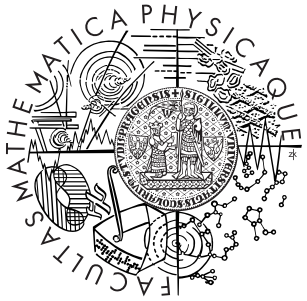
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martin Schmid

Konvoluční neuronové sítě a jejich implementace

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: doc. RNDr. Iveta Mrázová, CSc.
Studijní program: Informatika
Studijní obor: Programování

2011

Poděkování

Na úvod své bakalářské práce bych rád poděkoval všem, kteří mě podporovali v průběhu přípravy této práce.
V prvé řadě děkuji své vedoucí, doc. RNDr. Iveta Mrázové, CSc. za odborné vedení, cenné rady a přínosné připomínky.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 27. července 2011 Martin Schmid

Obsah

Název práce: Konvoluční neuronové sítě a jejich implementace
Autor: Martin Schmid
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: doc. RNDr. Iveta Mrázová, CSc.
E-mail vedoucího: Iveta.Mrazova@mff.cuni.cz

Abstrakt:

Tato práce ukazuje možnost využití *konvolučních neuronových sítí* pro rozpoznávání symbolů z obrázku. Popisuje tento model a také představuje jeho implementaci. Tato implementaci je následně použita v ukázkové aplikaci.

Nejprve je představen model neuronových sítí. Poté následuje popis učení tohoto modelu, včetně bližšího popisu algoritmu zpětného šíření. Nakonec je rozebrán model konvolučních neuronových sítí, kde jsou ukázány jeho přednosti při rozpoznávání obrázků

Dále se analyzují některé stávající implementace neuronových sítí. Každá je blíže představena a nakonec porovnány rychlosti. Protože žádná z těchto implementací nepodporuje model *konvolučních* neuronových sítí, je jedna z implementací o tento model rozšířena. Následuje bližší popis problematiky implementace tohoto modelu a představeno rozhraní rozšířené knihovny (tj. jak uživatel tuto knihovnu může použít).

Pro předvedení vlastností *konvolučních* neuronových sítí a funkčnosti této knihovny, je nakonec vytvořena ukázková aplikace. Aplikace je dostupná na webových stránkách a spustitelná v prohlížeči. Využívá této knihovnu pro rozpoznávání symbolů kreslených uživatelem přímo do této aplikace.

Klíčová slova: Konvoluční neuronové sítě, OCR, Encog

Abstract:

Bachelor thesis describes using convolutional neural networks for recognizing symbols from images. First describes this model and shows it's implementation. Then this implementation is used for sample application.

First, model of neural networks is described, then learning of this model (including backpropagation algorithm). Finally, convolutional neural networks are presented with it's advantages for symbol recognition.

Then some existing implementations of neural networks are analyzed, including speed comparison. None of these implementations support convolutional networks, so this model is added to one of them. Then this extension and it's interface (how to use it) is presented.

To show features of this model and to prove functionality of the implementation, sample application is created. This application is available on the web site and runnable using only a web browser.

Keywords: Convolutional neural networks, OCR, Encog

Úvod

Při scanování dokumentů, knížek, či při vyplňování různých dokumentů které se následně automaticky zpracovávají, je výhodou rozpoznávat z těchto obrázků různé symboly. Například při vyplnění jména a rodného čísla se jedná o symboly abecedy a cifry. Na tyto dokumenty narazíme na poštách či úřadech, kde by bylo čtení těchto dokumentů člověkem příliš časové náročné. Aktuálním příkladem je nedávné sčítání lidu, kde byly formuláře po ručním vyplnění automaticky zpracovány. Takto ručně vyplněné formuláře jsou v porovnání s naskenovanou knihou náročnější rozpoznat. Každý obyvatel totiž píše jinak a dokonce i stejný symbol nemusí napsat úplně stejně (třes ruky, spěch).

V této práci ukážeme možnost využití *konvolučních neuronových sítí* pro rozpoznávání symbolů z obrázku. Popíšeme tento model a také představíme jeho implementaci. Tuto implementaci použijeme v ukázkové aplikaci.

Členění práce

Model neuronových sítí

V první kapitole představíme model neuronových sítí. V druhé kapitole následuje bližší popis algoritmu zpětného šíření pro tento model. Třetí kapitola se zaobírá modelem konvolučních neuronových sítí, kde ukážeme jeho přednosti při rozpoznávání obrázků.

Implementace neuronových sítí

Ve čtvrté kapitole budeme analyzovat některé stávající implementace neuronových sítí, porovnáme zde i jejich rychlost. V kapitole číslo pět popíšeme problematiku implementace modelu konvolučních neuronových sítí a tento model implementujeme do knihovny *Encog*. Šestá kapitola popisuje vytvořenou ukázkovou aplikaci včetně jejího ovládání.

Kapitola 1

Neuronové sítě

V této kapitole představíme výpočetní model zvaný *neuronové sítě*. Ačkoliv se jedná o model inspirovaný stejnojmenným biologickým pojmem, nebudeme nijak tuto souvislost blíže rozebírat.

1.1 Neuron, neuronová síť

Definice 1.1.1 *(Neuron)* Je trojce $(f, \vec{w}, t)$ kde

- f je *přenosová funkce* $R \to R$
- $\vec{w} \in R^n$ je *vektor vah*
- $t \in R$ je *prah*

Neuron pro svůj vstup $\vec{x} \in R^n$ počítá výstup $y \in R$

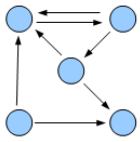
$$y = f(wx^T + t) \tag{1.1}$$

Dále $\xi = wx^T + t$ nazveme *potenciál*.

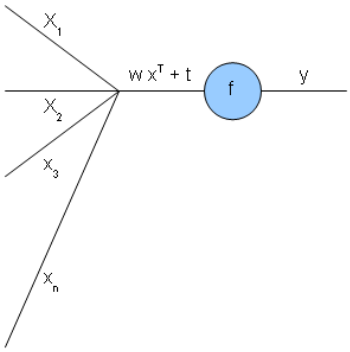
Takovéto neurony můžeme mezi sebou propojovat (výstup y může sloužit jako část vstupu $\vec{x}$ neuronu druhého) a tím vytvořit neuronovou síť Neuron tedy slouží jako základní výpočetní jednotka neuronových sítí.

Definice 1.1.2 *(Neuronová síť)* Je dvojce (N, C) kde

- N je *množina neuronů*
- $C \subseteq N \times N$ je orientovaná množina *spojů*



Obrázek 1.1: Neuronová síť

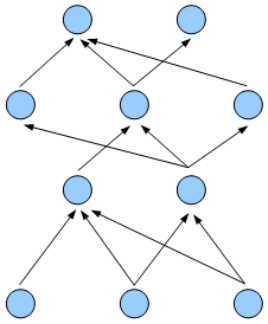


Obrázek 1.2: Modelu neuronu

Definice 1.1.3 *(Vrstevnatá neuronová síť)* Vrstevnatá neuronová síť s k vrstvami je neuronová síť, kde navíc

- C (orientovaná množina spojů) neobsahuje orientovaný cyklus)
- $L = \{l_1, l_2, ..., l_n\}$, kde $l_x \subset N$, jsou *vrstvy*
- Spoje vedou vždy pouze z vrstvy l_x do l_{x+1}, říkáme že vedou z nižší vrstvy do vrstvy vyšší
- První vrstvu nazveme vstupní I, poslední vrstvu výstupní O

Z definice je patrné, že do vstupní vrstvy nevedou žádné spoje. Podobně žádné nevedou z vrstvy výstupní.



Obrázek 1.3: Vrstevnatá neuronová síť

1.2 Výpočet výstupu

Podobně jako neuron počítá pro vstup $\vec{x} \in R$ svůj výstup $y \in R$, také neuronová síť pro vstup $x^i \in R$ počítá výstup $y^o \in R$ (i je počet vstupních neuronů, o počet výstupních).

Výpočet výstupu sítě probíhá iterativně z nejnižší (vstupní) vrstvy směrem vzhůru až do vrstvy nejvyšší (výstupní). V každé iteraci spočítají všechny neurony z vrstvy i vrstvy svůj výstup na základě vstupu z vrstvy $i - 1$.

Kapitola 2

Algoritmus zpětného šíření

Mějme vrstevnatou neuronovou síť

$$V = (N, C, I, O, w, t) \tag{2.1}$$

viz předchozí kapitola. Dále definujeme pojmy *vstupní vzor*, *požadovaný výstup* a *trénovací množina*

Definice 2.0.1 *(Vstupní vzor)* je vektor $\vec{x} \in R^m$, kde m je počet vstupních neuronů

Definice 2.0.2 *(Požadovaný výstup)* je vektor $\vec{y} \in R^n$, kde n je počet výstupních neuronů

Definice 2.0.3 *(Trénovací množina)* T je množina uspořádaných dvojic T = [*vstupní vzor*, *požadovaný výstup*]

Cílem libovolného algoritmu učení je nastavit váhy w sítě tak, aby byl skutečný výstup sítě co nejpodobnější výstupu požadovanému - a to pro celou trénovací množinu. Tuto podobnost nám charakterizuje chybová funkce ERR. Ta má na vstupu požadované výstupy sítě spolu se skutečnými, na svém výstupu hodnotu chyby (tj. jak moc se výstupy liší od požadovaných). Algoritmus učení se tedy snaží minimalizovat danou chybovou funkci.

Algoritmus zpětného šíření je algoritmus učení . Jedná se o gradientní metodu, která iterativně adaptuje každou váhu na základě parciální derivace chybové funkce.

$$w_{ij}(t+1) = w_{ij}(t) - \alpha \frac{\partial ERR}{\partial w_{ij}(t)} \tag{2.2}$$

Kde $\alpha \in R$ je parametr učení, a $w_{ij}(n)$ je váha w_{ij} v iteraci n. Posouváme tedy váhu proti směru derivace.

Jednotlivá iterace tohoto algoritmu probíhá

1. Výpočet výstupu sítě pro daný vzor
2. Výpočet parciálních derivací chybové funkce pro všechny váhy
3. Adaptace vah sítě dle (2.2)

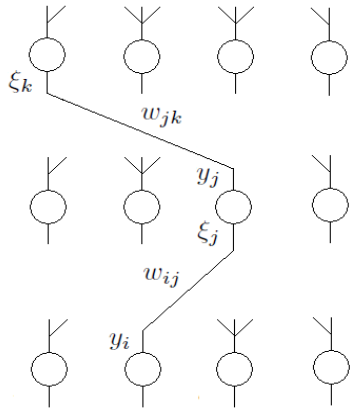
2.1 Výpočet parciálních derivací

Nyní si odvodíme adaptační pravidla pro algoritmus zpětného šíření, založeném na výpočtu parciálních derivací chybové funkce. V našem případě použijeme chybovou funkce SE (*squared error*)

$$SE = \frac{1}{2}\sum_{a=1}^{t}\sum_{b=1}^{m}(y_{ab} - d_{ab})^2 \tag{2.3}$$

Kde t je velikost trénovací množiny a m počet výstupních neuronů. Nejrpve si zavedeme některé symboly, které při výpočtu derivací budeme používat

$$\begin{aligned} w_{ij} &= \text{váha mezi neuronem i a j} \\ y_i &= \text{výstup neuronu i} \\ \xi_j &= \sum_i w_{ij} y_i (\text{vážený součet} \end{aligned}$$



Obrázek 2.1: Ilustrace jednotlivých symbolů použitých při vyjádření derivací

Nyní se již můžeme pustit do vyjádření

$$\frac{\partial SE}{\partial w_{ij}} = \frac{\partial \frac{1}{2} \sum_a \sum_b (y_{ab} - d_{ab})^2}{\partial w_{ij}}$$

$$= \sum_a \sum_b ((y_{ab} - d_{ab}) \frac{\partial y_{ab}}{\partial w_{ij}})$$

$$\frac{\partial y_{ab}}{\partial w_{ij}} = \sum_k \frac{\partial y_{ab}}{\partial \xi_k} \frac{\partial \xi_k}{\partial w_{ij}} = \sum_k \frac{\partial y_{ab}}{\partial \xi_k} \frac{\partial \xi_k}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}}$$

$$= \sum_k \frac{\partial y_{ab}}{\partial \xi_k} \frac{\partial \xi_k}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial w_{ij}} = \sum_k \frac{\partial y_{ab}}{\partial \xi_k} \frac{\partial \xi_k}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \sum_g w_{gj} y_g}{\partial w_{gj}}$$

$$= \sum_k \frac{\partial y_{ab}}{\partial \xi_k} \frac{\partial \xi_k}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} y_i = \sum_k \frac{\partial y_{ab}}{\partial \xi_k} \frac{\partial \xi_k}{\partial y_j} f'(\xi_j) y_i = \sum_k \frac{\partial y_{ab}}{\partial \xi_k} w_{jk} f'(\xi_j) y_i$$

Navíc

$$\frac{\partial y_{ab}}{\partial \xi_j} = \sum_k \frac{\partial y_{ab}}{\partial \xi_k} w_{jk} f'(\xi_j)$$

Tedy při výpočtu $\frac{\partial y_{ab}}{\partial w_{ij}}$, potřebujeme všechna $\frac{\partial y_{ab}}{\partial \xi}$ z vrstvy o jedna vyšší, ale zároveň jsme vypočítali $\frac{\partial y_{ab}}{\partial \xi_j}$ z naší vrstvy. Spočítáme-li parciální derivace dle vah pro všechny váhy v jedné vrstvě, dostaneme i všechny $\frac{\partial y_{ab}}{\partial \xi}$ z této vrstvy. Tyto členy budou poté potřeba pro výpočet parciálních derivací dle vah na nižší úrovni. Takto můžeme rekurentně pokračovat až do vstupní vrstvy. Potřebujeme však začít na výstupní vrstvě - pro ni spočítáme všechna $\frac{\partial y_{ab}}{\partial \xi}$ snadno

$$\frac{\partial y_{ab}}{\partial \xi} = f'(\xi)$$

Algoritmus zpětného šíření tedy potřebuje vyjádřit derivaci přenosové funkce. Například derivace sigmoidální přenosovou funkce (viz vrstevnaté neuronové sítě) vypadá následovně.

$$f'(\xi) = f(\xi)(1 - f(\xi))\lambda$$

Pro výpočet derivace sigmoidální přenosové funkce potřebujeme pouze násobení a odečítání (pokud známe hodnotu této funkce pro předložený vstupní vzor), a tato derivace je tedy snadno dopočítatelná.

2.2 Časová složitost jednotlivé iterace

Nechť N označuje počet vah učené sítě.

- Výpočet výstupu daného vzoru

  $\theta(N)$

- Výpočet parciálních derivací pro všechny váhy

  Postupujeme směrem shora dolů a v každé vrstvě počítáme derivace pro všechny její váhy. Potřebujeme výsledky z vrstvy vyšší, které jsme ale jíž v předchozí iteraci spočítali. Dále výstupy z vrstvy nižší (předchozí krok iterace). Celkem projdeme všechny vrstvy a jejich váhy. Dostáváme $\theta(N)$

- Adaptace vah dle 2.2

  $\theta(N)$

Celková časová složitost $\theta(N)$

2.3 Ukončovací kritéria

Iterativní proces učení musíme někdy zastavit. Protože se snažíme minimalizovat chybovou funkci, můžeme zastavit po dosažení požadované hodnoty chybové funkce. SE (2.3) však počítá chybu absolutně (suma přez všechny vzory), je vhodnější použít chybovou funkci MSE (*mean squared error*)

$$MSE = \frac{SE}{t} \tag{2.4}$$

Další možností je například zastavit proces učení při dosažení požadovaného maximálního množství iteraci. Algoritmus zpětného šíření nám nezaručuje konvergenci, a tedy nemusí nikdy požadované chyby docílit.

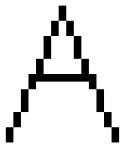
Kapitola 3

Konvoluční neuronové sítě

V této kapitole se nejprve se podíváme na problém, kde vrstevnaté neuronové sítě nemají vhodné vlastnosti. Poté si popíšeme model, jehož motivací je lepší chování právě na těchto typech problémů.

3.1 Motivace

Představme si použití modelu vrstevnatých sítí na rozpoznávání znaků z obrázku. Pro jednoduchost bude naším obrázkem černobílá mřížka/obrázek o rozměrech 20 na 20.



Obrázek 3.1: Příklad vstupu pro síť

Vstupní vektor sítě odpovídající tomuto obrázku bude mít velikost 20×20 a bude obsahovat pouze čísla 0 a 1. Jednička bude odpovídat černému pixelu, nula bílému. Vytvoříme síť s několika skrytými vrstvami a můžeme ji naučit například algoritmem zpětného šíření. Problém však může nastat při rozpoznávání vzorů, které jsou oproti naučeným posunuté či jinak zdeformované.

Při posunutí se nám i celý vstupní vektor posune, a výstup sítě nemusí být nijak blízký požadovanému. Samotné posunutí by se dalo vyřešit vystředěním každého obrázku (často se používá těžiště nebo střed nejmenšího obdélníku), ale při ručním psaní nejde určitě o jedinou deformaci. Ukažme si vše na příkladě.

(a) Vzor (b) Vstup (c) Rozdíl

Obrázek 3.2: Deformace vstupního obrázku. Šedou barvou jsou zobrazeny shodné pixely.

Tyto dva obrázky jsou si na první pohled velmi podobné, vstupní vektor se však bude velmi lišit. Motivací pro *KNS* je robustnost právě vůči takovýmto deformacím. Té je dosaženo pomocí techniky sdílení vah, kde je jedna a ta samá váha sdílena více neurony. Pokud jsme takovéto váhy naučili na náš vzor a vstupní obrázek bude zdeformovaný, nedostane se stejný vstup na stejné neurony. Může se však dostat na neurony, které s nimi šikovně sdílejí váhu. Vše si samozřejmě popíšeme mnohem podrobněji.
Základním stavebním kamenem *KNS* jsou *příznaková mapa* a *redukční vrstva*.

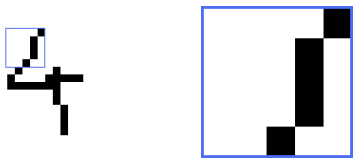
3.2 Konvoluční vrstva

Konvoluční vrstva se skládá z několika příznakových map. Může navazovat jak na vstupní vrstvu, tak na vrstvu subsamplingovou.

3.2.1 Příznaková mapa

Myšlenkou je získat lokální charakteristiky obrázku. Pod pojmem lokální máme na mysli, že z obrázku o rozměrech 20 na 20 budeme uvažovat podobrázek o velikosti například 5 na 5. Tato oblast bude vstupem jednoho neuronu, přesněji každý pixel tohoto podobrázku. Bude mít tedy 25 vstupů + práh, 26 celkem

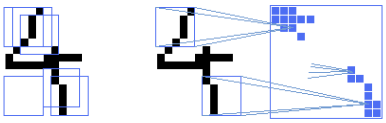
Nyní využijeme techniky sdílení vah. Tento neuron rozkopírujeme pro každou podoblast dané velikosti. (jednotlivé podoblasti se přitom překrývají). Podobrázku této velikosti je 16 krát 16, celkem 256. Bude li každý z těchto podobrázku sloužit jako vstup jednoho neuronu, dostaneme i stejné množství neuronů (256). Všechny tyto neurony sdílejí váhy, včetně prahu. Proto je unikátních vah v mapě pouze 26, zatímco neuronů 256. Díky takto rozkopírovaným neuronům se při posunutí lokální charakteristiky obrázku výstup z neuronů také pouze posune.



(a) Celý obrázek (b) Podobrázek

Obrázek 3.3: Podobrázek sloužící jako vstup neuronu

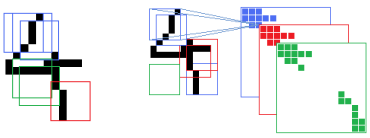
Příznak si můžeme představit například vertikální čáru, křivku či plně bílou oblast (vše na oblasti 5 na 5). Sada těchto 256 neuronů se nazývá příznaková mapa.



(a) Podoblasti (b) Příznaková

Obrázek 3.4: Napojení příznakové mapy

Při rozpoznávání obrázků je vhodné použití více příznaků. Pokud budeme rozeznávat například kromě vertikálních čar také horizontální čí další příznaky, může nám to při rozpoznání orbázku pouze pomoci. Proto těchto map použijeme více.



(a) Podoblasti (b) Příznaková mapa

Obrázek 3.5: Více příznakových map

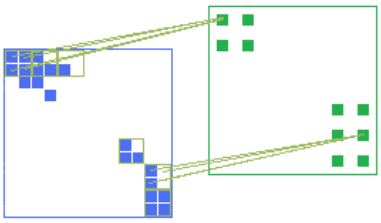
Každá mapa sdílí váhy svých neuronů, mezi sebou však nesdílejí žádnou. Mohou však obsahovat stejný počet neuronů a být napojeny na stejně velké podoblasti. Tedy při 6 mapách, vždy napojených na oblasti 5x5, bychom měli celkem 1536 neuronů (256*6). Vah by bylo pouze 156 (každá mapa pouze 26 vah sdílených mezi všemi neurony).

3.3 Subsamplingová vrstva

Subsamplingová vrstva je vždy napojena na předchozí konvoluční vrstvu a obsahuje stejné množství příznakových map jako tato konvoluční vrstva. Příznakové mapy z této subsamplingové budeme vrstvy značit $P2$, zatímco mapy z předchozí(konvoluční) vrstvy, $P1$. Každá příznaková mapa $P2$ ze subsamplingové vrstvy je spojena s jednou příznakovou mapu $P1$ z vrstvy předchozí. Neurony příznakových map $P2$ jsou tentokrát napojeny na nepřekrývající se podoblasti příznakových map $P1$. Protože se podoblasti nepřekrývají (a jsou větší než 1×1), je každá příznaková mapa $P2$ nutně menší než odpovídající $P1$. Tyto podoblasti jsou typicky čtvercové, v našem případě o velikosti 2×2.

3.3.1 Příznaková mapa

Opět všechny neurony příznakové mapy $P2$ sdílejí své váhy (včetně prahu), zatímco příznakové mapy nesdílejí váhu žádnou. V tomto případě však navíc každý neuron má pouze jednu váhu sdílenou spojeními na podoblast příznakové mapy $P1$. Tj. například pro podoblasti $2 * 2$, bude mít neuron 4 spojení a práh. Váhy pro tato 4 spojení budou však sdíleny (nikoliv práh) - neuron bude mít pouze 2 unikátní váhy (jedna pro práh). Příznaková mapa $P2$ se skládá z takovýchto neuronů pokrývajících svými podoblastmi celou odpovídající příznakovou mapu $P1$. Díky sdílení vah mezi neurony však naše celá příznaková mapa obsahuje pouze 2 váhy (a obecně stejně jako každý její neuron). Počet neuronů příznakové mapy $P2$ napojené podoblastmi o velikosti $2*$ na příznakovou mapu $P1$ o velikosti $16x16$ neuronů by byl $8x8$.



Obrázek 3.6: Podoblasti se v tomto případě nepřekrývají

Kapitola 4

Existující implementace

Implementací modelu neuronových sítí je velké množství a často se jedná o knihovny, které může programátor snadno použít. V této kapitole se blíže podíváme na tyto knihovny. U každé knihovny uvedeme navíc základní vlastnosti, jako je licence, použitý programovací jazyk a rozšířenost. Měřítkem rozšířenosti je počet výsledků, který vrátí vyhledávač *google* [10] po zadání názvu konkrétní implementace.

4.0.2 FANN

FANN je zkratkou pro "Fast Artificial Neural Network Library". Tato knihovna je napsána v jazyce C a jak již samotný název napovídá, mělo by se jednat o knihovnu kladoucí důraz na rychlost. Hned na úvodní stránce knihovny [3] se dokonce dočteme, že je až 150 krát rychlejší než ostatní knihovny. Knihovna podporuje pouze vrstevnaté neuronové sítě, a také seznam učících algoritmů není příliš rozsáhlý [8].

Vnitřně jsou vrstvy reprezentovány pomocí polí. Na stránkách knihovny najdeme možnosti integrace s jinými programovacími jazyky (Java, C#, PERL, Python, Prolog, Haskel a další) či prostředími jako Matlab.

Domovská stránka: `http://leenissen.dk/fann/`

Autor: Steffen Nissen

Jazyk: C

Licence: LGPL

Rozšířenost: 10,400 výsledků

4.0.3 Neuroph

Tento projekt původně vznikl jako diplomová práce na universitě v Bělehradu. Neuroph je framework napsaný v Jave, podporující velké množství modelů neuronových sítí. Z tohoto pohledu se jedná o mnohem silnější nástroj než kterým je *FANN*. K dispozici je také aplikace, která umožňuje sítě postavit grafickým návrhem.
Vnitřně jsou jednotlivé vrstvy a neurony implementovány pomocí tříd. Jednotlivá spojení synapsí jsou také řešeny objektově. To sice přináší pěkný objektový návrh, ale na druhou stranu rychlost je nutně menší než přes implementaci pomocí polí.

Domovská stránka: `http://neuroph.sourceforge.net/`

Autor: Zoran Sevarac

Jazyk: Java

Licence: Apache licenece V2.0

Rozšířenost: 72,700 výsledků

4.0.4 Encog

Zajímavostí tohoto projektu je nezávislá implementace ve 2 jazycích - Java a C#. Jedná se skutečně o 2 projekty se snahou sdílet funkcionalitu. Vzhledem k velké podobnosti zmíněných jazyků a také stejného hlavního vývojáře jsou obě implementace téměř totožné. K oběma projektům existují i samostatné knížky (z pochopitelných jsou si tyto knížky velmi podobné).
Na vrstvy a perceptrony se zde nahlíží ze dvou pohledů. První, objektový, je podobný jako u Neuroph-u a přináší jednoduché API. Druhý pohled je reprezentace pomocí polí a slouží hlavně pro rychlou práci se sítí (*FlatNetwork*). Uživatel nejprve vytvoří síť pomocí objektové API a poté sdělí, že je síť již finální. To způsobí vytvoření druhé sítě, tentokrát však typu *FlatNetwork*. Všechny váhy jsou tedy v paměti uloženy dvakrát a před začátkem učení dokonce vždy po dvojici stejné. Samotný algoritmus učení poté již pracuje s *FlatNetwork* (čili s poli). Jakmile je učení ukončeno, jsou váhy z polí nakopírovány zpět do objektů.

Domovská stránka: `http://www.heatonresearch.com/`

Autor: Jeff Heaton

Jazyk: Java, C#

Licence: LGPL, Apache licenece V2.0

Rozšířenost: 185,000 výsledků

Matlab

Nástroj, který se zdaleka neomezuje na práci s neuronovými sítěmi. Máme zde dispozici grafické prostředí, které nám umožňuje nejen síť postavit, ale také sledovat průběh učení. Právě druhá zmíněná funkcionalita má však nepříjemnou vlastnost - při základním nastavení proces učení velmi zpomaluje. Tato funkcionalita jde naštěstí vypnout a učení je poté výrazně rychlejší.

Pro zvýšení rychlosti podporuje Matlab vytvořit zdrojový kód v jazyce C (Matlab compiler). Tato možnost bohužel není dostupná pro učení neuronových sítí, ale pouze pro výpočet s již naučenou sítí [11]. Proto jsem tuto možnost nemohl do testu zahrnout.

Domovská stránka: `http://www.mathworks.com/products/matlab/`

Autor: Mathworks

Jazyk: Matlab

Licence: Proprietární

Rozšířenost: 136,000,000 výsledků

4.1 Porovnání rychlosti

Zde porovnáme rychlost představených implementací.

4.1.1 Podmínky testu

Hardware

Test probíhal na počítači s dvou jádrovým procesorem Intel Core 2 Duo s frekvencí jednoho jádra 2.26GHz a celkovou operační pamětí 2GB. Každá sada

testů byla spuštěna celkem desetkrát a výsledné číslo zobrazené v grafech je aritmetický průměr z těchto časů.

Musíme si uvědomit, že je-li implementace A rychlejší než implementace B na jednom počítači, nemusí být rychlejší na jiném. Příčinou může být podpora většího množství procesorů (či jader) nebo grafické karty.

Testované neuronové sítě

Celkem jsme testovali na třech neuronových sítích. Každá síť měla vždy vstupní vrstvu obsahující dva neurony a výstupní vrstvu s jedním neuronem. Skrytých vrstev měla první testovaná síť jednu, druhá dvě a třetí tři. Každá skrytá vrstva obsahovala 100 neuronů.

Trénovací množina

Trénovací množina vypadala následovně.

Vstup sítě	{0, 0}	{0, 1}	{1, 0}	{1, 1}
Požadovaný výstup	{0}	{1}	{1}	{0}

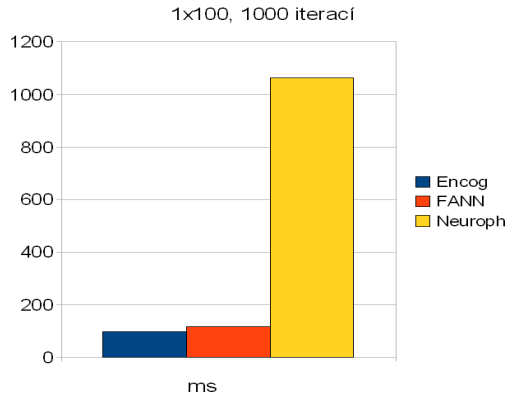
a jedná se tedy o funkci XOR.

Počet iterací a učení

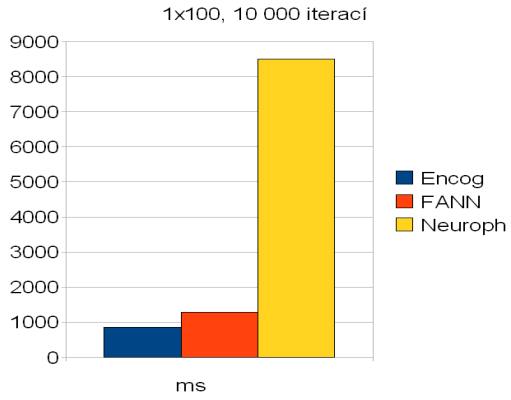
Celkový počet provedených iterací se lišil a u každého grafu je zobrazen zvlášť. Algoritmus učení jsme nastavili na zpětné šíření. Protože nás zajímá celkový čas daného počtu iterací a nikoliv počet iterací potřebných k naučení, počáteční nastavení vah nehraje roli. U každé knihovny jsme tedy postupovali podle tohoto schématu.

1. Vytvořit síť dané velikosti
2. Nastavit učící metodu na zpětné šíření
3. Nastavit učící koeficient na 0
4. Provést daný počet iterací

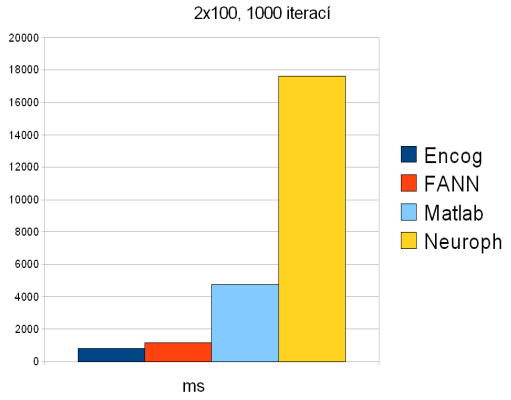
4.1.2 Výsledné grafy



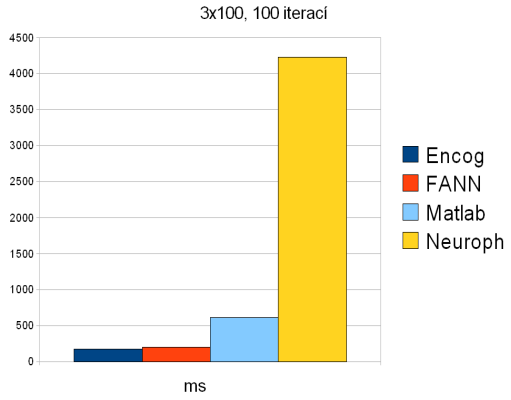
(a) Dvě vrstvy, 1000 iterací



(b) Jedna vrstva, 10 000 iterací



(a) Dvě vrstvy, 1000 iterací



(b) Tři vrstvy, 1000 iterací

4.1.3 Naměřené hodnoty

Encog	FANN	Neuroph
98ms	117ms	1063ms

(a) Jedna skrytá vrstva, 1000 iterací

Encog	FANN	Neuroph
858ms	12997ms	8501ms

(b) Jedna skrytá vrstva, 10 000 iterací

Encog	FANN	Matlab	Neuroph
839ms	1158ms	4756ms	17605ms

(c) Dvě skryté vrstvy, 1000 iterací

Encog	FANN	Matlab	Neuroph
167ms	204ms	615ms	4227ms

(d) Tři skryté vrstvy, 100 iterací

Z grafů je patrné, že *Encog* i *FANN* si vedou velmi dobře. Následuje *Matlab* a nejhorších výsledků dosahuje *Neuroph*. Je velmi zajímavé, že čím více vrstev síť obsahovala, tím více *Neuroph* zaostával. Při jedné skryté vrstvě byl přibližně 10 krát pomalejší (jak při 1000, tak 10 000 iteracích). Při dvou skrytých vrstvách 20 krát a při třech dokonce 25 krát.

Kapitola 5

Rozšíření knihovny *Encog* o model konvolučních neuronových sítí

V předchozí kapitole jsme si představili nejrozšířenější knihovny pro práci s neuronovými sítěmi. Ani jedna z těchto knihoven však nepodporuje model *konvolučních* neuronových sítí. Rozhodli jsme se tedy rozšířit jednu z těchto knihoven. Pro toto rozšíření jsme vybrali knihovnu *Encog*. Jednak dosáhla nejlepších výsledků v testu rychlosti, jednak je velmi rozšířená.

5.1 Implementace

Zde si představíme základní implementační problémy a jejich řešení týkající se jak reprezentace, tak následné práce s vahami. Zmíníme jednak problémy pro učení metodou zpětného šíření a rozebereme specifika pro konvoluční neuronové sítě.

5.1.1 Propojení vrstev v konvolučních NS

V jednoduchém modelu vrstevnaté NS jsou po sobě jdoucí vrstvy navzájem plně propojeny, tj. každý neuron z vyšší vrstvy je spojen s každým neuronem vrstvy nižší (a tedy i obráceně). Zamysleme se zde nad implementací. Pro výpočet výstupu sítě postupujeme směrem zdola nahoru. Protože máme plné spojení, nabízí se nám toto řešení (v pseudokódu)

Přímočarý výpočet

```
for all neuron z vrchní vrstvy do
  for all neuron z dolní vrstvy do
    proveď výpočet využívající váhu mezi těmito neurony
  end for
end for
```

Takto to skutečně funguje v Encog-u. Pro potřebu KNS nemůžeme tento přístup použít, protože zde nemáme plná spojení vrstev. Musíme si tedy nějak pamatovat, který neuron z vrchní vrstvy je spojen z kterým neuronem z vrstvy spodní. Máme dvě možnosti

1) Spojení ANO-NE Budeme mít síť reprezentovanou stejně jako v předchozím případě, ale pro každé spojení si budeme pamatovat, zda ve skutečnosti existuje či ne. Implementace by potom vypadala následovně:

Spojení ANO-NE

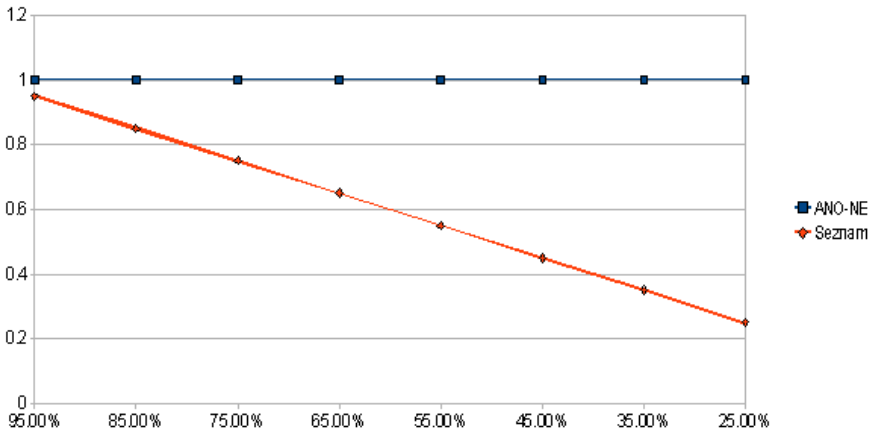
```
for all neuron z vrchní vrstvy do
  for all neuron z dolní vrstvyu do
    if existuje spojení then
      proveď výpočet využívající váhu mezi těmito neurony
    end if
  end for
end for
```

2) Seznam neuronů Pro každý neuron si pamatujeme, s kterými neurony ze spodní vrstvy je ve skutečnosti propojen.

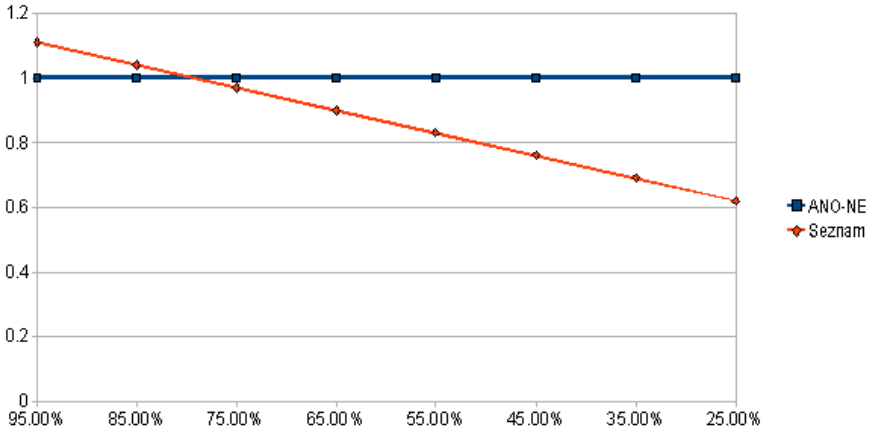
Seznam spojení

```
for all neuron z vrchní vrstvy do
  for all neuron ze seznamu spojení do
    proveď výpočet využívající váhu mezi těmito neurony
  end for
end for
```

Představme si, jak se budou chovat tyto dva přístupy pro vrstvy obsahující řádově stovky neuronů. První možnost by měla fungovat lépe, máme-li počet spojení relativně veliký. Chybí-li nám naopak jediné spojení mezi horní a dolní vrstvou, musíme se v druhé variantě zbytečně ptát s kterými neurony je spojen. Naopak pokud by byl spojen pouze jeden neuron, v první variantě by jsme se zbytečně mnohokrát zeptali, zda dané spojení existuje (a ono by neexistovalo). Dalším parametrem který musíme brát v úvahu je potřebná paměť. Pokud se pro každé spojení ptáme zda existuje, musíme si tuto informaci uložit v paměti. Zvolíme-li pole, bude mít velikost úměrnou součinu velikostí obou vrstev a jeho velikost nebude nijak souviset s počtem existujících spojení. Pokud si však pamatujeme pouze seznam existujících spojení, potřebná paměť je úměrná pouze počtu těchto spojení.
Otestujeme tyto dva přístupy pro dvě navazující skryté vrstvy s velikostí odpovídající konvoluční neuronové síti. Pokud máme konvoluční vrstvu například s 10 mapami, každá s velikostí okénka 10×10, tak tato vrstva by pro obrázek o rozměrech 24×24 pixelů obsahovala 2250 neuronů. Následující subsamplingová vrstva by poté obsahovala čtvrtinu z tohoto počtu neuronů (pro velikost subsamplingového okénka 2×2) Otestujeme tedy tyto dva přístupy na dvou následujících vrstvách, první s 2000 neurony, druhá s 500.



Obrázek 5.1: Paměť

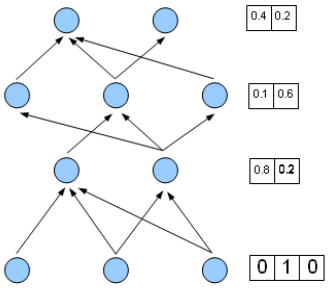


Obrázek 5.2: Čas

Vydíme, že druhé řešení je rychlejší, pokud je hustota spojení menší než 80% (kde 100% je plné spojení vrstev). Spočítáme hustotu spojení v síti, kterou použijeme v ukázkové aplikaci. Předpokládejme, že předchozí vrstva obsahuje 2000 neuronů. Nechť je následující vrstva konvoluční nebo subsamplingová. Potom hustota záleží pouze na velikosti okénka, protože neuron je spojen právě s neurony z předchozí vrstvy patřící do tohoto okénka. Pro okénko velikosti 10×10 je to 100 neuronů. Protože předchozí vrstva obsahuje 2000 neuronů, hustota je pouze 5%. Z tohoto důvodu jsme při implementaci zvolili variantu, kde si pamatujeme seznam spojených neuronů.

5.1.2 Výpočet výstupu

Při výpočtu výstupu sítě postupujeme iterativně směrem zdola nahoru. Pro výpočet výstupu aktuální vrstvy potřebujeme znát pouze výstup vrstvy předchozí (výstup každého neurony patřící do této vrstvy). Pro každou vrstvu si tedy můžeme v poli pamatovat výstup jednotlivých neuronů. Výstup vstupní vrstvy (kde začínáme) je roven vstupu. Při výpočtu výstupu neuronu z aktuální vrstvy dále potřebujeme vědet, s kterými neurony je vlastně spojen. Jak jsme již zmínili, tuto informaci máme uloženou v seznamu pro každý neuron. Dále musíme znát váhu pro toto spojení, tu ale není problém držet v seznamu spojení. Poté je již snadné vypočítat výstup každého neuronu z aktuální vrstvy, a tedy i výstup celé této vrstvy.



Obrázek 5.3: Pra každou vrstvu držíme v poli výstup

5.1.3 Výpočet derivací

Při výpočtu derivací postupujeme iterativně směrem shora dolů. Jak jsme zmínili v kapitole o algoritmu zpětného šíření (budeme se zde odvolávat na vzorečky a indexy vysvětlené v ní), při výpočtu $\frac{\partial y_{ab}}{\partial w_{ij}}$ potřebujeme všechny členy $\frac{\partial y_{ab}}{\partial \xi}$ z vrstvy vyšší. Tyto členy si můžeme pro každou vrstvu pamatovat v poli. Poté již snadno dopočítáme $\frac{\partial y}{\partial w_{ij}}$ pro každou váhu a mezi dvěma vrstvami pomocí.

$$\sum_k \frac{\partial y_{ab}}{\partial \xi_k} w_{jk} f'(\xi_j) y_i$$

Budeme tedy iterovat přez všechna k. Při této iteraci spočítáme i členy $\frac{\partial y_{ab}}{\partial \xi}$ z naší vrstvy, protože

$$\frac{\partial y_{ab}}{\partial \xi_j} = \sum_k \frac{\partial y_{ab}}{\partial \xi_k} w_{jk} f'(\xi_j)$$

a uložíme je do pole pro tuto vrstvu. Takto postupujeme až do nejnižší vrstvy.

Protože však některá spojení váhy sdílejí, musíme vyřešit jaká je derivace podle této sdílené váhy. Naštěstí jsou sdílené (tj. stejné) váhy vždy pouze ve stejné vrstvě. Nechť je váha w sdílena spojením $i-j$ a $r-s$ (spojením z i-tého neuronu z nižší vrstvy do j-tého vrstvy vyšší).

$$\begin{aligned}\frac{\partial y}{\partial w} &= \sum_k \frac{\partial y}{\partial \xi_k}\frac{\partial \xi_k}{\partial x} \\ &= \sum_k \frac{\partial y_{ab}}{\partial \xi_k} f'(\xi_j) y_i + \sum_k \frac{\partial y_{ab}}{\partial \xi_k} f'(\xi_r) y_s\end{aligned}$$

Tedy stačí spočítat derivace jako by se jednalo a nesdílené váhy, a poté váhy jednoduše sečíst (takovýto vztah by neplatil, pokud by váhy byli sdíleny i napříč vrstvami). V naší knihovně každé váze přiřadíme kategorii, a poté jednoduše stejné kategorie sečteme (takže sdílené váhy mají stejnou kategorii).

5.2 Použití knihovny

Snažili jsme se o co nejpřímočařejší a nejsnadnější použití modelu *KNS*. Je navíc důležité aby práce s tímto modelem byla co nejvíce podobná práci s modely, které již *Encog* obsahoval. Ukažme si nejprve příklad z dokumentace pro vytvoření vrstevnaté neuronové sítě na rozpoznání funkce *XOR*, uvedený v [12]. Tento kód vytvoří vrstevnatou neuronovou síť s 1 skrytou vrstvou obsahující 6neuronů. Vstupní vrstva bude mít neurony 2, výstupní 1 (síť se skládá směrem zdola nahoru).

```
BasicNetwork network = new BasicNetwork();

//Vstupni vrstva
network.AddLayer(new BasicLayer(
                new ActivationSigmoid(), true, 2));
//Skryta vrstva
network.AddLayer(new BasicLayer(
                new ActivationSigmoid(), true, 6));
//Vystupni vrstva
network.AddLayer(new BasicLayer(
                new ActivationSigmoid(), true, 1));
network.Structure.FinalizeStructure();
network.Reset();
```

A nyní se podívejme na vytvoření jednoduché *KNS* pomocí již upravené knihovny

```
//Vytvorime mapy charakteristik (okenko 2x2)
ConvolutionalLayer.FeatureMap featureMap1 =
                new ConvolutionalLayer.FeatureMap(2);
/A je propojena s nultou a prvni mapou z predchozi vrstvy
featureMap1.ConnectedTo = new List<int>() { 0, 1};

ConvolutionalLayer.FeatureMap featureMap2 =
                new ConvolutionalLayer.FeatureMap(2);
featureMap2.ConnectedTo = new List<int>() { 0, 1};
ConvolutionalLayer.FeatureMap[] featureMapList =
        new ConvolutionalLayer.FeatureMap[]
        {featureMap1, featureMap2};

ConvolutionalNetwork network = new ConvolutionalNetwork();
//Vstupni vrstva
network.AddLayer(new BasicLayer(
        new ActivationSigmoid(), false, 12 * 12));
//Nasleduje konvolucni, 2 mapy s okenkem 5x5
network.AddLayer(new ConvolutionalLayer(
        new ActivationSigmoid(), 2, 5));
//Subsamplingova vrstva
network.AddLayer(new SubsamplingLayer(
        new ActivationSigmoid(), 2));
network.AddLayer(new ConvolutionalLayer(
        new ActivationSigmoid(), featureMapList));
network.AddLayer(new SubsamplingLayer(
        new ActivationSigmoid(), 2));
//Skryta vrstva plnych spojeni, 10 neuronu
network.AddLayer(new FullyConnectedLayer(
        new ActivationSigmoid(), 10));
//Vystupni vrstva plnych spojeni, 4 neurony
network.AddLayer(new FullyConnectedLayer(
        new ActivationSigmoid(), 4));
network.Logic = new FeedforwardLogic();
network.Structure.FinalizeStructure();
network.Reset();
```

Vstupní vrstva obsahuje 12×12 neuronů. Následuje konvoluční vrstva se 2 příznakovými mapami (*feature map*) s okénkem 5×5. Dále subsamplingová vrstva s okénkem o velikosti 2×2, konvoluční s 2 příznakovými mapami (napojenými na obě příznakové mapy předchozí subsamplingové vrstvy). Potom ještě jedna subsamplingová vrstva a nakonec 2 vrstvy plných spojeních.

Podobně si uživatel může vytvořit libovolnou *KNS* dle svých představ. Síť může naučit stejným způsobem jako zdokumentovaný příklad *XOR* a nebudeme zde zdrojový kód ukazovat. Vstup sítě je však vhodné obdržet nějakým rozumným způsobem. V případě *XOR*-u je ještě snadné trénovací množinu ručně vypsat do kódu, pro obrázky však potřebujeme jiný přístup. Součástí rozšířené knihovny je také sada tříd, umožnující snadný převod bitmapových obrázků na vstup *KNS*

Kapitola 6

Ukázková aplikace

Rozhodli jsem se pro předvedení schopností KNS vytvořit aplikaci využívající naší upravené knihovny *Encog*. Inspirací byla domovská stránka projektu [8]. Na ní se nachází existující aplikace předvádějící některé aplikace této knihovny.

6.1 Existující ukázková aplikace *Encog*-u

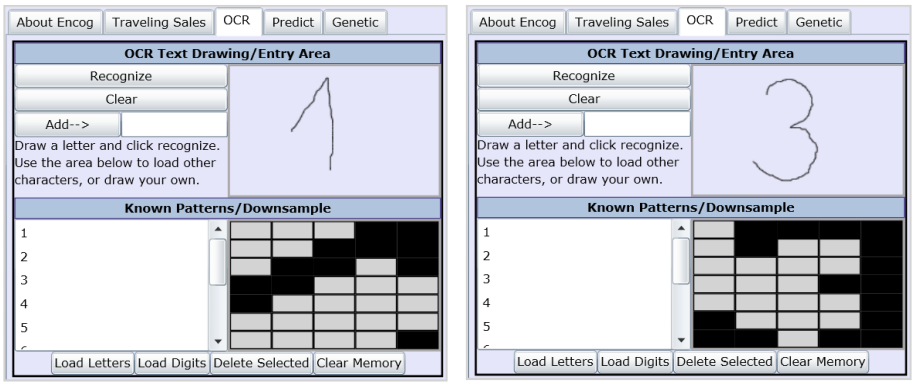
Na domovské stránce projektu *Encog* se nachází program [3], kde si uživatel může vyzkoušet některé aplikace NS. Tato aplikace je napsána pomocí technologie emphSilverlight, a tedy si ji uživatel může spustit přímo ve svém prohlížeči. Na několik těchto funkčních příkladů se podíváme blíže.

6.1.1 Predikce grafu

Je dána lomená čára procházející N body a aplikace se snaží odhadnout, jak bude tento graf dále pokračovat. V tomto případě se NS běžně učí tak, že vstupem je vždy k po sobě jdoucích bodů, a požadovaným výstupem bod následující. Tento příklad funguje v pořádku.

6.1.2 Rozpoznání cifer

Pro nás jistě zajímavější je příklad pro rozpoznávání cifer z obrázku. Tento obrázek si může uživatel přímo nakreslit do určené oblasti. Poté se program pokusí rozpoznat, o kterou cifru se jedná. Modelem NS jsou zvoleny asociativní sítě. Použití na OCR není však zvoleno příliš vhodně. Dle našich zkušeností nedosahuje o mnoho lepších výsledků než by se dosáhlo prostým náhodným číslem. Můžeme si také povšimnout, že se provádí zmenšení rozlišení obrázku na velikost 5x5 pixelů.



Obrázek 6.1: Oba tyto symboly byli špatně rozpoznány, první jako číslo čtyři, druhý jako devět

6.2 Požadavky ukázkové aplikace

Naše ukázková aplikace by měla fungovat pro rozpoznávání cifer lépe. Dále by měla umožňovat uživateli snadno se naučit jeho vlastní symboly. Tj. nakreslit do aplikace např. několikrát křížek, kružnici a písmeno F a po naučení je rozpoznávat.

Základní požadavky na aplikaci

- možnost uživatelského učení
- ukládání a načítání uživatelem učených sítí

6.2.1 Technologie aplikace

Tato aplikace musí využívat upravenou knihovnu Encog napsanou v jazyce C#. To nám výběr značně zužuje. Pro již zmíněnou aplikaci na stránkách Encog-u je zvolen Silverlight. Není to však jediná varianta. Máme v podstatě tři možnosti, jakým směrem se vydat

1. Klientská aplikace
2. Silverlight
3. Dynamická webová aplikace

Po zvolené technologii požadujeme, aby naší aplikaci poskytla:

- Rychlost. Protože učení je výpočetně náročné, a aplikace bude umožňovat jednotlivým uživatelům učení.
- Dostupnost. Tím máme na mysli, aby si pokud možno kdokoliv mohl na webových stránkách aplikaci vyzkoušet bez nutnosti její instalace.

Následuje bližší přiblížení těchto technologií.

1) Klientská aplikace Klasická aplikace ať již za použití WinForms či WPF

Klady

- Rychlost

Zápory

- nutnost instalace
- platformní závislost

2) Dynamická webová stránka Samotné jádro aplikace by běželo na serveru. Data by se posílala z prohlížeče ne server a zpět.

Klady

- platformní nezávislost
- uživateli stačí běžný prohlížeč bez dalších rozšíření

Zápory

- příliš velké zatížení serveru (především při používání aplikace více uživateli zároveň)

3) Silverlight Samotné jádro aplikace by běželo na serveru. Data by se posílala z prohlížeče ne server a zpět.

Klady

- běh v prohlížeči
- platformní nezávislost

Zápory

- nutnost mít nainstalovaný Silverlight

Závěr Z těchto kandidátů jsme se nakonec rozhodli vybrat Silverlight (jak již bylo zmíněno, je použit i při aplikace na stránkách projektu *Encog* [8]). Je rozumným kompromisem mezi rychlostí (pomalejší než klientská aplikace, rychlejší než webová stránka) a dostupností (méně dostupný než webová stránka, více než klientská aplikace).

6.3 Naše ukázkové aplikace

Hotová aplikace se nachází na webové adrese
`http://www.ms.mff.cuni.cz/~schmidm/bakalarka/`.
a kdokoliv si ji tedy může vyzkoušet.

6.3.1 Popis a ovládání ukázkové aplikace

Učení vlastních symbolů

Tato volba umožňuje uživateli nakreslit přímo do aplikace několik vzorových symbolů a poté je rozpoznávat. Vše si ukážeme pro případ, kdy chceme rozpoznávat čtyři symboly, a to: křížky, písmeno A, písmeno B a číslici 9. Po zvolení této možnosti je funkcionalita rozdělena do několika záložek. Po správném vyplnění každé záložky může uživatel postoupit na záložku další. Tento přístup je zvolen z důvodu přehlednosti a jednoduchosti - uživatel se vždy soustředí na jednu konkrétní věc.

Všechny záložky si postupně popíšeme a to v pořadí, v jakém jsou zapotřebí použít.

1) Nastavení Zde uživatel nastaví základní parametry sítě, která jsou dále v programu použita pro učení a rozpoznávání jeho symbolů. Nejprve je nutno nastavit, kolik různých symbolů chceme rozlišovat. Toto nastavení ovlivňuje poslední, výstupní vrstvu KNS. Počet neuronů této vrstvy je roven nastavené hodnotě. V našem případě nastavíme čtyři symboly. Dále si uživatel může zvolit koeficient učení a požadovanou hodnotu chybové funkce. Tyto parametry jsou již nastaveny na výchozí hodnotu.

2) Kreslení vzorů a učení V této záložce je k dispozici čtvercová oblast pro kreslení symbolů. Dále se zde zobrazí několik řádků, jeden pro každou skupinu symbolů (tedy nám se zobrazí 4 řádky). Tyto řádky se postupně plní nakreslenými obrázky získanými z čtvercové oblasti. Jedna skupina/řádek je vždy aktivní a je od ostatních graficky odlišena. Do této aktivní skupiny přidáváme vzorové obrázky.
Nakreslíme symbol do určené oblasti a poté ho pomocí tlačítka "Finish" přidáme do právě aktivní skupiny. Po přidání vzoru kreslíme dále, přičemž aktivní skupina zůstává nezměněna. V našem příkladě nejprve nakreslíme několik vzorových křížků. Jako minimální množství se ukázalo čtyři až pět vzorů.
Jakmile jsme dokončili kreslení křížků, je na čase změnit aktivní skupinu. Toho docílíme například stisknutím tlačítka "Add" v příslušné skupině. Obrázky do této skupiny přidáme stejným způsobem, jako křížky. Všechny nakreslené

symboly se zobrazují v náhledu v příslušných skupinách. Nepovedené máme možnost odstranit tlačítkem "X" umístěným pod každým náhledem.

3) Učení Ve stejné záložce, jako jsme kreslili vzory, probíhá i samotné učení. To z toho důvodu, aby bylo možné snadno graficky zobrazovat jeho průběh. U každého z náhledů je pozadí více či méně barevné, a to v závislosti na chybě tohoto vzoru. Toto však není jediná kontrola průběhu učení - zobrazuje se také počet iterací a celková chyba. Učení spustíme stisknutím "Learn"

Obrázek 6.2: Učení nakreslených vzorů

4) Rozpoznání Tato záložka slouží k rozpoznání nově nakresleného obrázku pomocí naučené sítě. Oblast určená pro kreslení je shodná jako ta na předchozí záložce. Nakreslíme symbol a stisknutím "Recall" se ho *KNS* pokusí rozpoznat a výsledný symbol je uživateli zobrazen.

5) Ukládání a načítání Naučenou síť máme možnost si uložit pro pozdější použití (rozpoznávání vzorů). Ukládá se jak nastavení vah sítě, tak i pojmenování jednotlivých skupin. Soubor držící tyto informace je uložen na disk uživatele, odkud si ho poté může také načíst. Obě tyto operace nalezneme v záložce "Saving and loading", kde se nacházejí tlačítka "Save" a "Load".

6.4 Použitá konvoluční neuronová síť

Popišme si nyní *KNS*, kterou jsme použili na učení rozpoznávání uživatelem kreslených symbolů. Tuto síť budeme dále nazývat pouze *EX-KNS*.

Vstupní vrstva I Vstupní vrstva má rozměr 12×12 neuronů a tedy vstupem je obrázek o stejných rozměrech.

Konvoluční vrstva C1 První konvoluční vrstva. Obsahuje 6 příznakových map s okénkami 5×5. Tedy každá tato příznaková mapa má obsahuje 64 neuronů ($12 - 5 + 1$ je šířka příznakové mapy)

Subsamplingová vrstva S1 První subsamplingová vrstva. Obsahuje 6 příznakových map s okénkami 2×2. Každá tato příznaková mapa je napojena na právě jednu příznakovou mapu z vrstvy *C1*. Počet neuronů v každé příznakové mapě je 4×4, tedy 16

Konvoluční vrstva C2 Druhá konvoluční vrstva. Obsahuje celkem 12 příznakových map, každá s okénkem s rozměry 3×3. Každá tato příznaková mapa je napojena na různou podmnožinu příznakových map vrstvy *S1*. Příznakové mapy z *C2* jsou napojeny několik příznakových map zároveň, aby bylo možné rozeznávat kombinace více příznaků. Tj. například pokud by byla $C2_1$ napojena na $S1_1, S1_2, S1_3$, může $C2_1$ rozeznávat kombinaci příznaků z těchto příznakových map a tím vytvořit příznak nový. Mohli by jsme vytvořit spojení s všemi příznakovými mapami *S1*. Takto však snížíme již tak velké množství spojení a navíc donutíme příznakové mapy *C2* rozeznávat různé příznaky. Následující tabulka zobrazuje napojení příznakových map
Počet neuronů v každé příznakové mapě je 2×2, tedy 4

1	2	3	4	5	6	7
1, 2, 3	2, 3, 4	3, 4, 5	4, 5, 6	1, 2, 3, 4	2, 3, 4, 5	3, 4, 5, 6
8	9	10	11	12		
1, 2, 3, 4, 5	2, 3, 4, 5, 6	1, 3, 5	2, 4, 6	1, 2, 3, 4, 5		

Obrázek 6.3: Napojení příznakových map *C2* na S1

Subsamplingová vrstva S2 Druhá subsamplingová vrstva. Obsahuje 12 příznakových map s okénky 2×2. Každá tato příznaková mapa je napojena na právě jednu příznakovou mapu z vrstvy *C2*.

Celkem 12 příznakových map. Každá příznaková mapa obsahuje již pouze jeden neuron.

Skrytá vrstva F1 Vrstva obsahující plná spojení. 10 neuronů.

Skrytá vrstva F2 Vrstva obsahující plná spojení. Počet neuronů je roven uživatelem zvolenému počtu rozpoznávaných symbolů

sectionTrénovací množina Trénovací množina je vytvořena z uživatelem nakreslených symbolů do určené oblasti v ukázkové aplikaci. Vstup je poté zmenšen na velikost odpovídající vstupu *EX-KNS*. Poté je proveden jednoduchý převod obrázku na čísla vstupního vektoru. Bíle pixely jsou reprezentovány 0, černé 1.
Jak jsme popsali výše, výstupní vrstva má stejný počet neuronů jako počet rozpoznávaných symbolů. Každému neuronu tedy přiřadíme jeden symbol a požadovaný výstup pro tento symbol bude vždy 1 na tomto neuronu a 0 jinde.

6.4.1 Použití knihovny pro vlastní data

Ukázkova aplikace má několik omezení. Pracuje pouze s černobílými obrázky o rozlišení 12×12 pixelů. Dále není vhodná pro zpracování velkého množství dat (uživatel je zadává ručně). Pokud chce uživatel využít knihovnu pro vlastní data, musí si napsat vlastní aplikaci nad touto knihovnou (podobně jako jsme napsali naši ukázkovou aplikaci). Uživatel může zvolit libovolnou technologii, musí být však schopna využít naší knihovnu napsanou v jazyce C#. Při tvorbě této aplikace musí uživatel vyřešit několik základních problémů.

Struktura neuronové sítě Na základě typu, velikosti a množství dat se uživatel musí zamyslet nad nejvhodnější strukturou sítě. Počet neuronů první vrstvy je určen velikostí dat/obrázků, ale například počet map charakteristik či výšku sítě musí zvolit dle svého uvážení. Je samozřejmě možné postupovat metodou pokus-omyl a vybrat síť, kterou uživatel nakonec uznal za nejvhodnější.

Načtení vstupu pro síť V naší ukázkové aplikaci načítáme vstup z kreslící oblasti. Uživatel bude potřebovat ze svých dat získat vstupní vektor sítě. Pokud se jedná o soubory obsahující bitmapové obrázky, je získání tohoto vektoru přímočaré (a knihovna poskytuje některé třídy usnadňující tuto činnost). Pokud však data pocházejí z jiného zdroje (internet, scan, kamera atd.), musí uživatel vyřešit tento převod sám. Je samozřejmě možné využít *konvoluční neuronové sítě* i na prací s jiným typem dat než s obrázky

Učení Koeficient učení stejně tak jako požadovanou chybu si uživatel musí zvolit sám. Pokud je množství dat velké, může učení trvat i několik dní. Pro tyto případy knihovna umožňuje učení kdykoliv přerušit a síť spolu s nastavenými vahami uložit. Poté může uživatel v učení kdykoliv pokračovat.

Závěr

V této práci jsme představili model konvolučních neuronových sítí. Popsali jsme blíže několik nejrozšířenějších knihoven pro práci s neuronovými sítěmi a porovnali jsme rychlost těchto knihoven. Protože však ani jedna z těchto knihoven nepodporuje model konvolučních *NS* rozšířili jsme jednu z nich (*Encog*) o tento model a popsali implementaci. Dále jsme vytvořili ukázkovou aplikaci využívající naší upravené knihovny. Tato aplikace ukazuje využití *KNS* na rozpoznávání symbolů, které umožňuje uživateli nakreslit přímo v prohlížeči. Ukázali jsme na ní robustnost tohoto modelu proti různým deformacím, což ho činí velmi vhodným při aplikaci na tento typ dat. Knihovnu lze využít v dalších aplikacích a nastínili jsme jak toho docílit.

KNS najdou svoje uplatnění především v oblastech, kde vrstevnaté sítě nedosahují dostatečné robustnosti. Věříme, že ukázková aplikace zvyší povědomí o tomto modelu a jeho vlastnostech. Pro následnou aplikaci tohoto modelu ve vlastních programech poté mohou využít programátoři právě naší rozšířené knihovny.

Další práce

Dále by jsme rádi aplikovali tento model na rozpoznávání barevných obrázků/fotografií či celých slov. Velmi zajímavá je také možnost aplikace na jiný typ dat než jsou obrázky, jako třeba video či zvuk. Konvoluční sítě jsou vhodné i na tato data [2].

Vhodné by bylo také doplnit podporu více učících algoritmů než jen zpětné šíření - *Encog* jich obsahuje několik, ale je potřeba zdrojový mírně upravit aby fungovali na tomto modelu.

Literatura

[1] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, v. 86, pp. 2278-2324, 1998.

[2] Somsak Sukittanon, Arun C. Surendran, John C. Platt, Chris J. C. Burges. CONVOLUTIONAL NETWORKS FOR SPEECH DETECTION [online]. `http://research.microsoft.com/pubs/68933/convnet_speechdetect.pdf` [cit. 2011-04-10]

[3] Steffen Nissen. FANN homepage [online]. `http://leenissen.dk/fann/` [cit. 2011-02-22]

[4] Neuroph homepage [online]. `http://neuroph.sourceforge.net/` [cit. 2011-02-22]

[5] Jeff Heaton. Encog homepage [online]. `http://www.heatonresearch.com/` [cit. 2011-02-22]

[6] Matlab homepage [online]. `http://www.mathworks.com/products/matlab/` [cit. 2011-02-22]

[7] Jeff Heaton. Encog sample application [online]. `http://www.heatonresearch.com/encog/benchmark.html/` [cit. 2011-02-22]

[8] Steffen Nissen FANN advanced usage [online]. `http://leenissen.dk/fann/html/files2/advancedusage-txt.html` [cit. 2011-02-22]

[9] Neuoph feature list [online] `http://neuroph.sourceforge.net/features.html` [cit. 2011-02-22]

[10] Google search engine [online]. `http://www.google.com/` [cit. 2011-01-10]

[11] MATLAB Compiler funcionality support [online].
http://www.mathworks.com/products/compiler/compiler_support.html
[cit. 2011-02-28]

[12] Jeff Heaton Encog introduction [online].
http://www.heatonresearch.com/dload/ebook/IntroductionToEncogCS.pdf
[cit. 2011-02-22]