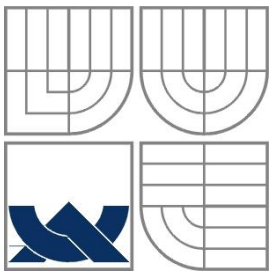
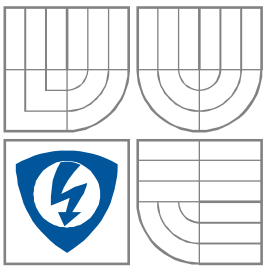


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ROZPOZNÁVÁNÍ ČÍSLIC
NUMBER RECOGNITION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Martin Gorgol
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Václav Jirsík, CSc.
SUPERVISOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Martin Gorgol	***ID:***	78177
Ročník:	3	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Rozpoznávání číslic

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s problematikou rozpoznávání číslic pomocí neuronové sítě.
2. Zvolte vhodnou sadu příznaků pro popis číslic.
3. Vyberte umělou neuronovou síť pro rozpoznávání číslic.
4. Proveďte rozbor ohledně počtu a jednotlivých typů příznaků pro rozpoznávání číslic umělou neuronovou sítí.
5. Dosažené výsledky zhodnoťte.

DOPORUČENÁ LITERATURA:

Šíma J., Neruda R.: Teoretické otázky neuronových sítí. Matfyzpress, Praha 1996
Duda R., O., Hart P., E., Stork, D., G.: Pattern Classification. John Wiley & Sons, New York, USA, 2001

Termín zadání:	7.2.2011	***Termín odevzdání:***	30.5.2011

Vedoucí práce: doc. Ing. Václav Jirsík, CSc.

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Tato práce popisuje základní pojmy a principy v oboru neuronových sítí. Blíže se pak zabývá problematikou rozpoznávání číslic pomocí těchto sítí, konkrétně pak pomocí metody back-propagation. Je zde rozebrán postup při volbě sady příznaků, typů příznaků a volbě topologie neuronové sítě. Cílem je získání konkrétních výsledků pomocí programu pro práci s neuronovými sítěmi.

Klíčová slova

Neuron, perceptron, Neuronová síť, back-propagation, rozpoznávání číslic, učení, tréninkový vzor, testovací vzor, extrakce příznaků

Abstract

This work describes the basic concepts and principles in the field of neural networks. Closer then this work deals with the identification numbers using these networks, in particular, using the back-propagation method. There is a broken process of choosing a set of signs, types of symptoms and of choosing a neural network topology. The aim is to obtain specific results by using the program for working with neural networks.

Keywords

Neuron, perceptron, neural network, back-propagation, number recognition, learning, training pattern, testing pattern, extraction of symptoms

Bibliografická citace:

GORGOL, M. Rozpoznávání číslic. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 40 s. Vedoucí bakalářské práce doc. Ing. Václav Jirsík, CSc..

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma Rozpoznávání číslic jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **30. května 2011** ……

podpis autora

Obsah

1 ÚVOD

Klasifikace (rozpoznávaní) číslic je problém při kterém se snažíme identifikovat předložený obrázek číslice, to znamená zařadit obrázek do příslušné klasifikační třídy. Využívají se k tomu různé metody, já jsem se zaměřil na klasifikaci pomocí dopředných neuronových sítí, konkrétně na metodu back-propagation.

Tato práce je inspirována několika různými pracemi podobného zaměření, na základě poznatků získaných z těchto prací jsem postupoval při řešení dílčích problémů souvisejících s problematikou klasifikace číslic. Výtah důležitých poznatků jsem rovněž v práci uvedl.

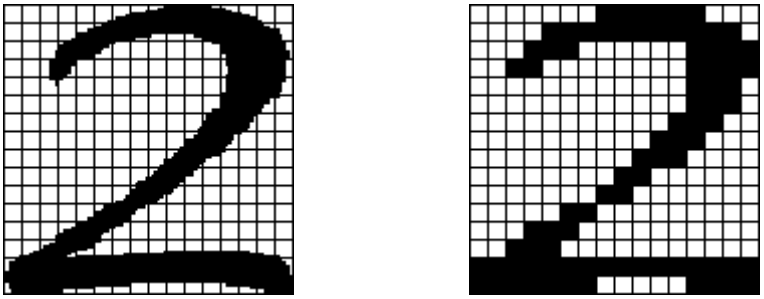
Nejdříve jsem se zabýval volbou vhodných sad vzorů. To obnášelo jak vytvoření obrázků číslic, tak vytvoření tréninkových a testovacích množin.

Na základě rešerže ohledně počtu a typů příznaků používaných pro klasifikaci číslic jsem se zaměřil na příznaky typu projekce (*Proj*) a posuvné okno (*Win*). Pak jsem se ale potýkal s problémem při extrakci příznaků. Jelikož jsem nesehnal vhodnou aplikaci, která by mi s tím ulehčila práci, vytvořil jsem si vlastní program v jazyce Java.

Nyní již jsem měl vytvořeny základy pro tvorbu samotné neuronové sítě. Použil jsem k tomu program *BPSIM*. Pro jednotlivé typy příznaků jsem pak simuloval různé topologie sítí a prováděl jejich vyhodnocení. Dosažené výsledky jsem zhodnotil v závěru.

2 EXTRAKCE PŘÍZNAKŮ

Pro popis binárních obrázku je potřeba zvolit pevnou množinu měřených veličin, kterou nazýváme *příznaky*. Hodnoty příznaků se získají z předloženého normalizovaného obrázku číslice pomocí *jasové funkce*, ta přiřadí jednotlivým pixelům hodnotu „0“ reprezentuje-li obrazový bod většinově bílou barvu podkladu nebo „1“ reprezentuje-li obrazový bod většinově barvu podkladu černou.



a) *b)*

Obr. 2.1) Příklad reprezentace číslice 2 v pravoúhlém rastru:
a) před aplikací jasové funkce
b) po aplikaci jasové funkce

Jestliže příznaky zapíšeme do posloupnosti tak vytvoří tzv. *příznakový vektor*. Počet složek tohoto vektoru určuje *dimenzi n příznakového prostoru P*.

Jestli zvolíme příliš malou množinu příznaků, může se stát, že nebude dostatečná pro dobré odlišení rozpoznávaných číslic. Postupným přidáváním příznaků lze obvykle dosáhnout lepšího oddělení.

Je důležité vybrat množinu příznaků, které daný objekt co nejlépe popisují a přitom ho dobře odlišují od ostatních. Preferují se příznaky odolné vůči šumu nebo očekávaným prostorovým transformacím. Také je výhodné jdou-li rychle vypočíst.

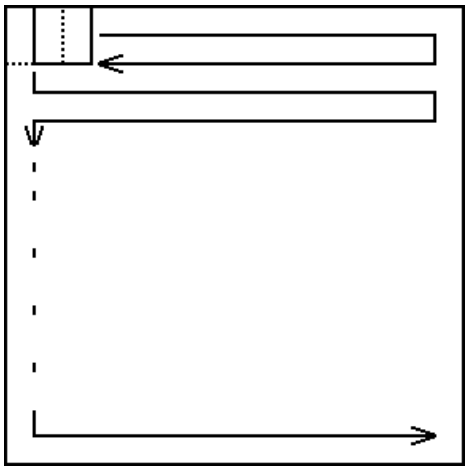
V ideálním případě by neměla hodnota jednoho příznaku záviset na hodnotě jiného. Avšak u metod zpracovávajících příznakový vektor sekvenčně, jako jsou Markovovy modely se závislost dvou po sobě jdoucích pozorování přímo předpokládá.

Normalizace příznaků je velice důležitá a ovlivňuje pozdější rozpoznávání. Rozsahy hodnot jednotlivých složek příznakového vektoru by měly být relativně stejné. Složky příznaku, které mají vysokou hodnotu, se většinou klasifikátor učí rychleji a učení ostatních položek zanedbává. Příslušným škálováním vstupu se tohoto nepříjemného efektu zbavíme.

Existuje několik typů příznaků, uvádím některé z nich:

1) *Jednotlivé pixely* – Samotné hodnoty jednotlivých pixelů zapsané po řádcích mohou tvořit binární příznakový vektor. V tomto případě je dimenze *n* rovna velikosti normalizačního čtverce (např. 16 x 16 = 256), takže je velmi velká [5].

2) *Posuvné okno* – Tento příznakový vektor poskytuje lepší odolnost vůči šumu. Obrázek je odshora postupně procházen oknem, jehož velikost musí být přizpůsoben velikosti normalizačního čtverce. To se děje směrem zleva doprava a zprava doleva tak, že se při dalším kroku vždy sousední pozice okna z 50% překrývají. Tím se zaručí zachování spojitosti měřených hodnot. V každé pozici se spočítá poměr nenulových pixelů v okně k ploše okna. Např. má li okno velikost 4 x 4 pixelů, hodnot v příznakovém vektoru (různých pozic) pak je 36 [5].



Obr. 2.2) Příklad postupu získávání příznakového vektoru typu Posuvné okno, tečkovaně je vyznačena první pozice okna, plnou čarou následující.

3) *Projekce* – Horizontální a vertikální projekce normalizovaného znaku jsou zapsány do jednoho vektoru za sebe (např. 16 + 16 = 32 hodnot). Hodnoty projekcí se pak dělí šířkou (resp. výškou) obrázku [5].

4) *Normalizované centrální momenty* – Jedná se o sofistikovanější typ příznaku. *Momenty* popisují obrázek jako celek, nebo jsou nasazeny lokálně pro různé části obrázku zvlášť [5]. Pro jejich výpočet platí vzorec:

$$m_{ij} = \sum_x \sum_y x^i x^j I(x,y) \; pro \; i, j \geq 0, \tag{2.1}$$

kde *x (y)* jde přes celou šířku (výšku) obrázku i *I(x, y)* značí hodnotu pixelu v daném bodě. Součet indexů *(i + j)* se nazývá řád nebo stupeň momentu.

Aby momenty nepodléhaly změně vůči běžným prostorovým transformacím, používají se tzv. *normalizované centrální momenty:*

$$\tilde{m}_{ij} = \frac{\sum_x \sum_y (x - x_t)^i (y - y_t)^j I(x, y)}{m_{00}^{\frac{i+j}{2}+1}}, \tag{2.2}$$

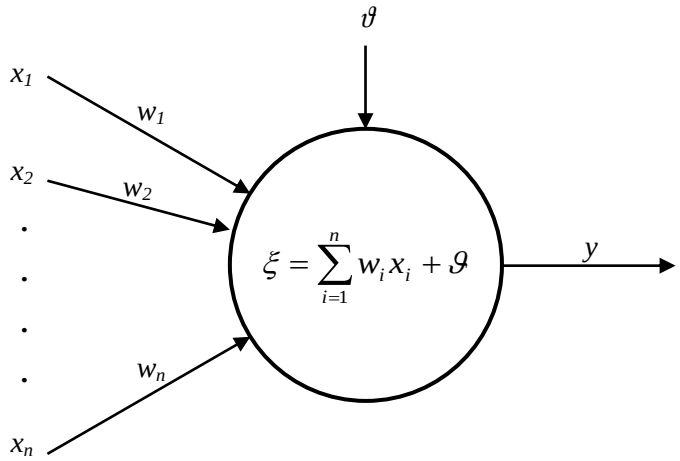
kde $x_t = \frac{m_{10}}{m_{00}}$ a $y_t = \frac{m_{01}}{m_{00}}$ reprezentují těžiště objektu na obrázku. Díky x_t a y_t jsou momenty $\tilde{m}_{ij}$ nezávislé na posunutí obrázku. Celý výraz se dělí mocninou momentu m_{00} (nazývanou plochou objektu).

Momenty tvoří dobrou charakteristiku binárního obrázku, ale s rostoucím stupněm *(i + j)* jsou stále více ovlivňovány šumem. V případě normalizovaných centrálních momentů se prvky $\tilde{m}_{00}$, $\tilde{m}_{10}$ a $\tilde{m}_{01}$ vynechávají, protože jejich hodnota je pro všechny obrázky přibližně stejná [5].

3 NEURONOVÉ SÍTĚ

3.1 Model neuronu

Matematický model neuronu je analogií skutečného biologického neuronu. Nervová buňka, čili n*euron (perceptron)* je základní stavební jednotkou každého složitějšího nervového systému. Neurony se specializují na přenos, zpracovávání a uchovávání informací [4].



Obr. 2.1) Model formálního neutronu s jedním výstupem y, n vstupy x_1, ..., x_n, váhami w_1, ..., w_n a prahem ϑ

Vzájemným spojením neuronů vzniká *neuronová síť*.

Formální neuron byl navržen roku 1943. každá taková jednotka má jen jeden výstup ale několik vstupů. Samotná provádí jen velmi jednoduchou operaci. Od váženého součtu svých vstupů odečte prahovou hodnotu. Tento mezivýsledek se nazývá *potenciál neuronu* a značí se ξ:

$$\xi = \sum_{i=1}^{n} w_i x_i + \vartheta , \tag{3.1}$$

kde w_1, ..., w_n jsou váhy pro jednotlivé vstupní hodnoty x_1, ..., x_n a ϑ je tzv práh neuronu. Výstup neuronu je dán vztahem:

$$y = f(\xi) = f(\sum_{i=1}^{n} w_i x_i + \vartheta), \tag{3.2}$$

kde f je tzv. *přenosová funkce (*nebo také *aktivační funkce)*. Existují různé typy přenosových funkcí. Mezi nejznámější patří následující:

1) *Lineární funkce (Identita)* – Tato funkce nechává potenciál neuronu nezměněn. Výstupem neuronu tedy může být libovolné reálné číslo:

$$f(\xi) = \xi . \tag{3.3}$$

2) *Skoková přenosová funkce* – Provádí velmi jednoduché prahování hodnot. Z matematického hlediska však nemá moc dobré vlastnosti. V bodě 0 není spojitá a má pouze jednostranné derivace což ji činí nepoužitelnou v dopředných neuronových sítích, ale zato se dá s výhodou použít v jednoduchém modelu neuronu. Její tvar je:

$$f(\xi) = \begin{cases} 1 & \text{pro } \xi > 0 \\ 0 & \text{jinak} \end{cases}. \tag{3.4}$$

3) *Logická sigmoida* – Je to asi nejpoužívanější funkce. V podstatě se jedná o o vyhlazení skokové funkce v okolí bodu 0. Její hodnoty se pohybují v intervalu <0, 1>, je spojitá, hladká a nelineární:

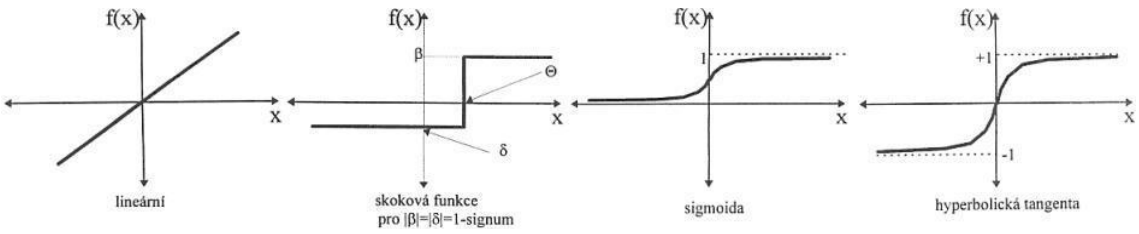
$$f(\xi) = \frac{1}{1+e^{-\xi}}. \tag{3.5}$$

Její derivace se dá vyjádřit pomocí funkce samotné:

$$f'(x) = f(x)(1-f(x)). \tag{3.6}$$

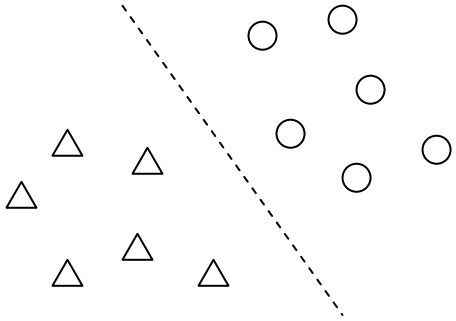
4) *Hyperbolický tangens* – Tato funkce je podobná logické sigmoidě, její obor hodnot se však pohybuje v intervalu <-1, 1>,:

$$f(\xi) = \frac{1+e^{-\xi}}{1+e^{-\xi}}. \tag{3.7}$$



Obr. 2.2) Průběhy jednotlivých přenosových funkcí [8]

Lze dokázat, že jediný neuron je schopen tzv. *lineární klasifikace* dvou množin (separace dvou množin pomocí jedné nadroviny) [1]. Podmínkou je *lineární separabilita* těchto množin. To znamená, že se dá příznakový prostor rozdělit nadrovinou na dvě části tak, aby v jedné části byly pouze vzory z první množiny a v části druhé jen vzory z druhé množiny.



Obr. 2.3) Příklad lineární separability dvou množin v příznakovém prostoru

3.2 Modely neuronových sítí

Vzhledem k sekvenčnímu zpracovávání dat procesorem, bylo nutné přistupovat takto i k neuronovým sítím. Byly vytvořeny modely, které maximálně zohledňují zmíněnou problematiku [6].

Příklady základních modelů:

- *model Sutton-Barto, model Hebb* – Tyto modely považují statický pohled na problematiku za nedostatečný. Jsou inspirovány Darwinovskou představou sobeckého chování základního elementu, který se snaží maximálně využít své vstupy podle svého měřítka hodnot. Byly navrženy tak, aby pro dostatečný počet neuronů byly schopny ovlivňovat vnitřní parametry, respektive že se neurony mohou upravit sami podle opakovaných vstupních podnětů a stát se stabilní. Učení nastává tehdy, když jsou neurony stabilizovány [6].

- *model Hopfield, model Bolzmann* – Topologie sítí je naprosto homogenní, každý neuron je spojen se všemi ostatními, spojení jsou symetrická. Hopfieldova síť používá dvě různé binární prahové jednotky (-1,1 a 0,1), takže existuji dvě možnosti její aktivace. Je vhodná pro data s binární reprezentací např. černo-bílé obrazy. Často se používá jako asociativní paměť nebo pro řešení optimalizačních problémů. Síť se velice rychle a snadno adaptuje – vytváří si tzv. stabilní stavy podle tréninkových vzorů. S jistotou konverguje k lokálnímu minimu, ale nezaručuje konvergenci k jednomu z uložených vzorů. Bolzmanův stroj (model) definuje síť jako soustavu stochastických částic, které se snaží dosáhnout stavu s nejmenší energií. V tomto (definicí částic) se od Hopfieldova modelu liší. Jinak jsou si ale oba modely velmi podobné, oba uvažují jednotky s energií, výpočetní vzorce pro celkovou energii jsou stejné [6].

- *Hammingův model* – Tato síť je nejjednodušším příkladem kompetičního modelu. Ten se skládá se ze dvou vrstev neuronů: z vrstvy receptorů a z vrstvy

vstupních neuronů (selfish neurony). Výstupní neurony nemají fiktivní vstup. Výstup každého vstupního neuronu je propojen se všemi výstupními neurony. Jedná se o variantu s učitelem a binárním vstupem. Lze ji také popsat jako třídič dle nejmenší chyby: k danému vstupu najde kategorii, jejíž reprezentant má od vstupu nejmenší tzv. Hammingovu vzdálenost (tj počet odlišných vstupů). První vrstva sítě počítá Hammingovu vzdálenost (respektive doplněk), druhá laterální inhibicí vybírá maximum (správnou kategorii) [6].

- *Kohonenův model* – Kohonenova samo-organizující se síť je typ sítě, které při učení nepotřebují učitele. Je založena na algoritmu shlukové analýzy, tj. na schopnosti nalézt určité navzájem závislé vlastnosti přímo v překládaných tréninkových datech bez přítomnosti nějaké vnější informace. Algoritmus vytváří nízko-dimenzionální (obvykle dvou-dimenzionální) rozlišitelné reprezentace vstupních tréninkových sad, které se nazývají mapy. Každý neuron ve výstupní vrstvě je výstupem a jejich počet (výstupů) je tedy roven počtu neuronů. Učení v Kohonenově síti probíhá tak, že se učící algoritmus snaží uspořádat neurony v mřížce do určitých oblastí tak, aby byly schopny klasifikovat předložená vstupní data [6].

- *Feed-forward model* – Jedná se o model dopředné neuronové sítě, je to nerekurentní, nejjednodušší a vůbec první model neuronové sítě. Informace se pohybuje pouze jedním směrem, od vstupní vrstvy přes vrstvu skrytých neuronů na vrstvu výstupní. Síť neobsahuje žádné smyčky ani cykly. Podrobněji se této metodě, a její konkrétní variantě back-propagation, budu věnovat v následujících kapitolách [6].

3.3 Dopředná neuronová síť

Obecná neuronová síť se dá definovat jako orientovaný graf, ve kterém uzly představují neurony a orientované hrany jsou spoje mezi nimi. Každé hraně je přiřazena *váha w (vážené spoje)*. Dále existuje neprázdná množina vstupních neuronů [5].

Vrstevnatá dopředná neuronová síť pak klade následující omezení [5]:

- Množina neuronů (vrcholů) je rozdělena do posloupnosti vzájemně disjunktních podmnožin zvaných vrstvy. Indexují se písmenem a a jejich počet se označuje L. Počet neuronů v každé vrstvě může být jiný.

- Hrany grafů vedou vždy pouze z vrstvy $a - 1$ do vrstvy a takovým způsobem, že vytvářejí úplný biparitní podgraf. Váha na hraně vedoucí z i-tého neuronu ve

vrstvě $a - 1$ do j-tého neuronu vrstvy a se značí $w_{ij}^{(a)}$. Symbol $\vartheta_j^{(a)}$ představuje prahovou hodnotu j-tého neuronu ve vrstvě a.

- První vrstva ($a = 1$) se nazývá *vstupní*. Obsahuje speciální vstupní neurony s identickou přenosovou funkcí a prahovou hodnotou 0. Hodnota výstupu těchto neuronů je rovna jejich hodnotě vstupní. Tyto neurony nemají v grafu spojů žádné předchůdce. Jejich úkolem je propagace vstupu pro celou neuronovou síť ($x_1, ..., x_n$).

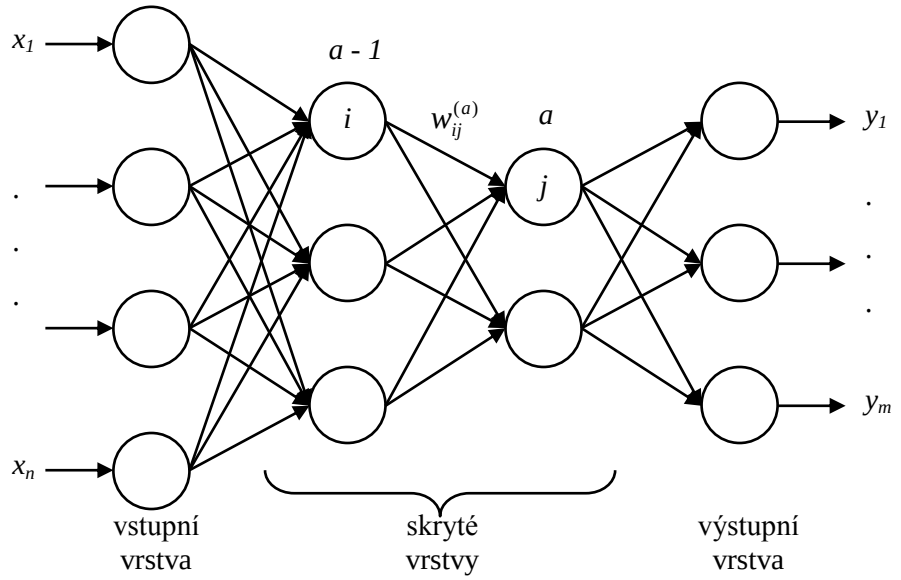
- Poslední vrstva ($a = L$) se nazývá *výstupní.* Neurony této vrstvy nemají v grafu spojů žádné následníky a z jejich výstupů je čten výstup celé neuronové sítě ($y_1, ..., y_m$).

- Ostatní vrstvy ($1 < a < L$) se nazývají *skryté*.

- Výstup j-tého neuronu ve vrstvě a se označuje $o_j^{(a)}$ a jeho potenciál $\xi_j^{(a)}$. Pro $a > 1$ se $o_j^{(a)}$ spočítá na základě výstupů všech neuronů z předchozí vrstvy $a - 1$ pomocí vzorce:

$$o_j^{(a)} = f(\xi_j^{(a)}) = f(\sum_i w_{ij}^{(a)} o_j^{(a-1)} + \vartheta_j^{(a)}). \tag{3.8}$$

- Pro vstupní vrstvu platí: $o_j^{(1)} = x_i$ a pro výstupní vrstvu platí: $o_j^{(L)} = y_i$



Obr. 2.4) Model dopředné neuronové sítě se čtyřmi vrstvami a architekturou 4-3-2-3 (čísla počtu neutronů ve vrstvách směrem od vstupní k výstupní)

Rozložení neuronů ve vrstvách charakterizuje tzv. architektura neuronové sítě.

Popis činnosti vrstevnaté neuronové sítě: Po předložení vzoru x se postupně směrem od vstupní k výstupní vrstvě počítají jejich výstupy. Příznakový vektor se pomocí skrytých vrstev vhodně nelineárně transformuje, aby poslední výstupní vrstva mohla provést lineární klasifikaci na transformovaném prostoru. Výstup poslední vrstvy představuje *skutečný výstup* neuronové sítě $y_1, ..., y_m$ [5].

Aby bylo možné síť učit, je zapotřebí mít k dispozici tréninkovou množinu X ve formě uspořádaných dvojic vzoru x a *požadovaného výstupu sítě* d. Pak se musí stanovit chyba odlišnosti skutečného výstupu neuronové sítě od požadovaného. Ke stanovení této chyby na celé tréninkové množině se užívá *kriteriální funkce* [5]:

$$E = \frac{1}{2}\sum_{k}\sum_{j}(y_{j,k} - d_{j,k})^2 , \qquad (3.9)$$

kde k je indexem uspořádané dvojice a $y_{j,k}$ (resp. $d_{j,k}$) je skutečný (resp. požadovaný) výstup j-tého neuronu ve výstupní vrstvě neuronové sítě po předložení k-tého tréninkového vzoru x_k.

3.4 Metoda back – propagation

Cílem učení je minimalizace kriteriální funkce pomocí algoritmu zpětného šíření. S jeho pomocí lze po každém předložení tréninkového vzoru x určit způsob modifikace váhy a prahové hodnoty neuronů v síti tak, aby došlo k poklesu kriteriální funkce.

Algoritmus zpětného šíření pracuje tak, že určí velikost chyby na výstupu každé vrstvy sítě a tuto chybu pak propaguje do nižších vrstev pomocí vážených spojů. Takže upravuje váhy směrem od výstupní vrstvy k vrstvě vstupní. Tyto změny jsou popsány vztahem [5]:

$$\Delta w_{ij}^{(a)} = \delta_j^{(a)} o_j^{(a-1)}, \tag{3.10}$$

kde $\Delta w_{ij}^{(a)}$ značí změnu váhy na hraně vedoucí z *i*-tého neuronu ve vrstvě *a – 1* do *j*-tého neuronu vrstvy *a*. V závislosti na aktuálně zpracovávané vrstvě pak platí:

$$\delta_j^{(L)} = (d_j - y_j) f'(\xi_j^{(L)}) \qquad \text{pro } j\text{-tý výstupní neuron,} \tag{3.11}$$

$$\delta_j^{(a)} = f'(\xi_j^{(L)}) \sum_j w_{ij}^{(a+1)} \delta_j^{(a+1)} \qquad \text{pro } i\text{-tý neutron ve vrstvě } a < L. \tag{3.12}$$

Nové váhy a prahové hodnoty neuronů v síti jsou pak upraveny takto:

$$w_{ij}^{(a)} \leftarrow w_{ij}^{(a)} + \Delta w_{ij}^{(a)}, \tag{3.13}$$

$$\vartheta_j^{(a)} \leftarrow \vartheta_j^{(a)} + \delta_j^{a0}, \tag{3.14}$$

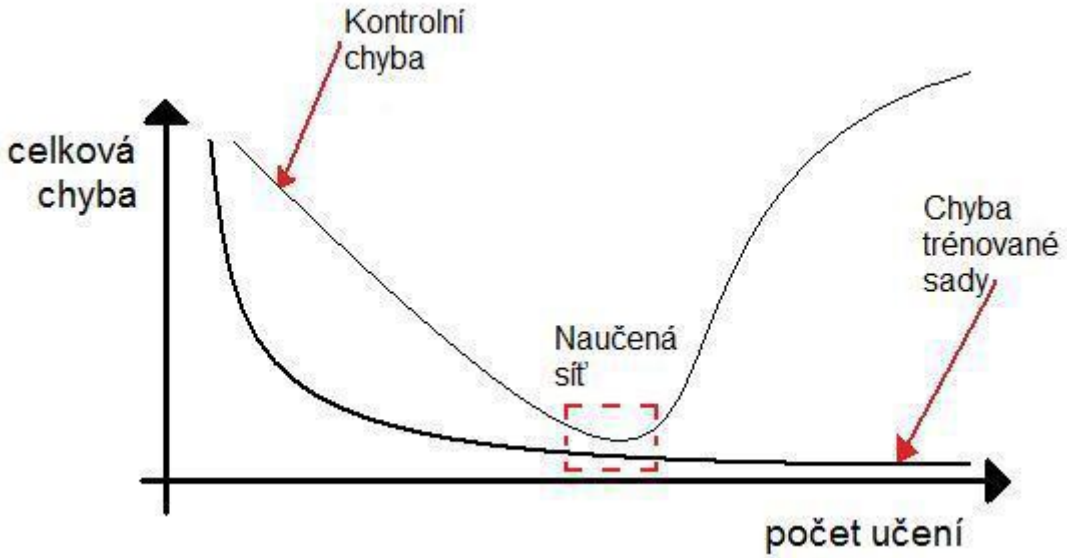
přičemž prahové hodnoty neuronů ve vstupní vrstvě se nemění.

Trénování sítě většinou probíhá v tzv. *epochách*. V každé epoše jsou síti postupně předloženy všechny vzory z tréninkové množiny. V další epoše se proces opakuje. Podle způsobu předkládání vzorů a adaptování vah se rozlišují dva způsoby učení [5]:

1) *Online učení* – Po předložení každého vzoru jsou okamžitě adaptovány váhy. Tento způsob je vhodný hlavně tehdy, je-li tréninková množina natolik velká, že se nemůže celá uchovávat v paměti.

2) *Dávkové učení* – Tento způsob je z hlediska učení výhodnější, změny vah totiž provádí až na konci každé epochy. Změny vah a prahových hodnot neuronů jsou v průběhu epochy přičítány do pomocných proměnných ($\Delta W_{ij}^{(a)}$ a $\Delta T_j^{(a)}$).

Jedním ze způsobů jak definovat podmínku ukončení učení je sledování poklesu kriteriální funkce pod stanovenou mez. Malá chyba na tréninkové množině nemusí hned znamenat dobře naučenou síť, protože chyba na datech mimo tréninkovou množinu může být naopak velká. To se nazývá *přeučení sítě* [5]. Řešení spočívá v zavedení kontrolní chyby. Ta po každém dokončeném cyklu učení přepočítá chybu v závislosti na vstupních vzorech. Algoritmus nepoužívá při učení zašuměná vstupní data, po zavedení kontrolní chyby je bude schopen v testovací fázi rozpoznat.



Obr. 2.5) Graf přesnosti učení

Používají se dvě metody k nalezení globálního minima chyby:

1) *Metoda nalezení minima* – Metoda, která se snaží měnit hodnoty vah tak, aby chyba klesala. Nastává ovšem problém, v případě, že chyba začne růst – došlo k nalezení lokálního minima funkce. Zde hrozí, že se metoda zasekne a chyba se nebude dále zmenšovat. Algoritmus chce ve vyhledávání pokračovat, chce dosáhnout globálního minima. Toho ale nedosáhne, protože není schopen procházet funkci směrem vzhůru. Existuje několik způsobů jak odstranit tento problém [3]:

   - Všem vahám nastavíme nové náhodné hodnoty, metoda se bude muset učit znova a bude mít jinou funkci, ve které bude hledat minima

   - Upravíme vzorce pro nastavení nových hodnot vah. Hodnoty nebudou závislé jenom na aktuální chybě, ale také na hodnotách z předešlého průchodu:

$$w_{aktuální} = w_{stará} + Ex + M\,, \tag{3.15}$$

   kde *Momentum M* vyjádříme následovně:

$$M(w_{aktuální} - w_{stará})\,. \tag{3.16}$$

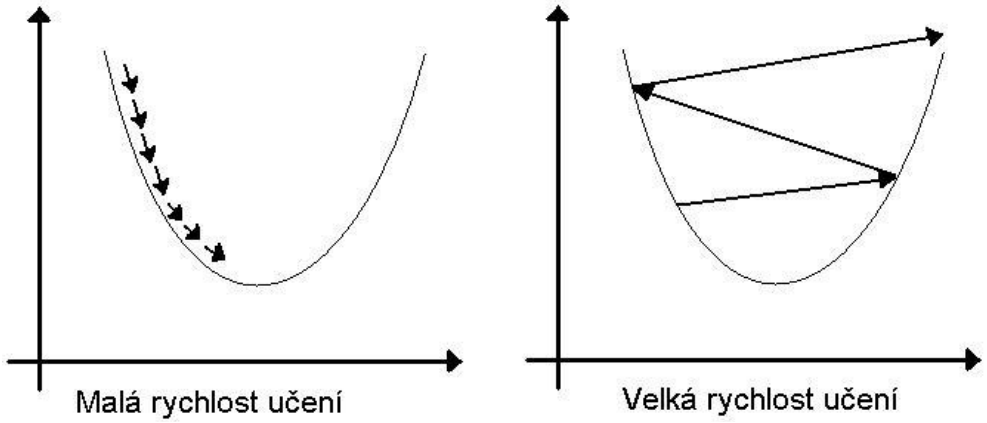
   Momentum (setrvačnost) definované v intervalu <0,1> určuje, jak dlouho bude váhový vektor pokračovat ve stejném směru, než překročí hodnotu gradientního horizontu (než začne počítat gradient pro jiné minimum). Nevýhodou back-propagation s momentem je fakt, že pokud máme funkci, kde

problém s lokálním minimem nenastává, pak je metoda bez momentu mnohem rychlejší.

2) *Metoda pracující s rychlostí učení* – Rychlost učení je jeden z nejdůležitějších aspektů pro nalezení globálního minima. Značíme η a rovnice pro výpočet nových hodnot vah vypadá následovně [3]:

$$w_{nová} = w_{aktuální} + \eta Ex\,. \tag{3.17}$$

Algoritmus pro příliš malé hodnoty rychlosti učení probíhá velmi pomalu (konverguje velmi pomalu), naopak pro velké hodnoty probíhá rychle, ale hrozí riziko oscilace mezi relativně nezajímavými lokálními minimy, případně divergence u globálního minima.



Obr. 2.6) Grafy rychlosti učení

Obě tyto možnosti mohou být vyhledány pomocí experimentování a testování vzorků po pevně daném počtu průchodů. Běžně používané hodnoty rychlosti učení leží v intervalu <0,1>. Ideální by bylo, použít největší hodnotu rychlosti učení, při zachování schopnosti konvergovat k minimálnímu řešení.

Neuronová síť se používá jako klasifikátor, který má neznámý vzor klasifikovat do jedné z klasifikačních tříd. Jednou z možností je použití pouze jednoho výstupního neuronu s identickou přenosovou funkcí. Naučená síť by měla na předložený vstupní vzor *x* dát (na výstup) index odpovídající třídy do které byl tento vzor klasifikován.

Relativně jednodušší (a lepší) je naučit síť tak, aby na vstupní vzor *x* z trénovací množiny dávala požadovaný výstup *d*:

$$d = e(i)\,, \tag{3.18}$$

kde vzor *x* z tréninkové množiny má být klasifikován do třídy *i*, vektor *d* má *c* složek (*c* tříd) a platí:

$$e(i) = (\underbrace{0, ..., 0}_{i-1}, \overbrace{1}^{i}, 0, ..., 0). \tag{3.19}$$

Neznámý vektor *x*, pro který dává neuronová síť výstup *y* je pak klasifikován do třídy *i* podle vztahu:

$$i = \arg\max_j \{y_j\}, \tag{3.20}$$

to znamená, že se vezme neuron z výstupní vrstvy, který má na výstupu maximální hodnotu, a jeho index bude představovat index klasifikované třídy.

Počet neuronů ve skrytých vrstvách závisí na konkrétní aplikaci. Typicky se určuje experimentálně, v některých může pomoci tzv. *zlaté pravidlo*. Počet všech vah a prahových hodnot (mimo těch ve vstupní vrstvě) uvnitř neuronové sítě určuje tzv. *stupeň volnosti*. Platí, že stupeň volnosti by neměl být větší než je celkový počet vstupních hodnot *N* (tzn. velikost trénovací množiny |*X*| násobená dimenzí vstupních vektorů *n*). Zlaté pravidlo říká, že pokud je použito nejvýše tolik skrytých neuronů, aby stupeň volnosti nepřesáhl *N/10*, dosáhne se při trénování nejnižší chyby na testovací množině [1].

Pro síť s jednou skrytou vrstvou platí:

$$q_h(q_{in} + q_{out}) + q_h + q_{out} \le \frac{N}{10}, \tag{3.21}$$

kde q_h označuje hledaný počet skrytých neuronů a q_{in} a q_{out} označují počet vstupních a výstupních neuronů. Celkový počet vah v síti je $q_h(q_{in} + q_{out})$ a prahových hodnot je $q_{in} + q_{out}$. Pak dostáváme:

$$q_h \le \frac{\frac{N}{10} - q_{out}}{q_{in} + q_{out} + 1}. \tag{3.22}$$

4 REŠERŽE ZDROJŮ

Abych dokázal vhodně zvolit sadu příznaků, použít určitý typ extrakce příznaku a zvolit vhodnou neuronovou síť s určitou topologií, udělal jsem si rešerži zdrojů. Vybral jsem takové zdroje, ve kterých se autor věnoval stejné či podobné problematice jako já, čili rozpoznávání číslic. Vycházel jsem ze tří prací, které jsem rovněž uvedl v použité literatuře, jedné diplomové a dvou bakalářských. Jedná se o *Rozpoznávání SPZ z jednoho snímku* [5], *Rozpoznávání číslic pomocí neuronové sítě* [6] a *Rozpoznávání textu v obraze* [7]. Obsah těchto prací a jejich přínos pro můj problém jsem rozebral níže:

4.1 Rozpoznávání SPZ z jednoho snímku

Jedná se o diplomovou práci. Autorovým cílem bylo rozpoznávání SPZ automobilů Spojených arabských emirátů z fotografií pořízených systémem pro kontrolu rychlosti vozidel. Autor se tímto úkolem zabýval komplexně, takže v práci je obsaženo předzpracování, lokalizace oblasti SPZ, segmentace a rozpoznávání číslic. Navíc samotnou klasifikaci číslic prováděl několika metodami, kromě řešení pomocí neuronových sítí také klasifikátorem minimální vzdálenosti a skrytými Markovovými modely. Takže rozpoznávání číslic pomocí neuronových sítí se zabýval jen částečně.

Zabýval jsem se pouze těmi částmi práce, které jsou pro mne důležité. Po segmentaci číslic z SPZ a všech úpravách včetně normalizace zde byly použity binární obrázky číslic o velikosti 30 x 30 pixelů Číslice však byly normalizovány pouze přizpůsobením výšky na velikost normalizačního čtverce, šířka byla dopočítána pro zachování původního poměru výšky ku šířce. Z těchto vzorů byly vytvořeny tři množiny, tréninkovou, ověřovací (testovací) a problémová. Tréninková množina byla vytvořena z 352 obrázků číslic.

Pro rozpoznávání pomocí neuronových sítí byla použita metoda back-propagation. Byly zde vyzkoušeny sítě s různou architekturou. Počet neuronů ve vstupní vrstvě byl závislý na použitém typu příznaku, výstupní vrstva měla vždy 10 neuronů (deset klasifikačních tříd). Množství neuronů ve skrytých vrstvách bylo získáno pokusem, který většinou korespondoval s výsledky zlatého pravidla. Učení probíhalo dávkově. Výsledky, kterých bylo v této práci dosaženo při rozpoznávání latinských číslic (výsledky emirátských číslic jsem ignoroval) jsou následující:

1) Příznaky typu *NCM* s momenty do řádu 3 nejsou dostačující, testovaná síť nebyla schopna si zapamatovat ani tréninkové vzory. Použití momentů vyšších řádů (až do řádu 7) již přineslo zlepšení (úspěšnost 99% na tréninkové množině). Další přidávání momentů už úspěšnost klasifikace nezvýšilo. Na ověřovací množině však nebyla úspěšnost dostačující (okolo 88%) [5].

2) Při použití příznaků *Pix* bylo dosaženo sítě velkých rozměrů. Při zachování třívrstvé topologie bylo dosaženo úspěšnosti rozpoznání uvedené v následující tabulce:

topologie	P
900-27-10	82%
900-34-10	85%

Tab. 4.1) Neuronová síť s příznaky Pix s úspěšností klasifikace na problémové (P) množině

3) Použitím příznaků typu *Proj* se nepodařilo snížit chybu na tréninkové množině na rozumnou hodnotu. Úspěšnost rozpoznání v závislosti na počtu neuronů ve skryté vrstvě je uvedena v tabulce:

topologie	T	O	P
60-20-10	90%	76%	71%
60-22-10	100%	94%	75%

Tab. 4.2) Neuronová síť s Příznaky Proj s úspěšností klasifikace na tréninkové (T), ověřovací (O) a problémové (P) množině

4) Čtyřvrstvé sítě s příznaky typu *Win* se svou úspěšností rozpoznání přiblížily k nejlepším třívrstvým topologiím. Výsledky jsou uvedeny v tabulce:

topologie	O
81-15-15-10	95%
81-20-20-10	85%

Tab. 4.3) Neuronová síť s příznaky Win s úspěšností na klasifikace ověřovací (O) množině

5) Nejlepších výsledků úspěšnosti rozpoznání bylo dosaženo pomocí tří-vrstvé sítě s příznaky typu *Win*. Výsledky jsou uvedeny v tabulce:

topologie	O
81-24-10	97%
81-22-10	89%

Tab. 4.4) Neuronová síť s příznaky Win s úspěšností klasifikace na ověřovací (O) množině

Největší problém nastával s konkrétními špatně klasifikovanými znaky 0 (chybně klasifikováno jako 9) a 5 (chybně klasifikováno jako 6).

4.2 Rozpoznávání číslic pomocí neuronové sítě

Jedná se o bakalářskou práci. Autor se v ní zabýval podobnou problematikou jako já, to znamená popisu neuronové sítě, bližšímu seznámení s metodou back-propagation používanou pro rozpoznávání číslic a věcmi s tím spojenými. Podobně jako autor diplomové práce uvedené výše se zabýval také předzpracováním obrazu, jako například vyhledáváním objektu v obraze, prahováním a podobně. Opět jsem se zaměřil pouze na pro mne důležité informace obsažené v této práci.

Co se týče neuronových sítí, podrobně se zabýval testováním počtu epoch, testováním rychlosti učení a testováním závislosti počtu vrstev a počtu neuronů neuronové sítě. K těmto testům byly použity vlastní obrázky číslic velikosti 5x7 pixelů reprezentující 10 číslic (vytvořené v malování). Tato sada byla podle potřeby různě upravována, např. přidáním šumu, posunutím pozice apod. Další sady číslic byly převážně převzaty z různých typů písem (fontů) a pospojovány pro vytvoření souboru sad.

K testu počtu epoch byla použita topologie sítě *7-7-10* s použitou aktivační funkcí logická sigmoida.

Z testování počtu vrstev a počtu neuronů neuronové sítě došel autor k závěru, že nejvhodnější topologií neuronové sítě pro další práci je *5-20-10*.

Při testu rozpoznávání jedné číslice byla použita znaková sada *MS PMincho* s velikostí písma 11 bodů, obrázky v rozměru 9 x 13 pixelů. Neuronová síť byla třívrstvá, vstupní vrstva měla 20 neuronů, jedna skrytá vrstva 100 neuronů a výstupní vrstva 10 neuronů. Testovala se zde číslice opatřená šumem, číslice posunutá v obraze, či číslice nakloněná. Testy bylo zjištěno, že algoritmus back-propagation výborně rozpoznává obraz se šumem. Problém nastal, pokud byl obraz nějakým způsobem vychýlen, ať už posunem, nebo rotací.

Konkrétní hodnoty výsledků dosažených v této práci zde nezveřejňuji, jelikož autor se zabýval spíše úspěšnostmi rozpoznávání několika jednotlivých (různě upravovaných) potenciálně zaměnitelných znaků než komplexními procentuálními výsledky např. na ověřovací či problémové množině.

4.3 Rozpoznávání textu v obraze

Jedná se o bakalářskou práci. Autor se v ní zabývá systémy OCR (systémy pro optické rozpoznávání znaku). Stejně jako v pracích rozebraných výše se i zde zabývá také předzpracováním obrazu (korekcí pootočení, lokalizací řádků a segmentací znaků). Ke klasifikaci bylo opět použito dopředné neuronové sítě s učícím algoritmem back-propagation.

Vstupním obrázkem byla v tomto případě 8 bitová bitmapa, tedy obrázek ve stupních šedi. Jednoduchým prahováním byl obrázek převeden do binarizovaného stavu.

Autorem byl vytvořen program na generování písmenek latinské abecedy do obrázku pro vytvoření tréninkové sady. Obrázky byly následně převedeny na tréninkové soubor obsahující vektor Hu momentů (příznak typu Hu momenty) každého písmenka a k němu odpovídající vektor identifikující znak.

Co se týká výběru topologie neuronové sítě, jsou zde rozebrány 3 návrhy, z nichž se dva zabývají klasifikací všech znaků UTF a jeden se zabýval klasifikací číslic. Tréninková množina byla rozdělena na 2 části, z nichž jedna sloužila pro učení a druhá pro kontrolu chyby aby nedošlo k přeučení. Výsledná topologii neuronové sítě byla 7-100-16, z čehož vstupní vrstva tvořilo 7 neuronů reprezentujících 7 Hu momentů a výstupní vrstvu tvořilo 16 neuronů, z nichž každý nesl hodnotu 1 nebo 0 (výsledný vektor udává binární číslo reprezentující každý znak).

V závěru bylo prezentováno zjištění, že kombinace Hu momentů a neuronových sítí při klasifikaci není ideální. Toto spojení je použitelné pouze pro menší počet rozpoznávaných znaků, např. právě jen pro rozpoznávání číslic. Úspěšnost rozpoznávání byla také hodně závislá na kvalitě zdrojového obrázku. Příčinou byly nejspíš zvolené Hu momenty, které nedokážou úspěšně rozlišit větší počet znaků a jsou velmi citlivé na sebemenší změnu vzhledu znaku.

5 VÝBĚR SAD VZORŮ

5.1 Vytvoření obrázků číslic

Rozhodnul jsem se pro vytvoření vlastních obrázků číslic. Využil jsem k tomu program „Malování“ systému Windows. Velikost normalizačního čtverce jsem zvolil 18x18 pixelů. Tato velikost byla dána především omezením počtu neuronů v jedné vrstvě programu *BPSIM* [9]. Pak jsem postupně vypisoval číslice různých znakových sad. Protože se číslice po prozkoumání lupou skládaly z různých barev o různých jasových hodnotách, uložil jsem tento obrázek jako bitmapu (*.bmp). Tímto jsem vlastně dostal obrázek skládající se pouze z pixelů černé a bílé barvy a efektivně jsem se tak zbavil nutnosti manuálního prahování. Hodnotu výšky fontů jsem volil tak, abych nemusel velikost jednotlivých číslic výškově ani šířkově upravovat. Když jsem totiž zvolil velikost fontu 18, neznamená to automaticky, že velikost číslice u každé znakové sady bude také 18 pixelů. Z toho plyne, že jsem „upravil“ velikost číslice na velikost normalizačního čtverce bez přizpůsobení šířky, takže poměr výšky ku šířce číslice zůstal zachován. Takto upravené číslice umístěné horizontálně ve středu obrázku o velikosti normalizačního čtverce jsem jednotlivě ukládal do paměti.

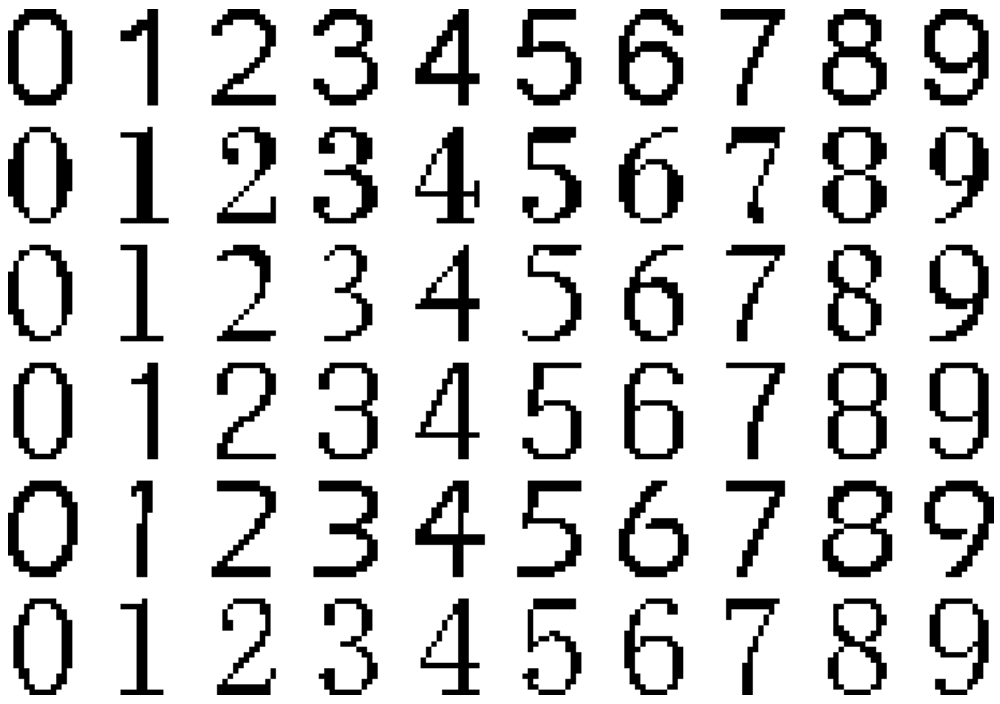
5.2 Vytvoření tréninkových a testovacích množin

Nyní již mám vytvořeny obrázky pro číslice z 11 znakových sad. Sad jsem měl původně připraveno více, ale opět jsem narazil na omezení programu *BPSIM*. Jedná se o tyto (v závorkách jsou uvedeny zkratky, které jsem pro ně použil a za nimi vzory jednotlivých sad):

- Arial (A): 0 1 2 3 4 5 6 7 8 9
- Bodoni MT (B): 0 1 2 3 4 5 6 7 8 9
- Californian FB (C): 0 1 2 3 4 5 6 7 8 9
- Dotum (D): 0 1 2 3 4 5 6 7 8 9
- Eras Medium ITC (E): 0 1 2 3 4 5 6 7 8 9
- Fang Song (F): 0 1 2 3 4 5 6 7 8 9
- Kartika (K): 0 1 2 3 4 5 6 7 8 9
- Levenim MT (L): 0 1 2 3 4 5 6 7 8 9
- NSimSum (N): 0 1 2 3 4 5 6 7 8 9
- OCR A Extended (O): 0 1 2 3 4 5 6 7 8 9
- Poor Richard (P): 0 1 2 3 4 5 6 7 8 9

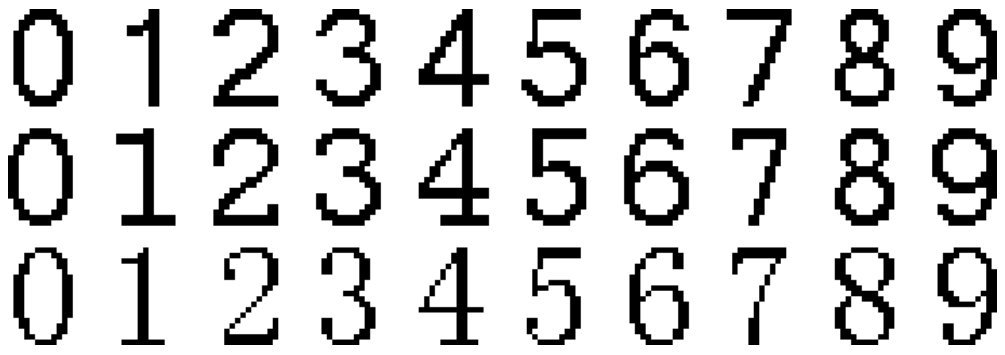
Z těchto sad jsem následně vytvořil 3 množiny:

1) *Tréninková množina* – Do této množiny jsem umístil 6 sad znaků (A, B, C, D, E, F). Jedná se o znaky, které jsou velmi dobře rozlišitelné a budou mi sloužit k učení (trénování) neuronové sítě.



Obr. 5.1) Připravené sady obrázků v tréninkové množině před extrakcí příznaků

2) *Ověřovací množina* – Do této množiny jsem umístil 3 sady znaků (K, L, N). Jedná se opět o znaky, které jsou velmi dobře rozlišitelné, avšak nejsou obsaženy v tréninkové množině. Tyto znaky mi budou sloužit k testování naučené neuronové sítě.



Obr. 5.2) Připravené sady obrázků v testovací množině před extrakcí příznaků

3) *Problémová množina* – Do této množiny jsem umístil 2 sady znaků (O, P). Jedná se o znaky, které jsou hůře rozlišitelné a budou mi sloužit k testování naučené neuronové sítě na hůře rozlišitelných číslicích.

Obr. 5.3) Připravené sady obrázků v problémové množině před extrakcí příznaků

6 REALIZACE EXTRAKCE PŘÍZNAKŮ

6.1 Vytvoření programu pro extrakci příznaku

Hledal jsem na internetu program, který by dokázal z připravených obrázků extrahovat příznaky a vytvořit z nich příznakový vektor. Bohužel jsem žádný takový nenašel. Našel jsem jen jeden, který dokázal převést obrázek na rastr jedniček a nul, což pro mne nebylo dostačující. Rozhodl jsem se tedy, že si program vytvořím sám. Rozhodnul jsem se že tento program vytvořím v programovacím jazyku Java.

Tento program jsem pojmenoval „bmp“. Je koncipován tak, aby mu mohl být předložen obrázek číslice (*.bmp) o libovolné velikosti (tzn. že není naprogramován jen pro jednu fixní velikost obrázku). Výstupem je matice jedniček a nul reprezentující černé a bílé pixely v obrázku, příznakový vektor typu Proj (vektor řádků a pod ním vektor sloupců) a příznakový vektor typu Win. Pro správné zobrazení výstupních hodnot příznakových vektorů je nutné mít v počítači nastaven jako výchozí jazyk klávesnice Angličtinu, jinak se zobrazí místo desetinné tečky desetinná čárka. To by mi ztížilo následný export těchto vektorů do programu pro simulaci neuronových sítí (podrobněji níže).

6.2 Příznaky typu Proj – model 1

První typ použitého příznaku byla projekce (*Proj*). Rozhodl jsem se tak na základě rešerží a také relativně snadné realizovatelnosti. Při tvorbě příznakového vektoru typu *Proj* jsem využil programu bmp. Jeho prostřednictvím se ze vstupního obrázku postupně načítaly hodnoty pixelů v jednotlivých řádcích, počet černých pixelů (jedniček) se pak sečetl a vydělil celkovým počtem pixelů v řádku (18). Tato výsledná hodnota byla zaokrouhlena na dvě desetinná místa a zapsána na výstup programu jako prvek příznakového vektoru. Obdobně se se postupovalo i při projekci sloupců. Hodnoty obou projekcí byly zapsány za sebe a vytvořily tak příznakový vektor který tedy obsahuje 36 hodnot (18 projekcí řádků a 18 projekcí sloupců). Z této hodnoty jsem v další kapitole vycházel při návrhu topologie neuronové sítě (počet vstupních hodnot odpovídá počtu neuronů ve vstupní vrstvě). Takto jsem postupoval pro všechny obrázky číslic z použitých množin.

Hodnoty příznakových vektorů všech obrazů jsem importoval do vstupních souborů pro program *BPSIM*. Soubory byly formátu *.*DAT*, ale daly se editovat v poznámkovém bloku, tvořil jsem je dle následujícího klíče: V názvu souboru jsem uvedl zkratku příznakových vektorů znakových sad které obsahuje a za podtržítkem typ příznakového vektoru „*PRO*“.

Pro příznaky typu *Proj* jsem vytvořil tyto soubory:

- *AB_PRO.DAT*
- *ABCD_ PRO.DAT*
- *KLN_ PRO.DAT*
- *OPV_ PRO.DAT*

6.3 Příznaky typu Win – model 2

Druhý typ příznaků, pro jehož použití jsem se rozhodl byl typ posuvné okno (*Win*). Z rešerží jsem zjistil, že je pro problematiku rozpoznávání číslic s oblibou využíván, a to také z důvodu úspěšné klasifikace obrazů s šumem a mírně pootočených obrazů. Při tvorbě příznakového vektoru typu *Win* jsem opět využil programu bmp. Jeho prostřednictvím se vstupní obrázek procházel oknem o velikosti 4x4 pixely, krok posunu byl 2 pixely (viz kapitola 2). V každém kroku posuvu okna se sečetlo množství černých pixelů v něm, a tento součet se vydělil celkovým počtem pixelů v okně (4x4=16). Výsledná hodnota byla zaokrouhlena na dvě desetinná místa a zapsána na výstup programu jako prvek příznakového vektoru. Hodnoty všech kroků byly zapsány za sebe a vytvořily tak příznakový vektor který tím pádem obsahuje 64 hodnot (64 kroků posuvného okna), neuronová síť proto bude mít 64 vstupních neuronů Takto jsem postupoval pro všechny obrázky číslic z použitých množin.

Hodnoty příznakových vektorů všech obrazů jsem stejně jako v předchozím modelu importoval do vstupních souborů pro program *BPSIM*. Názvy souboru jsem také volil obdobně, avšak za podtržítkem jsem uvedl typ příznakového vektoru „*WIN*“. Pro příznaky typu *Win* jsem vytvořil tyto soubory:

- *AB_WIN.DAT*
- *ABEF_WIN.DAT*
- *KLN_WIN.DAT*
- *OPV_WIN.DAT*

7 IMPLEMENTACE A VYHODNOCENÍ

7.1 Realizace neuronové sítě

Pro realizaci neuronové sítě jsem použil program BPSIM. Jedná se o program pro simulaci neuronových sítí od zadávání topologie až po prezentaci výsledků.

Umožňuje tvorbu různých topologií, zadávání parametru učení, inicializaci vah a inicializaci strmostí. Také je možné nastavit maximální počet předložení celé tréninkové množiny vzorů - epoch (sloužící jako zarážka algoritmu), počet opakování předložení jednoho vzoru a toleranci celkové chyby. K dispozici je i výběr mezi několika učícími algoritmy. Kromě metody back-propagation lze využívat ještě algoritmus s adaptující se přenosovou funkcí, algoritmus s párovým neuronem a algoritmus s proměnným počtem neuronů. Kromě učení lze síť i otestovat na uživatelem vybraných datech, či zobrazit a vytisknout průběhy učení (pravděpodobně díky problémy s kompatibilitou s novými systémy Windows nebyl tento modul funkční). Všechny potřebné informace i výsledky učení a testování jsou vypisovány na obrazovku programu. Program však má í některá omezení, nejvýraznější je omezení vrstev neuronové sítě na 5 (vyjma výstupní vrstvy), omezení maximálního počtu neuronů v jedné vrstvě na 64 a omezení počtu trénovacích vzorů na 40. Tato omezení jsem musel zohlednit při tvorbě obrazových vzorů a výběru typů příznakových vektorů.

Jak jsem již v předchozí kapitole uváděl, vstupním souborem tohoto programu jsou soubory formátu **.DAT*. Kromě hodnot příznakových vektorů musí tento soubor v hlavičce obsahovat také dimenzi vstupu (počet vstupních neutronů), dimenzi výstupu (počet výstupních neutronů), a počet vzorů uložených v souboru. Také je třeba přiřadit k jednotlivým vstupním datům (přízankovým vektorům) požadované hodnoty výstupu.

Na základech rešerží jsem jako hodnotu výstupu použil vektor délky 10, který obsahuje jedinou jedničku na místě odpovídající klasifikační třídy , jinak samé nuly. To znamená, že ve výstupní vrstvě navrhované neuronové sítě bude vždy 10 neuronů. Hodnoty výstupů pro jednotlivé číslice (klasifikační třídy) jsem určil takto:

0 :	1	0	0	0	0	0	0	0	0	0
1:	0	1	0	0	0	0	0	0	0	0
2:	0	0	1	0	0	0	0	0	0	0
3:	0	0	0	1	0	0	0	0	0	0
4:	0	0	0	0	1	0	0	0	0	0
5:	0	0	0	0	0	1	0	0	0	0
6:	0	0	0	0	0	0	1	0	0	0
7:	0	0	0	0	0	0	0	1	0	0
8:	0	0	0	0	0	0	0	0	1	0
9:	0	0	0	0	0	0	0	0	0	1

Na základech rešerží jsem se také rozhodl pro učící metodu back-propagation která je pro rozpoznávání číslic nejpoužívanější, jako aktivační funkci jsem použil logickou sigmoidu. Také jsem se rozhodl, že budu pracovat s topologií jen s jednou skrytou vrstvou. Co se týká nastavení parametrů učení, tak rychlost učení jsem po celou dobu používání programu měl nastavenou na hodnotu 0,8 a maximální počet epoch na 500. Počet opakování předložení jednoho vzoru jsem ponechal na hodnotě 1. Velikost tolerance celkové chyby jsem nastavil na 0,1. Když hodnota chyby klesla pod tuto hodnotu, síť byla považována za naučenou a mohlo se přistoupit k jejímu testování. Z výše uvedených informací vyplývá, že proces učení probíhal dávkově. Hodnotu momentu jsem nastavoval na základě výsledků učení zvlášť pro každý model. Hodnoty vah jsem určil náhodně pomocí tlačítka *Random* (v intervalu (-0,5; +0,5)) a uložil je pro opětovné použití zvlášť pro každou topologii.

7.2 Test a vyhodnocení modelu 1

Nyní jsem přešel k samotné tvorbě topologie sítě pro příznaky typu *Proj*. Počet vstupních i výstupních neuronů byl již jasně daný rozborem v předchozích kapitolách, takže jsem se omezil pouze na volbu vhodných parametrů učení a určení počtu neuronů v jedné skryté vrstvě.

Začal jsem s testováním počtu neuronů ve skryté vrstvě. Pro učení jsem použil vstupní soubor *AB_PRO.DAT* obsahující dvě znakové sady číslic (20 tréninkových vzorů). Náhodně jsem zvolil hodnotu momentu 0,8 a testoval jsem parametry učení a úspěšnost rozpoznávání na ověřovací (*KLN_PRO.DAT*) a problémové (*OPV_PRO.DAT*) množině pro různé topologie. Tři „nejlepší“ topologie jsem zobrazil v následující tabulce:

topologie (M=0,8)	epochy	O	P
36-8-10	154	63,3%	35%
36-10-10	143	70%	30%
36-12-10	123	66,7%	35%

Tab. 7.1) Neuronová síť s 20-ti tréninkovými vzory (příznaky Proj) s úspěšností klasifikace na ověřovací (O) a problémové (P) množině

Zvolil-li jsem větší, či menší počet skrytých neuronů než je uvedeno v tabulce, zvýšil se počet epoch nutných k naučení sítě a zároveň se snížila úspěšnost klasifikace na obou uvedených množinách. Pro úplnost dodávám, že úspěšnost klasifikace na tréninkové množině byla ve všech případech 100%. Pro další testování jsem se na základě těchto výsledků rozhodl pro neuronovou síť s topologií *36-10-10*.

V dalším kroku jsem se zabýval určením nejvhodnější hodnoty momentu. Pro vybranou topologii neuronové sítě jsem zkoušel měnit hodnoty momentu a sledoval

jsem počet epoch nutných k naučení sítě (poklesu celkové chyby pod 0,1). Výsledky jsem zobrazil v následující tabulce:

Moment (36-10-10)	epochy	O	P
0,6	262	73,3%	30%
0,7	195	76,7%	30%
0,8	143	70%	30%

Tab. 7.2) Neuronová síť s 20-ti tréninkovými vzory (příznaky Proj) s úspěšností klasifikace na ověřovací (O) a problémové (P) množině

Při testování vyšších či nižších hodnot momentu než uvedených v tabulce docházelo k nárůstu počtu epoch až na nastavenou limitní hodnotu (500 epoch), což znamenalo, že se síť nebyla schopná naučit (celková chyba neklesla pod hodnotu 0,1). Na základě těchto výsledků jsem se rozhodl pro použití hodnoty momentu 0,7 z důvodu nejvyšší úspěšnosti klasifikace z testovaných hodnot.

Nyní jsem přistoupil ke zvyšování počtu tréninkových vzorů. Pro učení jsem použil vstupní soubor *ABCD_PRO.DAT* obsahující čtyři znakové sady číslic (40 tréninkových vzorů). Hodnotu momentu jsem ponechal z předchozího kroku, čili 0,7. Pak jsem měnil počet skrytých neuronů do té doby, než jsem dosáhnul nejlepší možné klasifikace číslic. Dvě nejúspěšnější topologie jsem zobrazil v tabulce:

topologie (M=0,7)	epochy	O	P
36-14-10	178	80%	45%
36-16-10	197	86,7%	30%

Tab.7.3) Neuronová síť se 40-ti tréninkovými vzory (příznaky Proj) s úspěšností klasifikace na ověřovací (O) a problémové (P) množině

Z tabulky vyplývá, že síť s topologií *36-14-10* byla lepší pro klasifikaci číslic obsažených v problémové množině (45% proti 30% druhé sítě), kdežto síť s topologií *36-16-10* byla lepší pro klasifikaci číslic obsažených v ověřovací množině (86,7% proti 80% druhé sítě). Bohužel už jsem se nemohl zabývat učením sítě na větší tréninkové množině a jejím testováním, neboť to mnou používaný program *BPSIM* neumožňoval.

7.3 Test a vyhodnocení modelu 2

Při tvorbě topologie sítě pro příznaky typu *Win* jsem postupoval podobně jako při tvorbě sítě pro příznaky typu *Proj*.

Začal jsem s testováním počtu neuronů ve skryté vrstvě. Pro učení jsem použil vstupní soubor *AB_WIN.DAT* obsahující dvě znakové sady číslic. Náhodně jsem zvolil

hodnotu momentu 0,8 a testoval jsem parametry učení a úspěšnost rozpoznávání na ověřovací (*KLN_WIN.DAT*) a problémové (*OPV_WIN.DAT*) množině pro různé topologie. Tři „nejlepší“ topologie jsem zobrazil v následující tabulce:

topologie (M=0,8)	epochy	O	P
64-18-10	65	100%	60%
64-20-10	56	100%	60%
64-22-10	90	100%	60%

Tab. 7.4) Neuronová síť s 20-ti tréninkovými vzory (příznaky Win) s úspěšností klasifikace na ověřovací (O) a problémové (P) množině

Jelikož měly všechny tři zvolené topologie sítě stejnou úspěšnost klasifikace, pro další testování jsem se rozhodl vybrat síť s nejmenším počtem epoch potřebných pro naučení. Jedná se tedy o neuronovou síť s topologií *64-20-10.* Pro úplnost dodávám, že úspěšnost klasifikace na tréninkové množině byla ve všech případech 100%.

V dalším kroku jsem se opět zabýval určením nejvhodnější hodnoty momentu. Výsledky jsem zobrazil v následující tabulce:

Moment (64-20-10)	epochy	O	P
0,5	141	100%	60%
0,6	110	100%	60%
0,7	79	100%	60%
0,8	56	100%	60%

Tab. 7.5) Neuronová síť s 20-ti tréninkovými vzory (příznaky Win) s úspěšností klasifikace í na ověřovací (O) a problémové (P) množině

Při testování vyšších či nižších hodnot momentu než uvedených v tabulce docházelo opět k nárůstu počtu epoch až na nastavenou limitní hodnotu (500 epoch). Na základě těchto výsledků jsem se rozhodl pro použití hodnoty momentu 0,8 z důvodu nejmenšího počtu epoch potřebných pro naučení.

Nyní jsem přistoupil ke zvyšování počtu tréninkových vzorů. Pro učení jsem použil vstupní soubor *ABEF_PRO.DAT* obsahující čtyři znakové sady číslic. Hodnotu momentu jsem ponechal z předchozího kroku, čili 0,8. Pak jsem měnil počet skrytých neuronů do té doby, než jsem dosáhnul nejlepší možné klasifikace číslic.

Nejúspěšnější topologii jsem zobrazil v tabulce:

topologie (M=0,7)	epochy	O	P
64-20-10	49	100%	65%

Tab.7.6) Neuronová síť se 40-ti tréninkovými vzory (příznaky Win) s úspěšností klasifikace na ověřovací (O) a problémové (P) množině

Z tabulky vyplývá, že síť s topologií *64-20-10* se 40-ti tréninkovými vzory byla, co se týče klasifikace i počtu epoch lepší než topologie s 20-ti tréninkovými vzory.

8 ZÁVĚR

Touto prací jsem si ověřil, že příznaky typu *Win* (model 2) jsou vhodnější pro klasifikaci číslic než Příznaky typu *Proj* (model 1).

U obou modelů byla úspěšnost klasifikace na tréninkové množině 100%. Z toho usuzuji, že jsem do tréninkové množiny umístil "dobře rozpoznatelné" obrázky číslic.

U nejlepších topologií sítí modelu 1 jsem dosáhl úspěšnosti klasifikace 86,7% při testování na ověřovací množině (topologie *36-14-10*) a 45% při testování na problémové množině (topologie *36-16-10*) při učení s hodnotou momentu 0,8. Použitím vstupního souboru s větším počtem tréninkových vzorů se úspěšnost klasifikace zvyšovala. Z toho plyne, že použitím většího počtu tréninkových vzorů by se dalo dosáhnout ještě větší úspěšnosti klasifikace, ale z důvodů uvedených v práci to nebylo možné. Předpokládám, že úspěšnost klasifikace by se dala zvýšit také použitím jiných rozměrů normalizačního čtverce. Nečastěji byla na ověřovací množině chybně klasifikována či vůbec neklasifikována číslice 0 a 8, chybně klasifikovanými číslicemi na problémové množině jsem se vzhledem k jejich povaze nezabýval.

U nejlepší topologie sítě modelu 2 jsem dosáhl úspěšnosti klasifikace 100% při testování na ověřovací množině a 65% při testování na problémové množině při učení s hodnotou momentu 0,7. Jedná se o topologii *64-20-10*. U tohoto modelu byla úspěšnost klasifikace 100% na ověřovací množině při testování všech topologií. Použitím vstupního souboru s větším počtem tréninkových vzorů se mírně zvýšila úspěšnost klasifikace na problémové množině.

Nízké procento úspěšně klasifikovaných číslic z problémové množiny u obou modelů bylo způsobeno „hůře rozpoznatelnými" obrázky číslic umístěných v problémové množině, což bylo ostatně mým cílem při jejím návrhu.

Při snižování nebo zvyšování hodnoty momentu od ideální hodnoty docházelo k nárůstu počtu epoch až na nastavenou limitní hodnotu (500 epoch), což znamenalo, že se síť nebyla schopná naučit.

Bohužel jsem v této práci nezobrazil žádné průběhy učení sítě, jelikož tento modul programu *BPSIM* nefungoval.

Literatura

[1] Duda R., O., Hart P., E., Stork D., G. *Pattern Classification*, druhé vydání, Wiley, 2003.

[2] Baum L., E., Petrie T. *Statistical inference for probabilistic functions of finite state Markov chains*, Annals of Mathematical Statistic, 1966.

[3] The Robert Gordon university. *The Back Propagation Algorithm.* [online], URL: http://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf [citováno 17. května 2011]

[4] Novák M. a kolektiv: *Umělé neuronové sítě*. C. H. Beck, Praha, 1998

[5] Vala T. *Rozpoznávání SPZ z jednoho snímku*. Praha: Univerzita Karlova. Matematicko-fyzikální fakulta. 2006. 91 s. Vedoucí diplomové práce RNDr. Jana Štanclová.

[6] Doupovec Z. *Rozpoznávání číslic pomocí neuronové sítě.* Brno: Vysoké Učení Technické. Fakulta informačních technologií. Ústav počítačové grafiky a multimédií. 2009. 41 s. Vedoucí bakalářské práce Ing. Jana Šilhavá.

[7] Suchý V. *Rozpoznávání textu v obraze.* Brno: Vysoké Učení Technické. Fakulta informačních technologií. Ústav počítačové grafiky a multimédií. 2007. 32 s. Vedoucí bakalářské práce Ing. Michal Španěl.

[8] MARTÍNEK L., MIŠKA P. *Formální neuron, perceptron a ADALINE - aktivní fáze, adaptace.* [online], URL: http://neuron.felk.cvut.cz/courseware/data/chapter/36nan009/s03.html [citováno 17. května 2011]

[9] BP, *BPSIM* [počítačový soubor]. Verze 3.2.2011. KAMT FE VUT Brno: Pete Kasik, 1991. Počítačový program pro simulaci neuronových sítí. Vyžaduje Windows XP a nižší. Program poskytnut vedoucím bakalářské práce.

[10] *Manuál k simulačnímu programu BPSIM*. Brno: Pete Kasik, 1991. 31 s.

Seznam příloh

Příloha 1. Zdrojový text programu *bmp*

Příloha 2. CD obsahující:

- Elektronickou verzi bakalářské práce
- Obrázky sad použitých vzorů
- Zdrojový soubor programu *bmp* ve verzi spustitelné v programátorském prostředí Java
- program *BPSIM* obsahující uložené soubory příznakových vektorů pro oba typy příznaků

Příloha 1 - zdrojový text programu bmp

```
import java.awt.image.BufferedImage;
import java.awt.image.ColorModel;
import java.io.File;
import javax.imageio.ImageIO;

public class A {
   /*** @param args*/

          public static int SIRKA_OKNA = 4;
          public static int STEP_OKNA = 2;

   public static void main(String[] args) throws Exception {
                  File file = new File("C:\\Users\\Petr\\Desktop\\vzory\\a0.bmp ");

      BufferedImage img;img =
         ImageIO.read(file);
      int X = img.getWidth();
      int Y = img.getHeight();
      ColorModel model = img.getColorModel();
      System.out.println("X:" + X);
      System.out.println("Y:" + Y);
      int[][] bitmapa = new int[X][Y];
      for (int i = 0; i < Y; i++) {
         for (int j = 0; j < X; j++) {
            int pixel = img.getRGB(j, i);
            int r = model.getRed(pixel);
            int g = model.getGreen(pixel);
            int b = model.getBlue(pixel);
            double jas = 0.299 * r + 0.587 * g + 0.114 * b;
            int pix = (jas < 128) ? 1 :0;
            bitmapa[j][i] = pix;
            System.out.print(pix + "\t");
/* System.out.print(r + "-" + g + "-" + b + "-" + "\t");
*/
         }
         System.out.println();

      }
      System.out.println();
      for (int i = 0; i < Y; i++) {
          double soucet = 0.0;
         for (int j = 0; j < X; j++) {

            //System.out.print(bitmapa[j][i] + "\t");
```

```
            soucet += bitmapa[j][i];

        }
        //System.out.println();
        System.out.printf("%.2f ", soucet / X);
    }
    System.out.println();
    for (int j = 0; j < X; j++) {
         double soucet = 0.0;

        for (int i = 0; i < Y; i++) {
            //System.out.print(bitmapa[j][i] + "\t");
            soucet += bitmapa[j][i];

        }
        //System.out.println();
        System.out.printf("%.2f ", soucet / X);
    }

System.out.println("\n\n**********************************************************
******");
    for (int i = 0; i + SIRKA_OKNA <= Y; i+= STEP_OKNA) {
        if((i / STEP_OKNA) % 2 == 1) {
                //System.out.println("reverse:");
        }
        double[] radek = new double[X / STEP_OKNA];
        int jj = 0;
        for (int j = 0; j + SIRKA_OKNA <= X; j+= STEP_OKNA) {
        double soucet = 0.0;
                for (int l = 0; l < SIRKA_OKNA; l++) {
                for (int k = 0; k < SIRKA_OKNA; k++) {
                        soucet += bitmapa[j + k][i + l];
                                //System.out.print(bitmapa[j + k][i + l] + "\t");
                }
                        // System.out.println();
                }
                radek[jj] = soucet / (SIRKA_OKNA * SIRKA_OKNA);
                jj++;
                //System.out.println("prumer: " + soucet / (SIRKA_OKNA * SIRKA_OKNA));
            }
        if((i / STEP_OKNA) % 2 == 1) {
                for (int jjj = (jj - 1); jjj >= 0; jjj--) {
                    System.out.printf("%.2f ", radek[jjj]);
                }
        }
```

```
            else {
                    for (int jjj = 0; jjj < jj; jjj++) {
                            System.out.printf("%.2f ", radek[jjj]);
                    }
            }
                    //System.out.println("\n--------------------");
        }
    }
}
```