Általános információk, a diplomaterv szerkezete

A diplomaterv szerkezete a BME Villamosmérnöki és Informatikai Karán:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletes és pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő LaTeX-Diplomaterv sablon dokumentum tartalma.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2.5cm, baloldalon 1cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon - az első négy szerkezeti elem kivételével - szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2 táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le (A LaTeX ehhez kézenfekvő megoldásokat nyújt).

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben (a LaTeX-sablon a BibTeX segítségével mindezt automatikusan kezeli). Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

- A szakdolgozat készítő / diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető !
- Mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet !

Jó munkát, sikeres szakdolgozat készítést ill. diplomatervezést kívánunk !

FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Mesterséges neurális hálózatok fejlesztése TensorFlow alapon

Szakdolgozat

Készítette
Kemény Károly

Konzulens
dr. Strausz György

2016. október 11.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Kemény Károly*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ diplomatervet (nem kívánt törlendő) 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 átvettem, egyértelműen, a forrás megadásával megjelöltem.

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 diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2016. október 11.

Kemény Károly
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

Bevezető

0.1. Motiváció

A neurális hálózat A neurális hálózat egy számítási modell amelyet számos, algoritmikailag nehéz problémára sikeresen lehet alkalmazni. A fő alkalmazási területeik: osztályozási feladatok, regressziós feladatok, dimenzió csökkentés (kernel PCA), 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 rendszerektől kezdve egészen a rákos daganatok detektálásáig elterjedt a felhasználásuk.

2. *idősor előrejelzés:* Komplex idősor előrejelzésnél is előszeretettel használják óket, 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 kínaiban is képes volt tartani. [16]

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 a terület él, fejlődik és napról napra formálja a jelfeldolgozásról alkotott képünket.

Jelen Dolgozat célja A dolgozat fő csapásiránya egy áttekintő kép alkotása a neurális hálózatok felhasználásáról, különös tekintettel a neurális hálózatokkal történő képosztályozásra. A dolgozatomban arra törekszem hogy a neurális hálózatok gyakorlati alkalmazását ezen az alterületen keresztül ismerjem meg. Mivel tanulmányaim során ilyesféle - matematikai vonatkozású - programozással nem találkoztam jelentős mértékben, ezért ez egy jó alkalom arra hogy lássam hogy komplex matematikai absztrakciók hogyan öltenek testet szoftverként. Például a sztochasztikus generativ vagy a konvolúciós neurális hálózatok hogyan fordíthatóak le effektív programmá, és hogyan lehet belőlük értékes alkalmazásokat készíteni. A szakdolgozat egy hosszabb projekt része, ahol egy Tegra mobil chipen kell majd képosztályzó algoritmusokat alkalmazni. 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.

Érdeklődésem a neurális hálózatok iránt Az önálló labor munkám során és a szakirányomon oktatott kooperatív és tanuló rendszerek tárgyban találkoztam először ezzel a megközelítéssel. Az egyre modernebb ajánló algoritmusok keresése közepedte rá kellett eszmélnem hogy az ajánlórendszerek jövője is összefonódik a neurális hálózatokéval, így ez a dolgozat habár első látásra nem is látszik, de az önálló laboratóriumi projekt munkám továbbvitele. Egy potenciálisan érdekes, még alig kutatott terület, hogy a termékek hibás és hiányos leírását hogyan lehet augmentálni a termékek feltöltött képeiből kinyert címke felhővel. Erre nagy valószínűséggel alkalmazható hálózatok lennének a regionális konvolúciós hálózatok amiket érintőlegesen tárgyalok majd a konvulúciós hálózatokat taglaló fejezetben.

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 minnél 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 helyet és számítást igényelnek az egyes modellek. (Ez erősen példáúl függ a modell paraméter terének nagyságától)

A szakdolgozat végére reményeim szerint az egyetemi tárgyak anyagát továbbfejlesztve egy nagyobb rálátással fogok rendelkezni erre az izgalmas területre.

A dolgozat kontextusa Ahhoz hogy kontextusba helyezem jelen munkámat a bevezető további részében szeretnék 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 hadsereg "US

office of Naval Research" laboratóriumában. Már ezt is vizuális minták felismerésére alkották meg eredetileg.

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. 1985-ben egy forradalmi újítás, a hibavisszaterjesztéses algoritmus (backpropagation algorithm [14] 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 teljesíményt tudtak elérni, kevesebb tanítással mint a kor hálózatai.

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 kapacítás 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. A manapság a neurális hálózatok jelen vannak az élet minden területén ahol szükségünk van mintázatok intelligens felismerésére, lehet az a szolgáltatás hang, kép vagy akár szöveges dokumentumok. Google translate, Shazam, a netflix díj nyertes ajánlási algoritmus, csak hogy pár példát szemelvényezzek a számtalan közül.

A dolgozat felépítése A dolgozatomat az alábbi módon szeretném felépíteni:

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.

   (c) Mély hiedelem hálózatok.

   (d) Konvolluciós Neurális hálózatok

   (e) A legújabb fejlemények a neurális képfeldolgozás területén.

3. A saját fejlesztéseim bemutatása, amely egy két egyszerűbb struktúra a fent bemutatottak közül a tensorflow könyvtárral.

4. A mérési eredményeim kiértékelése, tanulságok levonása.

1. fejezet

A Tensorflow 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 és az információk 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 koránt sem helytálló. A szakdolgozatom nulladik lépseként számos más - a neurális hálózatok fejlesztését támogató - könyvtárat vettem szemügyre. A teljesség igénye nélkül: 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 a megfelelő 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 leszármazottjának tekinthető. A DistBelief csendben meghúzódva, de ott dolgozik a fejlett világ majdnem minden emberének az élete mögött, lévén hogy az Alphabet cégcsoport (A Google holding szerű vállalta) több mint 50 csapata adaptálta, és vértezte fel általa intelligenciával alkalmazását. Pár ismertebbet kiemelve: Google Search, Adwords, Google Maps, SteetView, Youtube, és természetesen a Google Translate. Miután évek tapasztalata gyülemlet fel az első generációs könyvtáruk 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.

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 munkamenetetek (?session?-ök) példányosítanak. Ezek a munkafolyamatok inicializálják a változókat, és a munkafolyamat 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 minnél 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 ismertetném.

1.3. A Tensorflow futás közben

A Tensorflow serving kiszolgáló rendszer Miután a kutatólaborokból kikerültek az új gépi tanulás modellek nem elég őket csak publikálni, a cégek komoly hasznot 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 kiprobálását, üzembe helyezését és A/B tesztelését, miközben egy stabil, verziózott API-t biztosít a kliensek számára. Természetesen ez mit sem érne ha nem skálázna gond nélkül hatalmas magasságokba, ezért egyszerűen integrálható a Kubernetes 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 gigabyteos sebességet érhetnek el, ugyanakkor a mátrix szorzás - ami a gépi tanuló algoritmusoknak egy kardinális eleme - kubikus 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 ugyan az, a súlyokat már megtanultuk, de most az egész modellünket egy mobil eszközön kell futtatni. Ez az eszköz az inferenciát jácint könnyedséggel 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. Ez jelenti az igazi skálázódást, asztali számítógépen fejlesztem, adatparkon tanítom, és egy mobil eszközön futtatom, mindezt jelentősebb kód újraírás nélkül. Ékes 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 jásztik, az egyik a szöveget ismeri fel a képen (feltehetően egy faster R-CNN), a másik pedig az effektív fordítást végzi.

2. fejezet

A Neurális hálózatokkal való képfeldolgozás lehetőségeinek áttekintése

A kutatómunka egy jelentős részét tette ki a dolgozatomnak, mivel az évek folyamán nagyon sok sikeres és sikertelen kísérlet született annak érdekében hogy hogyan lehetne neurális hálózatokkal képi adatokat feldolgozni. Talán mondhatjuk hogy ezek a hálózatok a legsikeresebb képosztályozó, de korántsem triviális hogy mik az előnyeik, hátrányaik és az egyes típusok milyen komplexitású jelekkel képesek megbírkózni. Először szeretném bemutatni az adathalmazokat amiken ezeket az algoritmusokat kiértékelik, majd eljutni az többrétegű perceptronoktól a konvolúciós hálózatokig, végül a legújabb trendek ismertetésével zárni 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 való kiértékelésére. Mivel ezekre az adathalmazokra sokat fogok hivatkozni, ezért szeretném őket egy-egy bekezdésben bemutatni.

MNIST Az MNIST adatbázis fekete fehért 20x20 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 legalapabb adathalmaz

CIFAR-10 A CIFAR-10 egy jóval összetettebb adathalmaz, 60'0000 annotált 32x32 pixeles, színes képet tartalmaz. A képek 10 osztályra vannak felosztva, osztályonként 6000 képpel. Az adathalmazból 50'000 kép van tanításra, és 10'000 tesztelésre fenntartva.

CIFAR-100 A CIFAR-100 felépítése megegyezik a CIFAR-10el, de osztályrendszere az előbbinél lényegesen összetettebb, 100 osztályt tartalmaz és minden osztályhoz 600 képt 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.

IMAGENET Az IMAGENET a világ legnagyobb képgyüjteménye, a WordNet lexikális adatbázis szinoníma halmazai szerint vannak annotálva a képek. Jelenlegi statiszikái:

- 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 puszta méreteivel is messze túlszárnyalja. Ezt mondhatjuk az etalon benchmarknak. Az évente megrendezett, a gépi látás "olimpiájának" számító ILSVRC(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 metrika

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.

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

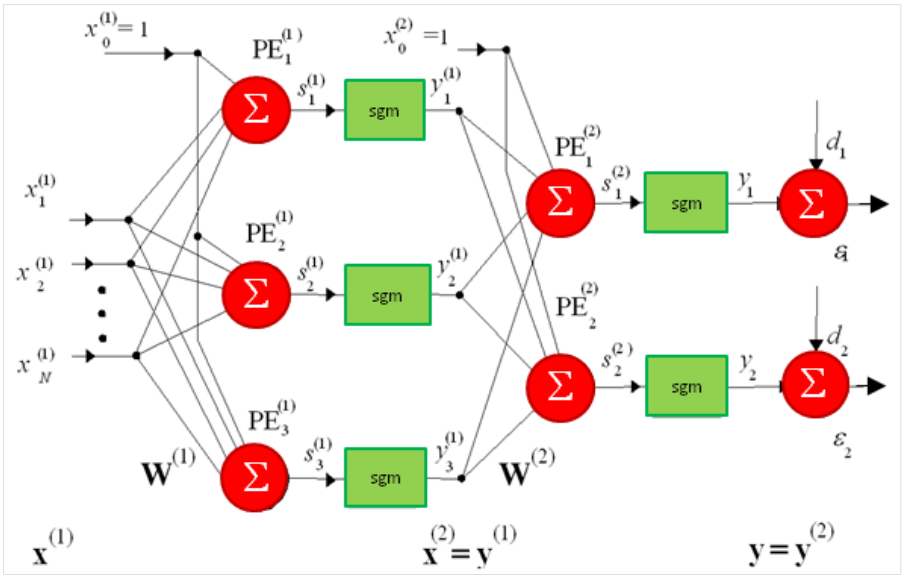
2.2. A Legelterjedtebb neurális hálózatok képfeldolgozáshoz

Bevezetés Az évek során számos neurális hálózattal kísérleteztek a kutatók annak érdekében hogy rájöjjenek melyek képesek legjobban megtanulni a képekeben 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 vagy a dropout metódus. Csak a hálózatok felépítésének és tanításának architektúrális 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, 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.



2.1. ábra. *Egy kétrétegű MLP hálózat.*

Backpropagation Az áttörés 1986-ban jött, amikor Geoffrey E. Hinton kollégáival sikeresen alkalmazta a hibavisszaterjesztéses algoritmust 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 pseudokód összefoglalását a 2.1.-ábrán látható hálózat tanításához a 2.1.-lista mutatja.

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-kiement(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
```

Az MLP teljesítménye képosztályozási feladatokra

Aktivítás a területen. Gondolhatnánk hogy ezt a témát már rég elfelejtették a kutatók, de mivel az MLP egy igen egyszerű struktúra, ezért folyik még néhány kutatás hogy a határait megtalálják.

MNIST Az MLP teljesítménye képosztályozási feladatok tekitetében igen szerény a többi hálózathoz képest, de az MNIST adathalmazzal még ez is egész jól megbírkózik, néhány figyelemre méltóbb eredményt a 4.4. táblázat foglal össze. A többi adathalmazon a naiv MLP nem hoz értékelhető eredményt, ennek az okait mindjárt megvizsgáljuk.

2.1. táblázat. *Az MLP teljesítménye az MNIST adathalmazon*

Rétegek száma	neuron struktúra	Teszt szet hiba százalék
3-réteg	768-300-10	4.7

MLP hiányosságai

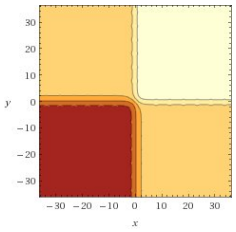
Ez az egyetlen eredmény amely értelmesen értelmezhető a naiv, csak felügyelt tanítással tanított MLP tekintetében. Ennek számos oka van, ezeket vizsgálom most meg.

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 hibafűggvény 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 az alább egyszerű függvény $\sigma^2(\sigma(x) + \sigma(y))$ kontúr diagrammja (2.2. ábra). Ez habár nem közvetlenül egy hibafüggvény, de ezt a tulajdonságát jól szemlélteti.

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 kapacítás kell a képekben megjelenő bonyolult struktúrák megtanulásához. Sajnos az előbbi cél csak a hálózat növelésével érhető el, az MLP paraméter tere viszont nagyon rosszl skálázódik. Ha veszünk egy 6 rétegű hálózatot aminek a rétegjei 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 nyilván valóan hatalmas. Ez is tanítható sikeresen, de ahhoz már a továbbiakban bemutott kisegítő neurális struktúrák szükségesek.

2.2.2. A Korlátozott Boltzmann Gép

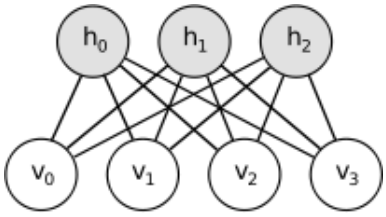
Bevezetés A korlátozott boltzmann gép (továbbiakban RBM - Restricted Boltzmann Machine, 2.3. ábra) egy szochasztikus generatív neurális számítási modell. Működése az eddig tárgyalt MLP-től gyökeresen eltér, funkciója a bemenet jellemzőinek (featureinek)



2.2. ábra. *Példa egy komponált szigmoid függvényre.*

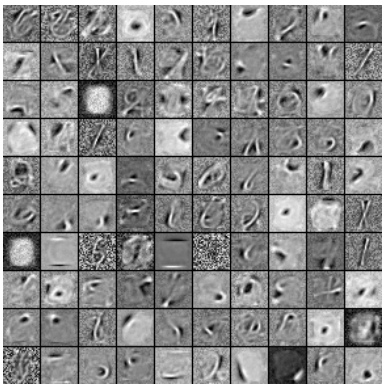
a megtanulása, és nem az egyes minták helyes osztályozása. Az intuitív jelentősége abban rejlik hogy feltehetjük hogy a naív MLP a kapzsi tanulása miatt nem képes megtanulni a minták valódi reprezentációját, de ha az MLP súlyait úgy tudnánk inicializálni úgy, hogy a mintákban lévő szabályosságokat már eleve ismerje, akkor ebből könnyebben meg tudja 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 méylebb hálókra az RBM és az Autoencoder többrétegű megfelelőit a mély hiedelem hálózatokat, és a "stacked" autoencodereket. 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. hozott ki egy áttekintő tanulmányt az előtanulásról a mélyebben érdeklődőknek "Exploring Strategies for Training Deep Neural Networks" [**?**] címmel.

Az RBM struktúrája Az RBM mint említettem egy sztochasztikus generatív számítási modell, amelyben fontos hogy az egyes neuronok egy páros gráfot alkotnak (2.3. 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 [9]-es referenciában talál bőséges irodalmat az RBM tanítását illetően. Itt csak az alapokra szorítkozok.



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

Az RBM tanítása Az RBM azért érdekes megközelítés a többi hálózathoz képest, mert probablisztikus alapokon nyugszik. Az úgynevezett energia alapú hálózatok felfoghatóak úgy, hogy a hálózat minden konfigurációjához tartozik egy $p(x)$ valószínűség, hogy mekkora



2.4. ábra. *Egy RBM által megtanult filterek. Forrás:* `http://www.pyimagesearch.com/wp-content/uploads/2014/06/rbm_filters.png`

valószínűséggel tartózkodik a háló az adott konfigurációban. Azt szeretnénk elérni hogy az alacsony energiájú konfigurációknak nagy legyen a valószínűsége. Ez formalizálva a következő képpen néz ki:

$$p(x) = \frac{e^{(-E(x))}}{Z} = \sum_h \frac{e^{(-E(x,h))}}{Z} \tag{2.1}$$

$$Z = \sum_x e^{(-E(x)}) \tag{2.2}$$

Ahol az E az energiafüggvényt 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, azért hogy a Hopfield hálózatok gyengeségeit kiküszöbölje. A (**??**) képletet felhasználva megalkothatjuk a hibafüggvényünket, amely a negatív logaritmikus valószínűségi függvény (negative log likelihood function) lesz:

$$\mathcal{L}(\theta, x) = \frac{1}{N} \sum_{x(i) \in \mathcal{D}} log(p(x^{(i)}) \tag{2.3}$$

Definiáljuk a szintén a temodinamikából származó szabad energia függvényt:

$$\mathcal{F} = -log \sum_h e^{-E(x,h)} \tag{2.4}$$

Ezzel újradefiniálhatjuk a valószínűségi függvényt:

$$p(x) = \frac{e^{-\mathcal{F}(x)}}{Z} \quad \text{ahol} \quad Z = \sum_x e^{-\mathcal{F}(x)} \tag{2.5}$$

Ami megengedi hogy a következőt írhassuk fel:

$$-\frac{\partial logp(x)}{\partial \theta} \approx \frac{\partial \mathcal{F}(\S)}{\partial \theta} - \frac{1}{|\mathcal{N}|} \sum_{\widehat{x} \in \mathcal{N}} \frac{\partial \widehat{x}}{\partial \theta} \tag{2.6}$$

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

$$-\frac{\partial logp(v)}{\partial W_{ij}} = E_v[p(h_i|v) * v_j] - v_j^{(i)} * sigm(W_i * v^{(i)} + c_i) \tag{2.7}$$

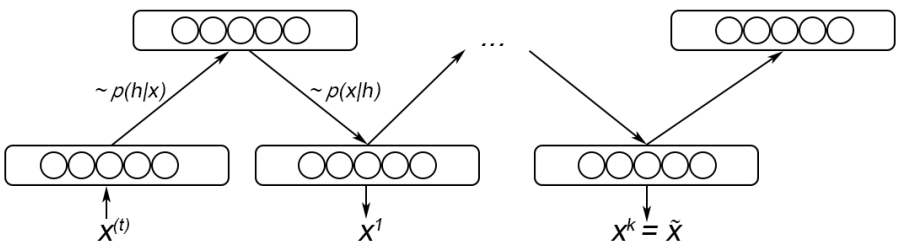
$$-\frac{\partial logp(v)}{\partial c_i} = E_v[p(h_i|v)] - sigm(W_i * v^{(i)}) \tag{2.8}$$

$$-\frac{\partial logp(v)}{\partial b_j} = E_v[p(h_i|v) * v_j] - v_j^{(i)} \tag{2.9}$$

Ebből látható hogy az egyes deriváltak kiszámításához szükségünk van a rejtett neuronok valószínűségére a bemeneti neuronok állapotától függően. Ezért szükséges hogy a boltzman gép korlátozott legyen, tehát egy páros gráp 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 valózsínűségi mintavételét a 2.5. ábra szemlélteti. A gibbs mintavételezéshez tartozó markov lánc lépéseinek a képleteit a (**??**) és a (**??**) képlet írja le.

$$h^{n+1} \quad \sim \quad sigm(W^T v(n) + c) \tag{2.10}$$

$$v^{n+1} \quad \sim \quad sigm(W^T h^{(n)} + b) \tag{2.11}$$



2.5. ábra. *A gibbs mintavételezési eljárás. Forrás:* `http://recognize-speech.com/images/nicolas/Gibbs.png`

Ahol ⌋ és (b) az eltolás súlyvektorokat jelzik.

Az RBM és DBN előtanulás eredményei Az hogy a hálót előtanítjuk fantasztikus eredményjavulást hozott, amely beigazolta hogy valóban a számok valós jellemző reprezentációihoz 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*

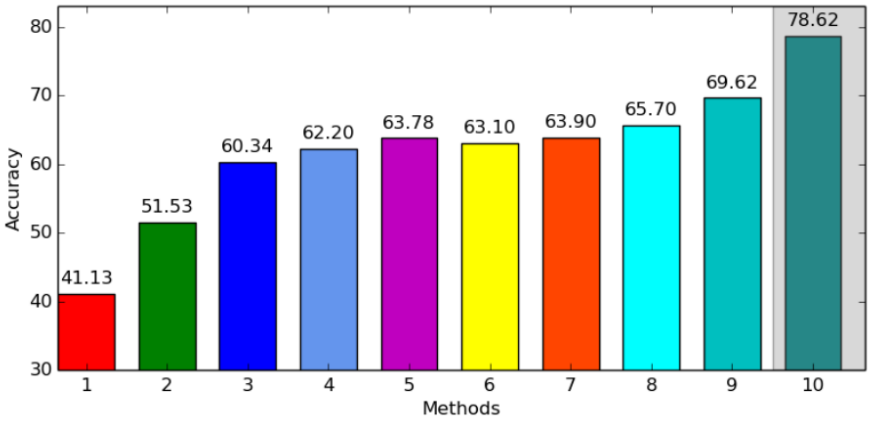
Rétegek száma	neuron struktúra	Teszt szet hiba százalék
3-réteg	768-800-10	0.7
6-réteg	784-2500-2000-1500-1000-500-10	0.35

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, viszony mint azt pár paragrafussal előbb megemlítettem a paraméter tér csökkentésében még van fejleszteni való ezeken a struktúrákon. Egy igen friss publikáció amelynek címe "How far can we go without convolution: Improving fully-connected networks"[15] 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 $1500 * 2000 = 3000000$, 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 terég 112 millióról 2.5 millióra csökkentették, összehasonlítás képpen egy modern konvolluciós hálónak 3.5 millió paramétere van. Látható hogy ezzel sikerült a kutatóknak megoldania a többrétegű perceptron gépek egyik legnagyobb problémáját, a mértéktelenűl burjánzó paraméter teret. Ezen felül az eltünő gradiensek problémáját is orvosolja, amivel itt bővebben nem foglalkozunk. Viszont az eredményeit a CIFAR-10 2.6. ábra szemlélteti. Az előbb említett táblázat nagyon jól összefoglalja az MLP-k teljesítményének a fejlődését a CIFAR-10es adathalmazt használva.

2.2.4. Konvolluciós hálózatok

Bevezetés A szakdolgozatom harmadik nagy részét a konvolluciós hálózatok 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, kénytelen voltam új irányok után nézni. Így találtam meg a konvolluciós architektúrákat. Ezek korunk legjobban teljesítő architektúrái, ennek oka hogy a jelek nagy része amit fel szeretnénk dolgozni az emberi érzékszervek által érzékelt jelek, például képek. Ezek viszont jelentős transzlációs invarianciával rendelkeznek, és ha ezt az invarianciát nem kell megtanulni hanem bele tudjuk építeni a hálózat struktúrájába, akkor lényegesen kevesebb paramétert kell megtanulnunk mint egy *elméleti síkon* hasonló teljesítményű MLP-nél. Mivel természetesen tudjuk Cybenko (1989)[8] és Kurt Hornik

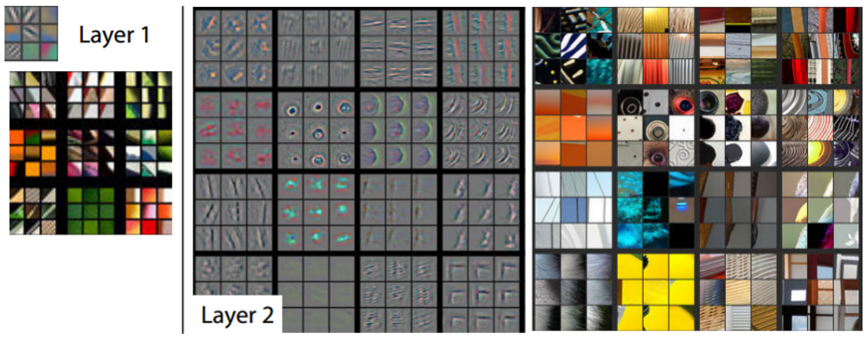


2.6. ábra. *Az MLP fejlődése a CIFAR-10 adahalmazon. (1) Logiszikus regresszió fehérített adatokon; (2) Tiszta backpropagation egy 782-10000-10 méretű hálózaton; (3) Tiszta backpropagation egy 782-10000-10000-10 méretű hálózaton. (4) Egy 10000-10000-10es hálózaton, RBM előtanítással, az utolsó réteg logisztikus regresszió; (5) Egyrétegű 10000 neuronos hálózat logisztikus regressziós kimenettel, RBM előtanítással; (6) "Fastfood FFT" model (7) Zerobias autoencoder hálózat 4000 rejtett 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. Az (1)-(5) eredmények Krizhevsky és Hinton 2009-es publikációjából származnak. A legutolsó azért szürkített, mert adat augmentációt használ.*

(1991)[10] 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 növelésével és hálózati kényszerek bevezetésével ez a paraméter tér jelentősen csökkenthető.

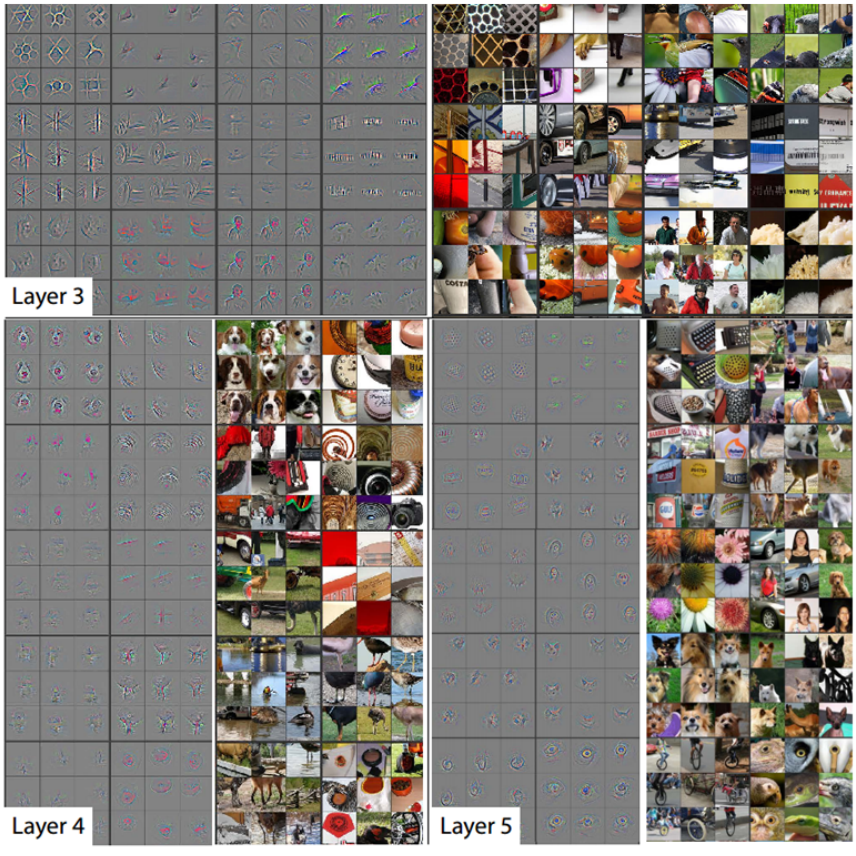
Az intuició A konvolúciós architektúra teljes mértékben biológiailag inspirált, a macsak vizuális kortexének a feltérképezésénél találtak hasonló kapcsolatokat az állat agyában, és ennek a mintájára építették fel a mesterséges hálózatot. Alapötlete hogy magába a hálózati struktúrába foglaljuk bele a jel transzlációs invarianciáját. A modellt eleinte képek feldolgozására alkották meg, és az előbbi mondat itt is szemléltethető a legintuitivabban. 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 össztetett jellemzők jelennek meg. És a hálózat az összetett jellemzők jelenlétéből következtet a kép osztályára. Ezt hivatott szemléltetni a 2.7. ábra és 2.8. 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[7] volt 1990-ben, amelyet irányítószámok, karakterek és hasonló dolgok felismerésére használtak, de a modell sokáig

nem kapott nagy érdeklődést.



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 labled Layer 2, we have representations of the 16 different filters (on the left)

2.7. ábra. *Az első két szint szűrői egy konvolúciós hálozatban. Forrás: "Visualizing and Understanding Convolutional Neural Networks" [13]*

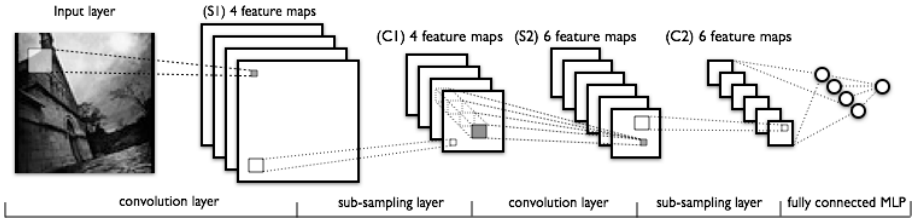


Visualizations of Layers 3, 4, and 5

2.8. ábra. *Az felsőbb rétegek szűrői egy konvolúciós hálozatban. Forrás: "Visualizing and Understanding Convolutional Neural Networks" [13]*

Matematikai interpretációja Tegyük fel hogy van egy $32x32x3$-as képünk. Ahhoz hogy egy teljes rétegbe kapcsoljuk bele mondjuk 1024 neuronnal ki kellene lapítanunk, és

egy $32 * 32 * 3 * 1024 = 3'145'728$ paraméterünk lenne az első rétegben. De tegyük fel hogy a legkissebb jellemző amit érzékelni akarunk az egy $3x3$ méretű patchen van a képen, viszont akárhol lehet, akkor ha az első rétegben 32 jellemzőt szeretnénk érzékelni, akkor csak $3 * 3 * 32 = 288$ paraméterre lesz szükségünk az első rétegben, ami jelentős redukció. Ezután a következő réteg ehhez a 288 neuronhoz fog kapcsolódni, és ha ott 64 neuron lesz akkor $3 * 3 * 64 = 567$ neuronra kell majd, amik az eredeti képből viszont már egy $5x5$-ös szeletet fognak indirekt módon lefedni. Ezt mutatja be egy klasszikus architektúra a leNet a 2.9. ábrán.



2.9. ábra. *Az felsőbb rétegek szűrői egy konvolluciós hálozatban. Forrás: "Visualizing and Understanding Convolutional Neural Networks" [13]*

Az IMAGNET 2012-ben az AlexNet nevű konvolluciós hálózat amelyet Alex Krizhevsky, Ilya Sutskever és Geoffrey Hinton alkottak fölényesen megnyerte az azévi ILSVRC versenyt. A háló top 5 hibája (az olvasó konzultáljon az adathalmazokat bemutató résszel a metrika leírásáért.) 16% volt, míg a második helyete ami egy SVM-eket használó modell volt 26%-os hibát produkált. Ez a 10%-os különség az egekbe emelte a konvolluciós hálózatok népszerűségét, és hivatalosan is elhozta a neurális képfeldolgozás korát. Innentől kezdve minden évben konvolluciós hálózatok nyerték meg az ILSVRC-t. A 2.3. táblázat bemutatja az egyes évek eredményeit a hálót néhány paraméterével együtt. Összehasonlítás képpen, egy átlagos ember teljesítménye az adathalmazon 5-10 hibaszázalék körül mozog.

2.3. táblázat. *Az ILSVRC győztesei*

Év	A struktúra neve	Tanítás ideje	Paraméter tér mérete	Top 5 hiba százalék
2012	AlexNet [2]	két GTX 580 GPU-n 5-6 nap	60 millió	16%
2013	ZF Net [13]	egy GTX 580 GPU-n 12 nap	60 millió	11.2%
2014	GoogLeNet [6]	"néhány high end GPU-n egy héten belül"	4 millió	6.7%
2015	Microsoft ResNet [3]	8 GPU-s gépen 2-3 hétig	N\A	3.6%

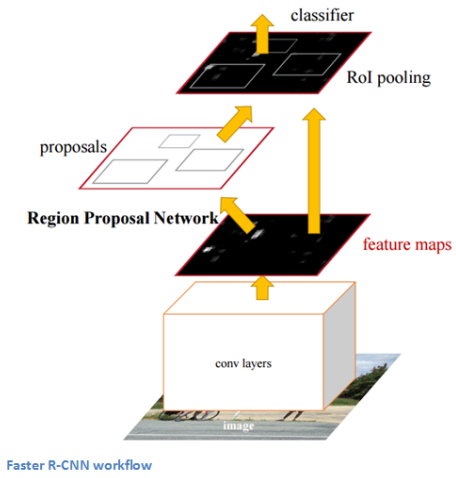
Értékelés Látható hogy a legújabb konvolluciós architektúrák már az emberi kiértékelésnél is pontosabb eredményt hoznak. Ráadásul a kezdeti naív megközelítést követően a paraméter tér is drasztikus csökkenésnek indult. Manapság ha valaki képosztályozási feladatot szeretne végezni neurális hálózatokkal, akkor egy ilyen előre elkészített architektúrát fog használni. Sajnos az is jól nyomonkövethető hogy a hálózatok fejlődésével a szükséges hardware kapacitás is meredek emelkedésnek indult. Ha az ember egy konvolluciós 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 éppen a tudomány tart, ebben a fejezetben röviden fel szeretném villantani a neurális képfeldolgozás legújabb és legérdekesebb irányait.

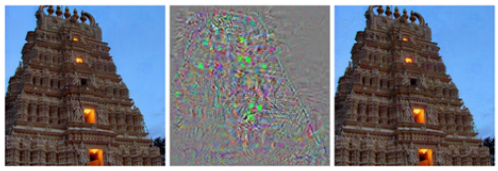
Régió alapú konvolúciós hálózatok[5] Sokan azt mondják hogy ez a publikáció csokor (R-CNN, Fast R-CNN, Faster R-CNN) hosszú idők óta az egyik legfontosabb amit új neurális architektúrákról olvashatott az ember. Eddig meg tudtuk mondani egy hálóval hogy tartalmaz-e a kép valamilyen objektumot. Az R-CNN hálózatok már az objektum pontos helyét is megmondják a képen, ami egy minőségbeli ugrást jelent. Az 2.10. 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.



2.10. ábra. *A Faster R-CNN munkafolyamata*

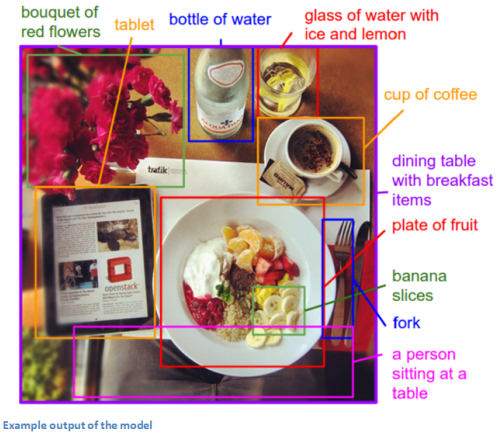
Generatív adverziális hálózatok[?] A LeCunn, a konvolúciós hálók megalkotója szerint ez az utóbbi 10 év legérdekesebb ötlete a területen. A lényeg hogy két hálózatot tanítunk tandemben, egy generatív és egy diszkriminatív modell-t. A diszkriminatív modell dolga edönteni egy képről hogy valódi-e vagy sem, a generatívé pedig hogy olyan képeket tudjon generálni amivel átveri a másik modell-t, ezért hívják adverziális hálozatnak. Az egész súlya abban rejlik hogy így a diszkriminatív hálózatnak meg kell tanulni az adat egy nagyon jó reprezentációját hogy képes legyen dönteni, ezzel mintegy nem felügyelt módon a

legfontosabb jellemzőket kiemelni a képből. A generátor a végére pedig képes lesz valósághű képeket "álmodni". A figrefadversarial-ábra mutat egy tipikus tanító példát.



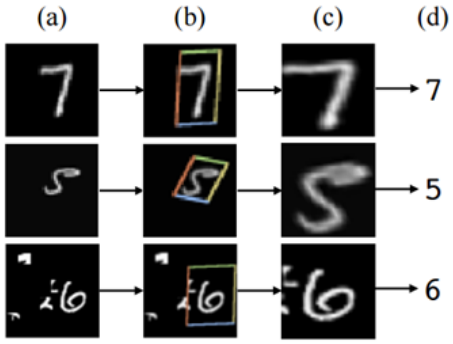
2.11. ábra. *Jobbra: Eredeti kép, középen: Pertubációk, balra: Pertubált kép. A jobb oldalit helyesen, a bal-t pedig hibásan osztályozná egy CNN.*

Képleírások generálása[1] A nem olyan távoli múltban nagyon sok érdekes publikáció jelent meg olyan neurális struktúrákról amelyek képek leírására alkalmasak. Lényegében egy CNN és egy RNN hálózat működik együtt, és fantasztikus dolgokra képesek. Tovább nem is taglalnám, mert nagyon sok előismeretet igényel a téma, az érdeklődők a megfelelő publikációt megtalálják az irodalomjegyzékben [?]. A 2.12. mutatja az eredményt.



2.12. ábra. *A Faster R-CNN munkafolyamata*

Térbeli transzformációs hálózatok (STN [4]) 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 2.13. ábra mutat a hálózat működésére egy példát.



2.13. ábra. *Egy térbeli transzformációs hálózat lépései.*

3. fejezet

Hálózatok impelmentálása és elemzése tensorflowban

A következő részben szeretném bemutatni saját mukámat és mérési eredményeimet. Az előző részben bemutatott hálózatfajtákból megvalósítottam számtalan 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. 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 volt, a másik pedig az SVM. Mindkettő bemenetére közvetlen a kép pixeljeit tettem.

3.2. A saját magam által kialakított fejlesztőkörnyezet

A Tensorflow ökoszisztéma nagyon jó pontja a csúcsra járatott monitorozási lehetőségek, 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, és a fejlesztéshez PyCharm IDE-t használni mert lényegesen gyorsabban lehet vele haladni komplex python kódnál mint a notebookokkal.

- *A modellek kiértékelése:* A modellek kiértékelésére úgy gondolom hogy a Tensorflowval érkező Tensorboard a legalkalmasabb, mint majd látni fogja az olvasó a modelleim 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 lenyúlni 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 saját implementációk bemutatása

3.3.1. 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 konvolúciós háló például 140 millió paramétert tartalmaz. A hálózat változóiról általánosságban a következő értékeket mentettem le: minimum érték, maximum érték, közép érték és a standardizált szórásukat. 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('stddev'):
            stddev = tf.sqrt(tf.reduce_sum(tf.square(var - mean)))
        tf.scalar_summary('sttdev/' + 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)
```

3.3.2. A logiszikus regresszió

Az első modell amit implementáltam Tensorflowban az a logisztikus regresszió volt. A célja az volt hogy összehasonlítsam a Tensorflow erőforrás igényét egy klasszikus python machine learning csomag, a Scikit-learn igényeivel. A struktúrát az <szám> ábra szemlélteti. Egyből látszódhat hogy az avatatlan szem számára a tensorboard eléggé nehezen értelmezhető lehet, ezért is szokták mondani hogy a tensorflownak a tanulási görbéje viszonylag nagy.

Szerencse hogy lehet benne névtereket létrehozni hogy a gráfok könnyebben átláthatóak legyenek. Az egész tanulási gráfot a járulékos nodeokkal az <szám> ábra szemlélteti.

3.3.3. 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 definicióját szemlélteti.

3.2. lista. *Egy MLP definiciója*

```
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 több félet 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 definicióját a (**??**)-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 definiciója a (**??**) 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 definiciója (**??**) képleten látszik.

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

$$reLu(x) = max(0, x) \tag{3.2}$$

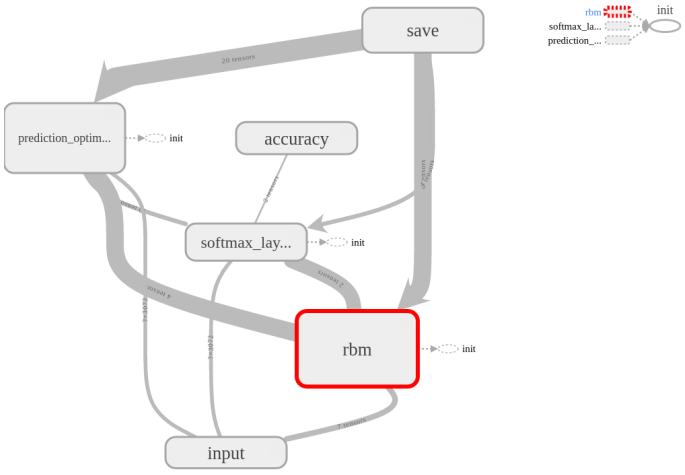
$$sgm(x) = \frac{1}{1 + e^{x}} \tag{3.3}$$

3.3.4. Az RBM hálózat

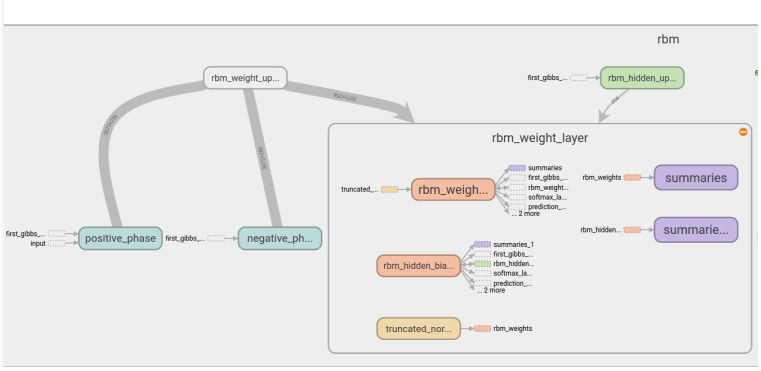
Mint mondtam a TF-Slim nagyon kényelmes volt addig amíg olyan struktúrákkal dolgoztam amik könnyen definiálhatóak. Viszont csak többrétegű perceptronok és konvolúciós hálózatokat lehet benne egyelőre alkotni. Az korlátozott boltzmann gép implementálásához le kellett mennem a Tensorflow alacsony szintű interfacéhez 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 gráfon belűl 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 tanító halmazom volt. Majd a bináris 0-1 döntést a következő képpen szimuláltam:

1. Kivontam a valósziníségi mátrixot a random számokból
2. Vettem az eljöjelét a keletkezett mátrixnak
3. Átvezettem egy relu rétegen, amitől 0 és 1 közé normalizálódott.

A tanítás A tanítást kétféle képpen végeztem el, egyrészről definiáltam a szabad energia függvényt és a keretrendszerrel számoltattam ki a (**??**) függvényből és különböző beépített optimalizátorokkal teszteltem, gradientdescenttel és adaptív gradiens (ADAM) optimizátorral. 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. Másrészt közvetlen kiszámoltam a gibbs sampling utáni értékekből (**??**), (**??**) és (**??**) függvényeket és egyszerűen csak hozzáadtam őket a súlyvektorhoz. A **??**. ábra a hálózat madártávlati struktúráját szemlélteti. 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 strukturját mutatja. Itt látszik igazán hogy a Tensorboard grafikonjai milyen kifonomult vizualizációt tesznek lehetővé. A tanítás optimalizálását a [9] alapján végeztem.



3.1. ábra



3.2. ábra

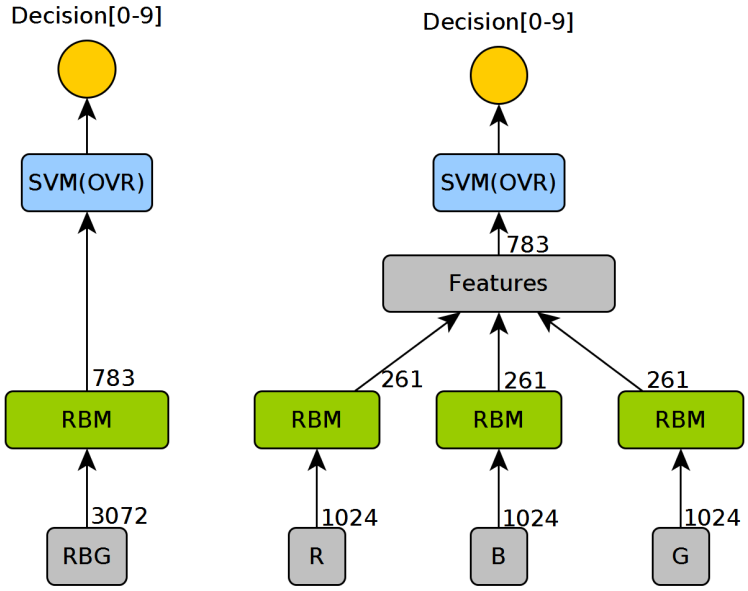
3.3.5. A DBN struktúra

Miután az RBM struktrát 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.

3.3.6. Hibrid modellek

A kísérletezésünk során elkezdtünk 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 hatalmas dimenzió csökkenést értünk el, pl 784-ről 64-re az MNIST esetében, ami jelentősen 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 sajnos nem is tudtam megtanítani reális idő alatt egy SVM-nek. Ezért letömörítettem az az 3072 dimenziós CIFAR-10 adathalmazt egy

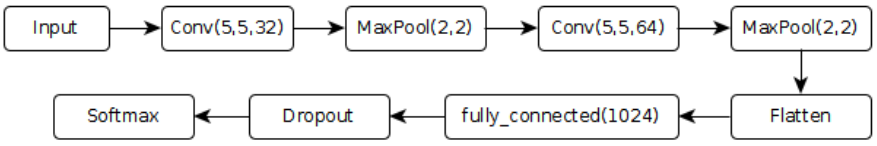
783 dimenziós térbe RBM-ek segítségével ami mindemellé még jellemző kiemelést is végzett, és ott tanítottam rajta a szupport vektor gépet, remélve hogy jó osztályozási eredményeket kapok. Ezt két megközelítésben próbáltam meg, az egyik az volt amikor egyetlen RBM-em volt, 3072 bemeneti és 783 kimeneti neuronnal, a másik az volt amikor 3 RBM-et tanítottam meg, minden színcsatornára egyet, és utána ezeket egy tömbbé összefűzve adtam át az SVM-nek. A két megközelítést a 3.3. ábra szemlélteti.



3.3. ábra

3.3.7. A Konvolúciós modell

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



3.4. ábra

Az egyes operációkat szeretném megymagyarázni 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 rétegben. A kétdimenziós konvolicó definicióját a (**??**)-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 konvolluciós rétegek egy magas absztrakcióval rendelkező, viszonylag transzláció invariáns jellemzőteret nyújtanak a képről, addig a hálózat végén lévő teljesen kapcsolt rétegek teszik lehetővé a nemlineáris leképzéseket ebben a jellemzőtérben. Az itteni teljesen kapcsolt réteget és a hozzá tartozó nemlineritásokat is az (**??**), (**??**), (**??**) írja le.

- *lrn:* Elvileg a reLu aktivációs függvény nem érzékeny annyira az adatok normalizálására, mégis kiderül a [2] publikációból hogy egy lokális normalizálási eljárás statisztikailag szignifikáns javulást képest hozni a hálózat osztályozó képességében. Ezt a matematikai struktúrát a (**??**) képlet szemlélteti. Ez is egy biológiailag inspirált eljárás. A bilógiai neve ennek az eljárásnak laterális inhibició.

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

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

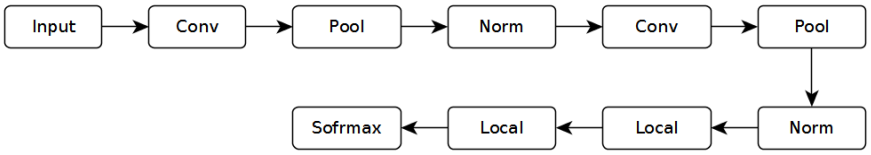
Ahol $a_{x,y}^i$ az i-edik kernel-t alkalmazzuk az (x,y) beli pozicióra.
$b_{x,y}^i$ a normalizáció utáni eredmény
És n a vizsgált pozició szomszédos kernel térképei, N az összes kernel a rétegben és α, β és k pedig további hiperparaméterek.

3.3.8. A konvolluciós modell skálázása

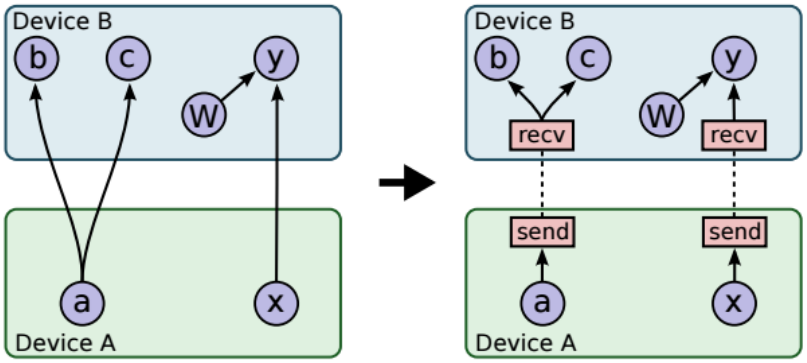
Az előbbiekben bemutatott konvolluciós modell könnyedén tanítható pár perc alatt egy korszerű CPU-n, viszont ha az ember nagyobb modelleket szeretne tanítani akkor ez már nem egy reális alternatíva. Ezekre az esetekre egy nagyobb hálózattal kísérleteztem a CIFAR-10 adathalmazon, aminek a felépítéséte a 3.5. ábra mutatja.

Ezt a hálózatot teszteltem CPU-n, 1 GPU-n, majd 2 GPU-n. 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 amúgy is tul nagy a hálónk, és nem férne el gradiens számítással

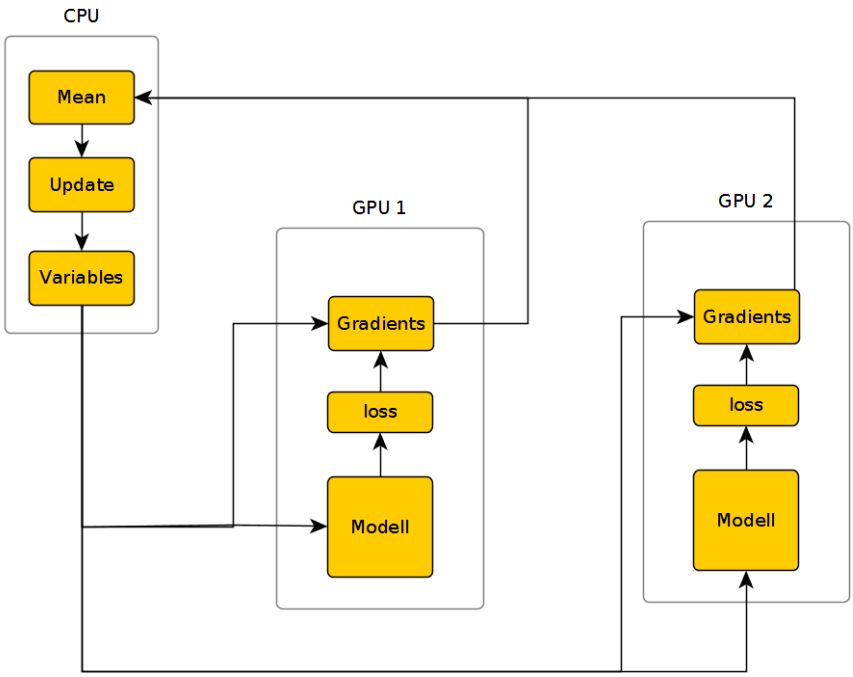


3.5. ábra



3.6. ábra

együtt rendesen egyetlen eszközön. De ha kissebb 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.



3.7. ábra

4. fejezet

A mérési eredményeim összegzése

Ebben a fejezetben szeretném bemutatni az általam elkészített hálózatok futásának mérési eredményeit. És ezeknek aspektusait:

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.

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

- 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.

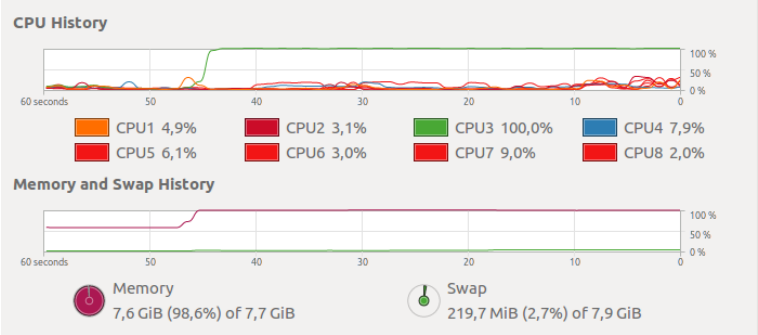
4.1. A keretrendszer általános teljesítménye

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

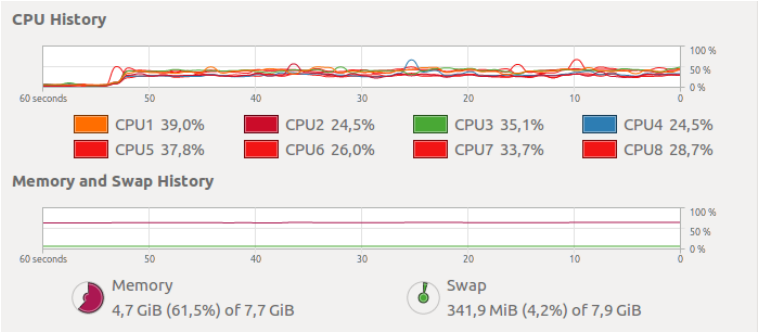
- Egyszerű rendszerszintű méréseket futtattam.
- A Tensorboardon keresztül vizsgáltam a hálók teljesítmény és erőforrás karakterisztikáját.
- Egy konvolúciós hálót kiskáláztam a CPU-ról 1 GPU-ra, majd 2 GPU-ra.

4.2. Rendszer szintű mérések.

Amikor nem sikerült lefuttatnom egy logisztikus regressziót Scikit segítségével a CIFAR-10 adathalmazon mert a teszt gép használhatlanná vált, akkor felütötte a fejét a kérdés nálam hogy mégis hogyan viszonyul teljesítményben a tensorflow a Python de facto gépi tanulás eszközéhez, a Scikit-learn könyvtárhoz. A két mérés erőforrás karakterisztikáját a 4.1. ábra és a 4.2. ábra mutatja, ahol az előbbi a Scikit-learn-ben elindított logisztikus regresszió, az utóbbi pedig a Tensorflowban megvalósított közelítő Logisztikus Regresszió.



4.1. ábra. *A Scikit-learnben idított logisztikus regresszió erőforrás igénye.*



4.2. ábra. *A Tensorflownban idított logisztikus regresszió erőforrás igénye.*

Mint látható a scikit-learnben elindított futtatás a teszt gép összes memóriáját felemésztette, illetve a magok kihasználása is nagyon rossz volt, mindössze 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 gép lefagyásának problémája eltünt, 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"[12]. Ezen felül még megpróbáltam az adathalmazt a newton konjugált-gradiens módszerrel megtanulni, de ez is sikertelen lett az erőforrások hiánya miatt. Ami érthető, hiszen a Newton-cg optimizátor gyors konvergenciát igér, 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 mindössze 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 szevereink 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 puszta nagysága miatt. Ezért is fontos hogy a mostani kutatások nagyrésze a numerikus optmalizáció terén a sztochatsztikus módszerekre koncentrál amik véletlenszerűen összeválogatott kis tanító halmazokkal approximálják a teljes adathalmaz tulajdonságait.

//TODO: GPU

4.3. A baseline eredmények ismertetése

Mint már említettem, a két alap metódus amihez a saját eredményeimet mértem a logsztikus regresszió, és egy egyszerű SVM volt, amihez polynomiális kernelt használtam 3 as fokszámmal és az első rendű tagokat használva. Az eredményeim az eredményiemet az MNIST adathalmazon a 4.1. táblázat, és a 4.2. táblázat szemlélteti.

4.1. táblázat. *A baseline mérések eremdénye 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. Ezeken sokat lehet javítani a képek előfeldolgozásával, de ez egy külön szakdolgozat témája lehetne, úgyhogy ezzel most itt nem foglalkozok. Minden struktúrát a nyers adatokon kezdtem el tréningezni, klasszikus előfeldolgozási lépések alkalmazása nélkül. A teljesség kedvéért megemlítem hogy a legjobb eredmény amit SVM-el értek el az MNIST adahalmazon annak a felépítése a következő volt: *Virtual SVM, deg-9 poly, 2-pixel jittered*, előfeldolgozási lépésként csak kiegyenesítést használt (deskewing), az elért teszt hibaszázalék pedig *0.56%* volt.

4.2. táblázat. *A baseline mérések eremdénye az CIFAR-10 adathalmaz nyers pixeljein, használt könyvtár: Scikit-learn*

tanuló metódus	Teszt halmaz hiba százaléka
Logisztikus regresszió	62%
Linear SVM with improved LCC	26%

Mint látni lehet itt ezek a metódusok már közel sem teljesítenek annyira fényesen. Az SVM-et nem én futtattam le, mert ahhoz a teszt gép nem volt elég erős. Az egy ICML 2010-es konferencián bemutatott eredmény volt, amit a "Improved Local Coordinate Coding using Local Tangents"[11]-ban publikáltak.

4.4. Az MLP-vel elvégzett méréseim eredményei

4.4.1. Az MNIST mérések

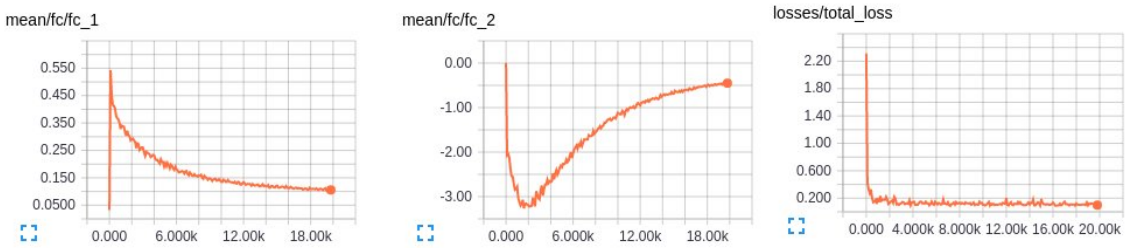
A többrétegű perceptronokról ismeretes hogy a rétegek között teljes kapcsolat van, és az adathalmaznak semmilyen tulajdonságát nem építik bele az architektúrába a priori. A konvolúciós hálózatok esetében, viszont a transzlációs invariancia előre feltételezett. és maga a hálózat tartalmazza a kényszereket. Ez alapvetően abban is megnyílvánul hogy a hálózat nem egy három dimenziós bemenetre (magasság, szélesség, mélység(RGB)) támaszaszkodik, hanem egy egyszerű vektorra. Az MNIST-en elért eredményeimet egy ilyen egyszerű struktúrával a 4.3. 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.

4.3. 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.*

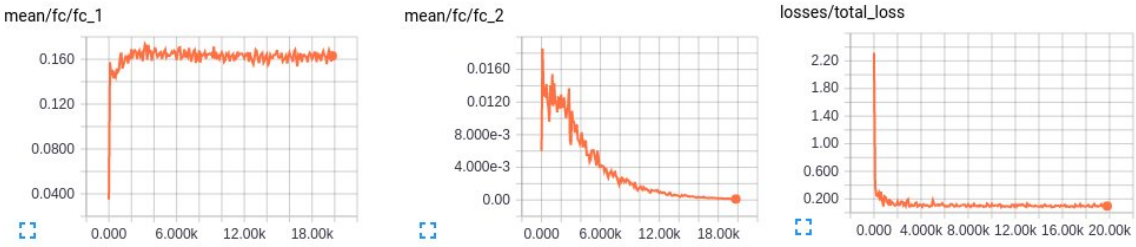
neuron struktúra	Optimalizáló algoritmus	Teszt szet hiba százalék
10	SGD learning-rate: 0.5	8%
1024-10	SGD learning-rate: 0.5	3.9%
400-10	SGD learning-rate: 0.5	2.02%
400-10	Adam	2.04%
800-10	Adam	2.21%
800-10	SGD learning-rate: 0.5	2.00%

Látható hogy az első struktúra a logisztikus regresszióval azonos eredményt ad, ami nem meglepő, hiszen egy egyrétegű struktúra softmax függvénnyel a végén praktikusan 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 képessegét, 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 elért 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 igénnyel szokott járni.

A 4.3. ábra és a 4.4. 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 4.4. tábláztból is látszik, az SGD-t használó hálózat sokkal közelebb került a teszt adathalmaz általános képéhez. Az is nagyon jól látszik a képeken hogy az ADAM optimizátor úgy alakította az együtthatókat, hogy lényegesen kissebb változtatásokat tegyen a felületen, ezért egyáltalán nem ugrált annyira a súlyok értéke mint az SGD-nél. Viszont mind a kettő a veszteségfüggvény stabilítási pontját olyan 2000 iteráció után érte el, onnantól már csak oszcilláltak a maradék 18000 lépésben.



4.3. ábra. *A veszteségfüggvény és a hálózat súly átlagainak a rétegenkénti alakulása ADAM optimizátor mellett.*



4.4. ábra. *A veszteségfüggvény és a hálózat súly átlagainak a rétegenkénti alakulása SGD optimizátor 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 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 négyszeresére nőtt. Ennek megfelelően a hálózatok ha ugyanazokat a hálózatokat probáljuk használni, akkor drasztikus teljesítmény csökkenést vagyunk kénytelenek tapasztalni.

4.4.3. Az előtanítás eredményeinek összefoglalása

Arra a számomra meglepő eredményre jutottam hogy az előtanítás az az én méréseim közben minden esetben csak rontott a hálózat klasszifikációs teljesítményén. Az általam

4.4. 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 vot. 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: 0.01	79 %
1024 - 10	SGD learning-rate: 0.01	90%

4.5. táblázat. *Az előtanított MLP hálozatok eredménye a MNIST adathalmazon, 25 elemű batchekkel.*

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

várt eredmény az lett volna, hogy az előtanítás jelentősen javítani fog a hálók teljesítményén, főleg nagy neuron számnál. A 4.5. táblázat és a 4.6. 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 tudodd kijönni egyik esetben sem. Ahhoz hogy ezt a feltevésemet igazolja mmegnéztem az mnist adathalmazon tanított nagyméretű (500-500-2000-30 neuron) hálónak a veszteségfüggvényét és a pontosságának a növekedését a validációs adathalmazon. Ez előtanítás nélküli háló mért eredményei a 4.5. ábrán, az előtanítással tanított hálónaé pedig a 4.6. ábrán láthatóak. Az ábrákon szépen kijön hogy amíg az előtanítás nélküli háló mindkét mértéket tenkintve egy viszonylag sima, egyenletes görbét ír le, ami egy minimumhoz konvergál, addig az előtanított háló beragadt egy rossz lokális minimumba, ahonnan nem tud kikerülni. Ebből azt szűrtem le, hogy érdekes módon nem feltétlen azok a jó jellemzők osztályozáshoz, amik a rekonstrukcióhoz. Pedig nekem egészen intutív lett volna, és az irodalomban is sokat olvastam róla hogy ez az előnye az RBM-ekkel való előtanításnak.

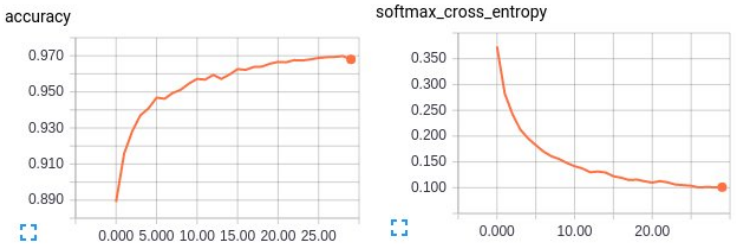
4.5. ábra. *A 500-500-2000-30 neuronú előtanított MLP veszteségfüggvényének, és a validációshalmazon való pontosságának alakulása.*

4.5. A konvolúciós hálózatokkal elvégzett méréseim eredményei

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

4.6. táblázat. *Az előtanított MLP hálozatok eredménye a CIFAR adathalmazon, 25 elemű batchekkel.*

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



4.6. ábra. *A 500-500-2000-30 neuronú nem előtanított MLP veszteségfüggvényének, és a validációshalmazon való pontosságának alakulása.*

is, ezért nagy reményekkel fordultam ezekhez a struktúrákhoz. Az eredményeimet a 4.7. táblázat mutatja az MNIST adathalmazon, és a 4.8. 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

4.7. 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 vot. 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%

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 CIFAR10-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 az én esetemben is 1%-ot javított a teszt

4.8. 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 vot. 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%

hiba százalékon, mint ahogyan azt Hintonék is tapasztalták a "ImageNet Classification with Deep Convolutional Neural Networks"[2] nevű publikációban. Cserébe a gradiensek kiszámolásának az idejét 174 ms-ről, 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, pusztán nehéz a deriváltját kiszámítani.

A Legjobb eredmény amit konvolúciós hálózattal a CIFAR adathalmazon elértem az *18%* százalékos hiba volt, a fent ismertetett lokális válasz normált struktúrával, csak 2 GPU-n 300'000 lépésen keresztűl tanítottam. Ez már egy egészen tisztességes eredménynek számít az adathalmazon.

4.6. A konvolúciós és az MLP hálózatok 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 CIFAR10-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 kissebb lesz, mégis az intuicióm becsapott.

Mindkét hálózatban a gradiens kiszámítása igényelte a legtöbb erőforrást. Ennek a számításnak a részleteit a 4.9. tábla mutatja az mlp hálózatra, és a 4.10. tábla a konvolúciós hálózatra. A táblázatokból jól látható hogy a kovolúciós hálózat lényegesen több erőforrást fogyasztott. Ha az erőforrások növekedése lineárisan skálázódna a paraméter tér beli növekedéssel, akkor jóval kissebb növekvény lett volna várható.

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

4.9. táblázat. *Az MLP hálózat megmagasabb erőforrás igényű részei az MNIST adathalmaznál*

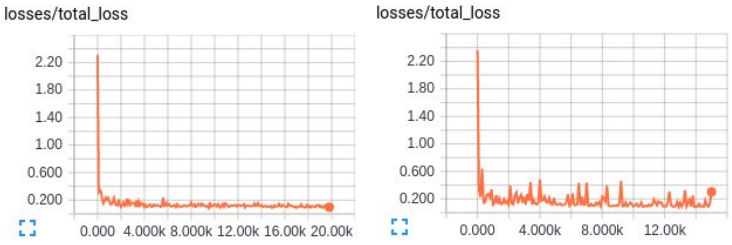
Hálózati rész	Memória	Számítási idő
gradiens	20.3 MB	12.5 ms
Az első teljes réteg gradiense	3.15 MB	12.1 ms
A második teljes réteg gradiense	2.03 MB	9.82 ms
ADAM optimizer	8 B	9.37 ms
Harmadik teljes réteg	750 KB	3.09 ms
Negyedik teljes réteg	11.3 KB	2.82 ms

4.10. táblázat. *A konvolúciós hálózat megmagasabb 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 előrecsatolt konvolúciós ág	2.3 MB	36.8 ms

- A konvolúciós háló paraméter tere lényegesen nagyobb lett, pedig én intuitívan kissebbnek 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 4.7. 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 beállni egy stabil pontba az adathalmaz kicsi mérete miatt, addig az MLP gond nélkül hoz konzisztens eredményeket a veszteségfüggvényben.



4.7. ábra. *A Tensorflownban idített logisztikus regresszió erőforrás igénye.*

Ugyanez a két metrika a CIFAR10 adathalmazon a **??**. 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 mutatják a 4.4. eredményei is.

4.7. A hibrid architektúrák eredményei

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 4.11. táblázat szemlélteti. A táblázaton látszik hogy a legtöbb kernel nagyjából ugyanúgy teljesít, kivéve a szigmoid kernel, mert az kiemelkedően rosszul teljesít a többihez képest.

4.11. táblázat. *Az RBM + SVM kombináció eredménye az MNIST adathalmazra különböző kernelekkel.*

A kernel	RBM mérete	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%
szigmoid coef0=0	64	89%
szigmoid coef0=1	64	89%

Miután az MNIST eredményeken látszik hogy az RBM által eszközölt dimenziócsökkentéssel 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 kiábránító eredményeket kaptam, az alábbi táblázat közli őket. 4.12.. 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 mnisten tapasztalt 0.17-0.12-höz képest. Hosszas lamentálás után rájöttem hogy a probléma ott van hogy amíg az MNIST adahalmaz normalizálva van, a CIFAR-ra ez nem elmondható ott 0 és 255 között terjednek az értékek.

4.12. táblázat. *Az RBM + SVM kombináció eredménye az CIFAR adathalmazra különböző kernelekkel.*

A kernel	RBM mérete	Teszt szet hiba százalék
poly coef0=1 degree=3	761	90%
poly coef0=1 degree=3	261 + 261 + 261	90%

Normalizált CIFAR10 RBM SVM Miután normalizáltam az adatokat a rekonstrukciós hiba leesett a várt értékre.

Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Irodalomjegyzék

[1] Li Fei-Fei Andrej Karpathy. Deep visual-semantic alignments for generating image descriptions. April 2015.

[2] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks.

[3] Kaiming He et al. Deep residual learning for image recognition.

[4] Max Jaderberg et al. Spatial transformer networks. February 2016.

[5] Ross Girschick et al. Rich feature hierarchies for accurate object detection and semantic segmentation. October 2014.

[6] Szegedy et al. Going deeper with convolutions.

[7] Yann LeCun et al. Gradient-based learning applied to document recognition. November 1998.

[8] Cybenko G. Approximations by superpositions of sigmoidal functions. 1989.

[9] Geoffrey Hinton. A practical guide to training restricted boltzmann machines, aug 2010. http://www.cs.toronto.edu/ hinton/absps/guideTR.pdf.

[10] Kurt Hornik. Approximation capabilities of multilayer feedforward network. 1991.

[11] Tong Zhang Kai Yu. Improved local coordinate coding using local tangents. 2010.

[12] Francis Bach Mark Schidt, Nicolas Le Rouy. Minimizing finite sums with the stochastic average gradient. September 2013.

[13] Rob Fergus Matthew D. Zieler. Visualizing and understanding convolutional neural networks. November 2013.

[14] Kishore Konda Zhouhan Lin, Roland Memisevic. Rumelhart, david e.; hinton, geoffrey e.; williams, ronald j. October 1986.

[15] Kishore Konda Zhouhan Lin, Roland Memisevic. How far can we go without convolution: Improving fully-connected networks. November 2015.

[16] Kishore Konda Zhouhan Lin, Roland Memisevic. Wavenet: A generative model for raw audio. September 2016.