

4.2. A baseline eredmények ismertetése

Mint már említettem, a két alap metódus amihez a saját eredményeimet mértem a logisztikus regresszió[22], és egy egyszerű SVM[13] volt, amihez polinomiális kernelt használtam 3-as fokszámmal és az első rendű tagokat használva. Az eredményeim az eredményeimet az MNIST és a CIFAR adathalmazon a 4.2. táblázat, és a 4.3. táblázat szemlélteti.

4.2. táblázat. A baseline mérések eredménye az MNIST adathalmaz nyers pixeljein, használt könyvtár: Scikit-learn

tanuló metódus	Teszt halmaz hiba százaléka
Logisztikus regresszió	8%
SVM poly kernel fok: 3 coef0: 1	6%

Az MNIST-en elért eredmények nem a legjobbak, de egészen optimális eredmények ilyen egyszerű eljárásokkal is. Ezekben sokat lehet javítani a képek előfeldolgozásával de a szakdolgozatomban nem erre szerettem volna fókuszálni. minden struktúrát a nyers adatokon kezdtettem el tréningezni, klasszikus előfeldolgozási lépések alkalmazása nélkül. A teljesség kedvéért megemlítem hogy a legjobb eredmény amit SVM-el értek el az MNIST adathalmazon annak a felépítése a következő volt: *Virtual SVM, deg-9 poly, 2-pixel jittered*[14], előfeldolgozási lépésként csak kiegyenesítést használt (deskewing), az elért teszt hibaszázalék pedig 0.56% volt.

4.3. táblázat. A baseline mérések eredménye az CIFAR-10 adathalmaz nyers pixeljein, használt könyvtár: Scikit-learn

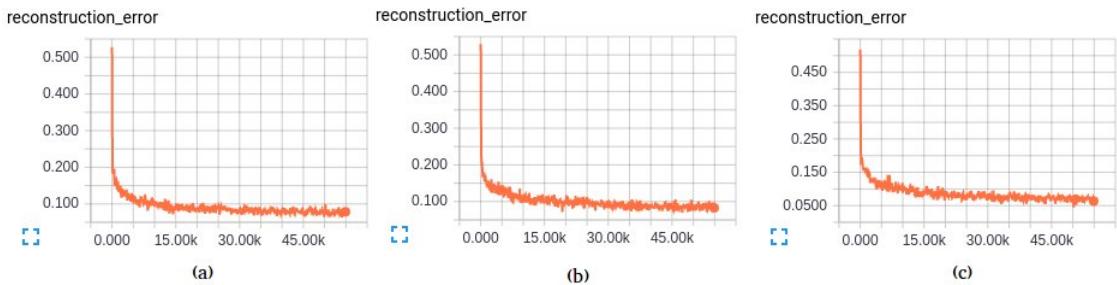
tanuló metódus	Teszt halmaz hiba százaléka
Logisztikus regresszió	62%
SVM poly kernel, coef0=1 fokszám=3	55%

Mint látni lehet itt ezek a metódusok már közel sem teljesítenek annyira fényesen. A legjobb eredmény amit elértek az adathalmazon SVM-el az 25,5% volt. Ez egy ICML 2010-es konferencián bemutatott eredmény volt, amit a "Improved Local Coordinate Coding using Local Tangents"[45]-ban publikáltak.

4.3. Az RBM struktúra optimalizálása

Ahogyan azt a saját munkám leírásánál is taglaltam többféleképpen végeztem el méréseket az RBM struktúrával hogy megfigyeljem hogyan hasonulnak egymáshoz. Először a szabad energia függvények különbségét definiáltam a (2.8) szerint, és ezt optimalizáltam, majd egyszerűen hozzá adtam az általam lederivált gradienseket. A rekonstrukciós hiba függvények

lefutását a 4.4.-ábra szemlélteti. Látható hogy a 3 megközelítésnek a lefutásában és erőforrás felhasználásában is jelentős különbség van, ezt a 4.4. táblázat szemlélteti. Jól látszik hogy az adaptív optimizátor megnövekedett számításigénnel jár, az SGD ugyanannyi memóriát igényel, mert körülbelül ugyanazt az implicit konstruált gráfot használja de kevésbé terhelő a processzort, amíg a kézi optimalizálás még ennél is lényegesen kevesebb erőforrást igényel minden tekintetben. Éppen ezért a kézzel kiegészített megoldást választottam. Ez a megoldás jobban is skálázódik több számítási egységre, ha nagyobb RBM-et akar tanítani az ember, mert a gradiens frissítő gráf miatt nem fog kommunikációs overhead fellépni az eszközök között, mivel ez a gráf nem jön létre.



4.4. ábra. Az RBM rekonstrukciós hibájának csökkenése a tanító batcheken.
Az (a) ADAM optimizátorral, (b) SGD optimizátorral, (c) a gradiensek manuális megadásával.

4.4. táblázat. Az RBM erőforrás mefelhasználása és hiba százaléka 10 epoch után, 256 neuron mellett különöző optimizátorokkal.

optimizátor	Batch idő	Memória	Rekonstruktív hiba
ADAM	11.8 ms	10.7 MB	0.077
SGD, tanulási ráta: 0.1	7.78 ms	10.7 MB	0.087
SGD, tanulási ráta: 0.01	7.78 ms	10.7 MB	0.084
SGD, tanulási ráta: 0.001	7.78 ms	10.7 MB	0.120
Manuális	2.67 ms	5.62 MB	0.070

4.4. Az MLP-vel elvégzett méréseim eredményei

4.4.1. Az MNIST mérések

A többrétegű perceptronokról ismeretes hogy a rétegek között teljes kapcsolat van, és az adathalmaznak semmilyen tulajdonságát nem építik bele az architektúrába a priori. A konvolúciós hálózatok esetében, viszont a transzlációs invariancia előre feltételezett és maga a hálózat tartalmazza a kényszereket. Ez alapvetően abban is megnyilvánul hogy a hálózat nem egy három dimenziós bemenetre (magasság, szélesség, mélység(RGB)) támaszkodik, hanem egy egyszerű vektorra. Az MNIST-en elért eredményemet MLP-vel a 4.5. táblázat mutatja be, a méréseket a fejezet elején ismertetett tesztgép CPU-ján végeztem. A bemenet egy 784 elemű vektor volt, ami praktikusan az MNIST adathalmaz egyetlen vektorba lapítva sorról sorra.

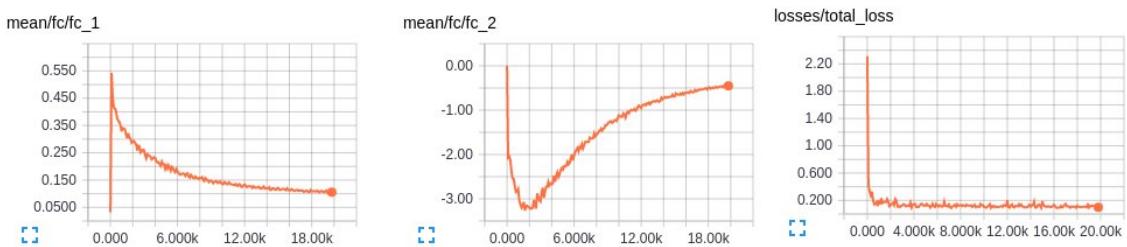
Látható hogy az első struktúra a logisztikus regresszióval azonos eredményt ad, ami nem meglepő, hiszen egy egyrétegű struktúra softmax függvényel a végén praktikusan

4.5. táblázat. A saját MLP teljesítménye az MNIST adathalmazon, 20 000 tanító batch után. Bemenetek száma: 784, reLu aktivációval, L2 reguralizációval, aminek az együtthatója 0.0005 volt. A súlyokat csonkolt normál eloszlással inicializáltam, standard szórás: 0.1, középérték: 0.0.

neuron struktúra	Optimalizáló algoritmus	Teszt szet hiba százalék
10	SGD learning-rate: 0.5	8%
1024-10	SGD learning-rate: 0.5	3.9%
400-10	SGD learning-rate: 0.5	2.02%
400-10	Adam	2.04%
800-10	Adam	2.21%
800-10	SGD learning-rate: 0.5	2.00%

ugyanakkora általánosító képességgel rendelkezik mint egy klasszikus módszerekkel tanított multinomiális logisztikus regresszor. Ezen az eredményen lényegesen tudtam javítani miután hozzáadtam egy 1024 elemű rejtett réteget, ami növelte a háló általánosító képességét, de 20'000 lépés alatt nem volt képes túltanulás nélkül megtanulni a feladatot. Ahogyan csökkentettem a hálózat méretét az csökkentette a túltanulás mértékét és jelentős teljesítmény növekedést hozott. A 2.00%-os hiba eredmény már elmondható hogy egészen közel van az eddigi legjobb elérte eredményhez 1 rejtett réteggel, ami 0.7%. Azt vártam volna hogy az Adaptív momentum közelítő (ADAM) módszer javít a hálózat eredményén, de nem így lett. Általában az Adam optimalizátor előnye abban rejlik hogy mivel a momentumot és a tanulási rátát adaptívan számítja minden batchre, ezért kevesebb hiperparamétert kell állítani. Ez természetesen megnövekedett számítás igénytelensége szokott járni.

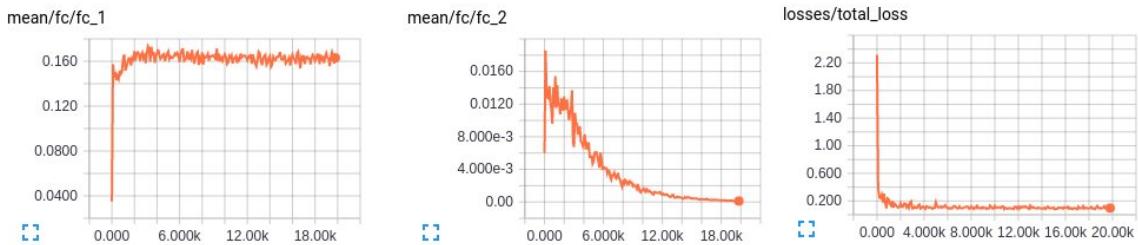
A 4.5. ábra és a 4.6. ábra mutatja hogy mekkora hatással van az gradiens keresési eljárás a megtanult súlyokra. Látszik hogy a két eredmény merőben más optimumra állt be. Mint ahogyan a 4.6. táblázatból is látszik, az SGD és az ADAM hasonló általánosítást ért el. Az is nagyon jól látszik a képeken hogy az ADAM optimalizátor úgy alakította az együtthatókat, hogy lényegesen kissembb változtatásokat tegyen a felületen, ezért egyáltalán nem ugrált annyira a súlyok értéke mint az SGD-nél. Viszont mind a kettő változat a veszeségfüggvény kvázi-stabilítási pontját 2000 iteráció után érte el, onnantól már csak oszcilláltak a maradék 18000 lépésben.



4.5. ábra. A veszeségfüggvény és a hálózat súly átlagainak a rétegenkénti alakulása ADAM optimalizátor mellett.

4.4.2. A CIFAR-10 mérések

Az MNIST és a CIFAR különbségei Miután az MNIST adathalmazon sikeres méréseket végeztem egészen egyszerű MLP hálózatokkal kíváncsi lettem hogy ezek a hálózatok mennyire viszik át a teljesítményüket egy fokkal bonyolultabb adathalmazra. A két adathalmaz



4.6. ábra. A veszteségfüggvény és a hálózat súly átlagainak a rétegenkénti alakulása SGD optimizátor mellett.

tulajdonságai egymáshoz képest:

- Mindkét adathalmaz 10 osztályt tartalmaz.
- Mindkét adathalmaz 10'000 teszt képet tartalmaz, 1000-et osztályonként.
- A CIFAR-10es adathalmaz 10'000-rel kevesebb tanító képet tartalmaz, ez osztályonként 1000 darab.
- Amíg az MNIST 28x28as fekete fehér képeket tartalmazott, addig a CIFAR-10 32x32-es színes képeket tartalmaz.

Ami rögtön szembetűnik hogy amíg az MNIST 784 dimenziót tartalmazott, addig a CIFAR-10 egy 3072 dimenziós adathalmaz, tehát az adathalmaz komplexitása ha csak tisztán a dimenziókat nézzük akkor a négyeszeresére nőtt. Ennek megfelelően a hálózatok ha ugyanazokat a hálózatokat próbáljuk használni, akkor drasztikus teljesítmény csökkenést vagyunk kénytelenek tapasztalni.

4.6. táblázat. A saját MLP teljesítménye az CIFAR adathalmazon, 20 000 tanító batch után. Bemenetek száma: 3072, reLu aktivációval, L2 reguralizációval, aminek az együtthatója 0.0005 volt. A súlyokat csonkolt normál eloszlással inicializáltam, standard szórás: 0.1, középérték: 0.0.

neuron struktúra	Optimalizáló algoritmus	Teszt szet hiba százalék
10	SGD learning-rate: 0.5	80%
10	SGD learning-rate: 0.001	75%
400-10	SGD learning-rate: 0.001	55%
500-500-2000-30	SGD learning-rate: 0.001	57.5%

4.4.3. Az előtanítás eredményeinek összefoglalása

Arra a számonra meglepő eredményre jutottam hogy az előtanítás az én méréseim közben minden esetben csak rontott a hálózat klasszifikációs teljesítményén. Az általam várt eredmény az lett volna, hogy az előtanítás jelentősen javítani fog a hálók teljesítményén, főleg nagy neuron számánál. A 4.7. táblázat és a 4.8. táblázat mutatja be az eredményeimet.

Úgy magyarázom ezeket az elszomorító eredményeket hogy az előtanítás következtében egy rossz lokális minimumba ragadt be a háló, ahonnan nem tudott kijönni egyik esetben sem. Ahhoz hogy ezt a feltevésemet igazolja megnéztem az MNIST adathalmazon tanított nagyméretű (500-500-2000-30 neuron) hálónak a veszteségfüggvényét és a pontosságának

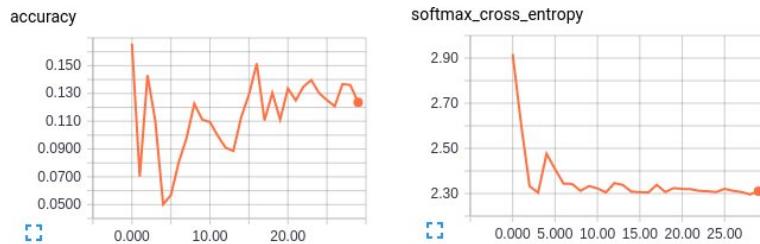
4.7. táblázat. Az előtanított MLP hálózatok eredménye a MNIST adathalmazon, 25 elemű batchekkel.

neuron struktúra	nem-felügyelt/felügyelt epoch szám	Optimalizáló algoritmus	Teszt szet hiba százalék előtanítással / nélküle
800 - 10	15/30	SGD learning-rate: 0.05	92% / 0.9%
500-500-2000-30	15/30	SGD learning-rate: 0.001	88.5% / 3.2%

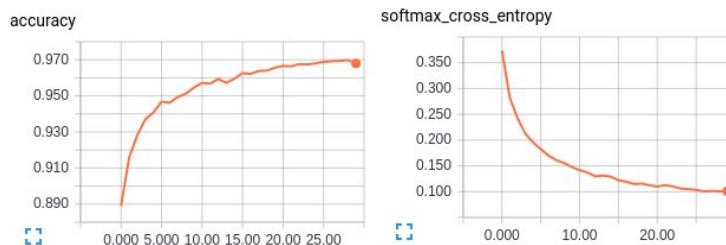
4.8. táblázat. Az előtanított MLP hálózatok eredménye a CIFAR adathalmazon, 25 elemű batchekkel.

neuron struktúra	nem-felügyelt/felügyelt epoch szám	Optimalizáló algoritmus	Teszt szet hiba százalék előtanítással / nélküle
400 - 10	10/10	SGD learning-rate: 0.001	91% / 55%
500-500-2000-30	10/10	SGD learning-rate: 0.001	88.5% / 57.5%

a növekedését a validációs adathalmazon. Ez előtanítás nélküli háló mért eredményei a 4.8. ábrán, az előtanítással tanított hálóé pedig a 4.7. ábrán láthatóak. Az ábrákon szépen kijön hogy amíg az előtanítás nélküli háló minden mértékben tekintve egy viszonylag sima, egyenletes görbület ír le, ami egy minimumhoz konvergál, addig az előtanított háló beragadt egy rossz lokális minimumba, ahonnan nem tud kikerülni. Ebből azt szűrtettem le, hogy érdekes módon nem feltétlen azok a jó jellemzők osztályozáshoz, amik a rekonstrukcióhoz.



4.7. ábra. A 500-500-2000-30 neuronú előtanított MLP veszteségfüggvényének, és a validációs halmazon való pontosságának alakulása.



4.8. ábra. A 500-500-2000-30 neuronú nem előtanított MLP veszteségfüggvényének, és a validációs halmazon való pontosságának alakulása.

4.5. A konvolúciós hálózatokkal elvégzett méréseim eredményei

Miután láthattuk hogy az MLP-k már nehezen birkóznak meg a CIFAR-10 komplexitású feladatokkal a figyelmemet az elméleti részben oly sok helyet elfoglaló konvolúciós architektúrák felé fordítottam. Mivel ezek az architektúrák érik el jelenleg a legjobb eredményeket a neurális jelfeldolgozás széles területén, nem csak képosztályozásban hanem akár beszédszintetizálásban

is, ezért nagy reményekkel fordultam ezekhez a struktúrákhöz. Az eredményeimet a 4.9. táblázat mutatja az MNIST adathalmazon, és a 4.10. táblázat a CIFAR-10 adathalmazon. Az alábbi felsorolás az általam használt hálózatokat írja le.

1. conv2d(5,5,32) -> flatten -> softmax
2. conv2d(5,5,32) -> maxpool(2,2) -> flatten -> fullyconnected -> dropout(0.5) -> softmax
3. conv2d(5,5,32) -> maxpool(2,2) -> conv2d(5,5,64) -> maxpool(2,2) -> flatten -> fullyconnected(1024) -> dropout(0.5) -> fullyconnected(1024) -> softmax
4. conv2d(5,5,32) -> maxpool(2,2) -> lrn -> conv2d(5,5,64) -> lrn -> maxpool(2,2) -> flatten -> fullyconnected(1024) -> dropout(0.5) -> fullyconnected(1024) -> softmax

4.9. táblázat. A saját CNN teljesítménye az MNIST adathalmazon, 20 000 tanító batch után. Bemenetek száma: 3072, reLu aktivációval, L2 reguralizációval, aminek az együtthatója 0.0005 volt. A súlyokat csonkolt normál eloszlással inicializáltam, standard szórás: 0.1, középérték: 0.0.

A struktúra sorszáma	Optimalizáló algoritmus	Teszt szet hiba százalék
1	ADAM	8%
2	ADAM	1.2%

4.10. táblázat. A saját CNN teljesítménye az CIFAR adathalmazon, 30 000 tanító batch után. Bemenetek száma: 3072, reLu aktivációval, L2 reguralizációval, aminek az együtthatója 0.0005 volt. A súlyokat csonkolt normál eloszlással inicializáltam, standard szórás: 0.1, középérték: 0.0.

A struktúra sorszáma	Optimalizáló algoritmus	Teszt szet hiba százalék
1	ADAM	33.23%
2	ADAM	33.51%
3	ADAM	26.9%
4	ADAM	26.0%

Látszik hogy az MNIST-en is egész jól teljesítenek a konvolúciós hálózatok, de arra az adathalmazra elég egy egyszerű MLP modellező képessége is. Ami viszont érdekesebb hogy a CIFAR-10-en, ahol láttuk hogy az MLP hálózatok igen szerény eredményeket érnek el, ott a CNN hálózatok már elkezdenek az MLP hálózatoknál lényegesen jobb eredményeket produkálni. Ennek a magyarázata a transzlációs invariancia által a súlyokra vetített kényszerek, amik természetesen jelen vannak a képi adatokban. Cserébe viszont a konvolúció miatt lényegesen megnő a számítási igény, és a hálózat végén található teljesen kapcsolt rétegek miatt bekövetkezett paraméter tér robbanás miatt hosszabb ideig is kell tanítani ezeket a hálózatokat. A lokális válasz normalizálás (Local Response Normalization) az én esetemben is 1%-ot javított a teszt hiba százalékon, mint ahogyan azt Hintonék is tapasztalták a "ImageNet Classification with Deep Convolutional Neural Networks"[27] nevű publikációban. Cserébe a gradiensek kiszámolásának az idejét 174 ms-ról, 355 ms-re emelték. Viszont megemelkedett memória igényük csak 3 MB volt. Ez érhető, hiszen ez az operáció nem foglal plusz memória területet mint a paraméterek, de a deriváltjának numerikus kiértékelése adott pontban igen számításigényes.

A Legjobb eredmény amit konvolúciós hálózattal a CIFAR adathalmazon elértem az 18% százalékos hiba volt, a fent ismertetett lokális válasz normált struktúrával (a negyedik a felsorolásban). A tanítás GPU-n 300'000 lépésen keresztül tartott. Ez már egy egészen tisztes eredménynek számít az adathalmazon.

4.6. A konvolúciós és az MLP hálózatok erőforrás igényének az összehasonlítása

Észrevételeim szerint az MLP hálózatok kis költségráfordítás mellett teljesítenek olyan jól az MNIST adathalmazon mint a konvolúciós hálózatok. Ugyanakkor úgy vélem hogy ez az előny csak az MNIST végtelen egyszerűségéből fakad. A CIFAR-10-es adathalmazon már jól észrevehető a konvolúciós hálózatok elsöprő fölénye, ha komplex képosztályozási feladatokról van szó.

Két struktúrának mértem össze a paramétereit az MNIST adathalmazon. Egy MLP-ét, és egy konvolúciós hálózatét. A hálózatok az alábbi struktúrával épültek fel, ebben a sorrendben:

- input(784) -> fc(500) -> fc(500) -> fc(2000) -> fc(30) -> fc(10)
- input -> conv2d(5,5,32) -> pool(2,2) -> conv2(5,5,64) -> pool(2,2) -> fc(1024)-> dropout(0.5) -> fc(1024) -> softmax

Érdemes megjegyezni hogy az MLP paraméter tere kb 1,8 millió paraméterből áll, amíg a konvolúciós hálózat kb 2,7 millió paraméterből. Habár eleinte úgy éreztem hogy a konvolúciós háló paraméter tere kisebb lesz, mégis az intuícióm becsapott.

Mindkét hálózatban a gradiens kiszámítása igényelte a legtöbb erőforrást. Ennek a számításnak a részleteit a 4.11. tábla mutatja az MLP hálózatra, és a 4.12. tábla a konvolúciós hálózatra. A táblázatokból jól látható hogy a kovoluciós hálózat lényegesen több erőforrást fogyasztott. Ha az erőforrások növekedése lineárisan skálázódna a paraméter tér beli növekedéssel, akkor jóval kissember növekvény lett volna várható.

4.11. táblázat. Az MLP hálózat legmagasabb erőforrás igényű részei az MNIST adathalmaznál

Hálózati rész	Memória	Számítási idő
gradiens	20.3 MB	12.5 ms
Az első teljes réteg gradiense	3.15 MB	12.1 ms
A második teljes réteg gradiense	2.03 MB	9.82 ms
ADAM optimizer	8 B	9.37 ms
Harmadik teljes réteg	750 KB	3.09 ms
Negyedik teljes réteg	11.3 KB	2.82 ms

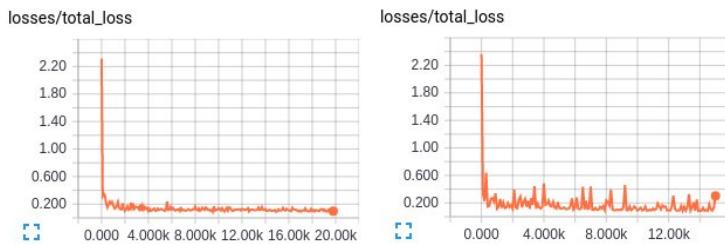
4.12. táblázat. A konvolúciós hálózat legmagasabb erőforrás igényű részei az MNIST adathalmaznál

Hálózati rész	Memória	Számítási idő
gradiens	114 MB	94.8 ms
A második konvolúciós réteg gradiense	44.6 MB	90.9 ms
Az első konvolúciós réteg gradiense	27.5 MB	94.7 ms
ADAM optimizer	8 B	50.1 ms
Első teljes réteg	192 KB	43.6 ms
Második előrekesztő konvolúciós ág	2.3 MB	36.8 ms

A fenti két táblázatból több érdekes következtetést is levontam, ezeket az alábbi felsorolásban összegzem:

- A konvolúciós háló paraméter tere lényegesen nagyobb lett, pedig én intuitívan kissébbnek gondoltam volna mint az MLP-ét. Azt sejtettem hogy a paraméter tér robbanását a két teljesen csatolt réteg fogja okozni, de ennyire nagyra nem számítottam.
- A számítást magasan a gradiens kiszámítása dominálja mind a két hálónál, viszont ahhoz képest hogy a CNN paraméter tere kb 1.5-szer akkora, a gradiens számítás ötször annyi memóriát, és kb nyolcszor annyi számítási időt igényel mint az MLP esetében.
- A CNN-ben a paraméter tér robbanásáért felelős egy batch számításánál elhanyagolható többletmunkát jelentenek.

A 4.9. ábra azt mutatja hogy az MNIST-en legjobban teljesítő, 800 neuronos MLP, és az előbb említett CNN veszteségfüggvényei hogyan viszonyulnak egymáshoz. Látható hogy amíg a CNN egy kicsit ugrál, nem tud beálni egy stabil pontba az adathalmaz kicsi mérete miatt, addig az MLP gond nélkül hoz konzisztens eredményeket a veszteségfüggvényben.

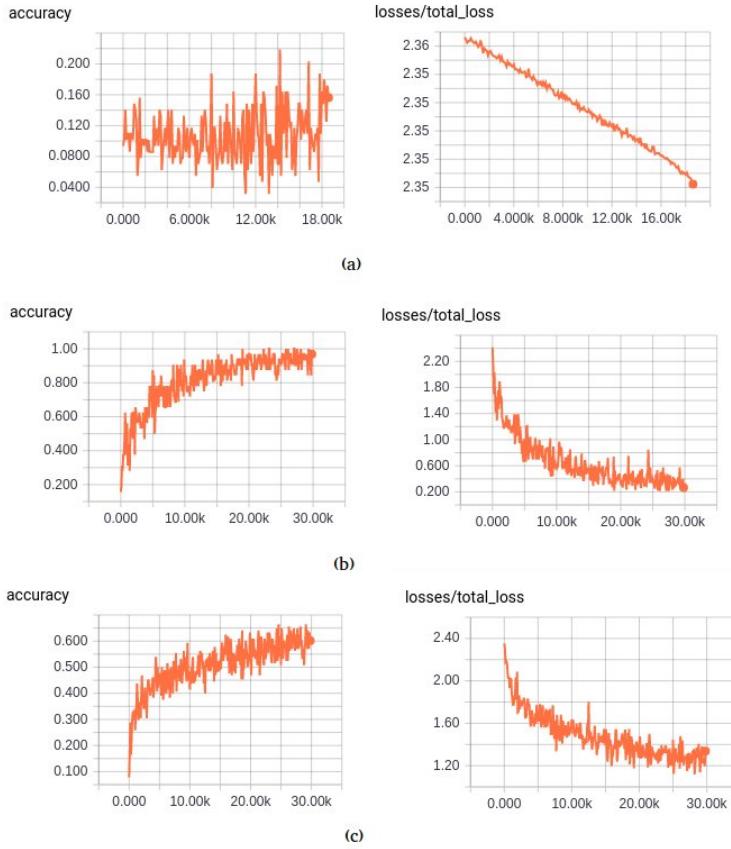


4.9. ábra. Egy MLP és egy CNN tanítása az MNIST adathalmazon..

Ugyanez a két metrika a CIFAR-10 adathalmazon a 4.10. ábrán látható. tisztán látszik hogy ide ez az MLP hálózat már kevés volt. Sőt, én sokkal mélyebb MLP függvényekkel sem tudtam lényegesen jobb eredményt elérni, ezt tisztán mutatja hogy az ábra (a) pontja egy mély MLP háló, megfelelő előtanítás nélkül beragadt egy lokális minimumba és képtelen volt onnan tovább haladni. A (b) ábra a konvolúciós háló eredményét mutatja ami lényegesen simább lefutású volt, jobb eredményekkel. A (c) rész az a 800 neuronos 1 rejtett rétegű MLP volt, ami a már megismert 50%-os hibánál kezdett el szaturálni. Ahogyan azt a 4.6. táblázat eredményei is mutatják.

4.7. A hibrid architektúrák eredményei

MNIST A fent ismertetett hibrid architektúrákból az MNIST adathalmazzal csak azt probáltam ki, ahol 1 darab RBM van. Ezt megtettem a scikit-learn SVM-jében (ez a libsvm python kötésekkel végülis) elérhető összes kernel implementációval, és 1-2 paraméter kombinációjukkal. Az eredményeket a 4.13. táblázat szemlélteti. A táblázaton látszik hogy a legtöbb kernel nagyjából ugyanúgy teljesít, kivéve a sigmoid kernel, mert az kiemelkedően rosszul teljesít a többihez képest. Ennek az oka feltehetőleg numerikus instabilitás volt.



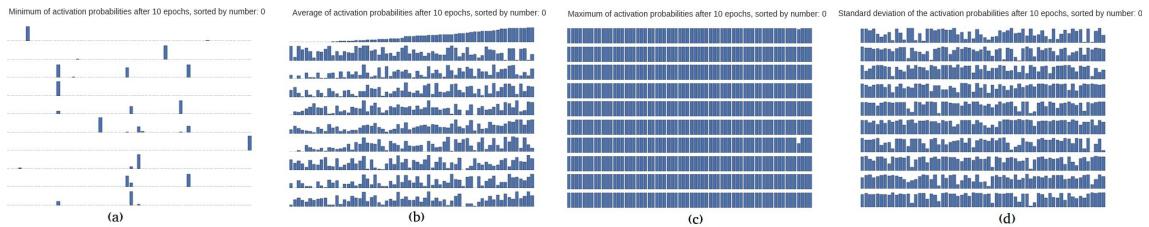
4.10. ábra. Az ábra (a) pontja egy mély MLP háló, megfelelő előtanítás nélkül beragadt egy lokális minimumba és képtelen volt onnan tovább haladni. A (b) ábra a konvoluciós háló eredményét mutatja ami lényegesen simább lefutású volt, jobb eredményekkel. A (c) rész az a 800 neuronos 1 rejtett rétegű MLP volt, ami a már megismert 50%-os hibánál kezdett el szaturálni. A méréseket a CIFAR-10 adathalmazon végeztem, a tanító pontokon.

Aktivációk vizsgálata Miután ilyen jó eredményeket hozott a 64 rejtett neuronos RBM az adathalmazon elkezdtem érdeklődni hogy vajon már a neuronok aktivációiból látszani fog-e az, hogy a kép melyik osztályba fog tartozni. Ehhez azt csináltam hogy az összes képre az MNIST-ben legeneráltam a rejtett neuronok aktivációs valószínűségét, és osztályonként vettem az átlagukat, a minimum és a maximum aktivációt. Ezeket az eredményeket a 4.11. ábra szemlélteti. Felülről lefelé helyezkednek el a számok 0-tól 9-ig, és a 0 átlagos aktivációs valószínűségei szerint van rendezve, hogy lássam esetleg szabad szemmel a triviális korrelációkat. Mint látható szabad szemmel semmilyen triviális korreláció nem vehető ki az értékekben, viszont azt érdekesnek találtam hogy a minimum aktivációknál van 1-2 nagyon erős marker ami mindenképpen jelen van az osztályban, ha egy osztályozó ezt a mintázatot megtanulja, akkor már ez alapján is egészen jó esélyteljes zárhat ki mintákat ha ezek az aktivációk hiányoznak. Abban is biztosak lehetünk hogy aminek ilyen magas a negatív értéke, azoknak nagyon kicsi lesz a szórása, mert egyszerűen nincsen hova szórniuk, mivel minden neuron legalább egyszer maximális értéket vesz fel. Persze egy valamire való algoritmus magasabb rendű összefüggéseket tanulna meg. Tehát összefoglalva azt szűrtetem le hogy az aktivációk véletlenszerűnek látszanak az aktivációk közpértékénél, de valójában

4.13. táblázat. Az $RBM + SVM$ kombináció eredménye az MNIST adathalmazra különböző kernelekkel.

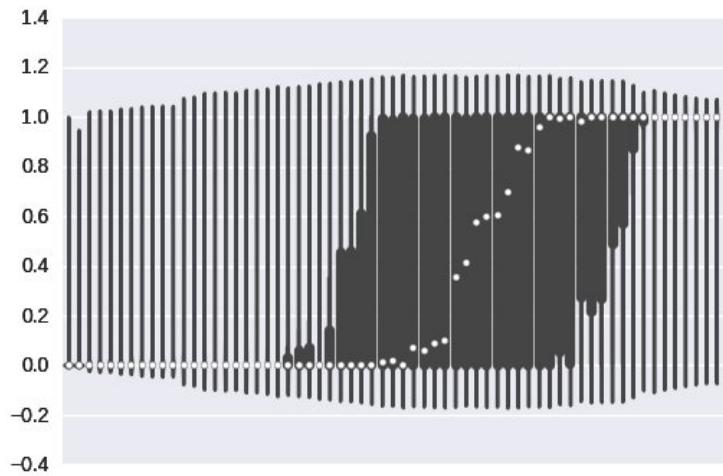
A kernel	RBM méréte	Teszt szet hiba százalék
linear	64	7.3%
rbf	64	6.0%
poly coef0=0 degree=2	64	7.1%
poly coef0=1 degree=2	64	5.8%
poly coef0=0 degree=3	64	6.0%
poly coef0=1 degree=3	64	5.6%
sigmoid coef0=0	64	89%
sigmoid coef0=1	64	89%

nagyon jól disztingváló aktivációk is keletkeznek.



4.11. ábra. Az RBM neuronjainak az aktivációs valószínűségeinek az (a) minimuma, (b) maximuma és (c) középértéke (d) szórása 10 epoch után. Felülről lefelé helyezkednek el a számok 0-tól 9-ig, és a 0 átlagos aktivációs valószínűségei szerint van rendezve.

Hogy ezt egy kicsit tovább vizsgáljam a nullás számra csináltam egy hegedű grafikont is, amit az 4.12. ábra mutat. A sok adat miatt nem lett egészen hegedű grafikon alakja, de nagyon jól szemlélteti amit látni szerettem volna. Van pár neuron aminek az aktivációja teljesen véletlenszerű az egész tartományban elterül, de van olyan is ami nagyon jól lokalizált. Sőt, nem egy neuron van ami egy pontba ömlik össze, tehát a nullás szám minden esetében 1-et vagy 0-át vesz fel.



4.12. ábra. Az RBM aktivációinak a hegedű grafikonja a nullás számra vetítve.

Naiv CIFAR Miután az MNIST eredményeken látszik hogy az RBM által eszközölt dimenzió csökkentéssel összekötött jellemző kiemelés lerövidítette a számítási időt, és a

nyers pixeleken alkalmazott SVM-hez képest nem rontott az osztályozás pontosságában elvégeztem ezeket a méréseket a CIFAR adathalmazon is. Ott kiábrándító eredményeket kaptam, az alábbi táblázat közli őket. 4.14.. Mint látszik ez a 90%-os teszt hiba elfogadhatatlan. Már tanítás közben gyanus volt hogy a 139.0-as pontos rekonstrukció hiba túl nagy az MNIST-en tapasztalt 0.17-0.12-höz képest. Hosszas lamentálás után rájöttem hogy a probléma az hogy a CIFAR-10 nem normalizált, hanem 0 és 255 között terjednek az értékek, és az RBM különösen érzékeny az adatok normalizálására, mert a háló 0 és 1 közötti valószínűségekkel dolgozik.

4.14. táblázat. Az $RBM + SVM$ kombináció eredménye az CIFAR adathalmazra különböző kernelekkel.

A kernel	RBM mérete	Teszt szet hiba százalék
poly coef0=1 degree=3	763	90%
poly coef0=1 degree=3	261 + 261 + 261	90%

Normalizált CIFAR-10 Miután normalizáltam az adatokat a rekonstrukciós hiba leesett a várt értékre. És a klasszifikációs eredmények is nagyon sokat javultak, ezt a 4.15. táblázat mutatja. Érdekes megfigyelni hogy a színek szerint szétszedett SVM lényegesen jobban teljesített mint az egyben tanított, pedig ugyanannyi neuront használtak és ugyanaddig tanítottam őket. Mindkét struktúra esetében jelentős teljesítmény javulást értem el a nyers adatokon tanított SVM-hez képest, egyenként kb 2 óra alatt futottak le a 7 helyett. Sőt ha az olvasó visszaemlékszik a nyers SVM teljesítménye 55% hibaszázalék volt, ettől az egyben tanított SVM alig marad le 1%-al, a szétdarabolt pedig 4%-al jobban teljesít. Ezen felül az MLP-nél is jobb teljesítményt nyújtott, ami meglepő annak fényében hogy ez a megoldás lényegesen kisebb paraméter térrrel rendelkezik.

4.15. táblázat. Az $RBM + SVM$ kombináció eredménye az CIFAR adathalmazra különböző kernelekkel.

A kernel	RBM mérete	Teszt szet hiba százalék
poly coef0=1 degree=3	763	56%
poly coef0=1 degree=3	261 + 261 + 261	51%

A hibrid architektúra értékelése Úgy gondolom hogy a jelentős sebességnövekedés, és ha jól ki van alakítva akkor a klasszifikációs teljesítmény növekedése indokolttá teszi az ilyen struktúrák alkalmazásának megfontolását egyes szcenáriókban.

4.8. A CNN és az SVM+RBM összehasonlítása

A neurális hálózatok és a kernel metódusok között van egy ciklusosság hogy éppen melyiknek a kutatása jár előrébb, és melyiket érdemes használni. Én személyes érdeklődésből hoztam be a dolgozatomba az $RBM + SVM$ kombinációját, pedig ez nem egy kifejezetten populáris kombináció. Egy féligen meddig építettem a konvolúciós hálózat a CIFAR-10 osztályozásának a problémájára kevesebb erőforrásból és kevesebb idő alatt 25%-al pontosabb eredményt tudott hozni mint az előbb említett hibrid megoldás, és még a legjobb SVM-es eredménynél [45] is fél százalékkal jobb eredményt volt képes hozni. Az <http://rodrigob.github.io/>

[are_we_there_yet/build/classification_datasets_results.html#43494641522d3130](#)-en összeszedett listából az látszik hogy az eddig elért legjobb eredmény az adathalmazon **3.47%** ami **21.43%-al** jobb mint a legjobb SVM-es megoldás. Az SVM-ek és a neurális hálózatok világa egyelőre viszonylag jól szeparált, ennek az az oka hogy a neurális hálózat kimenetén lévő softmax osztályozó réteget még nem sikerült lecserélni egy összetettebb klasszifikátorra, például egy SVM-re azért mert az ebből való gradiens visszaterjesztés egyelőre túlnyúlik a számítási kapacitásainkon, pedig ez is egy nagyon érdekes útvonal lenne hogy hogyan lehetne a két módszert kombinálni és vajon milyen eredményre vezetnének. A mai szeparált világnézzel vannak olyan adatstruktúrák ahol manapság is az SVM-ek hoznak dominánsan jobb eredményeket, viszont sok - az emberek számára fontos - klasszikus feldolgozási feladatnál, ahol spaciális kényszerek vannak jelen az adathalmazban ott a konvolúciós hálózatok hoznak lényegesen jobb eredményeket.

4.9. A három megközelítés végső összehasonlítása

Mint említettem, úgy gondolom hogy ha az ember ki tud a tanulás jellege miatt térbeli struktúrákat használni, akkor nagyon indokolt eredményekkel kell alátámasztani hogy az ember a konvolúciós neurális hálózatokkal szemben, egy másik architektúra javára döntsön. Ehhez viszont számolni kell a megnövekedett erőforrás igényekkel tanítás közben. Az MLP-ket úgy tudom elképzelni hogy a közeljövőben egy nagyobb architektúra részeként lehet majd használni, mint például az adaptív aktivációs függvényeknél ahogyan azt a "Learning Activations Functions to Improve Deep Neural Networks"[12] is taglalja. A kernel gépeknek most egy kicsit hidegebb korszakát éljük, mert ezeket a struktúrákat tudtommal lényegesen nehezebb nagyon nagy adathalmazokon tanítani, de úgy gondolom hogy mivel még ezeken a módszereken is nagyon sokan dolgoznak esetleg elköpzelhető hogy pár év múlva ugyanakkora népszerűségnek fognak örvendeni mint 2012-ben. De én most nem látnám indokoltnak hogy ezeket a struktúrákat használjam képosztályozási feladatokra, még fejlesztett lokalitású[45] kernelekkel sem.

5. fejezet

Az eredmények összefoglaló értékelése

Megszerzett tudás Ebben a szakdolgozatban úgy vélem hogy az egyetemen megszerzett tudásbázisomra alapozva képes voltam lényegesen tovább bővíteni az ismereteimet a gépi tanulás, ezen felül is a képosztályozás téma körében. A következő ismeretek megszerzését tartom kifejezetten előnyösnek:

- A domináns irányzatok és architektúrák megismerése a neurális hálózatokkal való képosztályozásban.
- Az egyes adathalmazok megismerése amin az ember érdemben össze tudja hasonlítani a munkáját másokéval.
- Több fejlesztési keretrendszer futólagos, és a Tensorflow -mint a neurális hálózatok fejlesztésére alkalmas eszköz- mélyebb megismerése
- Tapasztalat szerzése abban hogy hogyan kell teljesítmény méréseket végezni a hálózatokon, amik egy erőforrás korlátozott alkalmazásnál igen fontosak lehetnek.
- A neurális hálózatok tanítására használt hardverek lehetőségeinek és korlátjainak megismerése, illetve futólagos érintése az olyan új megoldásoknak mint az Nvidia Jetson platform.

A végső konklúziók levonása Mint azt a dolgozatom elején is említettem, ezeket az eljárásokat azzal a céllal értékeltettem hogy ki, hogy egy valós projektben legyenek majd majdan alkalmazva, ahol videofelvételeken szeretnénk majd objektumokat felismerni egy erőforrás korlátos rendszerben. Arra a konklúzióra jutottam hogy a Tensorflow a megfelelő keret rendszer lenne a céljainknak, mégpedig azért mert igen könnyen tudnánk vele a hálózatokat fejleszteni, finomhangolni, nagyteljesítményű GPU griden tanítani, majd egy erőforrás korlátos beágyazott rendszerre telepíteni. Architektúralis szempontból én úgy gondolom hogy a projektnek legmegfelelőbb alap struktúra a konvoluciós hálózatok lennének, mivel igazán nagy memória és számítási igényük csak tanítás közben van, de utána egy

korszerű beágyazott GPU gyorsító - mint az Nvidia Jetson - kb 300 képet tudnak kiértékelni másodpercenként, ami számunkra kielégítő sebesség.

További munka, és fejlesztési lehetőségek Az elsődleges célunk az lesz hogy a jelenlegi tudásunkra építve egy prototípus képosztályozó rendszert fejlesszünk a dolgozat elején említett robotrepülőgépes projekthez. Ehhez hozzá tartozik majd további irodalomkutatás a konvolúciós hálókkal való videofeldolgozás területén, ami egy nagyon izgalmas terület. Egy másik potenciális, lényegesen alapkutatás jellegűbb irány amivel külső konzulensem foglalkozik. Mint a láthattuk a modern hálózatoknak hatalmas paraméter terük van, és az ebben való optimalizálás komoly problémákat tud okozni algoritmikus komplexitása, és erőforrás igénye miatt. Egy megfelelően választott hibafüggvényel viszont esetlegesen el lehetne érni hogy csak egy kisebb, de azonos tulajdonságokkal rendelkező hálót keljen tanítani.

Köszönetnyilvánítás

Szeretném megköszönni Daróczy Bálintnak hogy elvállalta hogy a külsősként konzultáljon a szakdolgozatom elkészítésében, nagyon sokat tanultam tőle. Illetve szeretném még megköszönni dr. Strausz Györgynak aki belső konzulensként szerepet vállalt a szakdolgozatom és az önálló laborom elkészítésében is, valamint mindig segített racionalizálni az általam választott témat hogy ne vállaljam túl magamat.

Irodalomjegyzék

- [1] The cifar datasets. <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [2] Imagenet. <http://image-net.org/>.
- [3] Keras. <https://keras.io/>.
- [4] Kubernetes, production-grade container orchestration. <http://kubernetes.io/>.
- [5] The mnist dataset. <http://yann.lecun.com/exdb/mnist>.
- [6] Robust and parallel file reading and distributing pipeline in tensorflow. https://www.tensorflow.org/versions/r0.11/how_tos/reading_data/index.html.
- [7] Scaling convolutional networks. https://www.tensorflow.org/versions/r0.11/tutorials/deep_cnn/index.html.
- [8] Tensorboard, visualizing learning with tensorflow. https://www.tensorflow.org/versions/r0.12/how_tos/summaries_and_tensorboard/index.html.
- [9] Tensorflow-slim, a high level api for tensorflow. <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/slim>.
- [10] Emile Aarts and Jan Korst. Simulated annealing and boltzmann machines. 1988.
- [11] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- [12] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.
- [13] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [14] Dennis Decoste and Bernhard Schölkopf. Training invariant support vector machines. *Machine learning*, 46(1-3):161–190, 2002.

- [15] Horvath Gabor. *Neuralis halozatok*. Hungarian Edition Panem Könyvkiadó Kft., Budapest, 2006.
- [16] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.
- [18] G Gybenko. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.
- [19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.
- [20] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.
- [21] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- [22] David W Hosmer and Stanley Lemeshow. Introduction to the logistic regression model. *Applied Logistic Regression, Second Edition*, pages 1–30, 2000.
- [23] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.
- [24] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.
- [25] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.
- [26] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [28] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, pages 536–543. ACM, 2008.

- [29] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10(Jan):1–40, 2009.
- [30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [31] Zhouhan Lin, Roland Memisevic, and Kishore Konda. How far can we go without convolution: Improving fully-connected networks. *arXiv preprint arXiv:1511.02580*, 2015.
- [32] Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.
- [33] Marvin Minsky and Seymour Papert. Perceptrons. 1969.
- [34] J Moody, S Hanson, Anders Krogh, and John A Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4:950–957, 1995.
- [35] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.
- [36] Alexander G Ororbia II, C Lee Giles, and David Reitter. Learning a deep hybrid model for semi-supervised text classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP), Lisbon, Portugal, 2015a. UR L <http://www.david-reitter.com/pub/ororbia2015learning-deep-hybrid.pdf>*.
- [37] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [38] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.
- [39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [40] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *AISTATS*, volume 1, page 3, 2009.
- [41] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *arXiv preprint arXiv:1309.2388*, 2013.

- [42] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.
- [44] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.
- [45] Kai Yu and Tong Zhang. Improved local coordinate coding using local tangents. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1215–1222, 2010.
- [46] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.