Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky
Katedra Informatiky

Invariantná kategorizácia 2D objektov pomocou neurónovej siete DBN

2011 Bc.Juraj Barič

Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky
Katedra Informatiky

Invariantná kategorizácia 2D objektov pomocou neurónovej siete DBN

(Diplomová práca)

Bc. Juraj Barič

Študijný odbor: 9.2.1 Informatika
Vedúci práce: doc. Ing. Igor Farkaš, PhD.
Evidenčné číslo: 4c406060-a62e-46df-9de9-ba4ce56f4d37

Bratislava, 2011

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Juraj Barič
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Invariantná kategorizácia 2D objektov pomocou neurónovej siete DBN.

Cieľ: 1. Naštudujte si a implementujte model Deep Belief Network. 2. Otestujte základný model DBN v úlohe kategorizácie jednoduchých 2D objektov. 3. Navrhnite úpravu architektúry tak, aby podporovala čo najlepšiu kategorizáciu objektov (z troch tried) invariantne voči pozícii, škále a rotácii objektu.

Literatúra: Hinton, G. E, Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural Computation, 18:1527-1554.

Poznámka: Požiadavky: systematická práca, schopnosť čítať anglické texty, relatívna samostatnosť.

Vedúci: doc. Ing. Igor Farkaš, PhD.

Dátum zadania: 18.04.2011

Dátum schválenia: 18.04.2011

prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
vedúci

Čestne prehlasujem, že som túto prácu vypracoval samostatne s využitím uvedenej literatúry a podporou mojho vedúceho.

.....

Bratislava, apríl 2011

Poďakovanie

Na tomto mieste by som sa chcel poďakovať vedúcemu mojej dipl. práce, doc. Ing. Igorovi Farkašovi PhD., za odbornú pomoc, rady, a konzultácie, ktoré mi venoval. Zároveň by som sa rád poďakoval mojej rodine za podporu, ktorú mi poskytli počas tvorby práce, a obzvlášť otcovi aj za drobné odborné rady.

Abstrakt

Cieľom mojej diplomovej práce je analyzovať schopnosti modelu Deep Belief Network pri rozpoznávaní geometrických útvarov na čiernobielom obrázku pri rôznych transformáciach. Ďalšou úlohou je generovať naučené vzory. Úloha má niekoľko stupňov náročností: transformačná variabilita obrázkov, rotačná, škálová a pozičná invariantnosť a ich kombinácie.

V prvej časti mojej diplomovej práce som implementoval model, a pracoval na efektívnosti výpočtov. V ďalšej časti som sa venoval analýze šírenia signálov, a vplyvu parametrov siete na jej úspešnosť.

Výsledky testovania siete som zhrnul, a na základe správania navrhol rôzne zmeny. V mojej práci som zistil, že model DBN nie je vhodný na tento typ úloh, ale čiastočne ich dokáže riešiť.

Abstract

The aim of my thesis is to analyze the ability of Deep Belief Network model to recognize geometric shapes of binary pixel images under various transformations. Another challenge is to generate learned patterns. This task has several levels of difficulty: variability of images under rotation, scaling and position and their combinations.

In the first part of my thesis I have implemented the model, and worked on time efficiency of the used algorithms. In the next section I analyzed the data flow in the working network, and the impact of network parameters on network performance.

I have summarized the testing results and proposed some changes in the model structure. In my work I found that the DBN model is not very suitable for this type of task, but it can solve this problem partly.

Obsah

Kapitola 1

Úvod

Téma mojej diplomovej práce je invariantná kategorizácia 2D objektov pomocou neurónovej siete Deep Belief Network (ďalej iba DBN) [4] . DBN je nový model neurónových sietí. Vo svojej práci sa venujem jeho detailnému teoretickému popisu. V rámci praktickej časti som sa zaoberal jeho testovaním na rôznych obtiažnostiach invariantnosti - pozičnej, škálovej a rotačnej. Výsledkom je pozorovanie schopnosti DBN pri všetkých týchto úlohách, analýza správania sa siete, resp. zistenie, či je vôbec konkrétna úloha riešiteľná.

Keďže DBN patrí medzi tzv. "generatívne modely", venoval som sa aj analýze jej schopnosti rekonštrukcie vstupov.

Pri riešení tejto úlohy som vychádzal z už známych uskutočnených experimentov. Na základe ich analýzy som vypracoval teoretický model testovania, ktorý sa stal podkladom pre konkrétnu implementáciu simulácie učenia neurónovej siete. Vzhľadom k tomu, že táto simuláciaje pre dnes bežne dostupné počítače výpočtovo náročná, venoval som určitý čas svojho výskumu aj optimalizácii výpočtu na rýchlosť. Výsledkom mojej práce je aplikácia, ktorá vykonáva danú simulaciu v reálne použiteľných časoch.

Kapitola 2

Deterministické neurónové siete

2.1 Motivácia

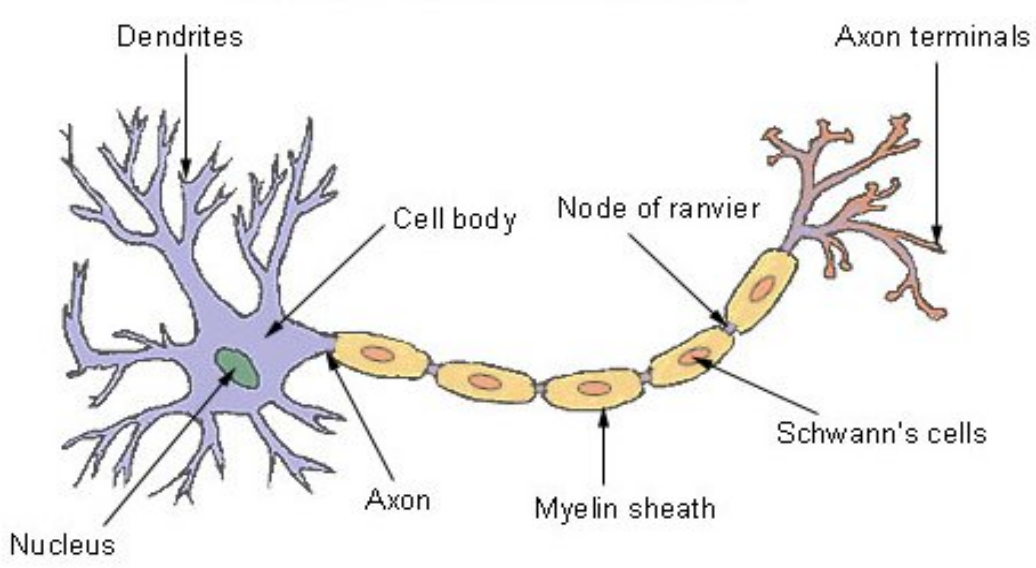
V prírode sa vyskytuje veľa rôznych živočíchov, ktoré sú nútené konať na základe vnemov z okolia. Už na prvý pohľad je zrejmé, že ich reakcie, schopnosť učiť sa, schopnosť pamätať si situácie a objekty, a následne ich rozoznávať, nefungujú ako stroje. Zatiaľčo stroj urobí presne to, čo mu programátor nadiktoval, človek sa vie rozhodnúť aj v situácii, s ktorou sa ešte nestretol, rozozná jedlo aj keď nemá tú istú chuť, vôňu, farbu a veľkosť ako inokedy, a dokáže si predstaviť také kombinácie vlastností na svojom okolí, ktoré ešte nevidel. S vývojom informačných technológií rastie však dopyt po strojoch, ktoré budú schopné konať bez pomoci človeka. Vedci preto začali skúmať ľudské správanie a činnosti jeho mozgu, aby zistili, čo u ľudí spôsobuje rozhodovanie. Dá sa povedať, že teória neurónových sietí sa snaží matematicky popísať fungovanie biologických neurónov, a s jej pomocou umožniť strojom do určitej miery ľudské správanie. Pre riešenie problémov však nie je potrebné zaoberať sa psychikou, preto ju pri skúmaní vynechávame. Tieto modely sa nazývajú tiež "Brain without mind"(mozog bez mysle).

2.2 Neurón

Neurón ako bunka je základný stavebný kameň celej nervovej sústavy. Jeho úloha je spracovávať informácie, ktoré dostáva v podobe elektrických impulzov, a výsledok spracovania ďalej posúvať. Nebudem tu popisovať jeho detailnú stavbu z chemicko-fyzikálneho hľadiska. Pre naše účely postačí následovné delenie:

- telo bunky, ktoré vysiela signál

- dendrity - výbežky, cez ktoré sa do neurónu dostávajú informácie (vzruchy). Pričom každý dendrit prináša inú informáciu, má iný vplyv na aktivitu neurónu. Niektoré ju zosilňujú, a iné zas tlmia, oboje sa deje v rôznych mierach.
- axon - výbežok, ktorým neurón vedie vzruch ďalej. Na konci axónu sa nachádza rozvetvenie, vďaka ktorému môže neurón poslať vzruch do viacero (rádovo 10^4) príjemcov (väčšinou ďalších neurónov).



Obr.: 1 (Schéma biologického neurónu) *náčrt neurónu s jeho základnými stavebnými prvkami. Obrázok prevzatý zo stránky wikipédie: http://en.-wikipedia.org/wiki/Neuron*

2.3 Umelý model neurónu

Z matematického hľadiska sa môžme na neurón pozerať ako na n-árnu matematickú funkciu dvoch typov: spojitú alebo diskrétnu. Základ majú oba typy rovnaký, rozdiel je len v obore hodnôt - najskôr sa vyráta akási sila vstupu. Jedná sa vlastne o váhovanú sumu vstupných signálov (x_1, ..., x_n). Výsledok sa označuje net, a posúva sa ďalej na spracovanie. Pre sumu teda potrebujeme n+1 konštantných hodnôt - n váh (w_1,...w_n) a prah (ďalej len bias, resp b):

$$net = b + \sum_{i=1}^{n} (w_i.x_i) \tag{2.1}$$

Všetky váhy aj bias nadobúdajú hodnoty reálnych čísel.

2.4 Diskrétny neurón

Používa 2-hodnotový, najčastejšie binárny 0, 1, alebo s opačnými pólmi -1, 1. Vo všeobecnosti nech sú to hodnoty A a B (nech $A < B$). Potom výstup neurónu vyrátame ako $A + (B\text{-}A).\text{sgn}(net)$. Výsledok teda bude opäť A alebo B.

2.5 Spojitý neurón

Požíva hodnoty z ľubovoľného intervalu (najčastejšie [0,1]), a výsledok musí byť opäť hodnota z tohto intervalu. Pre tieto účely sa používa sigmoidná funkcia

$$\sigma(x) : \mathbb{R} \to (0, 1) \tag{2.2}$$

definovaná:

$$\sigma(x) = 1/(1 + \exp(-x)) \tag{2.3}$$

Neurón n, ktorého aktivita sa má pohybovať v intervale (A,B), tak vypočíta silu svojho signálu ako

$$n = A + (B - A) * \sigma(net) \tag{2.4}$$

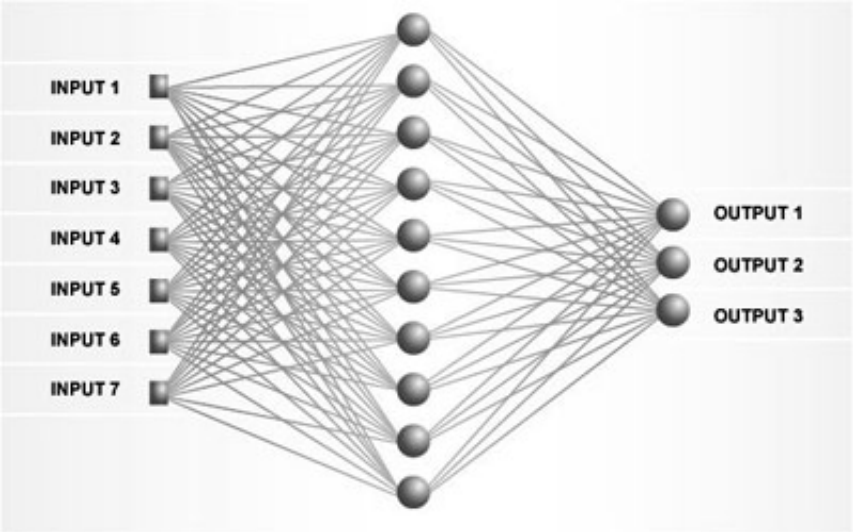
2.6 Neurónová sieť

Neuróny sú v tele navzájom poprepájané, posielajú signály jeden na druhý, a spolu tvoria nervový systém. Ľudský organizmus ich obsahuje veľmi veľa, len v mozgu ich je rádovo 10^{11}. Neurónovou sieťou by sme mohli nazvať ľubovoľnú malú podmnožinu husto poprepájaných neurónov, ktorá má v tele nejakú úlohu.

Formálne sa dá neurónová sieť definovať ako orientovaný graf, ktorého vrcholy reprezentujú neuróny a hrany prepojenia, pričom niektoré neuróny sú vstupy.

Uvediem tu dva typy neurónovej siete, ktorých grafy sú acyklické, teda neuróny sa dajú zgrupiť do vrstiev, pričom medzi neurónmi jednej vrstvy neexistuje žiadne spojenie, a výstupy jednej vrstvy predstavujú vstupy do nasledujúcej. Tieto siete sa nazývajú dopredné. Jej vrstvy sa dajú zoradiť a očíslovať od vstupu po poslednú vrstvu. Posledná vrstva je výstupná, jej výstupné signály sú zároveň výsledným vektorom celej siete. Ostatné vrstvy

sú skryté, a ich aktivity nie su zvonku pozorovateľné. Sieť dostane na vstup dáta, a signály jednotlivých vrstiev šíri "dopredu"z jednej na druhú.



Obr.: 2 (Schéma neurónovej siete) *Graf popisujúci umelú dvojvrstvovú NS so 7 vstupnými signálmi, 10 neurónmi v skrytej vrstve, a tromi výstupnými*

2.7 Učenie pomocou spätného šírenia chyby

Popísal som teda ako funguje výpočet na neurónovej sieti. Aby celý systém fungoval, a dokázal riešiť požadovaný problém, potrebujeme správne nastaviť váhy spojení. Optimálne hodnoty sa nedajú jednoducho vypočítať, a tak sa musí sieť "trénovať."To znamená, že na začiatku sa váhy nastavia náhodne (avšak v určitom rozumnom rozmedzí, napr od -1 po 1), a potom sa začne sieť spúšťať na vopred definovanej sade vstupov (hovoríme im trénovacia), pričom po každom výpočte sa váhy upravia.

Rozlišujeme tri druhy učenia:

1. Učenie s učiteľom - ku každému vstupu dostane sieť aj očakávaný výstup. Následne sa porovná skutočný výstup s očakávaným, a váhy sa posunú takým smerom, aby sa výstup pri danom vstupe viac podobal očakávanému výstupu. Pre daný typ učenia existuje tzv. chybová funkcia, ktorá sa učením minimalizuje.

2. Učenie bez učiteľa - sieť musí sama zistiť, aké výstupné signály treba priradiť ktorým vstupom. Väčšinou pritom sieť odpozoruje podobnosti vo vstupoch, a vytvorí prototypy vstupov.

3. Učenie posilňovaním (pomocou odmeny a trestu) pri tomto spôsobe sieť nepozná požadované výstupy, ale vie, či bola úspešná alebo neúspešna - a záleží od nej, ako zmení svoje aktivity pri nasledujúcom učení.

Na tomto mieste by som objasnil, ako prebieha najpoužívanejší spôsob učenia s učiteľom. Najprv zostrojíme chybovú funkciu siete:

$$E = \sum_{i=1}^{n} \frac{1}{2}(o_i - e_i)^2 \tag{2.5}$$

Táto funkcia predstavuje sumu štvorcov odchýliek. o_i je výstup i-teho neurónu, a e_i je očakávaný výstup i-teho neurónu. Cieľom siete je minimalizovať túto hodnotu. A to zmenami váh. Daný problém je vlastne hľadanie lokálneho minima chybovej funkcie. Avšak v tomto prípade nie sú funkčnými premennými vstupy siete, ale váhy výstupnej vrstvy. Našim cieľom je minimalizovať chybu pre konkrétny vstupný vektor, a teda vstupné hodnoty ostávajú nezmenené. Pre priblíženie sa k lokálnemu minimu zderivujeme chybovú funkciu v každom smere (podľa každej premennej) a podľa každej parciálnej derivácie určíme, či sa má váha zmenšiť alebo zväčšiť. Pre každý neurón i vyrátame zmeny váh:

$$\Delta w_{i,j} = \alpha. - \frac{\partial E}{\partial w_{ij}} \tag{2.6}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_i}.\frac{\partial o_i}{\partial w_{ij}} \tag{2.7}$$

kde

$$\frac{\partial E}{\partial o_i} = e_i - o_i \tag{2.8}$$

a

$$\frac{\partial o_i}{\partial w_{ij}} = o_i.(1 - o_i) \tag{2.9}$$

Kde α je kladná konštanta.

Je zrejmé, že takto sa dajú upraviť iba neuŕony výstupnej vrstvy, kde je u každého neurónu možné stanoviť odchýlku. Aby sieť správne fungovala, musíme natrénovať aj skryté vrstvy. Proces učenia prebieha takto po vrstvách, a nazýva sa backpropagation - chyba sa presúva na predchádzajúce vrstvy. Problémom ostáva, ako vyrátať vplyv každého neurónu na celkovú chybu siete. Riešenie funguje nasledovne:

Nech $\delta_i^{(v)}$ je chyba i-teho neurónu vo vrstve v, nech $W^{(v)}$ je váhova matica medzi vrstvami v-1 a v. Nech $o_j^{(v-1)}$ je výstup j-teho neurónu vo vrstve v-1. Potom chyba j-teho neurónu vrstvy v-1 je:

$$\delta_j^{(v-1)} = o_j^{(v-1)}.\sum_{i=1}^{n} W_{j,i}^{(v)}.\delta_i^{(v)} \tag{2.10}$$

Nakoľko celý výraz $\sum_{i=1}^{n} W_{j,i}^{(v)}.\delta_i^{(v)}$ je konštanta, parciálna derivácia podľa vstupnej váhy $W^{(v-1)}$ sa dá vyrátať pomerne jednoducho:

$$\frac{\partial E}{\partial w_{k,j}^{(v-1)}} = s_k^{(v-1)}.(o_j^{(v-1)}.(1 - o_j^{(v-1)})).\sum_{i=1}^{n} W_{j,i}^{(v)}.\delta_i^{(v)} \tag{2.11}$$

Tu je s_k k-ty vstup, pričom pre všetky vrstvy okrem prvej sú to výstupy predchádzajúcej (pri našom značení (v-2)-hej) vrstvy.

2.8 Radial Basis Function

Druhý často používaný typ siete je Radial Basis Function (RBF) Network. Jedná sa o 2-vrstvovú sieť, ktorej prvá vrstva sa aktivuje pomocou radial basis funkcie , a druhá vrstva výstupy prvej následne lineárne transformuje. Idea je, že neuróny RBF vrstvy predstavujú akési piliere siete ako rozpoznávacieho systému, s ktorými sa vstupné vektory signálov porovnávajú. Každý neurón svojim signálom vypovedá, ako "blízko," alebo ako podobný mu je vstupný vektor. Z toho plynie aj názov: neurón s radial basis function. Váhy každého neurónu predstavujú vektor (nazývaný aj "reference vector", prototyp, alebo centrum, ďalej referenčné vektory) v priestore vstupov, ktorý je akoby centrom vo svojom okolí. A so vzdialenosťou vstupu potom klesá aktivita neurónu.

Matematicky sa najprv vypočíta táto vzdialenosť:

$$x_i = \sqrt{\sum_{j=1}^{m}(u_j - c_{ij})^2} \tag{2.12}$$

kde x_i je vzdialenosť, u je vstupný vektor, a c je matica váh - teda c_i je vektor z priestoru vstupov zodpovedajúci i-temu neurónu RBF vrstvy.

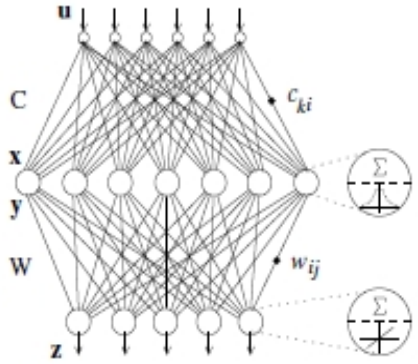
Výsledok sa následne použije v aktivačnej funkcii h(x), ktorá má svoje maximum v bode 0. Používa sa viac rôznych druhov aktivačných funkcií. Ako príklad uvádzam Gaussovu aktivačnú funkciu:

$$h(x) = \exp(-\frac{x^2}{2\sigma^2}) \tag{2.13}$$

Druhá-výstupná vrstva, ako už bolo spomenuté, je lineárna, teda výstup sa určí vzorcom:

$$z_k = \sum_{i=1}(y_i.w_{k,i}) + b_k \tag{2.14}$$

kde z je výstupný vektor, y_i je výstup i-teho neurónu skrytej vrstvy (y_i = h(x_i), w je váhová matica medzi prvou a druhou vrstvou, a $bias_k$ je prahová konštanta pre k-ty výstupný neurón.



Obr.: 3 (RBF sieť) *Štruktúra rovnaká ako u Double layer perceptron, ale prvky skrytej vrstvy majú miesto logistickej radiálnu aktivačnú funkciu a výstupné lineárnu funkciu*

Trénovanie RBF siete

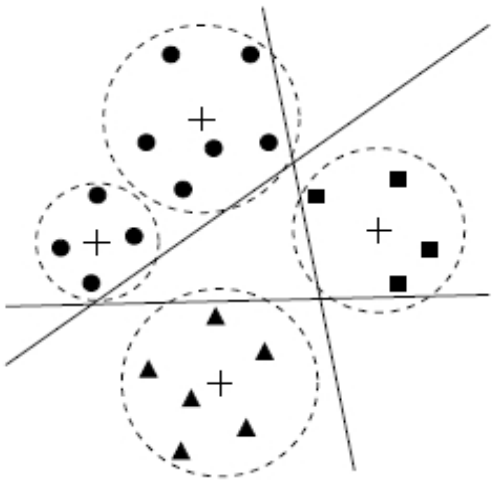
Rovnako, ako pri perceptrónoch, aj RBF sieť má svoju chybovú funkciu. Pre lineárnu aktivačnú funkciu výstupnej vrstvy je vypočítať parciálnu deriváciu triviálne. Chybová funkcia je suma štvorcov odchýliek od očakávaného výstupu, keďže sa jedná o učenie s učiteľom. Odchýlku j-teho výstupu značíme δ_j. V nasledujúcom vzorci je definovaná parciálna derivácia podľa váhy RBF neuóonov. Odvodenie prenechávam na čitateľa:

$$\frac{\partial E}{\partial c_{k,i}} = 2\frac{h'(x_i)}{x_i}(u_k - c_{k,i})\sum_j w_{j,i}\delta_j \tag{2.15}$$

$h'(x_i)$ je deriváciou aktivačnej funkcie h, a závisí od konkrétneho typu, ktorý bol zvolený.

2.9 Rozdiely medzi MLP a RBF

V tejto kapitole porovnáme, ako funguje klasifikácia pri MLP a RBF: MLP delí dátový priestor na hyperroviny, zatiaľčo RBF vytvára v tomto priestore hypergule, ktorých centrá tvoria spomínané referenčné vektory. Na nasledujúcom obrázku je znázornené vytváranie podmnožín dvojrozmerného vstupného priestoru.



Obr.: 4 (RBF and MLP recognition) *Trojuholníčky, štvorčeky a krúžky sú vstupné dáta troch rôznych tried. Priamky pretínajúce priestor sú deliace hranice vytvorené MLP. Krížiky značia centrá RBF siete, pričom okolo každého krížiku je vyznačené okolie prerušovaným kruhom.*

Kapitola 3

Stochastické modely

Teraz sa pozrime na nedeterministické (zvané aj stochastické) modely neurónových sietí, medzi ktoré patrí aj DBN. Skôr než sa dostaneme k popisu celej deep belief network, priblížme model, ktorý stojí za jej fungovaním.

Restricted Boltzmann machine (ďalej iba RBM) je umelá neurónová sieť s jednou skrytou vrstvou. Jej autorom bol Smolensky, a predstavená bola v roku 1986 [10]. Cieľom RBM je s čo najväčšou pravdepodobnosťou generovať vstupné vektory. Poďme sa pozrieť na RBM bližšie.

RBM sa skladá z dvoch vrstiev, skrytej (ďalej HL) a viditeľnej (ďalej VL). Medzi nimi je symetrické spojenie, čo je rozdiel oproti bežnej neurónovej siete. Vstupné signály prichádzajú na viditeľnú vrstvu, aktivujú neuróny v skrytej vrstve, a potom sa šíria naspäť po tých istých spojeniach na viditeľnú vrstvu.

Existujú diskrétne aj spojité RBM, ale keďže v mojej práci používam vo vstupoch aj výstupoch iba diskrétne hodnoty, spojitej RBM sa venovať nebudem.

V diskrétnej RBM, ako názov napovedá, majú všetky neuróny stav 0 alebo 1. Pre aktiváciu neurónu však nepoužijeme binárnu funkciu, ale vypočítame pravdepodobnosť toho, že bude "zapnutý" (ďalej použité aj termíny ako aktivovaný, svietiaci, páliaci - firing), t.j., že jeho hodnota sa nastaví na 1. Túto pravdepodobnosť vyrátame pomocou logistickej aktivačnej funkcie:

$$P(n_i = 1) = \frac{1}{1 + \exp(-\sum_{j=1}^{n}(w_{ij}.m_j) - b_i)} \tag{3.1}$$

kde n_i je neurón, ktorého hodnotu rátame, a m_j sú neuróny vrstvy, z ktorej prichádzajú signály.

V RBM teda najprv položíme vstup na viditeľnú vrstvu, a následne aktivujeme neuróny z HL s týmito pravdepodobnosťami:

$$P(h_j = 1) = \frac{1}{1 + \exp(-\sum_{i=1}^{m}(w_{ij}.v_i) - b_j)} \tag{3.2}$$

HL v RBM má za úlohu "popísať"vlastnosti vstupu. Preto sa neuróny v skrytej vrstve nazývajú aj feature detectors (ďalej len FD). Keď sa vstup transformuje na HL, tak sa signály prenesú naspäť na VL - hovoríme o rekonštrukcii vstupu. Rovnakým spôsobom ako pre HL vypočítame pravdepodobnosti pre viditeľné neuróny a následne každý z nich nastaví svoju hodnotu na 1 s pravdepodobnosťou:

$$P(v_i = 1) = \frac{1}{1 + \exp(-\sum_{j=1}^{n}(w_{ij}.h_j) - b_i)} \tag{3.3}$$

3.1 Učenie RBM

Ako bolo spomenuté, RBM má snahu každý vstup generovať s čo najväčšou pravdepodobnosťou. Teda často sa opakujúce vstupy budú najpravdepodobnejšie generované.

Pre každú RBM môžme vyjadriť pravdepodobnosť, že skrytý neurón j bude svietiť. Táto pravdepodobnosť je podmienená vstupným vektorom v:

$$P(h_j|v) = \sigma(\sum_{i=1}^{m}(w_{ij}.v_i) + b_j) \tag{3.4}$$

Následne vyjadríme pravdepodobnosť toho, že na viditeľnej vrstve dostaneme vektor v, ak máme na vstupe vektor h:

$$P(v|h) = \Pi_{i=1}^{m} P(v_i|h) \tag{3.5}$$

$$P(v_i|h) = \sigma(\sum_{j=1}^{n}(w_{ij}.h_j) + b_i) \tag{3.6}$$

je pravdepodobnosť každého jedného neurónu vo viditeľnej vrstve, že po prenose signálu zo skrytej vrstvy, bude zapnutý.

Pre naprávanie váh sa táto RBM nechá aplikovať dostatočne dlho na jednom vstupe, ktorý sa neustále generuje a potom aktivuje skrytú vrstvu. Po určitom čase sa dáta na oboch vrstvách ustália, avšak vektor na viditeľnej vrstve sa už líši od pôvodného vstupu. Keby sa po x iteráciach nelíšil, znamenalo by to dokonale naučenú RBM. Tento proces sa nazýva aj Gibbs sampling [4].

Týmto vznikne situácia analogická s abstraktným modelom Infinite Directed Model with Tied Weight [4], ktorej napodobením dostaneme

pomerne jednoduchú parciálnu deriváciu logaritmu pravepodobnosti pre vektor v na viditeľnej vrstve podľa každej jednej váhy:

$$\frac{\log p(v^0)}{\partial w_{ij}} =< v_i^0 h_j^0 > - < v_i^\infty h_j^\infty > \tag{3.7}$$

pričom $< v_i , h_j >$ značí počet prípadov, kedy sú vstupný i-ty a výstupný j-ty neurón spomedzi všetkých vstupných dát súčasne zapnutý.

Vzhľadom nato, že v praxi sa Gibbs sampling nedá simulovať nekonečne dlho, pri výpočtoch sa používa preto jednoduché max-likelihood-rule, ktoré je vlastne len skrátením Gibs Sampling procesu, a pri ktorom sa rekonštrukcia urobí len raz pre každý vstup.

Pre RBM sa toto učiace pravidlo používa vo viacerych formách. Jednou z foriem je urobiť rekonštrukciu vstupu a z neho rekonštrukciu výstupu. Potom dostaneme následovný vzorec pre nápravu váh:

$$\Delta w_{ij} = \alpha.(< v_i^0 h_j^0 > - < v_i^1 h_j^1 >) \tag{3.8}$$

Druhá verzia max-likelihood-rule vyráta iba pravdepodobnosť aktivácie viditeľného rekonštruovaného neurónu, a ráta s odchýľkou od žiadanej hodnoty:

$$\Delta w_{ij} = \alpha.(< (v_i^0 - p_i^1).h_j^0 >) \tag{3.9}$$

Kde p_i je pravdepodobnosť, že pri rekonštrukcii viditeľnej vrstvy sa i-ty neurón aktivuje.

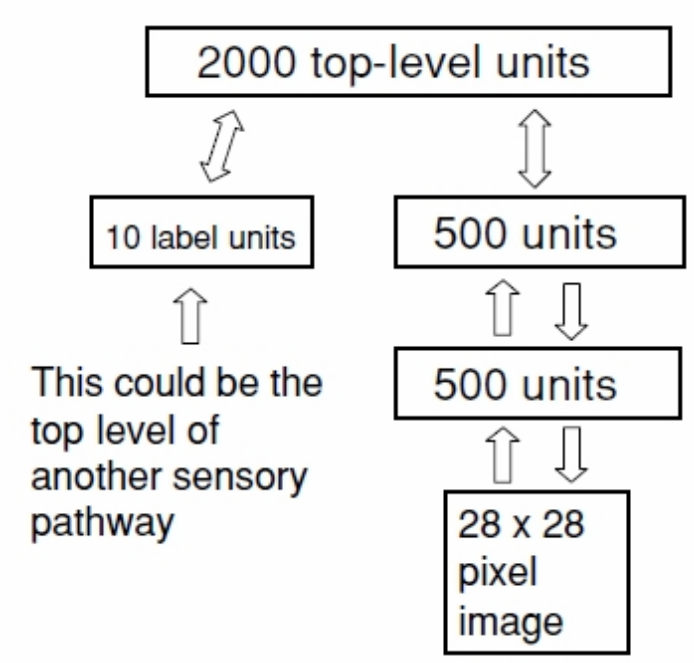
Počas implementácie a testovania som použil oba vzorce. Vyhodnotenie výsledkov ukázalo, že sa lepšie osvedčil prvý vzorec (číslo 15).

3.2 Deep Belief Network

Deep delief network je viacvrstvová sieť, ktorá má následovné vlastnosti:

- má symetrické váhové spojenia
- je stochastická
- je to generatívny model
- učenie prebieha po vrstvách (tzv. layer-wise learning)

Na malú ilustráciu prikladám konkrétnu štruktúru siete, ktorú použil Hinton a uverejnil vo svojom článku [4]. Úlohou tohto modelu bolo rozoznávať rukou písané číslice:



Obr.: 5 *Názorná ukážka štruktúry siete DBN. Výstupná vrstva s 10 vrcholmi (pre každú číslicu od 0 po 9) nie je úplne hore, ale pod najvyššou vrstvou vedľa predposlednej skrytej. Pri práci siete, či už učení, klasifikácii, alebo rekonštrukcii sa pre účely najvyššej RBM berie ako viditeľná vrstva spojený vektor výstupných neurónov a neurónov predposlednej skrytej vrstvy.*

Deep Belief Network je viacvrstvová sieť. V posledných rokoch sa používajú podobné hierarchické modely - v štruktúre siete sa dielčie úlohy

samovoľne rozdelia medzi vrstvy. Podobne to môže fungovať aj pri DBN. Hierarchické siete dokážu lepšie abstrahovať vlastnosti viacrozmerných dát [12]. Pozrime sa teraz, ako prebieha trénovanie tohto modelu.

3.3 Učenie DBN

Model Deep Belief Network používa "layer-wise learning", to znamená, že sa postupne učia jednotlivé vrstvy od viditeľnej až po výstup. Prebieha to tak, že vstupy sa uložia na vstupnú vrstvu, a prvé váhové spojenie (t.j. medzi vstupnou vrstvou a prvou skrytou) sa učí algoritmom max-likelihood-learning, podobne ako Restricted Boltzmann Machine.

Po naučení týchto váh sa aktivuje prvá skrytá vrstva, a učia sa váhové spojenia medzi prvou a druhou skrytou vrstvou, obdobne ako pri RBM. Pritom sa aktivácie prvej skrytej vrstvy berú ako viditeľná vrstva RBM, a druhá skrytá ako skrytá v RBM. Takto sa postupuje až po najvyššiu vrstvu, pričom spodnejšia vrstva sa berie pri max-likelihood algoritme ako viditeľná vrstva RBM, a vrstva nad ňou, ako skrytá vrstva RBM.

Výstup však nie je najvyššia vrstva, ale nachádza sa "vedľa"predposlednej. Samotná asociácia očakávaného výstupu k vstupu sa deje až pri učení váhových spojení pod najvyššou (ďalej iba Top) vrstvou. Na výstup sa položí očakávaný výstupný vektor a spolu s predposlednou vrstvou vytvorí "viditeľnú vrstvu"najvyššej RBM.

Táto RBM sa potom naučí generovať na viditeľnej vrstve, ktorej súčasťou je aj výstupný vetor, dané signály. Ak sú predchádzajúce vrstvy správne natrénované, potom aj na predposlednú vrstvu prichádzajú s veľkou pravdepodobnosťou príznaky, ktoré si sieť dokáže priradiť k výstupnému vrcholu.

3.4 Predošlé výsledky Deep Belief Network

DBN a jej algoritmus bol predstavený v roku 2006. Autorom bol profesor torontskej univerzity Geoffrey E. Hinton. Je to teda značne mladý, teoretický model, ktorý ešte nie je overený praxou. Medzičasom sa uskutočnilo niekoľko experimentov a pokusov riešiť niektoré problémy práve pomocou DBN, momentálne ale nie sú známe zatiaľ žiadne konkrétne príklady, ktoré by boli už aj v bežnej praxi použité. V nasledujúcej časti si popíšeme bližšie zrealizované pokusy testovať schopnosť DBN riešiť niektoré praktické problémy.

3.4.1 Filtrovanie spamov v emailoch

Autor: Grigorios Tzortzis, Aristidis Likas [5]
(Department of Computer Science, University of Ioannina - Grécko)

V pokuse bola použitá spojitá sieť, vstupné neuróný predstavovali slová z použitého slovníka, ich hodnota (z intervalu (0, 1)) predstavovala výskyt v danej správe. Pre trénovanie boli použité tri sady mailov, známe ako LingSpam, SpamAssassin, a EnronSpam. Slovník pozostával z najčastejšie sa vyskytujúcich slov vrámci trénovacej sady. (k bolo rozdielne pre každú trénovaciu množinu).

Miery úspešností:

Pre test bolo stanovených päť mier úspešností. Skôr, než ich matematicky definujem, objasním zápis zavedený autormi experimentu:

Nech X, Y sú množiny mailov, potom množina X $\rightarrow$ Y označuje všetky maily z X klasifikované sieťou ako Y. V pokuse sa pracovalo s množinami L (legitime mail) a S (spam mail).

Vráťme sa teraz späť k mieram úspešnosti. Najsúhrnnejšia je accurency, čo označuje úspešnosť správneho rozoznania mailov:

accurency = (|S $\rightarrow$ S| + |L $\rightarrow$ L|) / (|S $\rightarrow$ S| + |L $\rightarrow$ L| + |L $\rightarrow$ S| + |S $\rightarrow$ L|) teda počet správne rozoznaných k počtu úplne všetkých mailov

Ďalšie, pre uživateľa dôležité miery úspešnosti:

- ham recall (HR) = |L $\rightarrow$ L| / (|L $\rightarrow$ L| + |L $\rightarrow$ S|)
  úspešnosť rozpoznania legitímnych mailov
- ham precision (HP) = |L $\rightarrow$ L| / (|L $\rightarrow$ L| + |S $\rightarrow$ L|)
  spoľahlivosť pri klasifikácii mailu ako legitímny (pravdepodobnosť, že mail, o ktorom sieť povie "ok", naozaj nie je spam)
- spam reacll (SR) = |S $\rightarrow$ S| / (|S $\rightarrow$ S| + |S $\rightarrow$ L|)
  úspešnosť rozpoznania legitímnych mailov
- spam precision (SP) = |S $\rightarrow$ S| / (|S $\rightarrow$ S| + |L $\rightarrow$ S|)
  spoľahlivosť pri klasifikácii mailu ako spam (pravdepodobnosť, že mail, o ktorom sieť povie "spam", je naozaj spam)

Výsledky:

Najlepšie výsledky dosahovala sieť s tromi skrytými vrstvami o veľkosti 50, 50, 200 a výstupnou vrstvou, po 2 neuróny pre každú triedu. Všetky miery

úspešnosti pre všetky tri trénovacie sady dosiahli hodnoty nad 96% . Uvediem hodnoty pre mieru Accurency:

- LingSpam: 99.45%
- SpamAssassin: 97.50%
- EnronSpam: 97.43%

3.4.2 Generovanie výrazov tváre

Deep Belief Network, ako generatívny model, bol v jednom z pokusov ako prostriedok na generovanie výrazov tváre.[11]

Zo všetkých mne známych pokusov, bol tento jediný, ktorý používal generatívnu schopnosť modelu. Sieť bola natrénovaná na cca 3000 obrázkoch so 151 rôznymi osobami. Na výstupe bol a niekoľkými znakmi tváre (emócie a pohyby mimiky). Obrázky boli čiernobiele, pričom sa prvá vrstva učila generovať pravdepodobnosti, a nie binárne hodnoty. Zvyšné vrstvy boli binárne.

Zaujímavé pri tomto teste bolo generovanie tváre dvoch ľudí, teda rekonštrukcia vstupu z výstupu zodpovedajúcemu dvom osobám.

3.4.3 Rozoznávanie obrázkov

Pod tento experiment spadá viacero menších pokusov. Všetky mali rovnaký cieľ - klasifikáciu rôznych obrázkov. Primárne sa riešili tri rôzne úlohy. Tie isté úlohy boli riešené za tých istých podmienok aj inými modelmi. Výsledky boli následne porovnané.

Rozoznanie orientácie obdĺžnika

Úloha spočívala v klasifikácii dvoch typov obdĺžnikov podľa pomeru šírky a výšky. Najprv boli vygenerované iba obdĺžniky bielych pixlov, s rôznymi hrúbkami hrán, v čiernom pozadí. Ako sťaženie úlohy sa na čierne pozadie vložili obdĺžnikové výrezy čiernobielych obrázkov. V prvej, ľahšej verzii úlohy dosiahla chyba DBN hodnotu 2.60%, čo však nie je najlepší výsledok. Ten sa podaril na support vector machine (SVM), a to 2.15%. V ťažšej verzii sa naopak najviac osvedčila DBN, ktorej chyba, hoci 22.50%, bola zo všetkých modelov najnižšia.

Rozpoznanie konvexných geometrických útvarov

Na vstupe boli na čiernom pozadí množiny bielych pixlov. Úloha bola zistiť, či niektorá z nich netvorí geometrický útvar s nekonvexným uhlom. Pri hranách je treba poznamenať jeden problém. Ak ich smerový vektor v na pixlovej mape nemá aspoň jeden parameter 1 alebo 0, vytvárajú rôzne dlhé úsečky. Tým sa jedna hrana rozloží na viaceré hrany, pričom susedné sú rôznobežné. Môže tak nastať klamlivé zdanie, že sa stretávajú dve hrany zvierajúce nekonvexný uhol. O tejto úvahe sa v článku nič nespomínalo, takže mi ostáva iba odhadovať, nakoľko bola sieť schopná rozpornať jednotnú úsečku tvorenú rôzne dlhými úsekmi pixlov, ktoré lokálne vytvárajú konkávny uhol. Každopádne výsledok hovorí o pomerne veľkej chybe - ako najúspešnejšia sa ukázala práve DBN s 18,63%. Naproti tomu obyčajný single hidden layer perceptron mal v tomto experimente takmer dvojnásobnú chybu - 32,52%.

Klasifikácia rukou písaných číslic

Táto úloha sa podobala mojmu experimentu asi najviac, takže by som z nej mohol získať nejaké očakávania. Použitá bola pre účely pokusu databáza MNIST. V krátkosti by som prešiel cez všetky obtiažnostné variácie:

a) čierne číslice na bielom pozadí

b) číslice na pozadí s obrázkom

c) číslice otočené okolo stredového bodu

d) kombinácia dvoch posledných

Keďže moja úloha je práve klasifikácia obrázkov (teda vstupom bude sada pixlov), môžu mi výsledky tohto experimentu veľa napovedať o tom, čo môžem očakávať od mojej DBN. Dokonca rotácia obrázkov je presne to, čo testujem testovať aj ja. U číslic je navyše problém pri klasifikácií ”6” a ”9”. Databáza MNIST nemá pri týchto dvoch číslicach bodku, a či ju doplnili pri tomto pokuse, som sa nedozvedel.

Kapitola 4

Implementácia DBN

Na implementáciu modelu Deep Belief Network som si vybral jazyk c++ pre kombináciu performance a relativneho pohodlia pri písaní kódu. Písal som ho prevažne v prostredí MS Visual Studio, no kód je kompilovateľný aj gcc kompilátorom pod Linuxom. Na platforme windows som používal aj MinGV kompilátor.

Okrem samotnej DBN štruktúry s obsluhou na učenie a testovanie som urobil aj zopár iných rutín potrebných k diplomovej práci. V nasledujúcich odsekoch ich popíšem.

4.1 Generátor obrázkov

Na počiatku bolo potrebné vytvoriť tréningovú a testovaciu množinu. Keďže som pracoval s čiernobielymi obrázkami, na zápis pixla mi stačili 2 rôzne znaky. Rozhodol som sa preto generovať obrazce v textovej forme, s voliteľným znakom pre čiernu aj bielu. V mojom teste som si za pozadie zvolil medzeru , a pre čierny pixel "X".

Obrázky generovala samostatná aplikácia nazvaná "Invariant2DObject-Creator", ktorú prikladám na CD aj so zdrojovým kódom. Jej obsluhu som sa snažil urobiť čo najjednoduchšiu z hľadiska rýchlosti. Kedže sa jedná o pomerne jednoduchú aplikáciu, vystačí si aj bez grafického rozhrania, iba s príkazovým riadkom. Aplikácia počáva na tieto príkazy:

1. 'q' - vygeneruje štvorce
2. 't' - vygeneruje trojuholníky
3. 'c' - vygeneruje krúžky
4. 'e' - ukončí aplikáciu

5. 'n' - nastaví šum (noise)

   Používanie príkazov je následovné:

   - generujúci príkaz {q,t,c}: "command[output][position invariant[scalar invariant[rotation invariant]]] [output file name] count"

     output je buď "+"(výpis na konzole) alebo "−"(výpis do súboru)

     pre prepínače invariantnosti sa používajú len hodnoty y/n

     output file name sa používa v kombinácii s výpisom do súboru

     count - počet obrazcov, ktoré sa majú vygenerovať

   - 'e' sa používa bez prepínačov

   - 'n' - "n noise_density"kde noise_density má byť číslo z intervalu (0,1)

   K zdrojovému kódu som pridal prekompilačné makro "EASY-_POSITION_INVARIANT", ktoré pri zapnutí zľahčuje úlohu rozpoznávania pozične variabilných obrázkov. Toto zľahčenie objasním v sekcií o mojich pokusoch a výsledkoch.

4.1.1 Použité algoritmy

Na generovanie som použil jednoduché algoritmy, ktoré boli pre každý geometrcký útvar podobné:

1. urči veľkosť - pre škálovo nevariabilné obrázky je to presne polovica hrany okna
2. urči minimálnu vzdialenosť ťažiska od okraja pixlovej mapy. Ak sú obrázky pozične stabilné, tak je ťažisko presne v strede, ináč sa určí jeho vzdialenosť od okraja nasledovne:
   - Pre krúžok polomer + 1
   - Pre trojuholník polomer krúžku, ktorému je vpísaný, + 1
   - Pre štvorček, ktorý je rotačne stabilný, polovica hrany + 1, ak je rotačne variabilný, tak polovica diagonály + 1
3. vo vymedzenom pristore urči bod, kde leží ťažisko
4. urči uhol, pod ktorým sa obrázok otočí oproti normálu

Na rasteritáciu obrázkov som použil jednoduchý zaokrúhľovací algoritmus známy aj ako DDA . Do pozornosti dávam, že na presnejšiu rasterizáciu čiar a kriviek (kružníc a elips) sa používa v počítačovej grafike Bresenhamov algoritmus. Pre účely testovania DBN nám však stačí DDA.

4.2 Matrix tester

Pri svojej práci som sa nechcel nechať vyrušovať podružnými technickými drobnosťami a preto som použil model využívania už jestvujúcich algoritmov, miesto ich pracného programovania. Preto som použil standadnú STL knižnicu jazyka C++. Prvá implementácia používala pri práci s maticami a vektormi triedu vector<double>. Hoci som pri vývoji modelu Deep Belief Network testoval jeho funkčnosť na malých vstupoch, prevažne 2-rozmerných, už pri prvých testoch s obrázkami bolo jasné, že sa predsa len optimalizácii a vývoju vlastných tried pre prácu s maticami a vektormi predsa len nevyhnem. Prvý test siete veľkosti potrebnej pre účely diplomovej práce trval až 35 hodín, čo bola pre skúmanie daného modelu neakceptovateľne dlhá doba.

Upustil som teda od použitia STL knižnice ktorá je príliš robustná a triedy na správu vektorov a matíc ako aj ich operácii som naprogramoval sám. Pre tieto triedy som definoval funkcie aj operátory na násobenie a sčítavanie. Na otestovanie a porovnanie obidvoch implementácii mi poslúžila malá utilitka, ktorá automaticky inicializovala vektory a matice a potom v cykloch robila ich násobenie. Výsledok bol viac ako uspokojivý. Násobenie matíc prezentovaných stl vektormi bolo 20 až 30 násobne pomalšie, ako násobenie matíc prezentovaných jednoduchými mnou naprogramovanými triedami. Ďalším možným vylepšením bolo nepoužívať vlastné operátory nasobenia, ale funkcie. Pri operátoroch sa bolo totižto potrebné použiť copy koštruktor, čo zbytočne viedlo k alokácii matice. Testovanie však ukázalo, ze rozdiel medzi použitím operátorov a funkcii nebol až taký veľký a pohyboval sa rádovo pod 10 percent. Preto som pre lepšiu čitateľnosť kódu použil operátory násobenia. Samotná aplikácia je implementovaná oboma spôsobmi. Pomocou jednoduchého príkazu pre preprocesor sa dá skompilovať buď pomocou STL tried alebo pomocou vlastných implementovaných tried (pozri návod uvedený na CD). Zvedavý záujemca môže potom na vlastné oči porovnať performance oboch riešení na ľubovoľnom konkrétnom príklade.

Pre úplnosť treba spomenúť, že v sieti sa nikdy nenásobili štvorcové matice, ale jedna matica typu m x n s vektorom, takže nebolo možné použiť niektorý z rýchlejších algoritmov, ako napríklad Strassenov.

Ďalším spôosobom ako zrýchliť aplikáciu je použitie nastavení kompilátora. Okrem samozrejmého compilovania v release mode miesto debug modu treba použiť optimalizáciu kódu na rýchlosť. Toto zvýšilo performance aplikácie tiež niekoľko krát.

4.3 DBN tester

Samotnú neurónovú sieť som vyvíjal niekoľko mesiacov. Jej vývoj by som rozdelil do niekoľko častí:

1. Implementácia štruktúry siete a učiaceho algoritmu - v tomto kroku som sa snažil iba o to, aby som mal sieť, ktorá vykazuje známky učenia. Nakoľko som sa nechcel zdržovať dlhým učením, pre testovanie som používal boolovské funkcie, na ktorých som sieť potom testoval niekoľko tisíc krát a až ked dosiahla 100% úspešnosť počas všetkých týchto testov, mohol som postúpiť k ďalšej fáze testovania. Pre lepšiu analýzu som si naprogramoval logovaciu utilitku, ktorý mi pomáhala jednoducho logovať výsledky do niekoľkých úrovní: Error, Info, Trace a Debug.

2. S fungujúcou sieťou som pristúpil k naprogramovaniu čítania vygenerovaných dát zo súborov

3. Po prvom ťažkom učení obrázkov s rozmermi 30x30 pixlov, ktoré trvalo 35 hodín, hoci po jeho skončení mala sieť nulovú chybovosť, som prešiel k optimalizácií matíc.

4. Nakoniec som túto testovaciu aplikáciu refactoroval, pričom som všetky vstupy vložil do properties súbora, a pridal som niektoré prekompilačné makrá. Vďaka ním sa dá rozhodnúť pred kompiláciou, či sa použije stl vector, alebo moje vlastné triedy pre matice a vektory. Ďalej som portoval kód aby bol kompilovateľný kompilátorom minGV (MS verzia gcc), aby bol následne spúštateľný aj pod unix platformou.

Pri štarte aplikácie sa použijú ako argumenty dva prepínače. Prvý je meno vstupného súboru, ktorý popisuje sieť a učenie, a druhý prefix logovacích súborov.

Vstupný súbor obsahuje následovné atributy pre sieť a učenie:

1. **OutputSize** veľkosť výstupnej vrstvy, teda počet tired, ktoré budú charakterizovať dáta na vstupoch

2. **Size** šírka pixlovej mapy

3. **HiddenLayersCount** počet skrytých vrstiev

4. **Alpha** učiaca konštanta

5. **HiddenSize-1..n** n položiek, pre n skrytých vrstiev, pre každú z nich počet neurónov (dimenzia) danej vrstvy

6. **IterationCount** počet epoch učenia
7. **TrainingFileCount** počet súborov, skadiaľ sa vyčítajú trénovacie dáta
8. **TrainingFile-1..L** L položiek pre cesty k súborom, ktoré obsahujú trénovacie dáta
9. **TestFileCount** počet súborov, skadiaľ sa vyčítajú testovacie dáta
10. **TestFile-1..T** T položiek pre cesty k súborom, ktoré obsahujú testovacie dáta
11. **LineWidth** hrúbka čiar - nepárne celé číslo, vačšie ako 1 sa používa pre fuzzy hrany

4.4 Vizualizátor váh

Sieť neobsahuje veľké množstvo rôznych parametrov, a aj väčšina z nich je nemenná. Zato obsahuje obrovské množstvo váhových prepojení. Iba medzi vrstvami o rozmeroch 1200 a 2000 neurónov existuje vyše dvoch miliónov. Okrem toho, učenie prebieha samoorrganizovane, takže nie je možné určiť, ktoré FD sú za čo zodpovedné, a aké majú mať prepojenia s inými vrcholmi v susedných vrstvách.

Preto bolo potrebné urobiť nástroj na skúmanie váh. Okrem vecí ako priemerná hodnota, počet záporných/kladných váh, a podobných štatistík, som sa rozhodol urobiť grafický vizualizátor.

Prvý bol použitý pre prvé skryté vrstvy, a bol čiernobiely. Stredne silno sivá farba prezentovala mŕtve - nulové váhy. Biela vysoko kladné, a čierne nízko záporné a hodnoty medzi mali rôzne odtiene sivej. Pre skryté vrstvy nad vstupnými obrázkami bol tento spôsob postačujúci. Na mape váh pre každý FD sa rysovali tmavé alebo svetlé črty obrázkov na šedom pozadí.

Ale pre vyššie skryté vrstvy už tento spôsob nebol dosť prehľadný. Rozhodol som sa pridať farby. Odumretým nulovým váham som priradil čiernu, smerom do záporných hodnôt stúpala sýtosť červenej farby, a smerom ku kladným hodnotám stúpala sýtosť modrej a neskôr zelenomodrej farby.

Z týchto obrázkov už bolo možné vyvodzovať dôsledky a robiť prípadné zmeny, respektíve hľadať zdroje chýb pri učení.

Kapitola 5

Experimenty s DBN

Moje testovanie by som mohol rozdeliť do niekoľkých skupín.

5.1 Klasifikácia obrázkov s jemným šumom

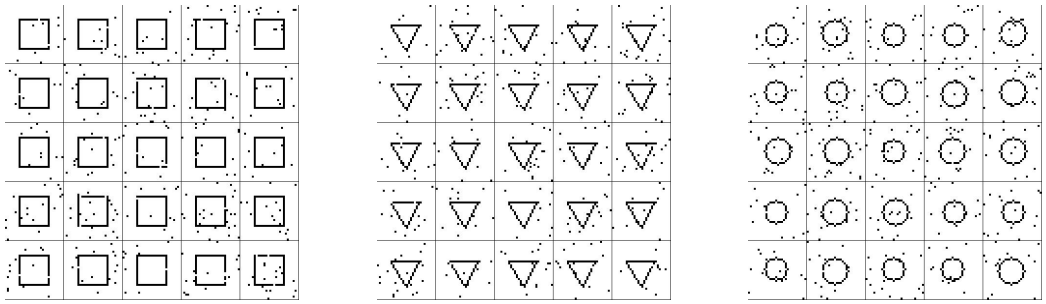
Ako prvý problém som si zvolil klasifikáciu troch vzorov s jemnou deformáciou. Ako vzory som si vygeneroval štvorce, trojuholníky a krúžky. Trénovanie prebiehalo na dokonalých obrázkoch, testovanie na obrázkoch so šumom 0,1 - to znamená, že každý pixel s pravdepodobnosťou 10 % zmenil svoju farbu. Obrázky mali rozmery 30x30 pixlov. Prikladám parametre siete a výsledky testov pre dané parametre. Test som urobil niekoľkokrát pre každú konštaláciu, a vybral som priemerný výsledok (keďže sa jedná o stochastický model, a pri malom počte testov by hrozilo, že dostanem nepresný výsledok):

Číslo testu	1	2	3
Počet výstupov	3	3	3
Alfa	0.7	0.15	0.15
Veľkost obrázkov	30 x 30	30 x 30	30 x 30
Počet skrytých vrstiev	3	3	3
Veľkosti skrýtych vrstiev	500 500 1500	100 100 200	100 100 200
Šírka hrán	1	3	3
Počet iteracií	900	30	150
Počet chýb počas testu	0	2	0
Z celkového počtu testov	150	300	300

Číslo testu	4	5	6
Počet výstupov	3	3	3
Alfa	0.15	0.05	0.15
Veľkosť obrázkov	30 x 30	30 x 30	30 x 30
Počet skrytých vrstiev	3	3	3
Veľkosti skrytých vrstiev	100 100 200	100 100 200	100 100 200
Šírka hrán	3	3	1
Počet iterácií	60	60	30
Počet chýb počas testu	0	0	0
Z celkového počtu testov	300	300	300

Výsledok 1 *V týchto dvoch tabuľkách je vidno správanie sa siete pri rôznych nastaveniach parametrov. Je zaujímavé, že pri tejto úlohe bolo obtiahnutie hrán kontraproduktívne - porovnanie testu č. 2 a 6.*

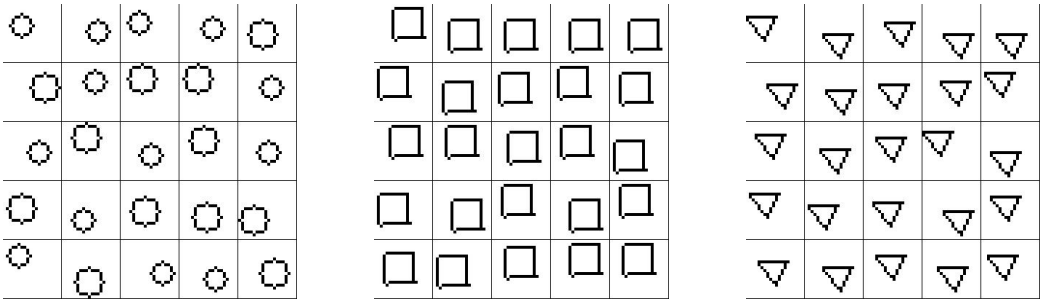
Problémom testu č.1 bola časová náročnosť. Učenie trvalo až 35 hodín. V tomto štádiu bolo potrebné venovať čas optimalizácii kódu a až následne som sa mohol vrátiť k vlastnej analýze problému. Znížil som taktiež počet iterácií učenia zhruba 10-15 násobne a veľkosti vrstiev na pätinu. Rýchlosť učenia týmto klesla na rádovo sekundy, teda asi 10 000 násobne.



Obr.: 6 *príklady testovacích, jemne zašumených obrázkov*

5.2 Klasifikácia pozične variabilných obrázkov

Pristúpil som k prvej vážnej úlohe, ktorá bola pripravená pre sieť v mojom pláne. Pozične variabilné obrázky mali všetky takmer rovnakú veľkosť, ale hýbali sa po celom zornom poli siete. Pre ilustráciu prikladám niekoľko obrázkov, ktoré som použil na testovanie:



Obr.: 7 *príklady testovacích, pozične variabilných, obrázkov*

Keďže išlo o náročnejšiu úlohu, zmenšil som veľkosť pixlovej mapy na 20x20 pixlov. Napriek viacerým pokusom však sieť nevykazovala dobré výsledky.

Nižšie uvádzam dva testy, jeden bez permutácie trénovacích obrázkov (to znamená, že v každej iterácii učenia sa brali obrázky v tom istom poradí), a druhý s permutovaním trénovacích obrázkov. Môj pôvodný predpoklad bol, že pri vysokom počte iterácii nie je potrebné permutácie robiť, keďže v každej iterácii je úprava váh malá, a v každej ďalšej sa prvé obrázky opäť dostanú k slovu, no zdá sa, že hrá dôležitú úlohu, tak som ju ponechal aj v ďalších testoch:

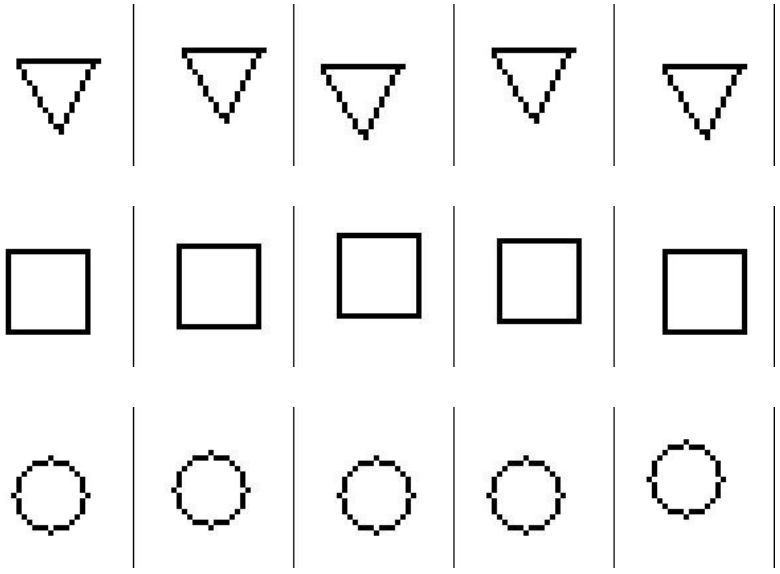
Číslo testu	1	2
Počet výstupov	3	3
Alfa	0.7	0.3
Veľkosť obrázkov	20 x 20	20 x 20
Počet skrytých vrstiev	3	3
Veľkosti skrytých vrstiev	500 x 500 x 1500	500 x 500 x 1500
Šírka hrán	1	1
Počet iterácií	30	60
Počet chýb počas testu - krúžky	19	12
Počet chýb počas testu - štvorce	0	1
Počet chýb počas testu - trojuholníky	9	7
Počet chýb počas testu - celkovo	28	20
Z celkového počtu testov	60	60

Výsledok 2 *Test č. 1 - bez permutácií trénovacích obrázkov, test č.2 - s permutáciami*

Napriek tomu, že sieť sa dokázala sčasti naučiť klasifikovať tieto tri geometrické útvary, obzvlášť štvorce, sú jej výsledky príliš neuspokojivé na to, aby sa dala sieť použiť v praxi. Nakoľko model Deep Belief Network zo svojej podstaty ignoruje 2D štruktúru vstupu a príznaky hľadá v kombináciach konkrétnych vstupných signálov, prišiel som k záveru, že v tomto prípade sieť nedokázala vyabstrahovať konktrétne znaky útvarov. Rozoznávanie poskytovalo dobré výsledky na prekryve čiar. Preto môžeme konštatovať dobrá úspešnosť na štvorcoch a nedostatočnú na krúžkoch.

5.3 Klasifikácia pozične variabilných obrázkov s obmedzenou variabilitou a fuzzy hranami

Pre zjednodušenie som v ďalšom teste obmedzil posuvy. To v praxi znamená, že geometircké útvary sa síce posúvali, ale nie po celom zornom poli. Od stredu sa mohli posunúť iba o jednu desatinu zorného pola v horizontálnom aj vertikálnom smere. Navyše som čiary obtiahol šedou farbou (teda vstupné signály okolo zapnutých pixlov boli nastavené na hodnotu 0.5) - tieto hrany budem v ďalšom texte označovať ako "fuzzy hrany". Tým som zvýšil prekryv obrázkov, ktoré sú veľmi blízko seba. Na druhej strane som však výrazne zúžil množinu všetkých možných obrázkov. Pri posuve oboma smermi o dva pixle (pri hrane dvadsať pixlov) môže byť stred obrázku vo vymedzenom priestore 5x5 pixlov. Na testovanie aj trénovanie mám teda spolu 25 rôznych obrázkov (samozrejme z každej triedy).



Obr.: 8 *Príklady testovacích, čiastočne pozične variabilných, obrázkov*

Pri tejto úlohe som urobil niekoľko desiatok testov. Každú triedu obrázkov som rozdelil na 20 trénovacích a 5 testovacích dát. Trvalo určitý čas, kým sa mi podarilo nájsť správne inicializačné hodnoty. S vhodnými inicializačnými hodnotami sieť dokázala obrázky rozoznávať bezchybne. Tie parametre uvádzam v tabuľke. Nakoniec som sa rozhodol úlohu opäť sťažiť, a vymazal som fuzzy hrany a nahradil ich tenkými čiarami. S takto definovanými

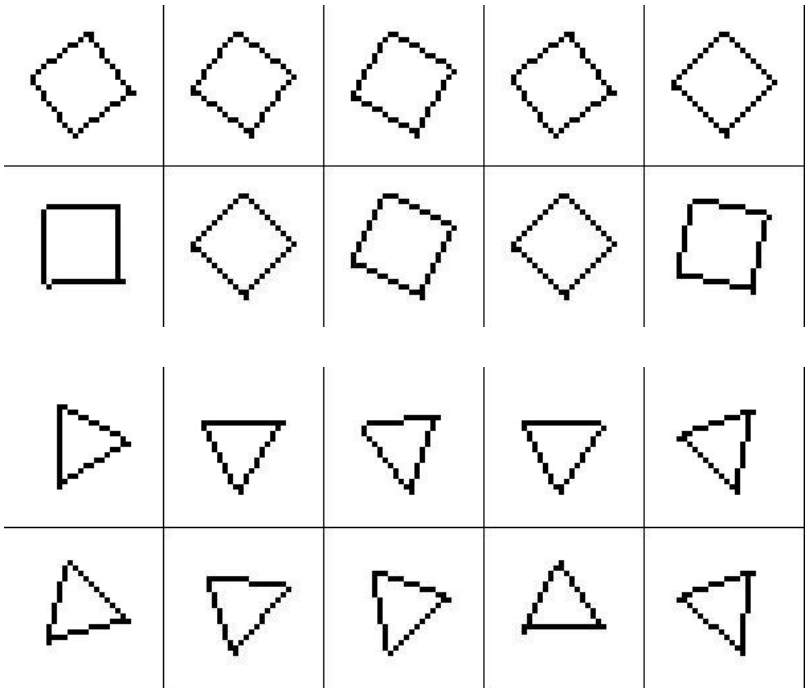
podmienkami mala sieť už značné problémy. Pre porovnanie prikladám výsledky:

Počet výstupov	3	3
Alfa	0.2	0.15
Veľkosť obrázkov	30 x 30	30 x 30
Počet skrytých vrstiev	3	3
Veľkosti skrytých vrstiev	700 x 700 x 1200	700 x 700 x 1200
Šírka hrán	**1**	**3**
Počet iterácií	40	100
Počet chýb počas testu	6	0
Z celkoveho počtu testov	15	15

Výsledok 3 *V ľavom stĺpci je test bez fuzzy hrán (so šírkou 1 pixel), vpravo s fuzzy hranami s rozptylom 1 pixel (hrany sa ťahajú jeden pixel od svojho stredu, teda ich celková šírka je 3)*

5.4 Klasifikácia rotovaných obrázkov s fuzzy hranami

Ďalšia úloha bola rozoznávanie uhlovo variabilných obrázkov. Pri malom rozlíšení je problém vygenerovať dostatočné množstvo rôznych trénovacích a testovacích obrázkov. Celkom som ich vytvoril dvadsať pri rozlíšení 30 x 30 pixlov. Opäť prikladám niektoré názorné príklady:



Obr.: 9 *Príklady testovacích, uhlovo variabilných, obrázkov*

Pri skúmaní schopnosti siete učiť sa uhlovo variabilné geometrické útvary bolo zbytočné hrať sa s kruhom, ktorý pri ľubovoľnom natočení vyzerá rovnako, preto som ho z trénovania vylúčil.

5.5 Rekonštrukcia

Deep Belief Network je generatívny model, teda vie na vstupe vytvárať dáta, ktoré už niekedy videl. Súčasť mojej úlohy bolo otestovať aj túto schopnosť. Prvý pokus som začal robiť na natrénovanej sieti, ktorá rozoznávala uhlovo variabilné obrázky.

Napriek tomu, že sieť rozpoznávala obrázky bezchybne, rekonštrukcia nič nevytvorila. Na vstupe som dostal prázdny obrázok. Keďže váhy sú

symetrické, predpokladal som, že som spravil chybu pri implementácií. Začal som teda upravovať kód na rôznych miestach. Tieto zmeny rozoberiem v nasledujúcich odsekoch.

5.5.1 Ignorácia rekonštrukčných biasov

Pri generovaní obrázkov som nepoužíval jeden parameter. A tým bol rekonštrukčný bias. Takže do začiatku rekonštrukcie vlastne nikdy nebol otestovaný. Rozhodol som sa teda tento bias vymazať. Tento krok ma síce neposunul správnym smerom k vyriešeniu problému, objavil som ale jeden zvláštny úkaz.

Keďže som rekonštrukciu spúšťal v DEBUG móde, prebiehalo učenie výrazne pomalšie ako v RELEASE móde. Aby som sa veľmi nezdržoval, a mohol pozorovať trace priebehu rekonštrukcie, znížil som počet epoch pri učení zo 100 na 30. Použil som pritom na učenie iba 16 obrázkov (zvyšných 24 som použil na testovanie). Sieť síce znovu nič nevygenerovala, ale jej úspešnosť bola 100%. Pri tak nízkom počte trénovacích iterácií sa sieť predtým pri uhlovo variabilných obrázkoch nebola schopná takto učiť.

Pokus som niekoľkokrát zopakoval, aby som zistil, či to nebola náhoda, a výsledok bol stále rovnaký. Počas všetkých testov nenastala ani jedna chyba. Skúsil som počet iterácií ešte viac znížiť, a ešte pri 6 sieť fungovala bezchybne. Pri nižšom počte bolo už trénovanie príliš krátke a sieť začala vykazovať chyby.

Za spomenutie stojí aj test s jedinou iteráciou (pri učení sieť analyzovala každý vstup iba raz), po ktorom som dostal 13 chybných testov z 24. Sieť mala síce iba 2 rôzne obrázky, avšak pre úplnosť som jej nechával výstupný neurón aj pre krúžok, hoci nikdy nebol zapnutý.

5.5.2 Gibbs sampling na najvyššej vrstve

Po prvom neúspešnom pokuse o opravu som sa rozhodol vypisovať všetky signály pri rozpoznávacom ako aj pri generatívnom procese. Zatiaľ čo pri klasifikácií boli v každej vrstve neuróny rovnomerne ale riedko zapnuté, pri generovaní bola každá druhá úplne neaktívna a každá druhá spolovice aktívna, čo je veľmi veľa, ak na vysokorozmernej sieti svieti každý druhý neurón.

Na najbližších stranách prikladám zopár príkladov z logov šírenia signálov pri práci siete.

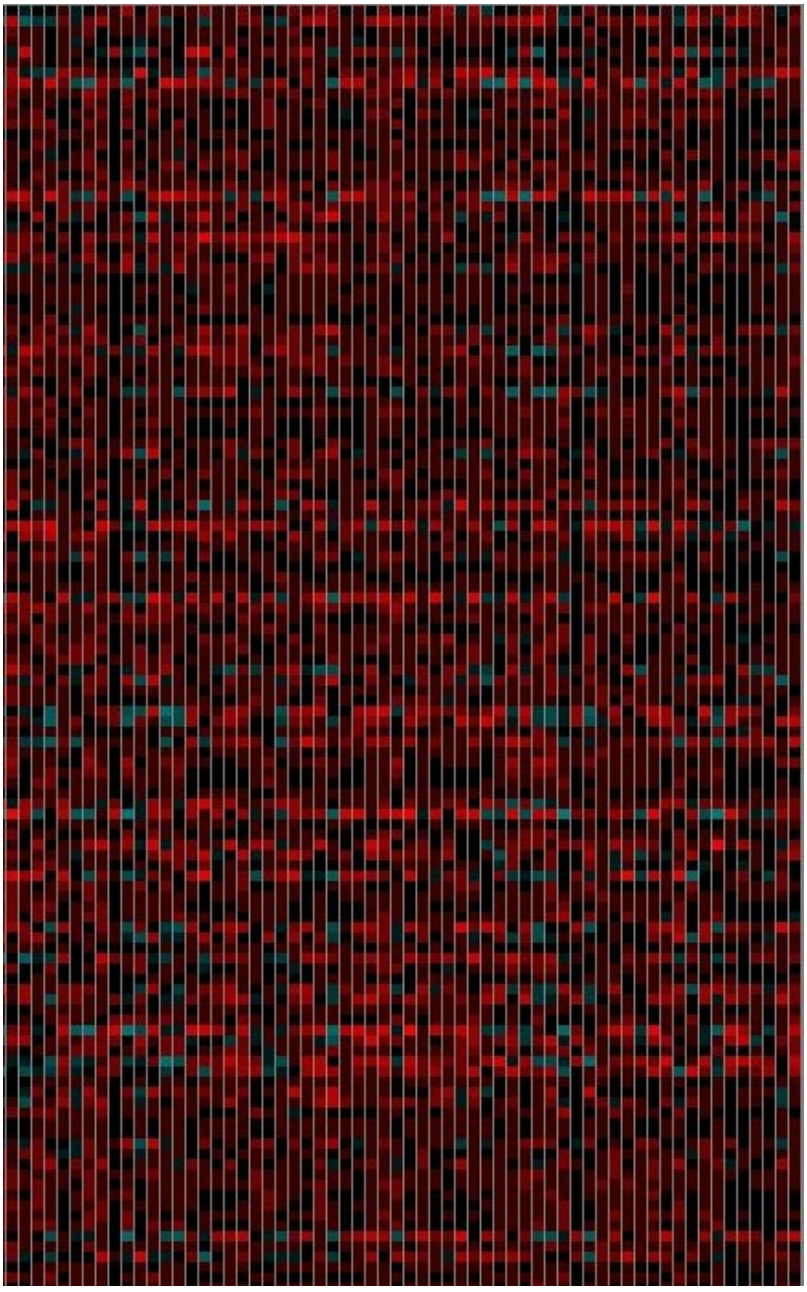
Pri rekonštrukcí bola práve spolovice aktívna najvyššia vrstva. Ako možné vysvetlenie bolo, že pri aktivácí tejto vrstvy z prázdnej predposlednej vrstvy a výstupnej, na ktorej je aktívny jediný neurón, dostávajú neuróny takmer

nulové net-hodnoty. Každý neurón dostáva signál iba od jediného spomedzi tých niekoľko stoviek neurónov, s ktorými je spojený, a teda iba jediná váha sa do sumy netu zarátava.

Napriek tomu však "správne"neuróny mali pravdepodobnosť o máličko vyššiu ako 50 %, že budú zapnuté, a ostatné o niečo málo nižšiu. Teda štatisticky by mala byť z množiny nesprávnych neurónov viac ako polovica neaktívna a spomedzi tých správnych o niečo málo viac ako polovica aktívna.

Môj ďalší pokus bol urobiť Gibbs sampling proces na najvyššej vrstve, k čomu ma donútila úvaha, že štatisticky bola zapnutá vždy o niečo viac ako polovica správnych neurónov, ktoré zvýšia pravdepodobobnosť nad 50 % . Pri dostatočne dlhej výmene medzi top vrstvou a tou pod ňou, ktorá sa skladá z výstupnej a predposlednej skrytej, by sa pravdepodobnosti a počet správne zapnutých a vypnutých neurónov mali pomaly vzdialovať od jednej polovice. Skúsil som testovať Gibbs sampling rôzny počet kôl. Postupne od 10 až po 10 000. Správanie siete bolo bohužiaľ stále rovnaké.

Ešte prikladám malú ukážku váhového profilu na vysvetlenie, prečo boli takmer všetky neuróny vypnuté vo vrstvách, do ktorých sa šírili signály. Na ňom je vidno, že váhy medzi vrstvami majú prevažne záporné hodnoty, a teda pri súčte náhodne vybratej aktívnej polovice prichádza do aktivačnej funkcie výrazne záporná hodnota. To spôsobuje, že pravdepodobnosť zobudenia každého neurónu sa blíži k nule.



Obr.: 10 *Váhový profil natrénovanej siete. V každom riadku sú znázornené váhy jedného neurónu, pre každý vstupný signál jeden štvorček. Nulové váhy sú vyznačené čiernou farbou, záporné majú odtieň červenej - čím bližsie k mínus nekonečno, tým červenšie, čím bližšie k nule, tým tmavšie. Kladné výhy naproti tomu sú označené tyrkysovou farbou. Čím vyššia hodnota, tým svetlejšia farba.*

TEST ROZPOZNÁVACÍCH SCHOPNOSTÍ

signály skrytých neurónov - vrstva 1:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0

signály skrytých neurónov - vrstva 2:

0 0 0 0 1 0 1 0 0 1 0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 1 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1
0 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0
1 0 1 0 0 0 0 1 0 1 1 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1
0 0 0 0 1 0 0 1 0 1 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0

signály skrytých neurónov - vrstva 3:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

Výsledok 4 *Takto sa správala natrénovaná sieť pri šírení signálu smerom od vstupnej vrstvy k výstupným klasifikačným neurónom. Vidieť pomerne riedko aktívne vrstvy, pričom najviac aktívnych neurónov je v druhej skrytej vrstve.*

TEST REKONŠTRUKČNÝCH SCHOPNOSTÍ

signály skrytých neurónov - vrstva 3:

```
1 1 0 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0
0 0 1 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0 1 0
0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 1 0 0
1 1 0 1 1 0 0 1 0 1 1 1 0 0 0 1 1 0 0 1 1 1 1 0 0 1 0 0 1 1 0 1 1 0 0 1 1 0 1 1 1
1 1 0 1 1 0 1 1 0 1 0 1 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 1 1 1 1 0 1 1 1 1 0 0 0 1
0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 0 0 0 1 0 0 1 0 0 0 1 1 1 0 1
0 1 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 1 0
0 1 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 0 1 0 0 1 0 0 0 0
0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 1 0 1 1 0 1
0 1 1 1 0 0 1 0 1 1 0 0 0 0 1 1 0 1 1 0 0 0 1 1 0 0 0 0 0 1 1 1 0 1 0 1 1 1 0 0 1
0 0 0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1 1 0 0
0 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 1 1 0 1 1 1 0 1
1 0 0 0 1 1 1 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 0 0 1 0 1 1
0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0
1 1 1 0 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 0 1 1
```

signály skrytých neurónov - vrstva 2:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
```

signály skrytých neurónov - vrstva 1:

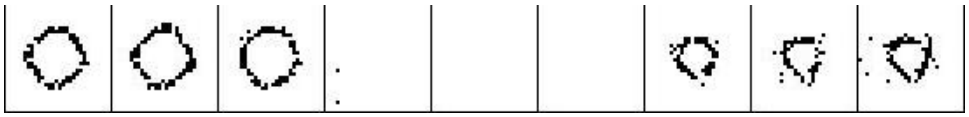
```
0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 1 0 1
1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 1 1 0 1 0 0
0 1 0 1 1 0 0 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1 0 1 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 0 0
0 0 1 1 0 0 0 1 0 0 0 1 1 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1
1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 1 0 1 1 1 0 1
```

Výsledok 5 *Takto sa správala natrénovaná sieť pri šírení signálu smerom od výstupných klasifikačných neurónov smerom k vstupnej vrstve. Najvyššia vrstva má takmer polovičné zastúpenie aktívnych vrcholov, druhá skrytá vrstva - tá pod ňou - žiadne páliace neuróny a ďalšia skrytá vrstva (prvá), rovnako ako najvyššia, je husto aktívna. Výsledná rekonštrukcia tohto procesu bola prázdna pixlová mapa, teda rovnako ako druhá skrytá vrstva, mala samé neaktívne neuróny.*

5.5.3 Zosilnenie výstupného signálu

Aby sa zvýraznilo posilnenie signálov do tých správnych neurónov, pokúsil som sa posilniť signál z výstupného neurónu, ktorý je ako jediný aktívny. Vyskúšal som teda prenásobiť hodnoty vo výstupnej vrstve vysokou konštantou. Začal som pri 10, a nič sa neudialo. Pri 100 začali vznikať prvé zoskupenia pixlov pripomínajúce snahu o nejaký obrazec. Avšak, hoci sieť bola naučená bezchybne (použil som tú, ktorá sa trénovala na uhlovo variabilných obrázkoch), napriek tomu nič zmysluplné nevyprodukovala.

Posilnil som signály 10 000 násobne, a sieť začala fungovať. Prikladám obrázky vygenerované sieťou - pričom som sa pokúsil generovať aj krúžky, ktoré som sieť vôbec neučil.



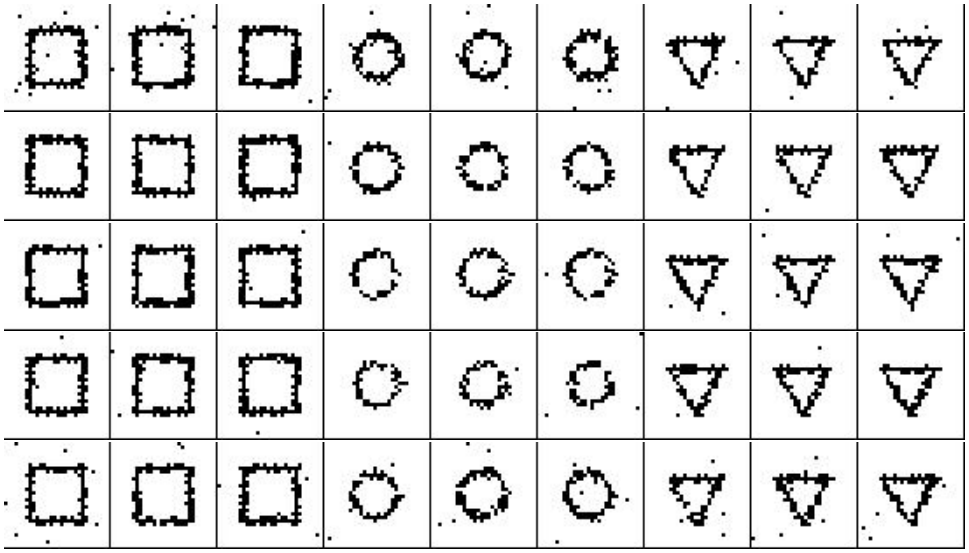
Obr.: 11 *Rekonštrukcia vstupných dát na naučenej sieti, zľava tri štvorce, tri krúžky a na konci tri trojuholníky. Na výstupe signály zosilnené 10 000 násobne*

Obr.: 12 *Rekonštrukcia vstupných dát na slabo naučenej sieti (tie isté parametre, ale iba šesť učiacich iterácii), zľava tri štvorce, tri krúžky a na konci tri trojuholníky. Na výstupe signály zosilnené 10 000 násobne*

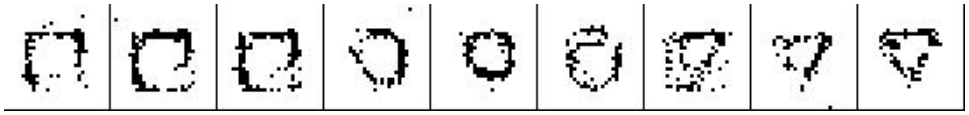
Obr.: 13 *Rekonštrukcia vstupných dát na naučenej sieti, zľava tri štvorce, tri krúžky a na konci tri trojuholníky. Na výstupe signály zosilnené 100 násobne*

5.5.4 Ďalšie rekonštrukcie

Pre úplnosť prikladám aj rekonštrukcie sietí naučených na iných typoch obrázkov. Ako prvé prikladám rekonštrukcie siete naučenej na zašumených obrázkoch bez invariantných transformácii.



Obr.: 14 *Rekonštrukcia vstupných dát na naučenej sieti bez transformácií, zľava tri štvorce, tri krúžky a na konci tri trojuholníky.*



Obr.: 15 *Rekonštrukcia vstupných dát na sieti naučenej na čiastočne polohovo variabilných obrázkoch.*

Z mojho pozorovania bol najväčší problém rekonštrukcie pri pozične variabilných obrázkoch. Kým pri ostatných úlohách bola rekonštrukcia úspešná hneď pri zosilnení výstupného signálu, pri tejto úlohe sieť začala generovať zmysluplné vstupy až pri vyladení viacerých detailov.

Kapitola 6

Model DBN so zdieľanými váhami

Ako už bolo spomenuté viackrát, model DBN ignoruje 2D štruktúru vstupu a tým pádom je absolutne nevhodný na typ úloh, aké sa snažím v mojej diplomovej práci vyriešiť. Testovať kategorizáciu škálovo variabilných obrázkov preto nemá význam. Prekryv je nulový a tým pádom sieť nie je schopná nájsť spoločné znaky obrázkov.

Ako riešenie sa javí použiť štruktúru back-propagation siete od Le Cuna [6], ktorou riešil kategorizáciu rukou písaných číslic. Rozhodol som sa vyskúšať rovnako namodelovať prvé dve alebo tri skryté vrstvy Deep Belief Network.

V pôvodnej verzií išlo o sieť s učením pomocou spätného šírenia chýb, ale samotné prepojenie medzi vrstvami môžme ponechať tak, ako bolo navrhnuté, iba zmeniť úpravy váh.

6.1 Štruktúra siete so zdieľanými váhami

Na tomto mieste ozrejmím štruktúru a myšlienku fungovania siete, ktorú Le Cun navrhol. Oproti bežnej siete s úplným prepojením medzi vrstvami prináša dve hlavné zmeny

6.1.1 Receptívne polia

V klasickej neurónovej sieti, kde sa každý neurón v skrytej vrstve naučí reagovať na určitý príznak (kombináciu vstupných signálov), alebo naopak zvykne reagovať vždy, pokiaľ určitý príznak nenájde, je každý FD spojený s každým signálom v nižšej vrstve. V praxi to však znamená veľkú redundanciu

a rušenie. Príznaky na tej najnižšej úrovni sa objavujú totiž ako zhluky pixlov, ale tie najprimitívnejšie znaky nevynikajú kombináciou pixlov na druhej strane zorného poľa. Okrem toho, že sa potom mnohé FD naučia reagovať na nezmyselnú kombináciu pixlov, ktoré sú príliš ďaleko od seba nato, aby tvorili súvislý príznak, sa pri učení musí v sieti upravovať zbytočne veľké množstvo parametrov - váh.

Ako obmedzenie týchto problémov sa javí usporiadať neuróny v skrytej vrstve metricky a každému neurónu dávať na vstup iba malú oblasť (nazývanú aj receptívne pole). Každá oblasť je rovnako veľká a pri 2D vstupoch (obrázkoch) sa jedná o štvorce resp. kruhy pixlov.

Oblasti sa prekrývajú (ich stredy sú vzdialené vždy iba o dva pixle), a každú "sleduje" rovnaký počet neurónov.

Druhá skrytá vrstva sa zoskupí rovnako a s prvou je spojená neúplne - každý FD sleduje iba oblasť neurónov. Tieto oblasti sú tvorené FD prvej skrytej vrstvy, ktoré sú z topografického hľadiska blízko seba. Týmto dosahuje sledovanie o niečo väčšej oblasti, v ktorej hľadá druhá vrstva kombinácie tých najmenších znakov.

Rovnako by sa dala zostrojiť aj ďalšia vrstva, ale zväčša sa už spájajú vrstvy úplne, teda každý neurón s každým vo vrstve oproti.

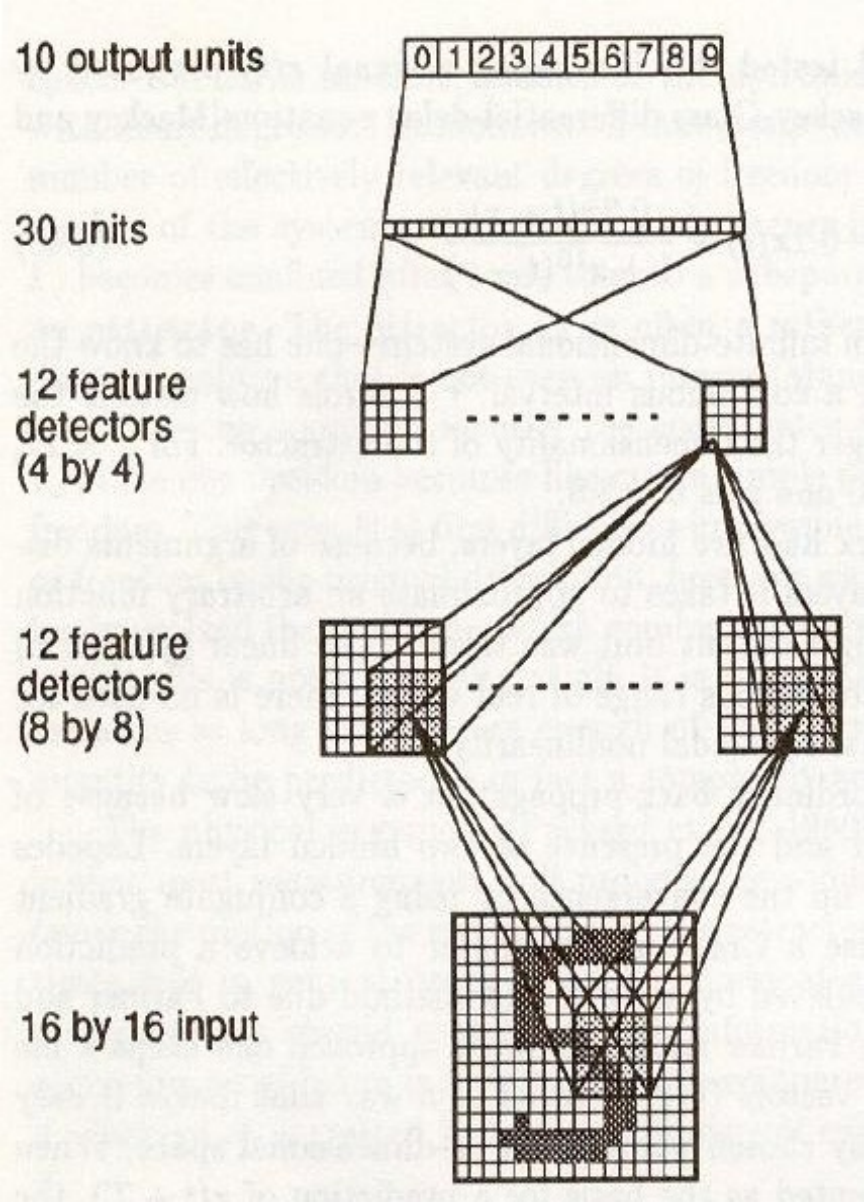
6.1.2 Zdieľanie váh

Tých najmenších príznakov, hlavne pri čiernobielych obrázkoch, nebýva veľa, a všetko ostatné sa skladá z nich. Na druhej strane, tieto príznaky sa nachádzajú všade. Teda prítomnosť jedného sa dá očakávať v ktorejkoľvek časti zorného poľa. Druhá zmena preto zredukovala počet príznakov na nízky počet - v danom pokuse Le Cuna na 12. Avšak každý FD sa naklonoval tak, aby sledoval všetky receptívne polia na vstupe. Takto vznikli skupinky FD. Každá skupinka sa naučila na jeden príznak a detekovala ho na ľubovoľnom mieste v zornom poli.

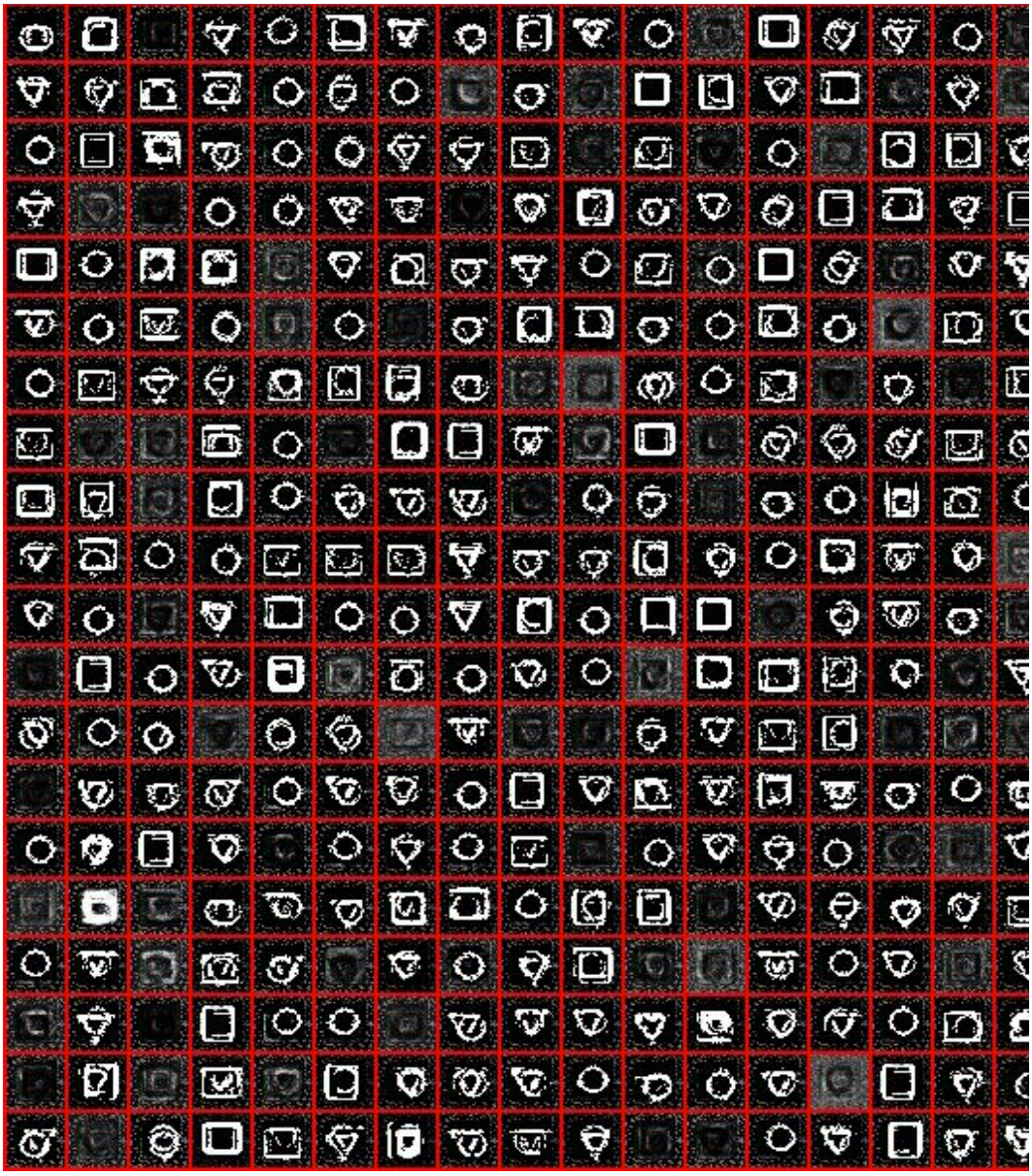
Pre prehľadnosť som urobil váhové profily prvej skrytej vrstvy. Tie prikladám na najbližších stránkach. Na nich je jasne vidieť, že takmer každý neurón sa naučil reagovať na celý obrázok, hoci prvé očakávanie bolo, že na pri vizualizácí váh sa budú ukazovať iba niektoré črty obrázkov.

Pôvodný úmyseľ nebol síce detekovať obrázky, ktoré by sa hýbali po zornom poli, ale myšlienka bola taká, že aj tak sa všetky obrázky skladajú z tých istých príznakov, ktoré sa nachádzajú rozmiestnené po celej pixlovej mape.

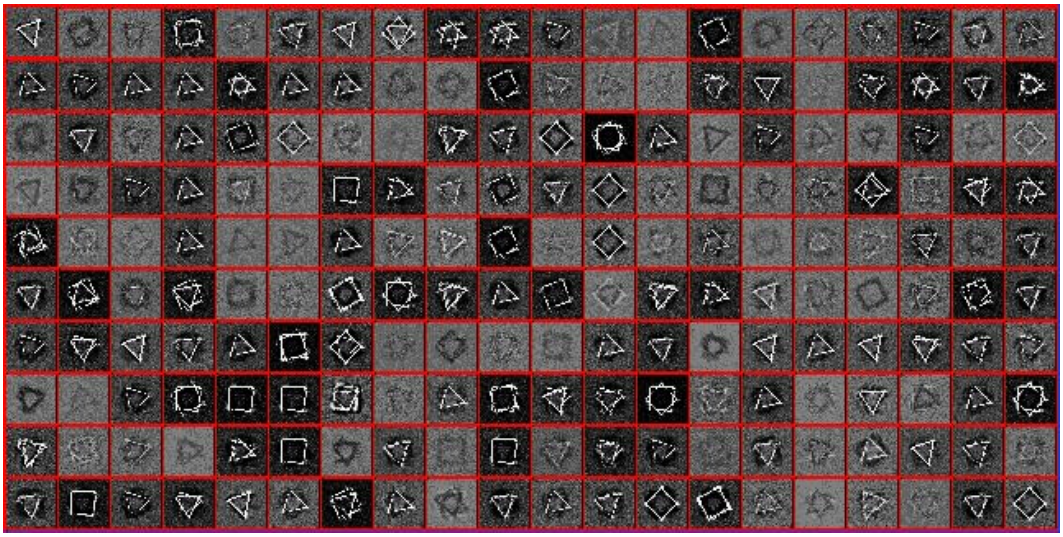
V každej skupinke je rovnaký počet neurónov, topologicky usporiadaných, každý z nich spojený s inou receptívnou mapou a vrámci skupinky sú váhové spojenia totožné. Viď obrázok na následujúcej stránke.



Obr.: 16 *Sieť so zdieľanými váhami podľa Le Cuna [6]. V prvej skrytej vrstve vidíme 12 FD, každý z nich namnožený 64 krát, sledujúci svoju receptívnu mapu o veľkosti 5x5 pixlov. V druhej skrytej vrstve je 12 FD, každý z nich sleduje 8 receptívnych polí z 8 skupiniek. Každý z 12 FD je rozmnožený 16 krát.*



Obr.: 17 *Časť váhového profilu prvej skrytej vrstvy siete, ktorá sa učila kategorizovať pozične variabilné obrázky. V každom okienku váhový profil jedného zo 700 neurónov. Je vidno, že väčšina neurónov sa naučila reagovať na jeden celý obrázok, alebo na dva rôzne zároveň, či naopak reagovať stlmením signálu pre nejaké obrázky.*



Obr.: 18 *Časť váhového profilu prvej skrytej vrstvy siete, ktorá sa učila kategorizovať uhlovo variabilné obrázky.*

6.2 Úprava DBN

Ako riešenie problémov s invariantými obrázkami som sa pokúsil implementovať sieť Deep Belief Network s takto upravenou štruktúrov. Učenie prebieha rovnako ako pri plne prepojenej DBN, ale zmeny váh sa vypočítajú pre každý klon FD zvlášť, a nakoniec sa spriemerujú.

6.2.1 Výsledky DBN so zdieľanými váhami

Prvé výsledky po implementovaní tohto modelu neboli povzbudivé. Sieť bola úplne zmätená pri polovici testových vstupoch. Vyskúšal som úpravy váh v rôznych fázach učenia - po všetkých iteráciach, aj okamžite po skončení jednej učiacej iterácie. Tieto úpravy učiaceho algoritmu mi nepomohli.

Ako počas celej práce, rozhodol som sa sledovať správanie sa siete počas šírenia signálov pri rozpoznávaciom procese. V prvej skrytej vrstve s FD mapami a zdieľanými váhami som objavil možný problém.

Pri mojich typoch obrázkov bola väčšina receptívnych polí vždy prázdna. Tým pádom boli vstupné net hodnoty signálov pre väčšinu neurónov blízke nule, a teda pravdepodobnosť ich aktivácie blízka 50 %. Táto vlastnosť spôsobila, že viac ako polovica signálov v prvej skrytej vrstve so zdieľanými váhami bola zapnutá náhodne a úplne nepredvídateľne.

Ako rozdiely oproti použitiu modelu LeCuna pri back propagation učení boli:

1. pri error back propagation sa neuróny správajú deterministicky - teda nemetú sieť meniacou aktivitou pri tých istých vstupoch

2. MNIST číslice zaberali celé zorné pole vstupu - teda každé receptívne pole bolo vyplnené nejakým vzorom, a nie prázdnym vstupom

Tieto dva rozdiely som zabudol zvážiť pred navrhnutím modelu. Ako možné riešenie teraz prichádza do úvahy zmena prezentácie signálov - namiesto hodnôt 0 a 1 by bolo lepšie použiť hodnoty -1 a +1, alebo -0.5 a +0.5. Takýto bipolárny neurón by tým pádom šíril obe hodnoty.

Kapitola 7

Záver

V tejto práci som sa zaoberal stochastickými hierarchickými modelmi neurónových sietí, konkrétne modelom DBN. Po implementácii modelu, a optimalizácií časovej zložitosti som analyzoval funkčnosť modelu a jeho správanie sa pri rôznych kategorizačných a generatívnych úlohách. Z výsledkov mojich testov usudzujem, že model Deep Belief Network nie je vhodný na riešenie kategorizácie invariantných 2D obrázkov, nakoľko v jeho podstate nezohľadňuje metriku roviny. Napriek tomu je však tento model dostatočne silný na to, aby sa naučil rozpoznávať obrázky pod silnou mierou deformácie.

Sieť dokázala celkom presne kategorizovať veľké množstvo rôznch obrázkov, ale len za tých podmienok, že dochádzalo k systematickým prekryvom.

Sieť dokázala niektoré vstupné dáta rekonštruovať. Bola taktiež schopná vyseparovať jednotlivé vstupy jednej triedy, a tak napríklad vygenerovať dva štvorce v rôznych natočeniach, bez toho, aby vznikol jeden obrázok ako zjednotenie dvoch rôznch. Túto schopnosť pripisujem práve hierarchickému učeniu siete.

V mojej práci som ďalej navrhol niekoľko úprav, ako napríklad riešenie rekonštrukcie pomocou algoritmu Gibbs sampling na vrchnej vrstve, zosilnenie signálu z výstupného neurónu, vytvorenie 2D receptívnych polí inšpirované modelom Le Cuna, zmena binárnych hodnôt neurónov, a podobne. Niektoré z týchto zmien som aj implementoval.

Literatúra

[1] Stuart Russel: Artificial Inteligence

[2] Ilya Sutskever, Geoffrey Hinton, and Graham Taylor: 2009, The Recurrent Temporal Restricted Boltzmann Machine

[3] Ilya Sutskever, Geoffrey Hinton: 2008, Deep, Narrow Sigmoid Belief Networks Are Universal Approximators

[4] Geoffrey E. Hinton, Simon Osindero, Yee-Whye Teh: 2006, A fast learning algorithm for deep belief nets

[5] Grigorios Tzortzis, Aristidis Likas: 2008, Deep Belief Networks for Spam Filtering

[6] Le Cun: 1990, Handwritten Digit Recognition with a Back-Propagation Network

[7] Le Cun: 2006, Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition

[8] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, Radford M Neal: 1995, The wake-sleep algorithm for unsupervised neural networks

[9] Honglak Lee, Chaitanya Ekanadham, Andrew Y. Ng: 2007, Sparse deep belief net model for visual area V2

[10] Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Rumelhart and J. L. McClelland (Eds.), Parallel distributed processing, 194–281. Cambridge: MIT Press.

[11] Joshua M. Susskind, Geoffrey E. Hinton, Javier R. Movellan, Adam K. Anderson: 2008, Generating Facial Expressions with Deep Belief Nets. Affective Computing, Emotion Modelling, Synthesis and Recognition. ARS Publishers.

[12] Y. Bengio and Y. LeCun: 2007, Scaling learning algorithms towards ai. $Large - ScaleKernelMachines.MITPress$