

UNIVERZITA KOMENSKÉHO V BRATISLAVE  
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

6ab5acd0-edab-4873-972a-e8ae50b302c1

# APLIKOVANÉ VYUŽITIE NEURÓNŮVÝCH SIETÍ

Bratislava, 2011

DOMINIKA FEDÁKOVÁ



KATEDRA INFORMATIKY

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

UNIVERZITA KOMENSKÉHO V BRATISLAVE

---

# APLIKOVANÉ VYUŽITIE NEURÓNOVÝCH SIETÍ

(Bakalárska práca)

---

**Študijný program:** Informatika

**Študijný odbor:** 9.2.1 Informatika

**Školiace pracovisko:** Katedra Informatiky

**Školiteľ:** Mgr. Ing. Michal Šefara

Bratislava, 2011

Dominika Fedáková

### ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Dominika Fedáková  
**Študijný program:** informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)  
**Študijný odbor:** 9.2.1. informatika  
**Typ záverečnej práce:** bakalárska  
**Jazyk záverečnej práce:** slovenský

**Názov:** Aplikované využitie neurónových sietí

**Cieľ:** Neurónové siete sa dajú využiť na riešenie rozličných problémov. Úlohou študenta je preniknúť do problematiky neurónových sietí a naštudovať teóriu tréningu neurónovej siete a tréningových algoritmov. Následne študent využije nadobudnuté poznatky na riešenie reálneho problému s využitím neurónovej siete.

**Literatúra:** 1. Kvasnička, V., Beňušková, E., Pospíchal, J., Farkaš, I., Tiňo, P., Kráľ A.: Úvod do teórie neurónových sietí  
2. Kačmáry, P., Malindžák, D.: Prognózovanie obchodu a výroby v dynamicky sa meniacich podmienkach trhu. Acta Montanistica Slovaca, 15 ročník (2010), mimoriadne číslo 1, s. 53-60  
3. Dereník Dávid: *Matematické modely v technickej analýze cenných papieroch*, Brno, 2006

**Vedúci:** Ing. Michal Šefara  
**Katedra:** FMFI.KAGDM - Katedra algebry, geometrie a didaktiky matematiky

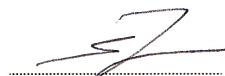
**Dátum zadania:** 29.10.2010

**Dátum schválenia:** 02.11.2010

  
doc. RNDr. Daniel Olejár, PhD.  
garant študijného programu



študent



Vedúci

Čestne prehlasujem, že bakalársku prácu som vypracovala samostatne s použitím uvedenej literatúry a pod dohľadom môjho vedúceho práce.

.....

# Pod'akovanie

Chcela by som poďakovať môjmu vedúcemu Mgr. Ing. Michalovi Šefarovi za jeho veľkú pomoc, za inšpiratívne rady a za dohľad nad mojou činnosťou.

# Abstrakt

**Autor:** Dominika Fedáková

**Názov bakalárskej práce:** Aplikované použitie neurónovej siete

**Škola:** Univerzita Komenského v Bratislave

**Fakulta:** Fakulta matematiky, fyziky a informatiky

**Katedra:** Katedra Informatiky

**Vedúci bakalárskej práce:** Mgr. Ing. Michal Šefara

**Rozsah práce:** 37 strán

**Bratislava, jún 2011**

Práca sa zaoberá aplikovaním konkrétneho problému na neurónové siete. Hľadáme ideálny typ siete a najvhodnejšiu architektúru pre daný problém. Po naštudovaní teórie testujeme rôzne programovacie prostredia a učiace algoritmy. Pre najvhodnejšiu neurónovú sieť naprogramujeme pomocné funkcie, ktoré uľahčia prácu s touto sieťou v budúcich projektoch. Ako ukážku doprogramujeme konkrétne GUI prostredie s aplikáciou novo naprogramovaných funkcií.

**KĽÚČOVÉ SLOVÁ:** Neurónová sieť, Back-propagation algoritmus, Levenberg-Marquardt, Resilient propagation

# Abstract

**Author:** Dominika Fedáková

**Thesis title:** Aplikované použitie neurónovej siete

**University:** Univerzita Komenského v Bratislave

**Faculty:** Fakulta matematiky, fyziky a informatiky

**Department:** Katedra Informatiky

**Advisor:** Mgr. Ing. Michal Šefara

**Thesis length:** 37 pages

**Bratislava, june 2011**

The thesis is focusing on applying concrete problem to the neural network. We look for the ideal type and architecture of network for the problem. After studying the theory we test the different programming environments and learning algorithms. For the best neural network we write some supporting functions that will make working with this network in future projects easier. We also write code for GUI environment with applied new programmed functions.

**KEYWORDS:** Neural network, Back-propagation algorithm, Levenberg-Marquardt, Resilient propagation

# Obsah

Úvod	1
<b>1 Základy nervovej sústavy a neurónovej siete</b>	<b>3</b>
1.1 Ľudský mozog . . . . .	3
1.2 Formálna definícia neurónovej siete . . . . .	5
1.3 História neurónových sietí . . . . .	7
<b>2 Typy neurónových sietí</b>	<b>9</b>
2.1 Jednovrstvové perceptróny . . . . .	9
2.2 Viacvrstvové perceptróny . . . . .	11
2.2.1 Back-propagation algoritmus (Algoritmus spätne šíriacej sa chyby)	13
2.3 Rekurentné neurónové siete . . . . .	14
2.4 Hopfieldove siete . . . . .	16
2.5 Samoorganizujúce sa (Kohonenove) mapy . . . . .	18
2.6 Evolučné neurónové siete . . . . .	18
<b>3 Návrh implementácie</b>	<b>20</b>
3.1 Charakteristika problému . . . . .	20



3.2	Metodika práce . . . . .	21
<b>4</b>	<b>Vyhodnotenie účinnosti a efektívnosti</b>	<b>23</b>
4.1	Riešenie v C++ . . . . .	23
4.2	Riešenie v Matlabe . . . . .	25
4.3	Riešenie v Jave . . . . .	26
4.4	Popisy naprogramovaných funkcií . . . . .	28
4.4.1	Načítavanie dát . . . . .	28
4.4.2	Normalizovanie dát . . . . .	29
4.4.3	Denormalizovanie dát . . . . .	29
4.4.4	Vytváranie novej siete . . . . .	30
4.4.5	Požítanie výstupu . . . . .	31
4.4.6	Čítanie vstupov zo súboru a zapisovanie výstupov do súboru .	31
4.4.7	Zaokrúhľovanie na dve desatinné miesta . . . . .	32
4.5	Využitie funkcií vo vytvorenom GUI prostredí . . . . .	32
<b>5</b>	<b>Diskusia</b>	<b>34</b>
5.1	Porovnanie rôznych algoritmov . . . . .	34
	<b>Záver</b>	<b>36</b>
	<b>Literatúra</b>	<b>38</b>
	<b>Prílohy</b>	<b>40</b>
	Príloha A: Zdrojové kódy . . . . .	41

Príloha B: Sreenshoty GUI aplikácie . . . . .	43
Príloha C: DVD so štruktúrou . . . . .	45

# Zoznam obrázkov

1.1	Znázornenie iónov v synapse [2, s.5] . . . . .	4
1.2	Model neurónov a ich spojení [5, s.35] . . . . .	6
2.1	Lineárne rozdelenie na 2 triedy [5, s.160] . . . . .	9
2.2	Perceptrón [5, s.158] . . . . .	10
2.3	Maelyho automat [2, s.121] . . . . .	16
2.4	Hopfieldova sieť so 4 neurónmi [5, s.45] . . . . .	17
4.1	Zdrojový kód funkcie <i>normalize()</i> . . . . .	29
4.2	Zdrojový kód funkcie <i>denormalize()</i> . . . . .	29
4.3	Zdrojový kód funkcie <i>round()</i> . . . . .	32
5.1	Graf priebehu veľkosti chyby MSE počas učenia siete [5, s.35] . . . . .	35
5.2	Zdrojový kód funkcie <i>create_RPROP()</i> . . . . .	41
5.3	Zdrojový kód funkcie <i>recognize_output()</i> . . . . .	42
5.4	Screenshotty vzorového GUI prostredia (1) . . . . .	43
5.5	Screenshotty vzorového GUI prostredia (2) . . . . .	43
5.6	Screenshotty vzorového GUI prostredia (3) . . . . .	44

# Úvod

Nachádzame sa v dobe, kedy počítače dokážu nahradiť prácu človeka. Vieme naprogramovať funkcie, ktoré počítač dokáže spracovať rýchlejšie a presnejšie ako človek. Ale inšpirácia na použitie modelu ľudského mozgu pre tvorbu programu nám dovoľuje program aj niečo naučiť a nielen nechať ho vykonávať presné inštrukcie. Medzi takéto modely patria neurónové siete. My im poskytneme dostatok vzorov (úlohy a ich riešenia), z ktorých sa môžu učiť daný problém, a siete sa časom naučia rozpoznávať a sami riešiť úlohy daného typu. Teda nie len že vedia odpovede na sadu úloh, ktoré sme im mi dali na naučenie, ale dokážu s veľkou presnosťou nájsť riešenie aj na úlohy tohto problému, ktoré ešte nevideli. Neurónové siete môžeme využiť aj v mnohých procesoch v bussines svete, aby sme čo najviac zredukovali manuálnu prácu, uľahčili a ušetrili čas pracovníkov. Je teda efektívne zautomatizovať problémy, ktoré sa počítač dokáže naučiť.

Cieľom práce bolo naštudovať a spracovať hrubú teóriu neurónových sietí. V dôsledku zistenia dostatočných informácií o týchto sieťach sme vybrali vhodný model pre riešenie nami určeného typu problémov, ktorých riešením sme sa zaoberali. Hlavnou úlohou bolo testovanie rôznych typov neurónových sietí s rozličnými tréningovými algoritmi a porovnanie výsledkov dosahovaných jednotlivými sieťami. Testovali sme neurónové siete s rôznymi parametrami a zisťovali, ktorá architektúra a parametre neurónovej siete sú najvhodnejšie.

Konkrétny problém, ktorému sme sa venovali, sa týka zisťovania času potrebného na realizáciu projektu navrhnutého firme. Každý projekt musí byť ohodnotený vhodnými premennými (vsupné dáta pre aplikáciu), následne neurónová sieť vyhodnotí tieto premenné a určí správny čas potrebný pre tento projekt. Tento proces je

veľmi rýchly a pri vhodnej neurónovej sieti aj dostatočne presný. Teda takto môžu byť projekty ohodnotené rýchlejšie, efektívnejšie a presnejšie, ako keby to mal prehodnocovať pracovník firmy. Okrem toho, závislosti medzi parametrami a časom projektu sú nelineárne, preto by musel pracovník firmy intuitívne odhadovať časy projektov pomocou porovnávania s podobnými už vypracovanými projektami. To ale zaberá mnoho času, a taktiež pri každom novom projekte je potrebné nanovo vyhľadávať podobné, už ohodnotené projekty. Tiež je potrebné myslieť na to, že na takéto aspoň z časti presné vyhodnocovanie časov je potrebná prax a skúsenosti s takýmito projektmi. Preto je to veľmi náročné pre nových, prípadne neskúsených zamestancov. Na druhej strane neurónová sieť dokáže z veľkého počtu dát nájsť závislosti, ktoré pre človeka môžu byť veľmi ťažko poznateľné, a preto bude vyhodnocovanie pomocou neurónovej siete lepšie. Využívanie aplikácie s integrovanou neurónovou sieťou bude nenáročné, skladajúce sa len z niekoľkých krokov, preto to môže využívať každý, kto dokáže vhodne určiť parametre projektov. Je však potrebné, aby bolo vyhodnotenie projektu čo najpresnejšie, a preto použijeme architektúru a parametre, ktoré budú vykazovať najlepšie výsledky. Tento problém je príkladom využitia neurónových sietí v praxi, pričom výsledky práce sa budú môcť použiť na riešenie podobných typov úloh<sup>1</sup>.

---

<sup>1</sup>Úloha, ktorá zisťuje závislosti medzi vstupnými premennými a výstupnou premennou. Musí obsahovať práve jeden výstup a ľubovoľný počet vstupov. Zameriavame sa na úlohy s menším počtom tréningových vzoriek, ale ak nie je povedané inak, neurónové siete pracujú lepšie s väčším počtom vzoriek.

# Kapitola 1

## Základy nervovej sústavy a neurónovej siete

### 1.1 Ľudský mozog

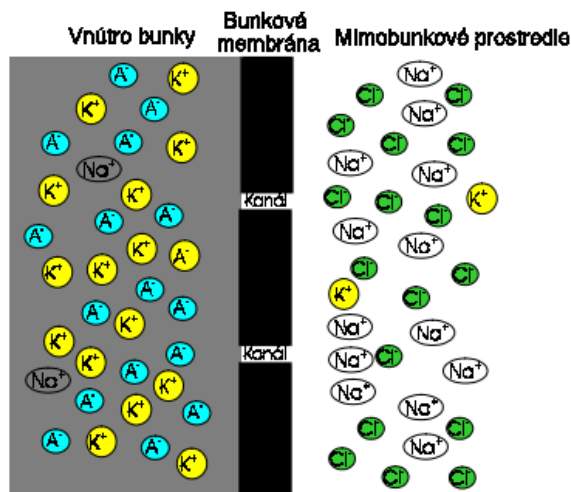
Ľudský mozog je veľmi komplexný. Paralelne spracúvava množstvo informácií. Vďaka veľkej výpočtovej sile nášho mozgu a hlavne kvôli vlastnosti učenia sa, inšpirovaný stavbou mozgu, bol zostavený model neuronových sietí bežiacich na počítačoch, ktoré dosahujú podobné vlastnosti ako náš mozog. Teda vie si zapamätať veľké množstvo informácií, ktoré mu poskytneme, a tieto informácie dokáže spracovávať, a teda si vyvinúť schopnosť učenia.

Základom procesu spracovávania informácií sú neuróny. Neuróny si medzi sebou prenášajú informáciu, spracujú ju a pošlú ďalej, až kým sa informácia nedostane k výstupným neurónom. I keď je rýchlosť neurónov veľmi pomalá, vďaka veľkému počtu neurónov a vysokému paralelizmu, dosahuje stále náš mozog vo väčšine oblastí lepšie výsledky ako počítače.

Neuróny sa skladajú z troch základných častí, telo bunky, dendrity a jeden axón. Dendrity sú zodpovedné za vstup informácie a axón slúži na šírenie informácie z bunky do iných buniek. Pri kontaktoch dvoch buniek vznikajú funkčné spojenia, nazývané synapsy. V synapse vzruch prechádza presynaptickým terminálom. Nasle-

duje niekoľko procesov spôsobených napätím a procesy vyúsťujú k otvoreniu iónových kanálov. Nimi prechádzajú kladné ióny a dochádza k depolarizácii alebo hyperpolizácii (Obr.1).

Obr. 1.1: Znázornenie iónov v synapse [2, s.5]



Pri otvorení kanálov, ktoré umožňujú prepúšťanie sodíka a kalcia do bunky, vzniká depolarizácia. Vďaka tomu, že sú to kladné ióny, znižuje sa záporný vnútro-bunkový potenciál, teda membránový potenciál sa posúva ku kladným hodnotám. Na druhej strane pri hyperpolizácii sa otvárajú draslíkové kanály. Kladné draslíkové ióny prúdia z bunky von, čo spôsobuje zvýšenie záporného membránového potenciálu, teda posun membránového potenciálu k negatívnym hodnotám vzhľadom na pokojový potenciál [2, s.4-7].

Tiež môže dochádzať k otvoreniu chloridových kanálov, a vtedy sa membránový potenciál dostáva na hodnoty pokojového potenciálu, čiže je to veľmi podobné hyperbolizácii. Pri depolarizácii vzniká excitačný postsynaptický potenciál, ktorý sa ďalej šíri pasívne. Pri ďalšom šírení sa znižuje amplitúda a dochádza k utlmovaniu takéhoto vzruchu. Keď sa membránový potenciál dostane na kritické hodnoty, dosiahne prah excitácie, dôjde k zatvoreniu sodíkových kanálov, pričom sa otvoria draslíkové kanály. Je prerušená možnosť depolarizácie a nastáva tzv. repolarizácia, kedy sa membránový potenciál dostáva späť na pôvodné hodnoty. Takto vzniká akčný potenciál (ner-

vový vzruch). Akčný potenciál sa vždy šíri smerom k axónu a nikdy sa nevracia, pretože bunková membrána je v tzv. refraktérnej fáze, čo znamená, že nevedie akčný potenciál. Teda ak sa spätne pozrieme na možnosť hyperbolizácie, po nej je malá pravdepodobnosť nastatia akčného potenciálu, keďže sa potenciály sčítavajú. Takéto hyperbolizácie voláme inhibičné postsynaptické potenciály. Zvyčajne je potrebné na vyvolanie akčného potenciálu viacero excitčných postsynaptických vzruchov z viacerých zdrojov, alebo z jedného zdroja, kde vzruchy nasledujú po sebe vo veľmi krátkych časových intervaloch [2, s.8-10].

Procesy na neurónoch sú veľmi komplikované. Sú asynchrónne a prenos vzruchu môže mať dlhodobé alebo krátkodobé účinky [2, s.12-13]. Jeden neurón môže mať niekoľko tisíc synáps, a vzdialenosť synapsy tiež určuje, aký veľký má synapsa vplyv. Preto je zatiaľ veľmi ťažké porovnávať výpočtové sily počítačových neurónových sietí k ľudskej nervovej sústave.

## 1.2 Formálna definícia neurónovej siete

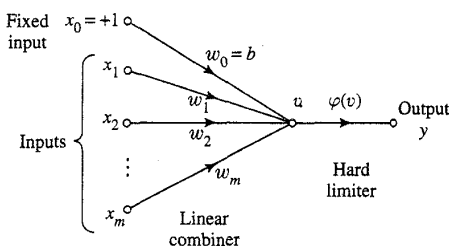
Model neurónovej siete môžeme opísať pomocou orientovaných grafov s presne danými pravidlami. Dá sa na ňom pekne ukázať ako prúdia informácie v neurónovej sieti a aká je základná architektúra siete. Začneme tým, že máme množinu vrcholov  $V$ , ktoré sú poprepájané orientovanými hranami  $E$ , ktoré budeme označovať ako usporiadanú dvojicu vrcholov  $(i, j)$ . Vrcholy predstavujú jednotlivé neuróny. Rozdeľujeme ich na vstupné vrcholy (neuróny, ktoré dostávajú informáciu z vonkajšieho prostredia), skryté vrcholy (neuróny, cez ktoré prúdia signály a spracúvajú informáciu) a výstupné vrcholy (neuróny, kde sa dostane výsledná spracovaná informácia). Celú sieť môžeme rozdeliť na vstupnú vrstvu, niekoľko skrytých vrstiev a výstupnú vrstvu. Hrany predstavujú synapsy medzi neurónmi. Každý vrchol má vstupný signál  $x_i$ , a po hrane  $(i, j)$  posiela signál do ďalšieho vrchola. K hranám sú priradené transformačné funkcie, ktoré hovoria akým spôsobom signál  $y_j$  závisí od signálu  $x_i$ . Signál putuje vždy len v orientácii hrany, nikdy nie opačným smerom. Túto vlastnosť pozorujeme aj v ľudskom mozgu.



Rozlišujeme dva druhy spojov medzi vrcholmi[2, s.72-73][5, s.37-42] :

1. Synaptické spoje - ide o lineárnu input-output väzbu, kde pôvodný signál  $x_i$  prenasobíme váhou synapsy  $w_{ij}$ , a tým dostaneme výsledný signál  $y_j$ .
2. Aktivačné spoj - nelineárna input-output väzba, kde  $y_j$  dostaneme dosadením signálu  $x_i$  prenasobený  $w_{ij}$  do aktivačnej funkcie.

Obr. 1.2: Model neurónov a ich spojení [5, s.35]



Výsledný signál na vrchole dostaneme sčítaním všetkých signálov, ktoré doňho smerujú. Teda signály najprv prenasobíme váhami synaps, potom ich sčítavame, pričom ešte odčítame prah excitácie, a tým získame membránový potenciál. Dosadíme ho do aktivačnej funkcie a dostaneme výsledný signál na danom vrchole [2, s.10].

$$y_j = \varphi \left( \sum_{i=0}^n w_{ij} x_i - \vartheta_j \right) \quad (1.1)$$

Každý vrchol (neurón) je reprezentovaný sadou lineárnych synaptických spojov, aktivačným spojom, ktorý môže byť nelineárny, a tiež možnosťou použitia pseudo inputu volaného bias. Predstavuje synaptický spoj inputu, ktorý je stále daný hodnotou +1. Požíva sa v prípade, keď na vstup dostaneme samé nuly, teda váhy na synapsách ostanú nezmenené a sieť sa nedokáže naučiť tento vzor. Touto funkciou opíšeme procesy na synpsách aj vo vnútri neurónu.

## 1.3 História neurónových sietí

Za začiatok modernej éry neurónových sietí považujeme priekopnícku prácu psychiatra a neuro-anatóma McCullocha a matematického génia Pittsa z roku 1943. V tejto prevratnej práci opisujú logické výpočty neurónových sietí, ktoré zjednocovali fyziológiu nervovej sústavy a matematickú logiku. Ich formálny model neurónu sa riadil pravidlom „všetko alebo nič“. Ukázali, že takto vytvorený systém (neurónová sieť), s dostatočným počtom takýchto jednotiek (neurónov) a ich správnym prepojením (zvolením synáps), dokáže v podstate vypočítať akúkoľvek vypočítateľnú funkciu. V prípade, že by takáto sieť bola neobmedzená, teda nekonečne veľká, ukázali ekvivalenciu s turingovým strojom. Táto práca ukázala významné výsledky, na ktorých ďalší mohli stavať nové poznatky ohľadne neurónových sietí a umelej inteligencie. Matematik von Neumann v roku 1956 zaviedol redundanciu, čo znamená, že na excitácii neurónu sa podieľa viacero neurónov. Presnejšie, ak má aspoň polovica neurónov podieľajúcich sa na aktivácii tohto neurónu hodnotu 1, v tom prípade sa neurón aktivuje [5, s.60].

V roku 1948 vyšla Wienerova kniha „Cybernetics“, ktorá opisovala spôsoby kontroly, komunikácie a spracovania štatistických signálov. V druhej edícii sa zameral na učenie a samo-organizáciu. Ďalším významným míľnikom v obore neurónových sietí bola Hebbova kniha „The organization of behaviour“ z roku 1949. Doteraz sa mnohí odvolávajú na jeho pravidlo, psychologické učiace pravidlo pre modifikáciu synáps: „When an axon of cell A ... excite(s) cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells so that A's efficiency as one of the cells firing B is increased“ [2, s.33], teda toto pravidlo hovorí o tom, že ak sa neurón A podieľa na excitácii neurónu B, zvyšuje sa jeho účinnosť v budúcich šíreniach sa signálu. Hebbova kniha bola inšpiráciou pre vytvorenie prvých modelov sietí, ktoré sa učia a adaptujú [5, s.60].

V roku 1958 Rosenblatt popísal siete s novým prístupom k riešeniu klasifikačných problémov. Tieto siete nazval perceptróny. Avšak v roku 1969 Minsky a Papert matematicky dokázali základné obmedzenia týchto sietí. Jednvrstvový perceptrón dokáže riešiť iba lineárne separovateľné problémy (napr. AND, OR) a nie je dokázané, že viacvrstvový perceptrón dokáže tento problém vyriešiť. Tieto problémy

boli vyriešené až v 90-tych rokoch 20. storočia, s príchodom metódy spätne šíriacej sa chyby. V ďalších rokoch boli vymyslené ešte mnohé ďalšie metódy riešenia sietí a špecializované siete na konkrétne typy problémov [2, s.35-37].

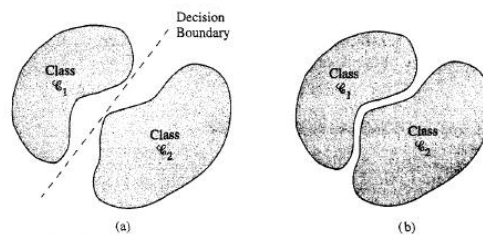
# Kapitola 2

## Typy neurónových sietí

### 2.1 Jednovrstvové perceptróny

Perceptrón je najjednoduchšia forma neurónovej siete, používaná na klasifikáciu vzorov, ktoré sú lineárne separovateľné (Obr.6), napr. booleanovský AND, OR, ... (problém XOR nie je lineárne separovateľný).

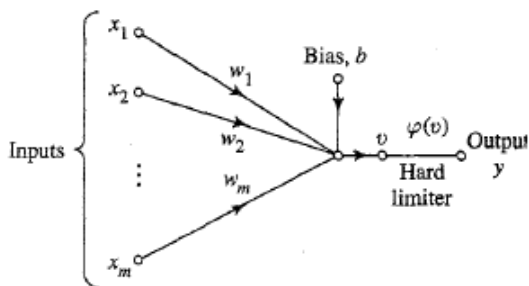
Obr. 2.1: Lineárne rozdelenie na 2 triedy [5, s.160]



Pozostáva z jedného neuróna s prispôsobiteľnými synaptickými váhami a doplnkového bias neurónu (bias neurón je doplnkový vstupný neurón s hodnotou +1. Ak si predstavíme lineárne separovateľné problémy na grafe, práve „bias“ nám umožňuje aby priamka rozdeľujúca dáta nemusela prechádzať stredom grafu). Algoritmus na

prispôsobovanie voľných parametrov tejto neurónovej siete sa prvýkrát objavil v učiacej procedúre vypracovanej Rosenblattom (1958,1962) pre jeho perceptrónový model mozgu. Perceptrón postavený z jedného neuróna je limitovaný na klasifikovanie vzoriek iba s dvoma triedami. Ak by sme chceli perceptrón použiť na klasifikáciu vzoru s viacerými triedami, musíme rozšíriť výstupnú („output“) vrstvu siete na viacero neurónov. Avšak problém musí byť stále lineárne separovateľný, aby perceptrón pracoval správne. Takýto neurónový model obsahuje lineárne sčítavanie nasledované tvrdým obmedzovačom, ktorý je uskutočňovaný funkciou signum. Sčítavacie uzly počítajú lineárnu kombináciu vstupov a pripájajú externý bias neurón. Na výslednú sumu je aplikovaný tvrdý obmedzovač, teda dostávame hodnotu v intervale  $\langle -1, 1 \rangle$ . Výstup je rovný hodnote  $+1$ , ak nám obmedzovač vráti kladnú hodnotu, výstup je  $-1$ , ak nám obmedzovač vráti zápornú hodnotu. To rozhodne, do ktorej z dvoch tried sa zaraďuje vstup [5, 157-165] [2, s.57].

Obr. 2.2: Perceptrón [5, s.158]



$$y = \varphi(v) = \varphi\left(\sum_{i=1}^m w_i x_i + x_{m+1} b\right) \quad (2.1)$$

## 2.2 Viacvrstvé perceptróny

Hovoríme o viacvrstvových dopredných („feedforward“) neurónových sieťach. Sieť pozostáva zo sensorických jednotiek (vstupné uzly), ktoré vytvárajú vstupnú vrstvu jednej alebo viacerých skrytých vrstiev s počítajúcimi uzlami a výstupnej vrstvy s tak tiež rátającimi uzlami. Signál sa šíri sieťou smerom dopredu cez jednotlivé vrstvy. Tieto siete sa bežne označujú aj ako viacvrstvé perceptróny, teda reprezentujú zovšeobecnenie jednovrstvových perceptrónov [5, s.178].

Tieto siete boli úspešne použité na riešenie zložitých a rôznorodých problémov, pomocou metódy učenia s učiteľom (sieti je daná vzorka vstupov a výstupov, podľa ktorých sa sieť bude učiť) a populárneho back-propagation algoritmu. V praxi sa dopredné siete so skrytými vrstvami používajú na riešenie klasifikačných problémov (problémy zaradenia vstupu do správnych výstupných tried, pričom na tréovanie dostaneme podmnožinu priestoru vstupov a výstupov) a na aproximovanie spojitých funkcií, teda sú dobrým prostriedkom na regresnú analýzu [2, s.70-72]. Algoritmus back-propagation (alebo gradientový algoritmus) je založený na „error-correction“ učiacom pravidle, teda učenie podľa chyby siete. Učenie pozostáva z dvoch prechodov cez rôzne vrstvy siete [5, s.178]:

- **Prechod dopredu („forward pass“)** – vstupný vektor je aplikovaný na vstupnú vrstvu a ďalej sa šíri vrstvami siete. Nakoniec sú vypočítané výstupné hodnoty siete. Počas tohto prechodu sú váhy na synapsách pevne dané a nemenia sa.
- **Prechod späť („backward pass“)** – Všetky váhy sú prispôsobované podľa pravidla opravovania chyby. Odčítaním výstupných hodnôt siete od chceného výstupného vektora k danému vstupnému vektoru, získame chybový signál. Tento signál putuje sieťou v spätnom smere od výstupnej vrstvy k vstupnej. Pri prechode cez jednotlivé synapsy sa upravujú váhy, aby pri ďalšom prechode dopredu podávali lepšie výsledky.

Vo viacvrstvovom perceptróne model každého neurónu obsahuje nelineárnu aktivačnú funkciu, nelineárnosť musí byť hladká. Najpoužívanejšou je sigmoidálna nelineárnosť, daná logaritmickou funkciou [5, s.179]:

$$y_j = \frac{1}{1 + e^{-\vartheta_j}} \quad (2.2)$$

$\vartheta_j$  označuje váhovaný súčet všetkých synaptických vstupov pre neurón  $j$  plus neurónu bias.  $y_j$  predstavuje výstup neurónu  $j$ , ktorý je v rozmedzí  $< 0, 1 >$  (alebo sa môže použiť hyperbolický tangens, s výstupom v rozmedzí  $< -1, 1 >$ ). Nelineárnosť je veľmi dôležitá, pretože bez nej by sa tieto viacvrstvové siete redukovali na jednovrstvové perceptróny (keďže kombináciou lineárnych úprav vstupu, dostaneme lineárnu úpravu  $(V.(W.\vartheta) = V.W(\vartheta))$ ).

Sieť obsahuje niekoľko skrytých vrstiev, ktoré nepatria ani vstupu ani výstupu. Umožňujú sieti naučiť sa komplexnejšie problémy tým, že extrahujú viacero dôležitých črt z vstupných vektorov.

Každý neurón má priradenú svoju hodnotu  $x_i$  a prahový koeficient  $\zeta_i$ , hrany medzi neurónmi sú ohodnotené váhami  $w_{ij}$ .  $\varphi$  predstavuje aktivačnú funkciu. Výslednú aktivitu skrytých a výstupných neurónov rátame nasledovne [2, s.74]:

$$x_j = \varphi \left( \sum_{i=0}^n w_{ij}x_i + \zeta_j \right) \quad (2.3)$$

Nie je nutne povedané, že pri rátaní aktivity neurónov musíme použiť iba konštanty a lineárne členy. Môžeme tu zaradiť kvadratické, kubické alebo členy vyššieho rádu. Tým dostaneme neurónovú sieť vyššieho rádu, napr. potenciál z neurónov predošlej vrstvy  $i$  a  $k$ , ktorý vložíme do aktivačnej funkcie neurónu  $j$  za použitia kvadratických členov vyzerať nasledovne [2, 76]:

$$\zeta_j + w_{ij}x_i + w_{kj}x_k + w_{iij}x_i^2 + w_{kkj}x_k^2 + w_{ikj}x_ix_k \quad (2.4)$$

### 2.2.1 Back-propagation algoritmus (Algoritmus spätne šíra- cej sa chyby)

Chybový signál výstupu neurónu  $j$  pri iterácii  $n$  ( prezentovanej  $n$ -tým príkladom tréningovej množiny dát) je definovaný ako [5, s.183]:

$$e_j(n) = d_j(n) - y_j(n) \quad (2.5)$$

pričom  $y_j(n)$  predstavuje hodnotu  $j$ -teho výstupného neurónu a  $d_j(n)$  predstavuje chcenú výstupnú hodnotu podľa tréningových dát. Definujeme okamžitú hodnotu chybovej energie neurónu  $j$  ako  $\frac{1}{2e_j^2(n)}$  a následne okamžitú hodnotu  $E(n)$  celkovej chybovej energie získame súčtom jednotlivých chybných energií všetkých neurónov vo výstupnej vrstve [5, s.183]:

$$E(n) = \frac{1}{2} \sum_{j=0}^n e_j^2(n) \quad (2.6)$$

Priemernú kvadratickú chybnú energiu získame sčítaním  $E(n)$  cez všetky vzory (príklady) v tréningových dátach vzhľadom na veľkosť týchto dát  $N$  [5, s.183]:

$$E_{av} = \frac{1}{n} \sum_{n=0}^N E(n) \quad (2.7)$$

$E_{av}$  predstavuje hodnotovú funkciu ako mieru na učenie. Cieľom učenia je prispôbovanie voľných parametrov, aby bola celková chyba siete  $E_{av}$  minimálna. Algoritmus back-propagation aplikuje korekciu  $\Delta w_{ij}(n)$  synaptickej váhy  $w_{ij}(n)$ , ktorá je proporcionálna k parciálnej derivácii  $\frac{\partial E(n)}{\partial w_{ij}(n)}$ . Podľa reťazového pravidla výpočtu („chain rule of calculus“) môžeme tento gradient vyjadriť ako [5, s.184]:

$$\frac{\partial E(n)}{\partial w_{ij}(n)} = \frac{\partial E(n)}{\partial e_i(n)} \frac{\partial e_i(n)}{\partial y_i(n)} \frac{\partial y_i(n)}{\partial \vartheta_i(n)} \frac{\partial \vartheta_i(n)}{\partial w_{ij}(n)} \quad (2.8)$$

Parciálna derivácia  $\frac{\partial E(n)}{\partial w_{ij}(n)}$  reprezentuje faktor citlivosti, ktorý rozhoduje smer hľadania synaptickej  $w_{ij}$  váhy v priestore váh.



Deriváciou a kombináciou rovníc dostaneme [5, s.185]:

$$\frac{\partial E(n)}{\partial w_{ij}(n)} = -e_i(n) \varphi'(\vartheta_i(n)) y_i(n) \quad (2.9)$$

Korekcia  $\Delta w_{ij}(n)$  aplikovaná váhe  $w_{ij}(n)$  je definovaná pomocou pravidla delta, kde  $\eta$  predstavuje parameter učiacej rýchlosti („learning rate“) siete a mínus predstavuje gradientový spád v priestore váh [5, s.185]:

$$\Delta w_{ij}(n) = -\eta \frac{\partial E(n)}{\partial w_{ij}(n)} \quad (2.10)$$

Keď dosadíme rovnicu (2.9) do rovnice (2.10), dostaneme [5, s.185]:

$$\Delta w_{ij}(n) = \eta \delta_i(n) y_i(n) \quad (2.11)$$

Kde lokálny gradient  $\delta_i(n)$  je definovaný ako [5, s.185]:

$$\delta_i(n) = e_i(n) \varphi'_i(\vartheta_i(n)) \quad (2.12)$$

Lokálnu gradient ukazuje na potrebné zmeny v synaptických váhach. Podľa rovnice (2.12), lokálny gradient  $\delta_i(n)$  pre výstupný neurón  $i$  je ekvivalentný súčinu korešpondujúceho chybového signálu  $e_i(n)$  a derivácie  $\varphi'_i(\vartheta_i(n))$  asociovej aktivačnej funkcie. Pomocou  $\Delta w_{ij}(n)$  budeme teda rekurentne upravovať synaptické váhy, až pokiaľ nedostaneme dostatočne malú priemernú kvadratickú chybu  $E_{av}$ , s ktorou budeme spokojní [5, s.186].

## 2.3 Rekurentné neurónové siete

Rekurentné neurónové siete sú neurónové siete s jedným alebo viacerými cyklami (spätnými slučkami). Spätná väzba môže byť lokálna (vcelku jednoduchá záležitosť, napríklad na úrovni jedného neurónu v sieti) alebo globálna (zahŕňa celú sieť). V tejto

podkapitole sme sa venovali rekurentným neurónovým sieťam s globálnou spätnou väzbou, keďže tie majú hlbší význam. Na aplikáciu spätnej väzby si však treba dávať väčší pozor. Ak je spätná väzba zle umiestnená, môže spôsobovať nežiadúce účinky a zo stabilného systému sa stane nestabilný. Téma neurónových sietí, videná ako nelineárne dynamické systémy s osobitným dôrazom na problém stability systému, sa nazýva neurodynamika („neurodynamics“) [5, s.686]. Viac o tejto téme sa môžete dozvedieť v knihe *Neural Networks, A comprehensive Foundation, Second Edition* (Simon Haykin), v kapitole 14.

Vezmime viacvrstvový perceptrón ako stavebný blok. Aplikácia globálnej spätnej väzby môže mať viacero foriem. Môžu sa poskytnúť spätné informácie z výstupných neurónov vstupným neurónom. Tiež sa môžu poskytovať spätné informácie zo skrytých neurónov vstupným neurónom. Pri viacerých skrytých vrstvách sa počet možností spätných slučiek zvyšuje. Teda rekurentné siete majú bohatý repertoár architektonických rozložení.

Predošlé typy neurónových sietí bohužiaľ nepokrývajú riešenia na všetky druhy problémov. Problém, ktorý ani nelineárna viacvrstvá sieť nedokáže riešiť, nastane napríklad, ak chceme spracovávať dáta, ktoré sú rôzne v závislosti od časovej jednotky (záleží, kedy boli dané na vstup), napr. dáta tvaru [2, s.118-120]:

$$A \rightarrow a, B \rightarrow b, C \rightarrow c, A \rightarrow c, B \rightarrow a, A \rightarrow a$$

V reálnom živote si to môžeme ukázať na príklade ceny akcií na burze. Môžeme si určiť parametre, ktoré by v sieti tvorili základ rátania ceny akcie, ale rovnaký vstup (teda aj keď má akcia rovnaké premenné) v rôznych časoch produkuje rôzne výstupy (rôzne ceny akcií) [3, s.6]. Treba dbať aj na časovú jednotku vstupu. Tento problém riešia siete s časovým posunom („Time Delay Neural Network“, TDNN), keď majú k dispozícii v jednom kroku posledných D vzoriek vstupov. Takýto typ siete môžeme realizovať pomocou vyššie popísanej doprednej siete so spätnou väzbou, ak k nej pridáme náhľad do D starých vstupov, pričom dáta musia byť správne časovo usporiadané. Existujú aj časovo – priestorové úlohy, s ktorými si TDNN nevie poradiť, napr. Maelyho automat (Obr. 2.3) [2, s.119-121].

Obr. 2.3: Maelyho automat [2, s.121]



Tu by sme mali vyžiť stavovú reprezentáciu časového kontextu. To využíva rekurentná sieť, skladajúca sa z asociačnej siete a stavovej siete. Teda neurónovej sieti sme pridali tzv. vnútornú pamäť, kde si neuróny budú pamätať informácie o svojich činnostiach, respektíve aktiváciach z minulosti. Úlohy vhodné pre tento typ siete môžeme zhrnúť do troch typov [2, s.130]:

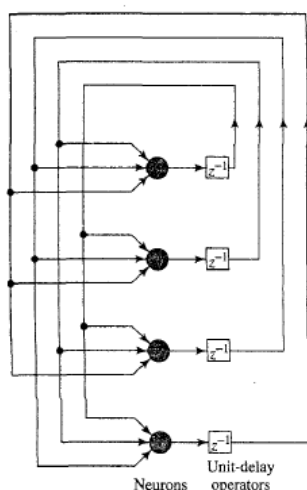
1. *Klasifikačné a asociačné úlohy s časovým kontextom* – problémy, ktoré riešia aj predošlé typy neurónových sietí pridané o časovú informáciu
2. *Predikčné úlohy* – chceme predpovedať budúce dáta na základe histórie D, teda hľadáme časovú štruktúru dát
3. *Generatívne úlohy* – úlohou je pokračovať v časovom rade dát na základe histórie úseku D

## 2.4 Hopfieldove siete

Hopfieldova sieť, inšpirovaná spinovými sklami, je príkladom asociatívnej pamäte bez skrytých neurónov. Je užitočná ako obsahom adresovateľná pamäť („content addressable memory“) alebo analógový počítač na riešenie kombinatorických optimalizačných problémov. Má v sebe zahrnuté mnohé funkcie nášho mozgu, napr. asociatívna pamäť, spracovávanie časových postupností informácií, zabúdanie, ... . Pozostáva zo súboru neurónov (môžu mať hodnotu  $-1 / +1$ ) a korešpondujúceho súboru oneskorovačov („unit delays“), ktoré slúžia ako synaptické váhy medzi  $i$ -tym a  $j$ -tym neurónom, pričom nasčítaním váhovaných signálov od všetkých ostatných

neurónov získame postsynaptický potenciál neurónu. Každý neurón ešte obsahuje hodnotu prahu excitácie, ktorá určuje, či sa daný neurón aktivuje a deterministické prechodové pravidlo, ktorá rozhadzuje hodnoty na  $-1 / +1$ . Týmto Hopfieldova sieť tvorí viacsľučkový systém so spätnou väzbou. Počet slučiek spätnej väzby je ekvivalentný počtu neurónov. Jednoducho povedané, výstup každého neurónu je spätne podávaný ostatným neurónom cez oneskorovače [2, s.190-191] (Obr. 9).

Obr. 2.4: Hopfieldova sieť so 4 neurónmi [5, s.45]



Použitie Hopfieldových neurónových sietí [2, s.190-191]:

- Rozpoznávanie známych vzorov
- Autoasociatívne vybavenie si z pamäti – rozpoznanie a rekonštrukcia známeho vzoru, pri čiastkovej informácii na vstupe o vzore
- Klasifikácia
- Optimalizačné problémy – cieľom je minimalizácia energie systému

Všeobecne povedané sa Hopfieldove siete používajú na modelovanie kognitívnych procesov, ktoré prebiehajú v našom mozgu.

## 2.5 Samoorganizujúce sa (Kohonenove) mapy

Jedná sa o siete, ktoré využívajú učenie bez učiteľa, teda dáta pripravené pre sieť neobsahujú dvojice vstup  $\rightarrow$  výstup. Keďže informácie o výstupoch nemáme k dispozícii (už existujú aj hybridné učenia s učiteľom, algoritmy LVQ, ktoré sa používajú na doladovanie). Tieto siete sú založené na súťaživom učení, výstupné neuróny medzi sebou súťažia o to, aby boli aktivované. Výstupné neuróny sú rozdelené do skupín a v každej skupine je vždy aktívny jeden víťazný („winner-takes-all“ alebo jednoducho „winning“) neurón. Jeden spôsob ako určovať víťazné neuróny je zavedením vedľajších tlmiacich spojení („negative feedback paths“) medzi nimi [2, 142].

V samoorganizujúcich sa mapách sú neuróny uložené na uzloch jedno alebo dvojrozmernej mriežky (existujú aj viacrozmerné, ale sú zriedkavé). Neuróny sa selektívne ladia na rôzne vstupné vzory, prípadne triedy vstupných vzorov, v priebehu súťaživého učiaceho procesu. Na mriežke sa vytvorí zmysluplný súradnicový systém pre rôzne vstupné črty („input features“). Samoorganizujúca sa mapa je charakterizovaná vytvorením topografickej mapy vstupných vzorov, v ktorých priestorové umiestenie neurónov na mriežke oznamuje skutočné štatistické črty obsiahnuté vo vstupných vzoroch. Takéto neurónové siete môžeme považovať za nelineárne zovšeobecnenie analýzy hlavných komponentov [5, 465-466].

## 2.6 Evolučné neurónové siete

Rovnako ako sa použila myšlienka biologického nervového systému na vytváranie silných umelých neurónových sietí, je v tejto dobre čoraz častejšie používaný nápad využiť prírodnú evolúciu na optimalizovanie neurónových sietí. Avšak tieto evolučné výpočty sú výpočtovo náročné, a preto na individuálne problémy je mnohokrát lepší a rýchlejší tradičný spôsob vyskúšania viacerých architektúr. Evolučný prístup je dobre využiť pri tvorbe neurónovej siete, ktorá by mala dobre pracovať na celej triede problémov (učiť sa rýchlo a dobre zovšeobecňovať). Potom by sme výsledky z evolučného algoritmu aplikovali na špecifické problémy, ktoré by nám boli predstreté [4].

Evolučné algoritmy sa podieľajú na optimalizácii neurónových sietí v dvoch základných smeroch. Vylepšujú a upravujú architektúru (topológiu) neurónovej siete, teda počet neurónov, počet skrytých vrstiev, pridávanie a odstraňovanie jednotlivých synáps, prípadne kontroluje parametre prechodovej funkcie neurónov alebo parametrov pre učenie back-propagation (napr. parameter „learning rate“). Druhý princíp spočíva v úprave váhových a prahových koeficientov v synapsách neurónovej siete, teda nahradí metódu back-propagation. Evolučný proces musí byť v rovnováhe medzi rýchlosťou nájdenia najbližšieho optima (väčšinou lokálneho) a najlepším prehľadom celého priestoru možností. Jednotlivé algoritmy sa líšia vzhľadom na ktorú podmienku sa zameriavajú. Príklady evolučných algoritmov: horolezecký algoritmus, „tabu search“, simulované žíhanie („simulated annealing“), evolučné stratégie, genetické algoritmy. V praxi sa na topologickú evolúciu využívajú genetické algoritmy a pre váhové a prahové koeficienty sa využíva simulované žíhanie a genetické algoritmy, prípadne sa používajú kombinácie rôznych algoritmov [2, s.248-252].

Keďže sme sa v tejto práci venovali konkrétnemu problému, je zbytočné využívať evolučné algoritmy na tento problém, a preto sme tieto algoritmy v práci nevyužívali.

# Kapitola 3

## Návrh implementácie

### 3.1 Charakteristika problému

Táto bakalárska práca sa zaoberá rýchlosťou a efektivitou neurónových sietí s rôznymi učiacimi algoritmi. Venovali sme sa problémom, kde je základom zisťovanie závislostí medzi vstupnými dátami a jedným výstupom, teda hodnoty vstupných dát určujú hodnotu výstupnú. Rátali sme s tým, že nemáme veľké množiny tréningových dát, keďže pri aplikovaní týchto neurónových sietí v praxi sú množiny dát na natreningovanie siete veľké iba niekoľko stoviek prípadne tisícok príkladov. Problémom je fakt, že aj v dnešnej programovo vyspelej spoločnosti sa stále nedá vopred povedať, že na určitý problém je najvhodnejšie použiť presný typ siete s istými parametrami. Preto našou úlohou bolo vyskúšať rôzne tréningové algoritmy na rôznych platformách a vybrať z nich ten, ktorý je najvhodnejší a najefektívnejší na riešenie daného typu úloh. Každé riešenie sme museli zhodnotiť, teda vypísať klady a zápory algoritmu a platformy a následne ich navzájom porovnať.

Očakávali sme, že nájdeme riešenie s dostatočne malou odchýlkou presnosti, aby sa neurónová sieť mohla využiť v konkrétnej aplikácii (v praxi). Ku tomuto najlepšiemu riešeniu sme naprogramovali funkcie, ktoré sa dajú využiť v aplikáciách na riešenie tohto typu problémov. Funkcie sú dostatočne všeobecné na to, aby dokázali samy zo vstupného súboru zistiť počet vstupných premenných pre sieť.

Pri týchto problémoch sme si stanovili podmienku jedného výstupu, a preto funkcia sama rozpoznáva správnu architektúru siete. Podmienka jedného výstupu je v dôsledku obmedzenia jedného z testovaných tréningových algoritmov. Ak by sme chceli riešiť problém s viacerými výstupnými premennými, funkcia by bola ľahko upraviteľná, pričom ale použitý tréningový algoritmus musí povoľovať viacero neurónov vo výstupnej vrstve. Taktiež by potom bolo potrebné sieti vopred povedať počet vstupov a výstupov, prípadne načítavať dáta z rôznych súborov, aby sa vedela vytvoriť vždy správna architektúra neurónovej siete. Na testovanie účinnosti a efektívnosti sme potrebovali konkrétny problém z daného typu úloh, a preto sme si najprv museli predpripraviť dáta z danej problematiky.

## 3.2 Metodika práce

Podľa vlastností sietí opísaných v prvej kapitole, sme si ako prvú na testovanie zvolili nelineárnu doprednú back-propagation sieť, vhodnú na regresnú analýzu. Túto sieť by sme mohli celú naprogramovať nanovo vo vybranej platforme, ale bolo by to zbytočné, keďže sú nám dostupné už naprogramované modely sietí tohto typu. Preto sme našli a využili už hotové externé triedy, ktoré nám umožňujú ľahko upravovať architektúru a parametre siete. Vhodne naprogramovanú back-propagation sieť sme našli v programovacom jazyku C++, pričom nám povoľovala priamy prístup ku kódu a ľahkú úpravu pre prispôbenie siete našim požiadavkám. Pre lepšie výsledky sme siahli po programovom prostredí Matlab od MathWorks, ktorý je vybavený nástrojom Nntool na prácu s neurónovými sieťami. Tu sme dosahovali skvelé výsledky, ale keďže je to komerčne platený program, nie všetci majú k nemu prístup. Preto sme využili programovací jazyk Java, ku ktorému existuje open source framework ENCOG, ktorý ponúka možnosť práce so sieťami a rôznymi tréningovými algoritmi.

Na začiatku sme si však museli predpripraviť dostatočné množstvo vhodných tréningových dát, aby v rámci možností pokryli celú sieť intervalu vstupných dát, pretože jedine na takej vzorke sa dokáže neurónová sieť správne natréňovať. Dáta nám boli poskytnuté firmou KOOL4 Solutions s.r.o. Bola to séria príkladov ohodnotenia projektov piatimi parametrami s jedným výstupom, dĺžkou času potrebného



na spracovanie tohto projektu. Týchto dát však nebolo dostatočné množstvo, preto sme si z týchto dát museli syntetizovať väčšie vzorky dát. Na to sme vybrali pracovné prostredie java a načítali a následne sme aplikovali vzorce na dáta. Využili sme náhodnosť, aby bola daná náhodná vzorka dát. Tým sme vytvorili vhodné tréningové dáta pre testovanie jednotlivých neurónových sietí a platform. Po vytvorení tejto vzorky sme sa pustili do vytvárania jednotlivých sietí, testovania a porovnávania. Porovnávali sme priemernú kvadratickú odchýlku siete, čas potrebný na natrénovanie siete a efektivitu práce v danom prostredí.

# Kapitola 4

## Vyhodnotenie účinnosti a efektívnosti

### 4.1 Riešenie v C++

Na začiatku sme sa zamerali na testovanie doprednej back-propagation siete, vďaka jej vlastnostiam vhodným na náš typ úloh. Využíva nelineárnosť a dokáže pomocou spätného šírenia chyby správne natréňovať sieť pri vhodných dátach. Zobrali sme si už naprogramovanú backpropagation sieť v jazyku C++ (autor: Bobby Anguelov [6]), kde je neurónová sieť veľmi dobre implementovaná. Pre naše potreby sme si museli túto sieť trochu upraviť. Neurónová sieť nám ponúka množstvo vlastných nastavení.

Ponúka nám ľahké nastavenie architektúry siete (počet neurónov v každej vrstve), nastavenie podmienok pre ukončenie - žiadanú percentuálnu úspešnosť siete a maximálny počet cyklov (epochov), načítavanie dát zo súboru csv, avšak vstupné a výstupné dáta musia byť v jednom súbore. Tieto dáta rozdelí na tréningové, validačné a testovacie, vo vami určenom pomere. Existuje možnosť premiešania vzoriek pred rozdelením.

Sieť ďalej ponúka pokročilejšie rozdelenie dát. Dáta môžeme rozdeľovať v troch variantách. Prvú variantu predstavujú statické dáta („static dataset“) - klasické rozdelenie dát v nami určenom pomere. Druhou variantou sú rastúce dáta („grow-

ing dataset“) - vždy, keď sieť skončí tréovanie na danej tréovacej množine (je splnená podmienka ukončenia tréovania), tréovacia sada sa zväčší o pevne daný percentuálny pomer celkového množstva vzoriek. Predom musíme zadať percento navyšovania sa dát. Poslednou variantou sú okienkové dáta („windowed dataset“) - v tréovacích dátach sa vytvorí okno pevnej dĺžky, teda podmnožina nami zvolenej dĺžky. Keď tréovanie skončí na danej podmnožine dát, podmnožina sa nahradí ďalšou podmnožinou rovnakej pevne danej dĺžky (posunie sa okno v množine dát).

Máme možnosť učenia s momentom - momentum slúži na zrýchlenie učiaceho procesu. Momentum nám pomáha udržiavať smer z predchádzajúceho kroku. Výhodou toho je, že chybné dáta nám teraz neovplyvnia priebeh učenia ako bez použitia momenta. Momentum môže mať hodnotu 0 až 1, pričom 0 znamená, že momentum má nulový efekt. Čím bude momentum väčšie, tým väčší dôraz dávame na predošlý krok. Natrénované váhy sa ukladajú do súboru, a pri ďalšom spustení tréovania sa načítajú. Celý priebeh tréovania sa ukladá do logového súboru.

Zápory, ktoré je potrebné v sieti zmeniť a vylepšiť:

- keďže sa používa sigmoidná aktivačná funkcia, výstup je z intervalu  $< 0, 1 >$ , preto na správne tréovanie siete musíme túto výstupnú hodnotu ešte roziahnuť do nášho intervalu výstupných dát.
- v sieti je primárne nastavená „clamping“ funkcia, teda výstup uzemňuje buď na nulu alebo jednotku, preto je potrebné vymazať túto funkciu ak je nežiadaná (v našom prípade je nežiadaná)
- môžeme pracovať iba s jednou skrytou vrstvou a je vždy nastavená sigmoidná funkcia. Nemáme možnosť zmeny.

Celkovo hodnotíme toto prostredie ako dobrý základ na tvorbu neurónovej siete, ktoré ponúka už mnoho predprogramovaných možností výberu vlastností neurónovej siete. Pre naše dáta však zd'aleka nedosahovala dobré výsledky, ktoré by sa dali použiť v praxi. Preto je to len vhodná alternatíva pre doprogramovanie, rozšírenie funkcií ponúkajúcich viacero skrytých vrstiev. Najlepšie výsledky sme dosiahli pri jednej skrytej vrstve a 20 skrytých neurónoch. Momentum sme nastavili na hodnotu 0.9 a učiaci parameter („learning rate“) na 0.01. Pre optimalizáciu sme použili podmienku

maximálneho počtu 50 000 epochov. Počet vzoriek, teda veľkosti dát je 2000 vzorov a určili sme si statické rozdelenie dát v pomere 0.6, 0.2, 0.2 (60% tréningové dáta, 20% validačné dáta, 20% testovacie dáta). Celkový čas učenia bol 24 minút, pričom sme sa dostali na úspešnosť 14.13% a priemernú kvadratickú odchýlku 112.86743 hodín.

## 4.2 Riešenie v Matlabe

Matlab je komerčné programové prostredie od MathWorks, s možnosťou tvorby neurónových sietí v integrovanom nástroji NNtool. Je tu možnosť programovať sieť priamo kódom, alebo vytvárať sieť v GUI prostredí. Taktiež tu máme možnosť percentuálne predom zvoliť rozdelenie dát na tréningové, validačné a testovacie. V tomto prostredí je tvorba siete veľmi intuitívna a jednoduchá. Dáta môžeme načítať z cvs súboru alebo nahodiť priamo v Matlabe. Je tu veľký výber neurónových sietí s rôznymi tréningovými algoritmami, možnosť výberu aktivačnej funkcie a počtu skrytých vrstiev aj počtu neurónov v jednotlivých vrstvách.

Matlab ponúka vytvorenie siete s učiacim algoritmom Levenberg-Marquardt (LMA algoritmus), ktorý je špeciálne určený na menšie sady dát. Čím viac je dát, tým horšie pracuje. Tento algoritmus interpoluje medzi Gauss-Newtonovým algoritmom a metódou gradientového zostupu. Je to algoritmus, ktorý pri malých dátach dokáže sieť natréňovať oveľa rýchlejšie ako backpropagation algoritmus. LMA algoritmus zakazuje viac ako jeden neurón vo výstupnej vrstve. Pre lepšiu účinnosť siete sme pridali jednu skrytú vrstvu s lineárnou aktivačnou funkciou. Znova máme možnosť zvoliť si percentuálne rozdelenie dát na tréningové, validačné a testovacie, pričom sú dáta najprv premiešané a až potom rozdelené. Váhy sú inicializované na náhodné hodnoty. Z týchto dvoch príčin bude každé tréningovanie prebiehať rôzne, a teda je lepšie sieť niekoľkokrát pretréňovať, aby sa došlo k čo najlepšiemu výsledku. Matlab má preddefinované funkcie na prácu s maticami, a preto prebieha tréningovanie siete rýchlejšie ako v C++.

Pri použití rovnakých dát ako v C++, teda pri počte vzoriek 2000, najlepšie výsledky vykazovala sieť s počtom skrytých neurónov 80, po 26 sekundách a 88 epochoch bola natréňovaná na priemernú kvadratickú odchýlku  $MSE=17.7$ . Záporom

je, že Matlab okrem MSE neukazuje percentuálnu úspešnosť siete. Keď sme dáta zredukovali na množinu 1000 vzoriek, pri rovnakej architektúre sa nám podarilo sieť natrénovať na MSE=5.37, za 10 sekúnd po 56 epochoch.

Ako bolo povedané, kvôli komerčnosti Matlabu sme siahli po inom prostredí.

### 4.3 Riešenie v Java

Existuje voľne dostupný framework ENCOG [7] v programovacom prostredí Java, ktorý ponúka predprogramované funkcie na tvorbu rôznych neurónových sietí, s použitím rozličných algoritmov. Ponúkajú veľký výber neurónových sietí: dopredné siete („Feedforward Network – Perceptron“), hopfieldove siete, kohonenove samoorganizujúce sa mapy, rekurentné siete, ...

Trénovacie algoritmy predprogramované vo frameworku ENCOG sú: back-propagation algoritmus, LMA algoritmus, resilient propagation, genetický algoritmus učenia, hopfieldovo učenie a mnoho ďalších.

Taktiež je možnosť nastavenia rôznej aktivačnej funkcie pre každú vrstvu siete: sigmoidná, hyperbolický tangens, lineárna, bipolárna, gaussova, súťažná („Competitive“).

Ponúka niekoľko spôsobov randomizácie, pri inicializovaní hodnôt váh. My sme používali dopredné siete, aktivačnú funkciu hyperbolický tangens a Nguyen-Widrow randomizačnú funkciu pri inicializačnej fáze siete. Vyskúšali sme tri rôzne trénovacie algoritmy, back-propagation s 2 skrytými vrstvami, LMA algoritmus s jednou skrytou vrstvou a Resilient propagation s jednou skrytou vrstvou. Výsledky z jednotlivých sietí sme porovnali v kapitole 5 a pracovali sme so sieťou, ktorá vykazovala najlepšie výsledky. Keďže sme používali aktivačnú funkciu hyperbolický tangens (výstup v intervale  $< -1, 1 >$ ), vstupné aj výstupné dáta bolo potrebné normalizovať. Pri zisťovaní výstupu siete a taktiež skutočnej chyby MSE, sme ich museli denormalizovať.

Back-propagation sme si vybrali ako základ, ale na rozdiel od predchádzajúceho riešenia v C++ sme pracovali s dvoma skrytými vrstvami. Taktiež sme zmenili aktivačnú funkciu na hyperbolický tangens. Po niekoľkých spusteniach tréningu siete,

sme si mohli všimnúť, že backpropagation algoritmus ľahko zapadne do lokálneho riešenia a nepokračuje v tréňovaní siete. Teda je to riešenie, pri ktorom je veľmi otáznne, či sa sieti podarí natréňovať správne.

Po dosiahnutí lepších výsledkov v riešení v Matlabe sme sa inšpirovali a vyskúšali tréňovať s LMA algoritmom. Použili sme jednu skrytú vrstvu a taktiež aktivačnú funkciu hyperbolický tangens. Keďže je normalizačná funkcia nastavená na interval  $\langle -1, 1 \rangle$ , bolo by zbytočné meniť normalizačnú funkciu na interval  $\langle 0, 1 \rangle$ , aby sme mohli použiť sigmoidnú aktivačnú funkciu. Výsledky sme zapísali do tabuľky a porovnali.

Vyskúšali sme pracovať ešte s resilient propagation učiacim algoritmom, ktorý funguje podobne ako back-propagation algoritmus, ale snaží sa ešte vylepšiť nedostatky tejto siete. Magnitúda čiastkovej derivácie je zvyčajne priveľká alebo primalá a navyše je parameter učiacej rýchlosti konštantný na celej back-propagation sieti. Resilient propagation učiaci algoritmus používa namiesto parametra učiacej rýchlosti špeciálnu update hodnotu pre každé synaptické spojenie medzi neurónmi. Vďaka tomu zaručí oveľa menšiu pravdepodobnosť, že sa resilient propagation algoritmus zasekne lokálne. Táto hodnota sa po každom epochu aktualizuje na nové hodnoty, podľa chyby siete pomocou tohto vzorca [8]:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ \cdot \Delta_{ij}^{(t-1)} & , \text{ak } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^- \cdot \Delta_{ij}^{(t-1)} & , \text{ak } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \Delta_{ij}^{(t-1)} & , \text{inak} \end{cases}$$

Následná úprava váh bude podľa vzorca [8]:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)} & , \text{ak } \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\Delta_{ij}^{(t)} & , \text{ak } \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ 0 & , \text{inak} \end{cases}$$

Tab. 4.1: Vysvetlivky jednotlivých znakov [8]

Premenná	Význam
$\Delta_{ij}^{(t)}$	Update hodnota pre iteráciu $t$
$\frac{\partial E}{\partial w_{ij}}^{(t)}$	Gradient váhy z nerónu $i$ do $j$ pre iteráciu $t$
$\Delta w_{ij}^{(t)}$	Zmena vo váhe z neurónu $i$ do $j$ spôsobená iteráciou $t$
$\eta^+$	Pozitívny krok, zvyčajne 1.2
$\eta^-$	Negatívny krok, zvyčajne 0.5

## 4.4 Popisy naprogramovaných funkcií

### 4.4.1 Načítavanie dát

Pre naše potreby sme si museli pripraviť funkciu na čítanie dát z csv súboru. Je to void funkcia `read_csv_file(File file)`. Tieto dáta načítavame do dynamického poľa, kde si podľa súboru zistíme počet vstupných premenných a celkový počet vzoriek v dátach. Najprv si súbor načítavame pomocou `BufferedReader` a čítame z neho po riadkoch do premennej `line`. Počet vstupných premenných v jednom riadku sa určí pomocou funkcie `split(arg0)`, ktorá nám rozdelí `String` do poľa podľa porežazca `arg0` (v našom prípade čiarky, prípadne bodkočiarky). Musíme zistiť, či v csv súbore sú premenné v riadku delené čiarkou alebo bodkočiarkou pomocou funkcie `contains(arg0)`, a v prípade, že sú reálne čísla v tvare `10,02` nahradíme čiarku bodkov do tvaru `10.02`, aby sa dali konvertovať premenné v `Stringu` do hodnôt `double` pomocou funkcie `Double.parseDouble(premenná)`.

Popri načítavaní dát si taktiež zisťujeme minimálnu a maximálnu hodnotu každej premennej, pre potreby normalizácie v neskorších funkciách tréovania siete. Dáta hotové pre potreby siete máme v konečnom kroku načítané v globálnych dvojrozmerných poliach `input[][]` a `output[][]`. Počet premenných, podľa ktorého sa určuje počet neurónov na vstupnej vrstve, máme uložený v premennej `num_inputs`.

## 4.4.2 Normalizovanie dát

V sieti sme používali aktivačné funkcie hyperbolický tangens, preto sme museli vstupné aj výstupné dáta najprv normalizovať do intervalu  $\langle -1, 1 \rangle$ . Funkcia má tri vstupné parametre, číslo na normalizáciu a ďalšie dve premenné, ktoré určujú interval, z ktorého normalizujeme. Maximálne a minimálne hodnoty jednotlivých premenných máme uložené v poliach `data_max[]` a `data_min[]`, ktoré upravujeme pri načítavaní dát vo funkcii `read_csv_file()`. Výstupom je double normalizovaná hodnota.

Obr. 4.1: Zdrojový kód funkcie `normalize()`

```
public static double normalize(double number, double d_min, double d_max) {  
    double norm_number;  
    norm_number = (number - d_min) * 2 / (d_max - d_min) - 1;  
    return norm_number;  
}
```

## 4.4.3 Denormalizovanie dát

Pre používanie už natrénovanej siete a pre reálny odhad priemernej kvadratickej odchylky MSE, bolo potrebné výstup siete z intervalu  $\langle -1, 1 \rangle$  denormalizovať na reálne hodnoty, a na to je potrebná inverzná funkcia k normalizácii. Znova má tri vstupy, číslo na denormalizáciu a interval, v ktorom pracujeme. Výstupom je double denormalizovaná hodnota.

Obr. 4.2: Zdrojový kód funkcie `denormalize()`

```
public static double denormalize(double number, double d_min, double d_max) {  
    double denorm_number;  
    denorm_number = ((d_max - d_min) * (number + 1)) / 2 + d_min;  
    return denorm_number;  
}
```



#### 4.4.4 Vytváranie novej siete

Funkcia na vytvorenie a natréňovanie siete je najdôležitejšia. Využívali sme tam funkcie frameworku ENCOG verzie 2.4.3. Funkcia *create\_RPROP()*, ktorá vytvorí novú doprednú sieť s trénovacím algoritmom resilient propagation, má tri vstupy:

- **File súbor** - kde máme uložené dáta (vzorky), podľa ktorých budeme tréňovať sieť
- **int maxEpoch** – obmedzenie na maximálny počet cyklov (epochov), pri zadaní hodnoty -1, nechceme obmedzovať na maximálny počet cyklov (teda teoreticky nekonečno)
- **double minError** – obmedzenie na minimálnu chybu, teda podmienka ukončenia tréňovania siete
- **int num\_Hneuron** – počet skrytých neurónov v skrytej vrstve

Na začiatku sieť musí načítať všetky dáta a uložiť ich do dvojrozmerného poľa, vstupy a výstupy sú v oddelených poliach pomocou vyššie opísanej funkcie *read\_csv\_file(file)*. Po načítaní dát vieme počet vstupných premenných, ktorý máme uložený v premennej *num\_input*, teda môžeme vytvoriť architektúru siete (pridať vrstvy s počtom neurónov). Pridali sme podmienku kontroly dostatočného počtu skrytých neurónov. Zisťujeme, či je zadaný počet neurónov v skrytej vrstve *num\_Hneuron* aspoň dvojnásobnej veľkosti ako je počet vstupných neurónov *num\_input*. Ak táto podmienka nie je splnená, nie je zaručený správny beh siete. Preto je predvolené zvýšiť počet skrytých neurónov na dvojnásobný počet vstupných neurónov.

Nasleduje zicializovanie siete, nastavenie váh na náhodné hodnoty. Je potrebné nastaviť vlastnosti siete, v našom prípade doprednú sieť s algoritmom resilient propagation. Začíname tréňovanie z náhodných hodnôt váh, a preto je každé tréňovanie trochu odlišné. Máme možnosť nastaviť stratégiu zresetovania siete a tréňovania nanovo, ak sa nezlepší chyba siete aspoň o jedno percento počas päťdesiatich iteráciach a naša natréňovaná sieť stále nedosiahla požadovanú chybu. Avšak my sme túto možnosť nevyužili a v zdrojovom kóde je to zakomentované. Následne v

cykle iterujeme a trénujeme sieť, pokiaľ nedosiahneme žiadanú chybu. Prípadne ak dosiahneme maximálny počet epochov, použijeme funkciu `break` a zastavíme učenie siete. Počas trénovania sa do konzoly vypisujeme chyba po každom epochu, aby bolo priebeh učenia ľahko sledovateľný. Po skončení trénovania vypíšeme konečnú priemernú kvadratickú odchýlku siete MSE. Funkcia vráti natrénovanú sieť. Pre veľkú dĺžku kódu funkcie sme ho museli presunúť do prílohy Obr. 5.2.

#### 4.4.5 Požitánie výstupu

Aby sa dala používať natrénovaná sieť, potrebujeme funkciu `recognize_output()`, ktorá zo vstupných premenných bude počítat (aproximovať), podľa natrénovanej siete, správny výstup. Ako vstupné parametre pre funkciu je pole vstupných premenných a natrénovaná sieť, v ktorej chceme rátať výstup. Tieto vstupné premenné musíme najprv normalizovať. Potom ich vložíme do podoby `NeuralData` a následne príkazom `compute(input)` dostaneme výstup siete v podobe `NeuralData`. Odtiaľ získame chcené dáta pomocou príkazom `getData(0)`, v podobe `double`. Túto hodnotu denormalizujeme a vrátime ako výstup funkcie. Konkrétny kód funkcie môžeme nájsť v prílohe Obr. 5.3.

#### 4.4.6 Čítanie vstupov zo súboru a zapisovanie výstupov do súboru

Doprogramovali sme ešte funkciu pre vyhodnocovanie viacerých sád vstupných premenných, pričom vstupné premenné sú uložené v csv súbore. Každý riadok obsahuje jednu sadu vstupných parametrov, ktoré treba vyhodnotiť v natrénovanej sieti. Funkcia void `recognize_file_outputs()` má tri vstupné parametre, súbor zo vstupnými sadami, String obsahujúci názov súboru, do ktorého chceme zapisovať výsledky siete na dané vstupy a natrénovanú sieť. Rovnakým spôsobom ako v podkapitole 4.4.1 **Načítavanie dát** načítame a rozdelíme vstupné premenné do poľa a následne z nich pomocou predchádzajúcej funkcie `recognize_output()` zistíme aproximovaný výstup k daným vstupným premenným. Následne zapíšeme do csv súboru vstupné premenné

aj s vyrátaným výstupom.

#### 4.4.7 Zaokrúhľovanie na dve desatinné miesta

Denormalizovaný výstup je typu `double` a má 13 desatinných miest. V aplikáciách je však vhodnejšie a praktickejšie pracovať ďalej iba s reálnym číslom s dvoma číslicami za desatinnou čiarkou. Preto sme si vytvorili funkciu `round(double x, int pocet)`, kde vstupný parameter `x` určuje hodnotu, ktorú chceme zaokrúhliť, a počet určuje na koľko desatinných miest chceme zaokrúhľovať. Výstup je `double`, už zaokrúhlené číslo.

Obr. 4.3: Zdrojový kód funkcie `round()`

```
public static double round(double x, int pocet) {  
    return Math.round( x * Math.pow(10,pocet))/ (double)Math.pow(10,pocet);  
}
```

### 4.5 Využitie funkcií vo vytvorenom GUI prostredí

Poskytnuté dáta, na ktorých sme testovali rôzne neurónové siete a platformy, sa dajú vhodne využiť na vytvorenie GUI prostredia. Toto GUI prostredie nám dovoľuje vhodne pracovať s týmito dátami a pretestovať funkčnosť funkcií opísaných v predošlej podkapitole. Toto prostredie slúži aj ako ukážka využitia týchto funkcií a teda aj neurónovej siete. Tlačidlá v GUI prostredí aplikujú jednotlivé naprogramované funkcie pre neurónové siete:

- `recognize` v záložke `Recognize output` – používa funkciu `recognize_output()`
- `open` – otvorí nové okno, s možnosťou prehliadania si súborov na disku a výberom koncového súboru
- `retrain` – natrénuje sieť nanovo, využíva funkciu `create_RPROP()`

- recognize v záložke Recognize outputs files – využíva *recognize\_file\_outputs()*

Prikladáme screenshoty z daného GUI prostredia v prílohe Obr. 5.4-5.6. Zdrojové kódy k tejto časti sú rozsiahle, preto sa dajú nájsť na priloženom DVD. Taktiež v ňom nájdeme skompilované GUI prostredie.

# Kapitola 5

## Diskusia

### 5.1 Porovnanie rôznych algoritmov

Tab. 5.1: Porovnanie jednotlivých učiacich algoritmov

použitý algoritmus	skr. vrstvy	neuróny	epochy	MSE
resilient propagation	1	5	83000	0.197
resilient propagation	1	10	50000	0.079
resilient propagation	1	20	27702	0.075
resilient propagation	1	40	21791	0.075
resilient propagation	1	80	10621	0.099
backpropagation	2	20;20	100	104.567
LMA algoritmus	1	20	39	3681.430
LMA algoritmus	1	40	30	2067.696
LMA algoritmus	1	80	84	1136.717

Z tabuľky 5.1 vidíme, že zďaleka najlepšie je použiť resilient propagation s jednou skrytou vrstvou a použitými 40 neurónmi v skrytej vrstve. Od 20 neurónov vyššie, počet neurónov v skrytej vrstve ovplyvnil už len rýchlosť učenia siete. Výsledná priemerná kvadratická odchylka MSE bola rovnaká. Menil sa len počet epochov, ktoré

boli potrebné na naučenie siete. Ak by sme použili viac ako 40 neurónov, učenie by trvalo príliš dlho v závislosti od času (nie počtu epochov).

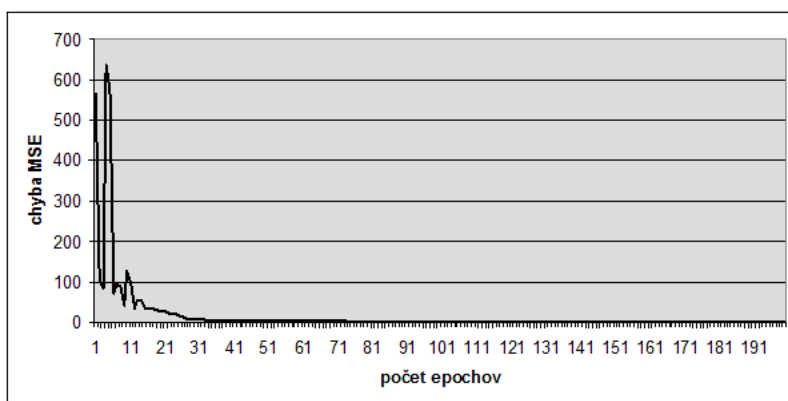
Back-propagation s dvoma skrytými vrstvami dosahoval lepšie výsledky ako s jednou skrytou vrstvou v C++. Učil sa aj oveľa rýchlejšie, ale bol dosť nepredpovedateľný a neporovnateľný s učiacim algoritmom resilient propagation.

Pri algoritme LMA sme dosahovali zlé výsledky. Nedokázali sme presne napodobniť architektúru, ktorú si vytváral Matlab a pri našej bežnej architektúre s jednou vrstvou sa tento algoritmus nemôže použiť v praxi.

Z týchto troch algoritmov sme si preto vybrali resilient propagation a naprogramovali sme funkcie, ktoré vytvárajú a trénujú takýto druh neurónovej siete, dokážu zistiť výstup pre zadané vstupné premenné alebo zisťovať niekoľko výstupov naraz a zapisovať ich do súboru. Tieto funkcie sú prístupné v jar súbore spolu s Java dokumentáciou, dostupné v priloženom DVD. Teda sú ľahko importovateľné do akéhokoľvek nového projektu. Je potrebné si nainportovať aj framework ENCOG, ktorý je voľne dostupný na internete a taktiež na priloženom DVD.

Pre názornú ukážku rýchlosti trénovania siete vidíme priebeh zmeny chyby siete MSE pri učení resilient propagation a maximálnom počte epochov 200. Je tu znázornené kolísanie chyby pri určení, ale po viacerých epochoch chyba konverguje k nule.

Obr. 5.1: Graf priebehu veľkosti chyby MSE počas učenia siete [5, s.35]



# Záver

V práci sme zhrnuli poznatky o neurónových sieťach z rôznych zdrojov. Tým sme vysvetlili pojem neurónovej siete, opísali krátku históriu a rôzne druhy neurónových sietí s dôrazom na ich využitie. Podrobne sme sa venovali dopredným sieťam s back-propagation algoritmus. Back-propagation algoritmus sa považuje za všeobecne najvhodnejší algoritmus na regresnú analýzu, ktorá zahŕňa aj náš typ problémov s vyhodnocovaním vstupných premenných do jednej výstupnej premennej.

V druhej časti sme sa venovali využívaniu rôznych programovacích prostredí a rôznych algoritmov. Využili sme už predprogramované prostredia na tvorbu neurónových sietí, keďže sme zhodnotili, že sú dostatočne výkonné pre naše potreby. Každé prostredie malo svoje výhody a nevýhody. Prostredie C++, s naprogramovanou možnosťou vytvorenia doprednej siete s back-propagation algoritmom malo mnoho možností nastavenia neurónovej siete a práce s tréningovými dátami. Bohužiaľ nám dovoľovalo tvoriť neurónovú sieť len s jednou skrytou vrstvou, a preto sme dosahovali nedostatočné výsledky, s priemernou kvadratickou odchýlkou 112.86743, čo pri priemernej výstupnej hodnote 130.117, je veľmi zlá úspešnosť. Použili sme dáta s počtom vzoriek 2000. Pre nedostatočnú úspešnosť takúto sieť nemôžeme použiť v praxi. Je to však vhodná sieť pre budúce doprogramovanie viacerých možností lepšej architektúry siete, ktorá by mohla poskytovať lepšie výsledky.

Rovnaké dáta sme použili pri tréningu neurónovej siete vytvorenej v Matlabe. Matlab je vynikajúce prostredie na tvorbu neurónových sietí v špecializovanom nástroji nntool, ktorý ponúka aj GUI prostredie na jednoduché vytváranie sietí. Najlepšia sieť špecializovaná na práve náš typ problému bola dopredná sieť s Levenberg-Marquardt algoritmom (LMA algoritmus) a dvoma skrytými vrstvami. Jedna skrytá

vrstva so sigmoidnou a druhá s lineárnou aktivačnou funkciou. Výsledky dosiahnuté touto sieťou bol priemerná kvadratická odchýlka MSE 5.37. Matlab má predprogramované funkcie na prácu s maticami a preto je veľmi rýchly. Taktiež má integrovanú normalizáciu dát do potrebného intervalu, preto je s ním jednoduché pracovať a môžeme využívať dáta v rozmedzí reálnych čísel. Prostredie matlab je však komerčne platené, a preto nevhodné pre používanie širokej verejnosti a budúce rozvíjanie tejto práce.

Zistili sme, že najvhodnejšie je použiť programovacie prostredie java s využitím open-source frameworku ENCOG. Vyskúšali sme v ňom tri rôzne trénovacie algoritmy: back-propagation s dvoma skrytými vrstvami, LMA algoritmus a resilient propagation. Najlepšie výsledky dosahovala dopredná sieť s učením resilient propagation (vylepšený back-propagation algoritmus). Pri 40 neurónoch v skrytej vrstve sme dosahovali po vyše 20000 epochoch priemernú kvadratickú odchýlku MSE 0.075. Preto sme takúto sieť zobrali ako základ a naprogramovali sme funkcie na ľahkú tvorbu takejto neurónovej siete, pričom sme museli doprogramovať pomocné funkcie na načítavanie dát z csv súborov a normalizáciu dát. Výstupom je jar súbor, ktorý je ľahko importovateľný do akéhokoľvek nového projektu. Ponúka taktiež funkcie ako je rátanie výstupu s daných vstupov alebo rátanie skupiny výstupov zo skupiny vstupov čítaných zo súboru a ich následné zapísanie do súboru. Pre testovanie a budúcu inšpiráciu sme naprogramovali aj GUI prostredie, ktoré využíva všetky funkcie a pôsobí ako aplikácia na riešenie nami daného problému.



# Literatúra

- [1] KAČMÁRY, P. - MALINDŽÁK, D. 2010. *Prognózovanie obchodu a výroby v dynamicky sa meniacich podmienkach trhu*. In Acta Montanistica Slovaca. 2010, ročník 15, mimoriadne číslo 1, s. 53-60
- [2] KVASNIČKA, V. a kol. 1997. *Úvod do teórie neurónových sietí*. Iris:Bratislava,1997. ISBN 80-88778-30-1
- [3] DERENÍK, D. 2006. *Matematické modely v technickej analýze cenných papierov*: bakalárska práca. Brno: Masarykova univerzita, 2006. 37 s.
- [4] BULLINARIA, JOHN A. 2005. *Evolving Neural Networks: Is it Really Worth the Effort?* The University of Birmingham Edgbaston, ESANN 2005: 267-272
- [5] HAYKIN, S. 1999. *NEURAL NETWORKS: A Comprehensive Foundation* 2nd ed. McMaster University Hamilton, Ontario, Canada: Pearson Education, 1999. ISBN 81-7808-300-0
- [6] ANGUELOV, B. 2008. *Basic Neural Network Tutorial : C++ Implementation and Source Code*. <http://takinginitiative.net/2008/04/23/basic-neural-network-tutorial-c-implementation-and-source-code/>
- [7] ENCOG 2.4.3 framework from Heaton Research.  
<http://www.heatonresearch.com/encog>
- [8] HEATON, J. 2011. *Resilient Propagation*.  
[http://www.heatonresearch.com/w/index.php?title=Resilient\\_Propagation](http://www.heatonresearch.com/w/index.php?title=Resilient_Propagation)
- [9] MATLAB R2009b: program na pracu s matematikou a neurónovými sieťami

- [10] TAHERI, T. 2010. *Comparing Neural Networks in Neuroph, Encog and JOONE*.  
<http://www.codeproject.com/KB/recipes/xor-encog-neuroph-joone.aspx>
- [11] FARKAŠ, I. 2011. *Neurónové siete: prezentácie k prednáškam*.  
<http://ii.fmph.uniba.sk/farkas/Courses/ns.html>
- [12] FRÖHLICH, J. 2004. *Neural Networks with Java*. <http://www.nnwj.de/>

# Prílohy

## Príloha A: Zdrojové kódy

Obr. 5.2: Zdrojový kód funkcie *create\_RPROP()*

```
public static BasicNetwork create_RPROP(File file, int maxEpoch, double minError,
    int num_Hneuron) throws IOException {
    Logging.stopConsoleLogging();
    //testovanie dostatočného počtu skrytých neuronov
    if (num_Hneuron < 2 * num_inputs) {
        num_Hneuron = 2 * num_inputs;
    }
    //nacistame data
    read_csv_file(file);
    //vlozime data do tvaru neuronovej siete
    NeuralDataSet trainingSet = new BasicNeuralDataSet(input, output);
    //zalozime novu siet
    BasicNetwork network = new BasicNetwork();
    //pridame jej vrstvy
    network.addLayer(new BasicLayer(new ActivationTANH(), true, num_inputs));
    network.addLayer(new BasicLayer(new ActivationTANH(), true, num_Hneuron));
    network.addLayer(new BasicLayer(new ActivationTANH(), true, 1));
    //nastavime, aby to bola dopredna siet
    network.setLogic(new FeedforwardLogic());
    //a mozeme ukonsit strukturu siete
    network.getStructure().finalizeStructure();
    //zinicializuje, zaciatočne vahy nastavime nahodne z intervalu <-1,1>
    (new NguyenWidrowRandomizer(-1, 1)).randomize(network);
    //nastavime uciaci algoritmus resilient propagation
    final Train train = new ResilientPropagation(network, trainingSet);
    /*
    strategia reset, po 50 iteraciach s nezmenenou chybou
    train.addStrategy(new RequiredImprovementStrategy(50));
    */

    //zacneme sa ucit po jednotlivých iteraciach
    int epoch = 1;
    do {
        //priami prikaz na trenovanie siete v ďalšej iterácii
        train.iteration();
        //vždy vypiseme MSE siete, asledujeme priebežný stav ucenia siete
        System.out.println("Epoch #" + epoch + " Error:" + denormalize(
            train.getError(), -data_max[num_inputs], data_max[num_inputs]));
        epoch++;
        //musime kontrolovat, aby sme nepresiahli maximalny počet epochov
        if ((epoch > maxEpoch) && (maxEpoch > -1)) break;
        //podmienka skoncenia treningu je urcene chcene minimalne MSE
    } while (denormalize(train.getError(), -data_max[num_inputs],
        data_max[num_inputs]) > minError);

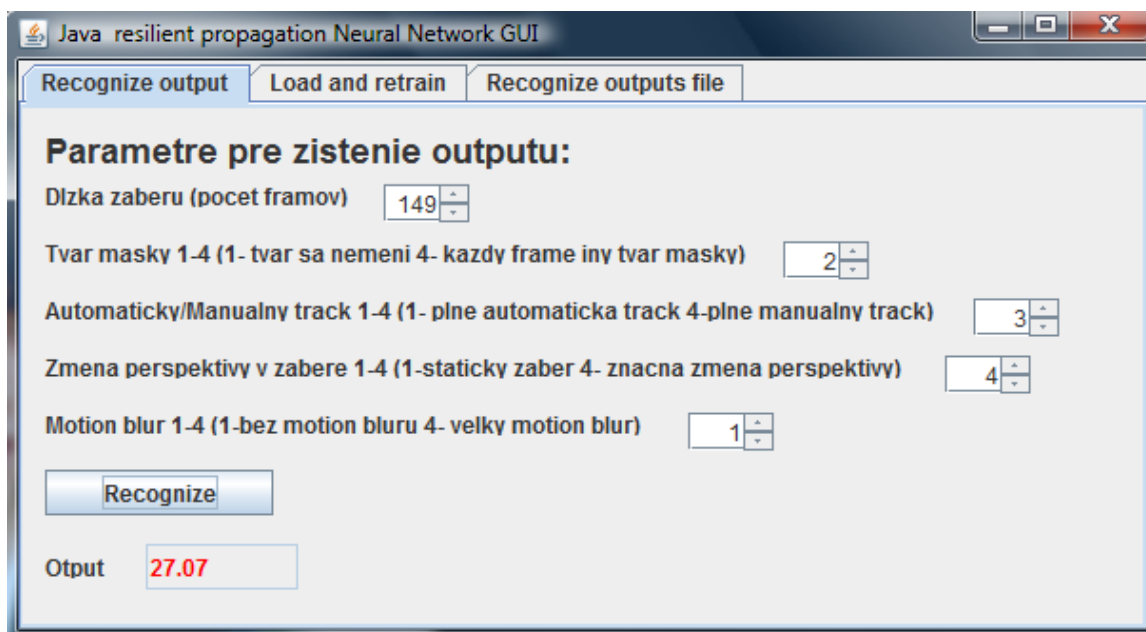
    //nakoniec este vypiseme koncovu MSE chybu siete
    System.out.println("Neural Network Results:");
    System.out.println("error=" + denormalize(train.getError(), -data_max[num_inputs],
        data_max[num_inputs]));
    //funkcia vracia natrenovanu siet
    return network;
}
```

Obr. 5.3: Zdrojový kód funkcie *recognize\_output()*

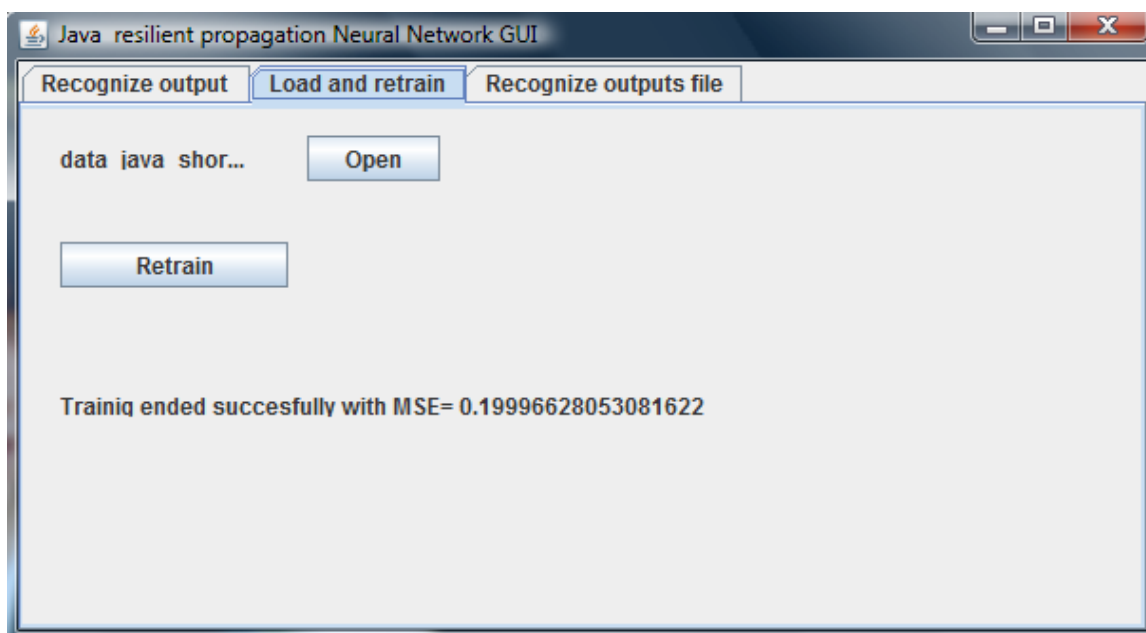
```
public static double recognize_output(double[] inputs, BasicNetwork network1 ) {
    double NN_output; double temp;
    for (int i=0; i<inputs.length; i++){
        temp=normalize(inputs[i], data_min[i], data_max[i]);
        inputs[i]=temp;
    }
    NeuralData input=new BasicNeuralData(inputs);
    NeuralData output=network1.compute(input);
    NN_output=denormalize(output.getData(0), data_min[num_inputs], data_max[num_inputs]);
    return NN_output;
}
```

## Príloha B: Screenshoty GUI aplikácie

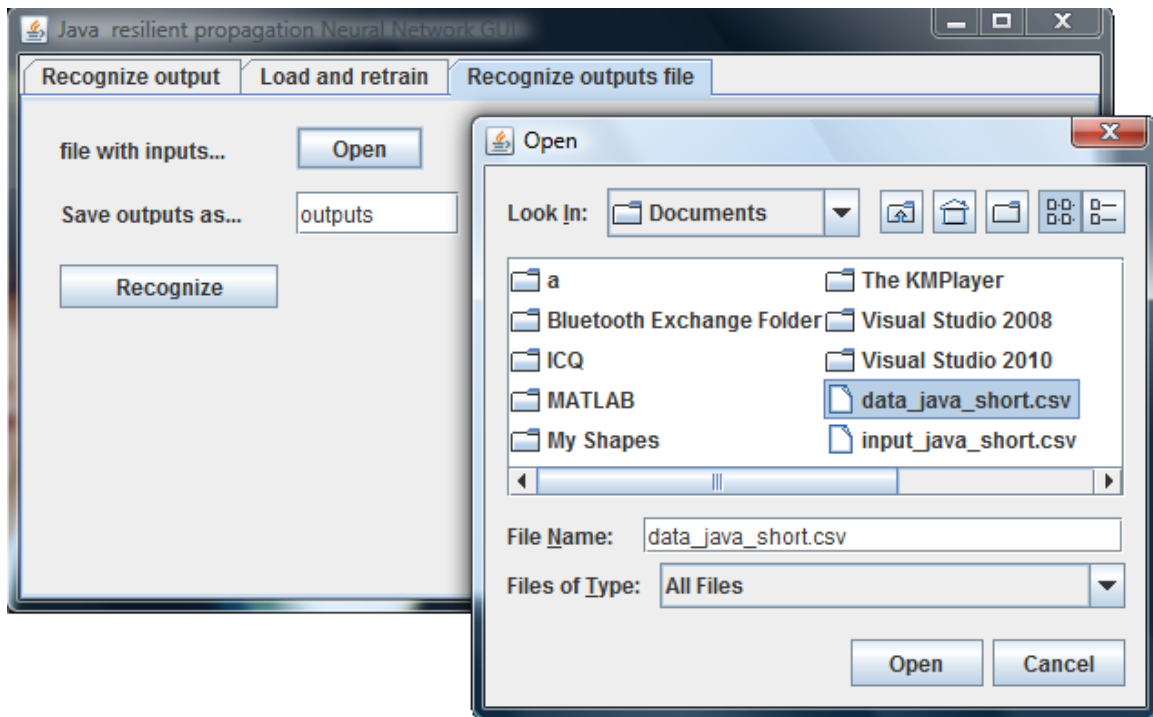
Obr. 5.4: Screenshoty vzorového GUI prostredia (1)



Obr. 5.5: Screenshoty vzorového GUI prostredia (2)



Obr. 5.6: Screenshoty vzorového GUI prostredia (3)



**Príloha C: DVD so štruktúrou:**

**src:**

-zdrojové kódy aplikácie aplikacia.zip

-zdrojové kódy naprogramovaných funkcií kniznica.zip

**doc** - dokumentácia k jar súboru doc.zip

**exe:**

- spustiteľná aplikácia NN\_GUI

- knižnica s naprogramovanými funkciami

create\_resilient\_NN.jar

- knižnica ENCOG encog-core-2.4.3.jar

**data** - dáta použité pri testovaní sietí, 2000 vzoriek data.csv

**bc** - bakalárska práca bc.pdf