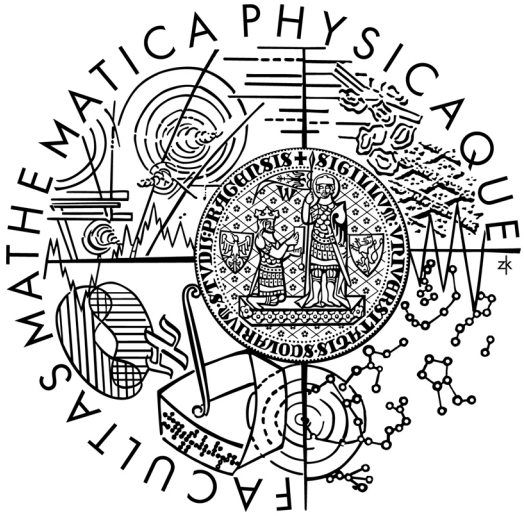
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta



Michal Drobný

Vrstevnaté neuronové sítě a vizualizace jejich struktury

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: doc. RNDr. Iveta Mrázová CSc.

Studijní program: Informatika

Studijní obor: Programování

Praha 2011

Poděkování

Rád bych poděkoval všem, kteří mi byli jakkoli nápomocni při psaní této práce. Děkuji vedoucí mé bakalářské práce docentce Ivetě Mrázové za odborné konzultace a rady, děkuji své rodině a blízkým za jejich trpělivost a všestrannou podporu. Zvláště potom děkuji Elišce Tiché za neustálou psychickou podporu a důvěru v době psaní této práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …..... dne .....

Název práce:	Vrstevnaté neuronové sítě a vizualizace jejich struktury
Autor:	Michal Drobný
Katedra:	Katedra teoretické informatiky a matematické logiky
Vedoucí práce:	doc. RNDr. Iveta Mrázová CSc., Katedra teoretické informatiky a matematické logiky

Abstrakt: Model vrstevnatých neuronových sítí je v současné době známý především díky svým univerzálním aproximačním schopnostem a již základní algoritmus zpětného šíření dává výsledky aplikovatelné v reálných problémech. Tato práce je věnována problematice vrstevnatých neuronových sítí, popisuje vybrané algoritmy pro adaptaci vah a prahů na základě předložených vzorů, především algoritmus zpětného šíření a algoritmus škálovaných konjugovaných gradientů, který řadíme mezi konvergenčně extrémně rychlé algoritmy druhého řádu. Součástí práce je také aplikace pro vizualizaci struktury vrstevnatých neuronových sítí implementující výše zmíněné algoritmy, jejíž řešení je navrženo s ohledem na její využití při výuce v oblasti umělé inteligence.

První část práce je věnována úvodu do problematiky a popisu obou algoritmů, následuje stručné pojednání o dalších variantách algoritmů a jejich analýza. V další části je zdůvodněn výběr vhodného programovacího jazyka pro implementaci aplikace, stanovení cílů a popis implementačních prací. Závěrem jsou pak shrnuty výsledky rychlostního a implementačního porovnání implementovaných algoritmů s vybranou nekomerční implementací ENCOG.

Klíčová slova: Umělé neuronové sítě, vrstevnaté neuronové sítě, algoritmus zpětného šíření, škálované konjugované gradienty.

Title:	Multi-layered Neural Networks and Vizualization of Their Structure
Author:	Michal Drobný
Department:	Department of Theoretical Computer Science and Mathematical Logic
Supervisor:	doc. RNDr. Iveta Mrázová CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: The model of multi-layered neural networks of the back-propagation type is well-known for their universal approximation capability and even the standard back-propagation training algorithm used for their adjustment often provides results applicable to real-world problems. The present study deals with the issue of the multi-layered neural networks. It describes selected variants of training algorithms, mainly the standard back-propagation training algorithm and the scaled conjugate gradients algorithm, which ranks among the extremely fast second-order algorithms. One of the parts of the present study is also an application for the visualisation of the structure of multi-layered neural networks whose solution is designed with respect to its potential utilization in the education of artificial intelligence.

The first part of the study introduces the subject matter and formally describes both algorithms, followed by a short description of other variants of the algorithms and their analysis. The next part discusses the selection of the appropriate programming language for the implementation of the application, specifies the goals and describes the implementation works. The conclusion summarizes the test results of the speed and implementation comparison with the selected noncommercial-based software ENCOG.

Keywords: Artificial Neural Networks, Multi-Layered Neural Networks, Back-Propagation, Scaled Conjugate Gradient.

Úvod

Jednou z významných oblastí umělé inteligence jsou neuronové sítě. Jedná se o matematické modely, jejichž struktura vychází z principů fungování lidského mozku a nervové soustavy. První model umělé neuronové sítě popsal již v roce 1958 psycholog Frank Rosenblatt [7]. Byla nazvána *Perceptron* a smyslem její existence mělo být modelování postupu, při kterém lidský mozek zpracovává vizuální data a učí se rozeznávat objekty. Rosenblattovi následovníci testovali a používali podobné sítě ke studiu rozpoznávání a procesu učení v biologických neuronových sítích. V současné době představují umělé neuronové sítě zavedený prostředek pro řešení široké škály úloh z mnoha oborů od finančnictví přes medicínu až po optimalizaci rezervací letenek v letecké dopravě. Je až pozoruhodné, kde všude se můžeme s tímto pojmem setkat.

Umělé neuronové sítě si však neberou za cíl pouze simulovat funkci nervového systému, spíše se snaží tyto principy aplikovat při efektivním řešení nejrůznějších úloh a problémů nejen z oblasti umělé inteligence, ale především na reálných úlohách. Základním a nejrozšířenějším modelem umělých neuronových sítí jsou vrstevnaté neuronové sítě typu zpětného síření, dále jen vrstevnaté neuronové sítě. Jejich struktura je tvořená sítí základních výpočetních jednotek, neuronů. Způsob zpracování vstupních informací je velmi podobný principu lidského myšlení, z čehož vycházel i Frank Rosenblatt. Obvykle se nejprve učí na základě trénovací množiny a posléze se tato znalost aplikuje při procesu rozpoznávání vzorů, které nebyly obsaženy v trénovací množině.

S velmi rychle rostoucí výkonností počítačových systémů a rozvojem nových poznatků o vrstevnatých neuronových sítích roste i rozšíření nejrůznějších aplikací a systémů určených pro řešení úloh pomocí vrstevnatých neuronových sítí. Často se jedná o velmi složité a rozsáhlé úlohy, jejichž úspěšné řešení klade vysoké požadavky jednak na rychlost učícího algoritmu, jednak na schopnost generalizace. Při implementaci simulátorů pro neuronové sítě je kladen důraz na paralelizaci výpočtů neuronové sítě a efektivitu zpracování dat, což vede ke značným rychlostním rozdílům mezi implementovanými knihovnami zabývajícími se touto problematikou.

Pokud chceme vrstevnaté neuronové sítě použít k řešení složitých úloh s více parametry, je třeba se vypořádat s následujícími komplikacemi. Základní algoritmus zpětného šíření se sice může danou úlohu naučit velmi dobře, přitom ale využívá při výpočtu poměrně velký počet skrytých neuronů, jejichž význam je mnohdy nejasný. Nevýhodou algoritmu je také značná závislost na zadaných parametrech, jejichž vhodná volba je pro úspěšný proces učení

stěžejní. Taktéž při použití na velkých sadách dat můžeme pozorovat výrazné zpomalení procesu učení a zhoršenou schopnost generalizace.

Přestože algoritmus zpětného šíření má při použití na složitých problémech výše zmíněné problémy, existují jiné algoritmy, které se dokáží s jednotlivými dílčími nedostatky vyrovnat. Mezi algorimy, které nejsou tolik citlivé na zadané parametry, řadíme např. algoritmus škálovaných konjugovaných gradientů společně s jeho variantami.

První část práce je věnována úvodu do problematiky vrstevnatých neuronových sítí, následuje stručné shrnutí procesu učení a nejznámějších algoritmů pro učení neuronových sítí s učitelem. Podrobněji je pojednáno o algoritmu zpětného šíření s momentem a algoritmu škálovaných konjugovaných gradientů. V další části práce je zdůvodněn výběr vhodného programovacího jazyka pro implementaci aplikace, stanovení cílů a popis implementačních prací. Závěrem jsou pak shrnuty výsledky rychlostního porovnání implementovaných algoritmů s vybranou nekomerční knihovnou ENCOG.

1 Úvod do problematiky

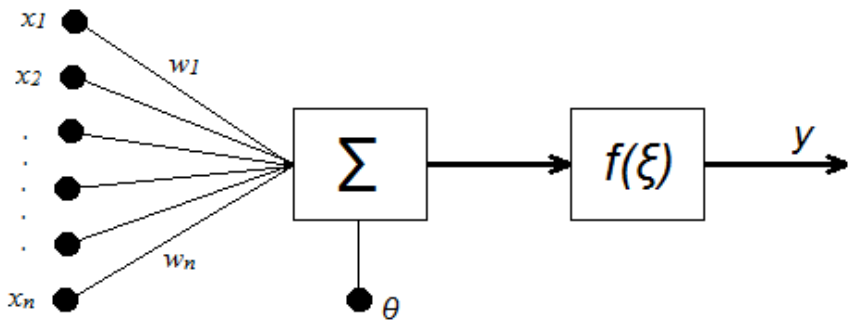
Za umělou neuronovou síť obecně považujeme takovou strukturu pro distribuované a paralelní zpracování dat, jejímiž základními stavebními prvky jsou tzv. formální neurony. Každý z těchto neuronů může přijímat libovolný konečný počet vstupních informací a předávat je dalším neuronům prostřednictvím jediného rozvětveného výstupu.

Nejjednodušší model neuronu počítá součet vážených vstupů, který předává dále pomocí nelineární přenosové funkce. Formální neuron je tedy charakterizován vahami, které vedou informaci ze vstupů tohoto neuronu, dále prahem a typem použité přenosové funkce, kterou může být například skoková přenosová funkce (1.1a)).

V biologickém pojetí neuronu označuje práh, θ, prahovou hodnotu aktivace neuronu. Matematický model neuronové sítě se neomezuje pouze na skokovou přenosovou funkci, proto práh neslouží pouze jako aktivační mez. Právě adaptace hodnoty prahu a vah je nezbytná při implementaci algoritmů pro proces učení neuronových sítí podle předložených vzorů.

1.1 Formální model neuronu

Formální neuron (Obr. 1.1) má konečný počet n vstupů, které přenášejí vstupní hodnoty x_i, $1 \leq i \leq n$, do neuronu. Každý neuron nejprve vypočte hodnotu svého vnitřního potenciálu ξ, který odpovídá váženému součtu složek vstupního vektoru neuronu a tzv. prahové hodnoty θ (Vz. 1.1). Na výsledek potom aplikuje nelineární přenosovou funkci $f(\xi)$, kterou může být např. skoková přenosová funkce (1.1a)).



Obr. 1.1 - Formální model neuronu

$$\xi = \sum_{i=1}^{n} x_i w_i + \theta \qquad \textit{(Vz. 1.1)}$$

$$y = f(\xi) \qquad \textit{(Vz. 1.2)}$$

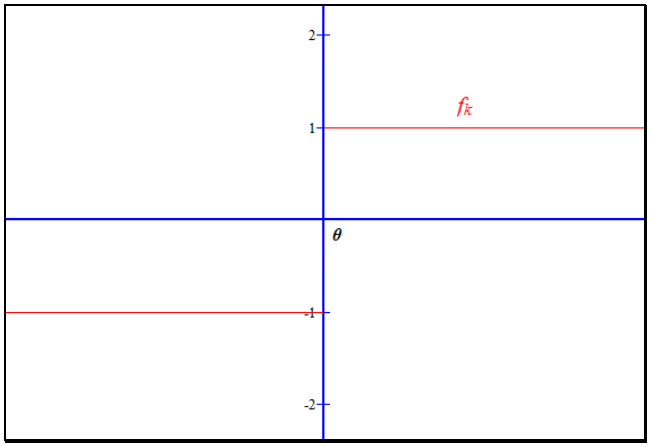
1.2 Základní přenosové funkce

Přenosová funkce je aplikována na vnitřní potenciál neuronu a její výstupní hodnota je posílána rozvětveným výstupním kanálem do dalších neuronů, případně se jedná o výstupní hodnoty neuronové sítě, pokud se neuron nachází ve výstupní vrstvě. Zde je výčet některých nejpoužívanějších přenosových funkcí pro umělé neuronové sítě:

a) *Skoková přenosová funkce f_k*

Nelineární přenosová funkce (Obr. 1.2), kterou zavedl již v roce 1943 McCulloch a Pitts [6]. Výstup *y* je binární, závisí na velikosti vnitřního potenciálu neuronu. Pokud potenciál překročí danou hodnotu prahu *θ*, pak výstupem je *1*, jinak *-1*.

$$\begin{aligned} f_k(\xi) &= -1 \quad pro \quad \xi < \theta \\ f_k(\xi) &= 1 \quad pro \quad \xi \geq \theta \end{aligned} \qquad \textit{(Vz. 1.3)}$$



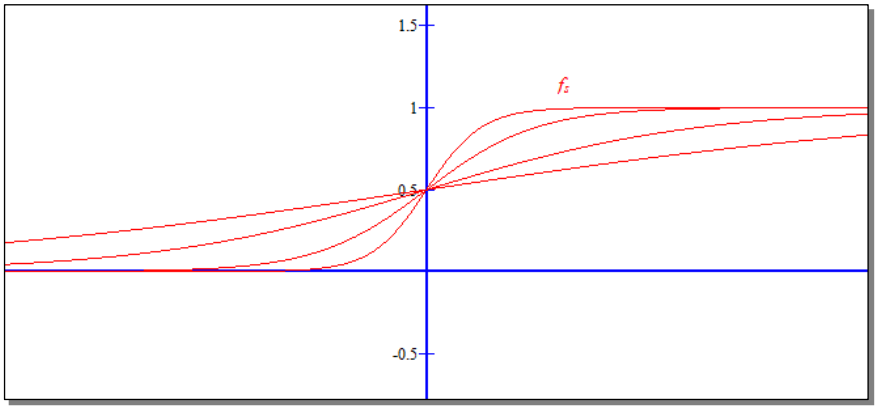
Obr. 1.2 - Skoková přenosová funkce

b) *Sigmoidální přenosová funkce f_s*

Nelineární přenosová funkce (Obr. 1.3), která má spojitou první derivaci v každém bodě definičního oboru. Tvar funkce závisí na nezáporném parametru λ, který nazveme strmostí funkce. Pro vyšší hodnoty λ se tvar funkce více blíží skokové přenosové funkci (Obr. 1.2). Funkci skokovou nahrazujeme funkcí sigmoidální zejména při práci se spojitými vstupními daty.

$$f_s[\lambda](\xi) = \frac{1}{1+e^{-\lambda\xi}} \qquad \textit{(Vz. 1.4)}$$

$$\lim_{\xi\to-\infty} f_s(\xi) = 0$$
$$\lim_{\xi\to\infty} f_s(\xi) = 1$$
$$f_s(0) = 0{,}5$$



Obr. 1.3 - Sigmoidální přenosové funkce s hodnotami strmosti 1,5 a 10.

c) *Přenosová funkce hyperbolický tangens f_h*

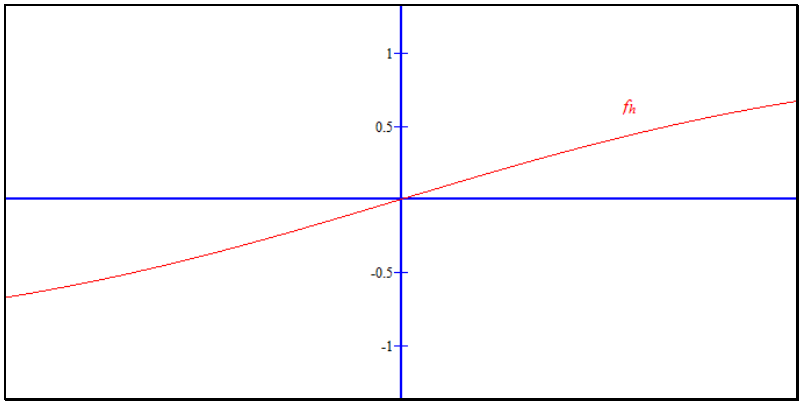
Přenosová funkce hyperbolický tangens (Obr. 1.4) je symetrická podle počátku, což je výhodná vlastnost pro výpočtové operace procesu učení neuronové sítě. Podle (Vz. 1.5) je její tvar podobný sigmoidální přenosové funkci.

$$f_h\left(\frac{\xi}{2}\right) = 2\,f_s[1](\xi) - 1 = \frac{1-e^{-\xi}}{1+e^{-\xi}} \qquad \textit{(Vz. 1.5)}$$

$$\lim_{\xi \to -\infty} f_h(\xi) = -1$$

$$\lim_{\xi \to \infty} f_h(\xi) = 1$$

$$f_h(0) = 0$$



Obr. 1.4 - Přenosová funkce hyperbolický tangens

Přenosovou funkci hyperbolický tangens pro hodnoty blížící se nule zleva i zprava lze snadno aproximovat pro $f_h(x) \approx x$.

2 Vrstevnatá neuronová síť

Umělá neuronová síť je prostředkem pro zpracování složitých dat, využívajícím ke své práci množství jednoduchých navzájem propojených výpočetních jednotek. Ve velké míře je inspirována architekturou lidského mozku, proto je schopna se učit a později tyto znalosti využívat.

První model umělé neuronové sítě popsal v roce 1958 psycholog Frank Rosenblatt [7]. Byla nazvána *Perceptron* a smyslem její existence mělo být modelování postupu, při kterém lidský mozek zpracovává vizuální data a učí se rozeznávat objekty. Rosenblattovi následovníci testovali a používali podobné sítě ke studiu rozpoznávání a procesu učení v biologických neuronových sítích.

Postupem času se umělé neuronové sítě začaly využívat nejen k biologickému studiu, ale i k řešení obecně nelineárních problémů, kde je vhodné pracovat s pravděpodobností a generalizací. Neuronové sítě totiž umožňují postihnout mnoho problémů, které byly a jsou velice obtížné, nebo zcela neřešitelné standardními výpočetními a statistickými metodami. Do konce 80. let 20. století začalo mnoho institucí používat umělé neuronové sítě pro rozličné účely. Dnes hrají nezastupitelnou roli v ekonomice, zpracování dat, biologii, grafice a mnoha jiných oborech.

2.1 Umělá neuronová síť

Umělá neuronová síť je matematický model, který se skládá z konečné neprázdné množiny neuronů navzájem propojených synaptickými spoji.

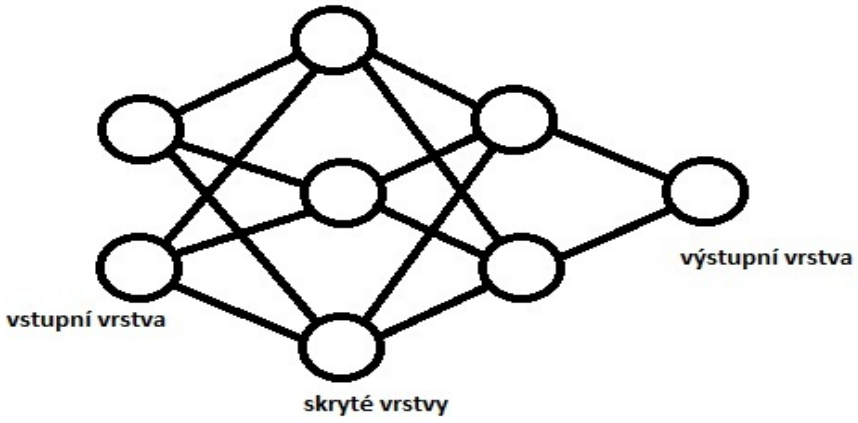
Přesněji ji lze definovat jako uspořádanou šestici $M = (N,C,I,O,w,t)$, kde:

- N konečná neprázdná množina neuronů,
- $C \subseteq N \times N$ neprázdná množina orientovaných spojů mezi neurony,
- $I \subseteq N$ neprázdná množina vstupních neuronů,
- $O \subseteq N$ neprázdná množina výstupních neuronů,
- $w : C \rightarrow \Re$ váhová funkce, $\Re$ značí reálná čísla,
- $t : N \rightarrow \Re$ prahová funkce.

Podle variant propojení a rozdělení neuronů do vrstev rozlišujeme různé modely neuronových sítí.

Jako první byl popsán již zmíněný *Perceptron*, což je původně jednovrstevná neuronová síť, jejíž váhy a prahy mohou být adaptovány pro dosažení požadovaného výstupu. Později se tento model zobecnil na vícevrstvé perceptrony. Mezi dalšími modely můžeme zmínit například Hopfieldův model, Kohonenovy samoorganizující se mapy aj. (více viz [1]).

V této práci se omezíme pouze na vrstevnaté neuronové sítě (Obr. 2.1).



Obr. 2.1 – Vícevrstvá neuronová síť architektury 2-3-2-1.

2.2 Vrstevnatá neuronová síť

Vrstevnatá neuronová síť se skládá z konečného počtu vrstev, z nichž každá obsahuje nenulový konečný počet neuronů. Synapse jsou vedeny pouze z neuronu jedné vrstvy do neuronu vrstvy bezprostředně následující, nikoli ovšem v rámci jedné vrstvy.

První vrstvu budeme označovat jako vrstvu vstupní a poslední jako vrstvu výstupní, ostatní vrstvy jsou skryté. Aktivita neuronů v každé vrstvě se určí podle vnitřního potenciálu ξ daného neuronu (Vz. 1.1), na který je aplikována sigmoidální přenosová funkce (Vz. 1.4):

$$y_j = f_s(\xi_j) = \frac{1}{1+e^{-\lambda \xi_j}} \qquad \textit{(Vz. 2.1)}$$

Neurony uvnitř jedné vrstvy přitom mění svou aktivitu postupně po jednotlivých vrstvách od vstupní vrstvy směrem k výstupní.

2.3 Trénovací množina dat

Trénovací množinu dat pro neuronovou síť můžeme definovat jako konečnou neprázdnou množinu uspořádaných dvojic

$$P = \{[\vec{m}_i, \vec{n}_i] | 1 \leq \mathrm{i} \leq \mathrm{k}\},$$
$$\vec{m}_i \in R^u, \quad \vec{n}_i \in R^v,$$

kde k je velikost množiny, $\vec{m}_i$ je vektor dimenze u vstupní vrstvy a $\vec{n}_i$ vektor dimenze v výstupní vrstvy neuronové sítě. Trénovací množinou dat tedy chápeme množinu vzorových vstupních dat společně s jejich požadovanými výstupy.

Skutečné výstupy neuronové sítě lze porovnat se vzorovými výstupy z trénovací množiny dat a pomocí vhodné účelové funkce vyjádřit jejich odchylku.

2.4 Využití

V současné době patří vrstevnaté neuronové sítě k nejpoužívanějším modelům neuronových sítí s rozsáhlým využitím v oblasti medicíny, ekonomie, letecké dopravy, umělé inteligence a mnoha jiných.

Jiným využitím může být predikce časových řad ve finančnictví, na toto téma vznikla řada odborných studií [8].

3 Proces učení vrstevnatých neuronových sítí

Proces učení neuronové sítě chápeme jako optimalizační proces, ve kterém optimalizujeme tzv. účelovou (cílovou, chybovou) funkci. Tato funkce vhodným způsobem reprezentuje rozdíl mezi skutečným a požadovaným výstupem. Nejčastěji používaným druhem cílové funkce je drobná modifikace střední kvadratické odchylky:

$$E = \frac{1}{2|P|} \sum_{d} \sum_{j} (y_{d,j} - n_{d,j})^2 \qquad \textit{(Vz. 3.1)}$$

kde *d* indexuje předkládané vzory z trénovací množiny *P*, *j* indexuje výstupní neurony, *y* představuje skutečný výstup příslušného neuronu a *n* jeho požadovaný výstup.

Proces učení tedy můžeme definovat jako adaptaci vah a prahů neuronové sítě podle předložených vzorů a zvoleného algoritmu učení tak, abychom minimalizovali odchylku skutečných a požadovaných výstupů. Optimalizace chyby v tomto smyslu znamená ukládání předložené informace do neuronové sítě.

3.1 Způsoby dělení procesu učení neuronových sítí

Proces učení neuronové sítě lze dělit dlemnoha kritérií. S ohledem na studium vzájemných vztahů a souvislostí předkládaných vzorů dělíme proces učení na asociované a neasociované učení. Při neasociativním učení jsou síti opakovaně předkládány vzory bez studia jejich vzájemných souvislostí a podobných vlastností. Cílem tohoto učení je uložení informace a možnost jejího následného vybavení. Naopak při asociativním učení se extrahují vzájemné vztahy mezi jednotlivými vzory nebo skupinami vzorů.

Jiný způsob dělení si všímá opakovaného předkládání vzorů. Jednorázové učení lze charakterizovat tím, že nejlepšího výsledku je dosaženo v jednom učícím cyklu, proto nedochází k žádným dodatečným úpravám parametrů. Vžité představě o učení odpovídá opakované učení, při kterém jsou opakovaně předkládány neuronové síti vzory ve snaze o optimalizaci účelové funkce. Tento typ učení může mít různé varianty ve způsobu předkladaní vzorů. Jedním z nich je tzv. posilované učení. U tohoto způsobu se zasahuje i do průběhu učebního procesu na zakladě průběžných výsledků.

Nejčastější způsob dělení si všímá zpětné vazby - učení s učitelem, resp. bez učitele. Při učení s učitelem předkládáme síti trénovací množinu dat, pomocí které opakovaně adaptujeme váhy a prahy neuronové sítě tak, aby odchylka (účelová funkce) byla co nejnižší.

Jsou-li předložené výstupní a vstupní hodnoty stejné, pak proces nazýváme auto-asociativní učení, jinak se jedná o heteroasociativní učení.

Dle časování iterace algoritmu může proces učení probíhat dvěma způsoby, a sice off-line a on-line. Při variantě off-line se adaptují synaptické vahy a prahy až po přečtení všech předložených vzorů. Iterace algoritmu tedy proběhne až tehdy, když neuronová síť zpracuje všechny vzory z trénovací množiny. Naproti tomu při variantě učení on-line jsou parametry sítě adaptovány vždy po zpracování každého vzoru z trénovací množiny.

Učení bez učitele, nebo také samoorganizace, je založeno na schopnosti neuronových sítí rozeznat ve vstupních vzorech stejné nebo podobné vlastnosti a zpracovávat, případně třídit vstupní data dle nabytých znalostí. Podobná vstupní data jsou pak sdružována do shluků (clusters), proto tento druh učení také nazýváme klastrování. Pro tuto metodu vzniklo mnoho algoritmů [11], které jsou však nad rámec této práce.

V této práci se zaměříme pouze na algoritmy pro učení vrstevnatých sítí s učitelem.

3.2 Algoritmy pro adaptaci vah a prahů

Mezi nejznámější algoritmy pro adaptaci vah a prahů neuronové sítě v procesu učení s učitelem řadíme algoritmus zpětného šíření, který popsali již v roce 1968 Arthur Earl Bryson a Yu-Chi Ho [12]. Tento algoritmus využívá zpětného síření chyby a gradientní metodu, která na základě vypočtené záporné parciální derivace účelové funkce podle konkrétní váhy adaptuje hodnotu dané váhy. Metoda zpětného šíření chyby se později ukázala jako velmi užitečná a implementačně nenáročná, proto její momentová modifikace se stala jedním z nejrozšířenějších algoritmů pro adaptaci vah a prahů ve vrstevnatých neuronových sítích.

Na principu zpětného šíření chyby byly založeny mnohé další algoritmy, které se liší svými parametry a adaptačním pravidlem. Tyto algoritmy můžeme dělit dle výpočtové náročnosti a využívané informace. Základní algoritmus zpětného šíření a jeho modifikace řadíme mezi gradientní algoritmy prvního řádu, jelikož pro adaptaci vah využívá výpočet první parciální derivace účelové funkce podle dané váhy. Mezi nejznámější varianty algoritmu zpětného šíření řadíme algoritmy Delta-bar-delta, Silva & Almeida, Super SAB, které jsou stručně popsány níže (3.3).

Jelikož algoritmy druhého řádu se ukázaly jako výpočetně i paměťově náročné a složitostně nevyhovující, zrodila se aproximační větev, která pro adaptaci neuronové sítě využívá aproximace částí Hessovské matice. Do aproximačních algoritmů druhého řádu řadíme algoritmy konjugovaných gradientů nebo Levenberg-Marquardtův algoritmus, který aproximuje Hessovu matici pomocí Jacobiho matice prvních parciálních derivací účelové funkce podle vah a prahů.

Jiný přístup k adaptaci vah a prahů používají tzv. relaxační metody. Tyto metody se snaží vyhnout výpočtu gradientů neuronové sítě v každé iteraci algoritmu a proto zavádějí do váhového vektoru náhodnou pertubaci. Porovnáváním nové a staré účelové funkce aktualizují parametry a vhodně adaptují příslušnou váhu.

3.3 Vybrané algoritmy pro učení vrstevnatých sítí

Pro důkladnější pochopení problematiky procesu učení je v následující podkapitole stručně popsáno několik vybraných algoritmů. Jedná se o algoritmus Quickprop, Levenberg-Marquardtův algoritmus, algoritmus Super SAB, algoritmus Silvy&Almeidy a algoritmus Delta-bar-delta.

a) *Algoritmus Quickprop*

Algoritmus je založen na nezávislých optimalizačních krocích pro každou váhu, přičemž využívá kvadratickou aproximaci účelové funkce. Adaptační pravidlo využívá hodnoty parciálních derivací z poslední iterace, což zvyšuje paměťovou náročnost.

$$\Delta^{(k)} w_{ij} = \Delta^{(k-1)} w_{ij} \cdot \left(\frac{\nabla_{ij} E^{(k)}}{\nabla_{ij} E^{(k-1)} - \nabla_{ij} E^{(k)}} \right) \qquad \textit{(Vz. 3.2)}$$

kde $\Delta^{(k)} w_{ij}$ je změna váhy w_{ij} v *k*-té iteraci, $\nabla_{ij} E^{(k)}$ značí parciální derivaci účelové funkce *E* podle váhy w_{ij} v *k*-tém kroku algoritmu.

Pokud přepíšeme (Vz. 3.2) do podoby

$$\Delta^{(k)} w_{ij} = \frac{\nabla_{ij} E^{(k)}}{\left(\nabla_{ij} E^{(k-1)} - \nabla_{ij} E^{(k)} \right) / \Delta^{(k-1)} w_{ij}} \qquad \textit{(Vz. 3.3)}$$

potom vidíme, že jmenovatel odpovídá diskrétní aproximaci parciální derivace druhého řádu účelové funce *E* podle váhy w_{ij}.

V závislosti na hodnotě parciálních derivací, aktualizace váhy vypočtená adaptačním algoritmem může být poměrně vysoká. Bylo publikováno několik drobných modifikací algoritmu, které tento problém částečně řeší, lze totiž do algoritmu zavést další parametr, který hlídá velikost kroku [1].

b) ***Levenberg-Marquardtův algoritmus***

Algoritmus patří mezi adaptivní aproximační algoritmy druhého řádu, který kombinuje gradientní a Newtonovu metodu. Za předpokladu, že účelová funkce je vyjádřena vybraným druhem kvadratické ochylky, můžeme Hessovskou matici aproximovat jako

$$H = J^T J \qquad \textit{(Vz. 3.4)}$$

kde J je Jakobiho matice obsahující první parciální derivace účelové funkce podle vah a prahů.

Určení Jacobiho matice je v tomto případě výpočetně méně náročnější, než vypočtení Hessovské matice.

Adaptační pravidlo poté může být vyjádřeno jako

$$\Delta^{(k)} w_i = -\left[J^T J + \mu I\right]^{-1} J^T \vec{e} \qquad \textit{(Vz. 3.5)}$$

kde μ je vhodně zvolený skalár kontrolující chování algoritmu, J je Jakobiho matice obsahující první parciální derivace účelové funkce podle vah a prahů, I je jednotková matice a $\vec{e}$ je vektor chyb neuronové sítě.

Pro $\mu = 0$ algoritmus implementuje Newtonovu metodu aproximující Hessovskou matici, pro vysoké μ algoritmus snižuje velikost kroku algoritmu.

c) ***SuperSAB***

Rozšíření algoritmu zpětného šíření, které je řádově rychlejší a robustní vzhledem k volbě počátečních parametrů. Algoritmus využívá momentové modifikace, což umožnuje urychlení

konvergence v plochých oblastech váhového prostoru, v příkrých oblastech naopak tlumí oscilace [12].

Využívá parametry učení pro každou váhu a práh neuronové sítě a multiplikativní konstanty pro zvětšování a zmenšování těchto parametrů (obvykle $\alpha^+ = 1{,}05, \alpha^- = 2$).

Na začátku procesu učení jsou nejprve nastaveny všechny parametry učení na počáteční hodnotu (obvykle $\alpha_{ij} = 1{,}2$).

Krok I. Pro *k*-tý krok algoritmu je nejprve proveden k-tý krok algoritmu zpětného šíření s momentem, který využívá příslušný parametr učení pro danou váhu, moment učení je volen nejčastěji jako hodnota 0,3.

Krok II. Pokud se nezměnilo znaménko parciální derivace účelové funkce podle dané váhy, pak jsou všechny parametry učení modifikovány pomocí multiplikativní konstanty pro zvětšování parametru učení.

$$\alpha_{ij}^{(k+1)} = \alpha^+ \cdot \alpha_{ij}^{(k)} \qquad \textit{(Vz. 3.6)}$$

Krok III. Pokud se znaménko změnilo, pak je anulována předchozí změna vah, která způsobila změnu znaménka parciální derivace

.

$$\Delta^{(k+1)} w_{ij} = -\Delta^{(k)} w_{ij} \qquad \textit{(Vz. 3.7)}$$

Dále zmenšíme parametr učení

$$\alpha_{ij}^{(k+1)} = \alpha_{ij}^{(k)} / \alpha^- \qquad \textit{(Vz. 3.8)}$$

a vynulujeme změnu vah tak, aby se při dalším kroku algoritmu tato změna neprojevila

.

$$\Delta^{(k+1)} w_{ij} = 0 \qquad \textit{(Vz. 3.9)}$$

Krok IV. Přejdeme ke **Kroku II**.

d) ***Algoritmus Silvy & Almeidy***

Modifikace algoritmu zpětného šíření, která urychluje nebo zpomaluje proces učení v závislosti na znaménku parciální derivace účelové funkce podle příslušné váhy. Pro každý práh a váhy je využit parametr učení, který je aktualizován pomocí vhodně zvolených skalárních konstant $u > 1, d < 1$. Důležitým předpokladem je kvadratická chybová funkce.

V k-té iteraci je aktualizován parametr učení následující způsobem

$$\begin{aligned} \alpha_{ij}^{(k+1)} &= \alpha_{ij}^{(k)} \cdot u \qquad pokud\ \nabla_{ij} E^{(k)} \cdot \nabla_{ij} E^{(k-1)} \geq 0 \\ \alpha_{ij}^{(k+1)} &= \alpha_{ij}^{(k)} \cdot d \qquad pokud\ \nabla_{ij} E^{(k)} \cdot \nabla_{ij} E^{(k-1)} < 0 \end{aligned} \qquad \textit{(Vz. 3.10)}$$

kde $\alpha_{ij}^{(k)}$ je parametr učení pro váhu w_{ij} v *k*-tém kroku algoritmu a $\nabla_{ij} E^{(k)}$ je parciální derivace účelové funkce *E* podle váhy w_{ij} v *k*-tém kroku.

Adaptační pravidlo pro změnu váhy je zvoleno jako

$$\Delta^{(k)} w_{ij} = -\alpha_{ij}^{(k)} \cdot \nabla_{ij} E^{(k)} \qquad \textit{(Vz. 3.11)}$$

Algoritmus nemusí dávat uspokojivé výsledky, pokud se opakuje více urychlovacích kroků po sobě [1].

e) ***Algoritmus Delta-bar-delta***

Drobná modifikace algoritmu Silvy & Almeidy, která klade větší důraz na urychlení učení. Toto urychlení je žádoucí především v případně malých počátečních vah.

Opět jsou vhodně zvoleny skalární konstanty $u > 1, d < 1$, které v každé iteraci algoritmu aktualizují parametr učení pro příslušnou váhu následujícím způsobem

$$\begin{aligned} \alpha_{ij}^{(k+1)} &= \alpha_{ij}^{(k)} + u \qquad pokud\ \nabla_{ij} E^{(k)} \cdot \delta_{ij}^{(k-1)} > 0 \\ \alpha_{ij}^{(k+1)} &= \alpha_{ij}^{(k)} \cdot d \qquad pokud\ \nabla_{ij} E^{(k)} \cdot \delta_{ij}^{(k-1)} < 0 \\ \alpha_{ij}^{(k+1)} &= \alpha_{ij}^{(k)} \qquad jinak \end{aligned} \qquad \textit{(Vz. 3.12)}$$

kde $\alpha_{ij}^{(k)}$ je parametr učení pro váhu w_{ij} v *k*-tém kroku algoritmu, $\nabla_{ij} E^{(k)}$ je parciální derivace účelové funkce *E* podle váhy w_{ij} v *k*-tém kroku a

$$\delta_{ij}^{(k)} = (1 - \Phi) \nabla_{ij} E^{(k)} + \Phi \delta_{ij}^{(k-1)} \qquad \textit{(Vz. 3.13)}$$

kde Φ je vhodně zvolená skalární konstanta.

Aktualizace vah potom odpovídá pravidlu

$$\Delta^{(k)} w_{ij} = -\alpha_{ij}^{(k)} \nabla_{ij} E^{(k)} \qquad \textit{(Vz. 3.14)}$$

3.4 Strategie učení

Existuje řada algoritmů pro adaptaci vah a prahů v procesu učení vrstevnatých neuronových sítí s učitelem. Pro korektní naučení informace obsažené v předložených vzorech je potřeba sledovat průběh účelové funkce během procesu učení. Vhodný průběh učení se projevuje hladkou účelovou funkcí, která ukazuje na vhodně zvolenou velikost trenovací množiny a parametry učení, pokud nějaké algoritmus vyžadoval. Naproti tomu nevhodný průběh učení se pozná podle příliš plochého nebo oscilujícího průběhu účelové funkce [1].

Častým problémem při procesu učení neuronových sítí je stav přetrénovaní. Je to stav, kdy se neuronová síť specializuje na rozpoznávání předkládaných vzorů na úkor schopnosti generalizace a částečně tak ztrácí nabytou informaci. Při přetrénovaní roste účelová funkce na vzorech, které sice nejsou obsaženy v předložené trénovací množině, ale obsahují stejnou či podobnou vlastnost nebo informaci jako předkládané vzory. Abychom se vyhnuli procesu přetrénování, je nutno všechny dostupné vzory rozdělit na část validační a testovací. Validační množina zpravidla obsahuje 10% - 50% všech dostupných vzorů, tento údaj záleží na konkrétním řešeném problému. Ostatní vzory jsou většinou obsaženy v testovací množině.

Při procesu učení je vhodné sledovat průběh účelové funkce nejen na testovací, ale i na validační množině. Ve chvíli, kdy účelová funkce na validační množině dosáhne svého minima, proces učení ukončíme. Pokud bychom pokračovali dále, neuronová síť by se specializovala pouze na předložené vzory a ztrácela by schopnost generalizace.

V ideálním případě má účelová funkce na validační množině hladký průběh, neosciluje a její jediné minimum je právě globální minimum. V obecném případě je ale průběh účelové funkce oscilující nebo dokonce rostoucí. Neuronová síť se v tomto případě učí velmi složitě, případně vůbec.

3.5 Prořezávání

Pokud v neuronové síti s většími hodnotami vah existují váhy s řádově nižší hodnotou, bývá jejich činnost výrazně utlumena a nepodílejí se významnější měrou na výsledku. Lze je tedy bez náhrady eliminovat. Dojde tak k částečnému zjednodušení modelu a obvykle ke zlepšení generalizace. Čím méně parametrů, tím více se neuronová síť zaměřuje na obecnou podstatu dat a stoupá výpočetní rychlost. Jako další důsledek také klesá nebezpečí přeučení [9]. Tato problematika je více pojednána např. [11].

3.6 Metody regularizace

Regularizace je technika, která se při procesu učení s pomocí dodatečné informace snaží vyhnout stavu přeučení neuronové sítě.

Mezi nejznámější metody regularizace řadíme predikci počtu efektivně využitých neuronů a návrh architektury neuronové sítě. Jelikož architektura sítě má na proces učení velký vliv, je nutné správně odhadnout počet neuronů ve skrytých vrstvách tak, aby se neuronová síť dokázala správně adaptovat na předložené vstupní vzory. Pokud je počet efektivně využitých prahů a vah nízký, dochází k častým oscilacím, naopak, pokud je vysoký, neuronová síť se nedokáže dostatečně adaptovat na vstupní vzory [10].

Mezi další metody regularizace řadíme modifikace výpočtu účelové funkce pomocí rozkladu vah, normalizaci vstupních a výstupních dat, případně jejich kombinace [10]. V současné době je velice používaná metoda Bayesovské regularizace, která minimalizuje kombinaci účelové funkce a vah tak, aby vytvořená neuronová síť byla schopna co nejlépe generalizovat.

Mezi alternativní způsoby předcházení přeučení neuronové sítě řadíme již zmíněné prořezávání (3.5) a předčasné ukončení procesu učení na základě validační množiny (3.4).

3.7 Zhodnocení a implementace zvolených algoritmů

Součástí této práce je softwarová implementace vybraných algoritmů, které vhodným způsobem reprezentují nejrozšířenější algoritmy pro adaptaci vrstevnatých neuronových sítí.

Po důkladném zvážení byly vybrány 2 algoritmy, a sice algoritmus zpětného šíření s momentem a algoritmus škálovaných konjugovaných gradientů.

Algoritmus zpětného šíření je implementačně nenáročný a metodu zpětného šíření chyby lze využít pro výpočet parciálních derivací, což je stěžejní pro případné rozšíření implementace o další algoritmy. Algoritmus je navíc dostatečně rychlý a univerzálně použitelný.

Jako další implementovaný algoritmus byl zvolen algoritmus škálovaných konjugovaných gradientů, který řadíme mezi extrémně rychlé algoritmy druhého řádu. Byl zvolen především kvůli své paměťové a výpočtové nenáročnosti, neboť nevyžaduje ukládání matic v průběhu iterace algoritmu. Algoritmus je založen na aproximaci Hessovské matice, ale využívá pouze parciální derivace prvního řádu, proto není složitostně náročnější než výše zmíněný algoritmus zpětného šíření s momentem, ale konvergenční rychlost je na vybraných problémech řádově vyšší [5].

Oba algoritmy jsou podrobně popsány v následujících kapitolách a implementovány v rámci aplikace NeDro.

4 Algoritmus zpětného šíření

Algoritmus zpětného šíření je jedním z nejpoužívanějších algoritmů pro adaptaci vah vrstevnatých neuronových sítí. Cílem tohoto algoritmu je nalézt množinu vah takových, aby každý vstup z trénovací množiny vydal jako výsledek pokud možno požadovaný výstup, tzn. aby odchylka mezi skutečným výstupem a požadovaným výstupem odpovídajícího trénovacího vzoru byla co nejmenší.

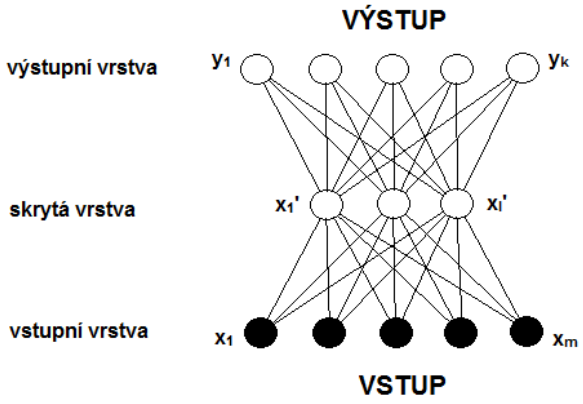
4.1 Adaptace vah a prahů

K optimalizaci účelové funkce se používá tzv. gradientní metoda. V průběhu učení je nutné nastavit hodnoty vah a prahů v síti tak, aby byla hodnota účelové funkce *E* co možná nejmenší. Přitom budeme prahy reprezentovat jako váhy vycházející z fiktivního neuronu s konstantním výstupem rovným *1*. Dále tedy budeme mluvit jen o adaptaci vah.

Vidíme, že hodnotu každé váhy je třeba změnit o zápornou parciální derivaci *E* podle této váhy, abychom zmenšili chybu *E*. Označíme-li tuto hodnotu jako $\Delta_E w_{ij}$, platí:

$$\Delta_E w_{ij} \cong -\frac{\partial E}{\partial w_{ij}} \qquad \textit{(Vz. 4.1)}$$

Pro jednoduchost dále nebudeme uvažovat index *d* pro trénovací množinu dat ani u požadovaných, ani u skutečných výstupních hodnot neuronů, *n* a *y*. Z podobných důvodů budeme dále indexovat neurony z vrstvy pod příslušným neuronem *j* jako *i* a neurony z vrstvy nad neuronem *j* jako *k*. Váhy mezi neuronem indexovaným i a neuronem bezprostředně následující vrstvy indexovaným j budeme označovat jako w_{ij}, y_i značíme skutečné výstupy, kde *i* indexuje výstupní neurony.



$$y_j = f\left(\sum_{i=1}^{l} w'_{ij}\, x'_i + \theta'_j\right) \qquad 1 \le j \le k$$

$$x'_i = f\left(\sum_{h=1}^{m} w_{hi}\, x_h + \theta_i\right) \qquad 1 \le i \le l$$

Obr. 4.1 – Třívrstvá síť s m vstupními a k výstupními neurony a jednou skrytou vrstvou s l skrytými neurony. Jako nelinearitu lze použít např. sigmoidální přenosovou funkci (1.1b)). Hodnoty x_i' zde představují výstupy neuronů ze skryté vrstvy, θ_j' jejich prahy, w_{hi} značí váhu synapse mezi vstupními neurony a neurony skryté vrstvy, w'_{ij} jsou vahami mezi neurony skryté vrstvy a neurony vrstvy výstupní. Hodnoty x_h představují vstupní hodnoty neuronů vstupní vstvy, θ_j jejich prahy, y_j značí hodnoty výstupních neuronů výstupní vrstvy.

K výpočtu hodnot $\Delta_E w_{ij}$ lze použít algoritmus zpětného šíření. Nejdříve pro předložený trénovací vzor určíme skutečný výstup sítě (Obr. 3.1), který porovnáme s požadovaným výstupem. Na základě zjištěné odchylky lze pro každou váhu vyjádřit zápornou parciální derivaci E podle této váhy.

Pro výstupní vrstvu potom platí

$$\Delta_E w_{ij} = -\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial \xi_j}\frac{\partial \xi_j}{\partial w_{ij}} = -\frac{\partial E}{\partial \xi_j}\frac{\partial}{\partial w_{ij}}\sum_{i'} w_{i'j}\, y_{i'} =$$

$$= -\frac{\partial E}{\partial \xi_j} y_i = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \xi_j} y_i = (n_j - y_j)\lambda y_j (1 - y_j) y_i =$$

$$= \delta_j y_i$$

a pro skryté vrstvy

$$\Delta_E w_{ij} = -\sum_k \frac{\partial E}{\partial \xi_k}\frac{\partial \xi_k}{\partial y_j}\frac{\partial y_j}{\partial \xi_j} y_i = -\left(\sum_k \frac{\partial E}{\partial \xi_k}\frac{\partial}{\partial y_j}\sum_j w_{jk}\, y_j\right)\frac{\partial y_j}{\partial \xi_j} y_i$$

$$= -\left(\sum_k \frac{\partial E}{\partial \xi_k} w_{jk}\right)\frac{\partial y_j}{\partial \xi_j} y_i = \left(\sum_k \delta_k w_{jk}\right)\lambda y_j (1 - y_j) y_i =$$

$$= \delta_j y_i$$

Vztah pro aktualizaci vah ve vrstevnatých neuronových sítích tedy bude mít tvar:

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} + \alpha \Delta_E w_{ij} = w_{ij}^{(k)} + \alpha \delta_j y_i \qquad \textit{(Vz. 4.2)}$$

kde $0 \leq \alpha \leq 1$ je parametr učení a člen δ_j odpovídá

$$(n_j - y_j) y_j (1 - y_j) \lambda \quad \text{pro výstupní neuron}$$

a $$\lambda y_j (1 - y_j) \sum_k \delta_k w_{jk} \quad \text{pro skrytý neuron.}$$

Z praktických důvodů je vhodné volit hodnotu parametru učení α co možná největší, umožňuje to rychlejší učení. Na druhou stranu však právě velká hodnota α může způsobit lokální oscilace v procesu učení.

Standardní algoritmus zpětného šíření má mnoho předností. Mezi hlavní přednosti vrstevnatých neuronových sítí učených algoritmem zpětného šíření řadíme jeho aproximační vlastnosti a schopnost vytvářet interní reprezentaci znalostí a správně generalizovat. Mezi další pozitiva patří robustnost sítě a možnost opětovného využití již natrénovaných sítí. Na druhou stranu však musíme zmínit existenci lokálních minim, pomalou konvergenci a obecně je velice obtížné odhadnout význam jednotlivých skrytých neuronů, přičemž konvergenci algoritmu také nelze zaručit [1]. Částečným řešením výše zmíněných problémů může být rozšíření algoritmu o další parametr – tzv. *moment učení*.

4.2 Momentová varianta

Základní algoritmus zpětného šíření rozšíříme o další parametr, který nazveme momentem učení. Adaptační pravidlo bude modifikováno následujícím způsobem:

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} + \alpha \delta_j y_i + \alpha_m (w_{ij}^{(k)} - w_{ij}^{(k-1)}) \qquad \textit{(Vz. 4.3)}$$

(uvažujeme tedy i změny vah v předchozím cyklu)

Modifikovaný algoritmus řadíme ke skupině heuristických optimalizačních metod. Moment snižuje citlivost učení na detaily chybové funkce, pomáhá vyhnout se uvíznutí v lokalním minimu a bývá zpravidla konvergenčně rychlejší než základní varianta algoritmu [1]. Tento typ algoritmu byl implementován v systému NeDro.

4.3 Algoritmus zpětného šíření

Cílem algoritmu zpětného šíření s momentem ve vrstevnatých neuronových sítích je minimalizace účelové funkce *E*. Jeho použití předpokládá nelineární přenosovou funkci, zde je použita sigmoidální přenosová funkce (1.1b)) se strmostí $\lambda = 1$.

Požadovaný vstupní, resp. výstupní vektor z trénovací množiny dat budeme značit $\vec{m}_i$, resp. $\vec{n}_i$ a skutečné výstupy y_i, kde *i* indexuje výstupní neurony.

I. *Inicializace vah a prahů*:

Nastav hodnoty vah a prahů jako malé náhodné hodnoty.

II. *Předkládání nového vzoru sítě*:

Předlož síti vstupní vektor $\vec{m}_i$ z trénovací množiny a specifikuj požadovaný výstupní vektor $\vec{n}_i$.

III. *Výpočet skutečných výstupů*:

Skutečné výstupy $y_1, y_2,...,y_k$, kde *k* je dimenze výstupní vrstvy sítě, se určí pomocí sigmoidální přenosové funkce a vzorců uvedených v (Obr. 3.1). Dále podle (Vz. 3.1) určíme hodnotu účelové funkce *E*. Pokud tato hodnota je dostatečně malá nebo bylo dosaženo limitního počtu opakování cyklu algoritmu, pak skonči.

IV. *Aktualizace vah*:

Postupně od výstupní vrstvy směrem ke vstupní aktualizuj váhy dle vzorce (Vz. 4.2):

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} + \alpha\Delta_E w_{ij} = w_{ij}^{(k)} + \alpha\delta_j y_i$$

V tomto vztahu představuje $w_{ij}(k)$ váhu ze skrytého anebo vstupního neuronu v čase *k*, y_i označuje skutečný výstup neuronu *i*, $0 < \alpha < 1$ je parametr učení a δ_j je chybový člen odpovídající neuronu *j*.

Pokud leží neuron *j* ve výstupní vrstvě,

$$\delta_j = y_j(n_j - y_j)(1 - y_j)$$

kde n_j je požadovaný výstup neuronu j a y_j je jeho skutečný výstup.

Jestliže leží neuron j ve skryté vrstvě, pak

$$\delta_j = y_j(1-y_j)\sum_k \delta_k w_{jk}$$

kde k je index pro neurony z vrstvy nad neuronem j.

Konvergence bývá daleko rychlejší, pokud použijeme modifikované adaptační pravidlo (Vz. 4.3):

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} + \alpha\delta_j y_i + \alpha_m (w_{ij}^{(k)} - w_{ij}^{(k-1)})$$

přičemž $0 < \alpha_m < 1$ nazveme moment učení.

Pokračujeme krokem **II.** a opakujeme cyklus.

5 Algoritmus SCG

Jednotlivé varianty algoritmů pro adaptaci prahů a vah ve vrstevnatých neuronových sítích se navzájem liší modifikací pravidel používaných gradientní metodou, popsanou v (4.1). Tyto algoritmy často závisí na parametrech, které jsou zcela zásadní pro rychlost konvergence daného algoritmu. Typickým příkladem je právě algoritmus zpětného šíření s momentem, popsaný v (4.3), který závisí na *parametru učení* a *momentu učení*.

Využijeme-li faktu, že odchylky skutečných a požadovaných výstupních hodnot neuronové sítě mohou být popsány pomocí účelové funkce, můžeme využít znalostí z jiných oborů matematiky, např. z matematické analýzy a teorie optimalizace. Jelikož proces učení neuronové sítě podle předložených trénovacích vzorů často zahrnuje adaptaci několika tisíc vah, potom je nutné uvažovat pouze metody optimalizace, které jsou aplikovatelné na problémy velkých rozsahů.

K těmto metodám patří i metody konjugovaných gradientů, které jsou mnohdy řádově rychlejší než dosud používaný algoritmus zpětného šíření, složitost těchto algoritmů nicméně zvyšuje hledaní směru největšího poklesu účelové funkce a dále stanovení počtu kroků adaptace vah v tomto směru.

Algoritmus škálovaných konjugovaných gradientů, dále jen *SCG*, který popsal v roce 1993 M. F. Möller [3], eliminuje nutnost hledat směr největšího poklesu díky vlastnostem Hessovy matice. Počet kroků pro daný směr je odhadnut pomocí Levenberg-Marquartovy aproximace a algoritmus se tak stává nedeterministickým z hlediska vstupních parametrů zadávaných uživatelem.

Následující popis algoritmu je převzat z [3].

5.1 Notace

Nechť je dána vrstevnatá neuronová síť. *Váhový vektor* je vektor v reálném euklidovském prostoru $\Re^N$, kde N je počet vah a prahů v neuronové síti. Prahy neuronů budeme opět reprezentovat jako váhy vycházející z fiktivního neuronu s konstantním výstupem rovným *1*, jak tomu bylo i při popisu algoritmu zpětného šíření (4.1).

Nechť $\vec{w}$ je váhový vektor definovaný následujícím předpisem

$$\vec{w} \quad = \quad (\ldots, w_{ij}^{(l)}, w_{i+1,\,j}^{(l)}, \ldots, w_{N_l\,j}^{(l)}, w_{ij+1}^{(l)}, w_{i+1\,j+1}^{(l)}, \ldots) \qquad \textit{(Vz. 5.1)}$$

kde $w_{ij}^{(l)}$ je váha z neuronu *i* ve vrstvě *l* k neuronu *j* ve vrstvě *l+1*.

Účelová funkce *E* nechť je střední kvadratická odchylka, nebo jiná vhodná chybová funkce, pak její derivaci chápeme jako

$$E'(\vec{w}) \quad = \quad \left(\ldots, \sum_{p=1}^{k} \frac{dE_p}{dw_{ij}^{(l)}}, \sum_{p=1}^{k} \frac{dE_p}{dw_{i+1\,j}^{(l)}}, \ldots, \sum_{p=1}^{k} \frac{dE_p}{dw_{N_l\,j}^{(l)}}, \sum_{p=1}^{k} \frac{dE_p}{dw_{ij+1}^{(l)}}, \ldots \right) \qquad \textit{(Vz. 5.2)}$$

kde *k* je počet vzorů z trénovací množiny (2.3) a E_p je chyba spojená se vzorem indexovaným *p*.

Nyní definujme některé vektorové operace, které jsou použity dále v popisu algoritmu: *váhové sčítání, váhový rozdíl, váhový součin, váhový podíl*.

$$\begin{aligned} \vec{w} + \vec{y} &= \left(w^1 + y^1, \ldots, w^i + y^i, \ldots, w^N + y^N\right)^T \\ \vec{w} - \vec{y} &= \left(w^1 - y^1, \ldots, w^i - y^i, \ldots, w^N - y^N\right)^T \\ \vec{w}^T \vec{y} &= \sum_{i=1}^{N} w^i y^i \\ \frac{\vec{w}}{\sigma} &= \sum_{i=1}^{N} \frac{w^i}{\sigma} \end{aligned} \qquad \textit{(Vz. 5.3)}$$

kde $\vec{w} \in \Re^N$ a $\vec{y} \in \Re^N$ jsou váhové vektory dimenze *N* a w^i je *i*-tá složka váhového vektoru; $1 \le i \le N$; $\sigma \in \Re$.

Euklidovská norma je definována jako

$$\left\|\vec{w}\right\| \quad = \quad \left[\sum_{i=1}^{N} \left(w^i\right)^2 \right]^{\frac{1}{2}} \qquad \textit{(Vz. 5.4)}$$

Účelová funkce *E* pro modifikovaný váhový vektor $(\vec{w} + \vec{y})$ může být vyjádřena Taylorovou řadou jako

$$E(\vec{w} + \vec{y}) \quad = \quad E(\vec{w}) + E'(\vec{w})^T \vec{y} + \frac{1}{2} \vec{y}^T E''(\vec{w}) \vec{y} + \ldots \qquad \textit{(Vz. 5.5)}$$

N x N matici *A* nazveme *pozitivně definitní*, pokud platí

$$\vec{y}^T A\vec{y} > 0 \quad \forall \vec{y} \in \Re^N, \vec{y} \neq 0 \qquad \textit{(Vz. 5.6)}$$

Nechť $\vec{p}_1,\ldots,\vec{p}_k$ je množina nenulových váhových vektorů v $\Re^N$. Množinu nazveme konjugovaným systémem vzhledem k nesingulární symetrické *N x N* matici ***A*****,** pokud platí

$$\vec{p}_i^T A\vec{p}_j = 0 \qquad \textit{(Vz. 5.7)}$$

kde $i \neq j, i = 1,\ldots,k$.

Množinu bodů $\vec{w}$ v $\Re^N$, která splňuje

$$\vec{w} = \vec{w}_1 + \alpha_1\vec{p}_1 + \ldots + \alpha_k\vec{p}_k, \quad \alpha_i \in \Re \qquad \textit{(Vz. 5.8)}$$

kde $\vec{w}_1$ je váhový vektor a $\vec{p}_1,\ldots,\vec{p}_k$ je podmnožina konjugovaného systému, nazveme *k-rovinou* nebo π_k.

5.2 Škálované konjugované gradienty

Většina optimalizačních metod je založena na lokálním iterativním procesu, během něhož se minimalizuje hodnota aproximace účelové funkce. Pro aproximaci se většinou používá Taylorův rozklad prvního, příp. druhého stupně dané účelové funkce.

Každá iterace zahrnuje dva nezávislé procesy, které je potřeba vykonat. Prvním je určení směru adaptace, druhým je stanovení velikosti prováděného kroku.

Algoritmus SCG je založen na podobné optimalizační strategii, ale vhodně volí směr adaptace a velikost prováděného kroku užitím informace z druhého stupně Taylorova rozkladu účelové funkce (Vz. 5.5).

Mějme tedy Taylorovu kvadratickou aproximaci účelové funkce pro okolí váhového vektoru $\vec{w}$.

$$E_{\vec{w}}(\vec{y}) = E(\vec{w}) + E'(\vec{w})^T\vec{y} + \frac{1}{2}\vec{y}^T E''(\vec{w})\vec{y} \qquad \textit{(Vz. 5.9)}$$

V případě dosažení minima musí být derivace této funkce nulová

$$E'_{\vec{w}}(\vec{y}) = E''(\vec{w})\vec{y} + E'(\vec{w}) = 0 \qquad \textit{(Vz. 5.10)}$$

Vektory, které splňují tyto podmínku, nazveme kritickými body.

Nechť $\vec{p}_1,\ldots,\vec{p}_k$ je konjugovaný systém. Jelikož tento systém představuje bázi pro $\Re^N$, přechod od počátečního váhového vektoru $\vec{y}_1$ ke kritickému bodu $\vec{y}_*$ může být vyjádřen jako lineární kombinace $\vec{p}_1,\ldots,\vec{p}_k$

$$\vec{y}_* - \vec{y}_1 = \sum_{i=1}^{N} \alpha_i \vec{p}_i \qquad \alpha_i \in \Re \qquad \textit{(Vz. 5.11)}$$

Součinem (Vz. 5.11) a $\vec{p}_j^T E''(\vec{w})$ a substitucí $E'(\vec{w})$ za $-E''(\vec{w})\vec{y}_*$ dostáváme

$$\vec{p}_j^T\left(-E'(\vec{w}) - E''(\vec{w})\vec{y}_1\right) = \alpha_j \vec{p}_j^T E''(\vec{w})\vec{p}_j \quad \Rightarrow$$

$$\alpha_j = \frac{\vec{p}_j^T\left(-E'(\vec{w}) - E''(\vec{w})\vec{y}_1\right)}{\vec{p}_j^T E''(\vec{w})\vec{p}_j} = \frac{-\vec{p}_j^T E'_{\vec{w}}(\vec{y}_1)}{\vec{p}_j^T E''(\vec{w})\vec{p}_j} \qquad \textit{(Vz. 5.12)}$$

Kritického bodu $\vec{y}_*$ může být dosaženo v *N* iteračních krocích použitím (Vz. 5.11) a (Vz. 5.12). Tento bod ovšem nemusí nutně představovat minimum, ale může být sedlovým bodem nebo maximem. Pouze pokud je Hessova matice $E''(\vec{w})$ pozitivně definitní, potom $E_{\vec{w}}(y)$ dosahuje globálního minima [4].

$$\begin{aligned}
E_{\vec{w}}(\vec{y}) &= E_{\vec{w}}(\vec{y}_* + (\vec{y} - \vec{y}_*)) \\
&= E(\vec{w}) + E'(\vec{w})^T(\vec{y}_* + (\vec{y} - \vec{y}_*)) + \frac{1}{2}(\vec{y}_* + (\vec{y} - \vec{y}_*))^T E''(\vec{w})(\vec{y}_* + (\vec{y} - \vec{y}_*)) \\
&= E(\vec{w}) + E'(\vec{w})^T y_* + E'(\vec{w})^T(\vec{y} - \vec{y}_*) + \frac{1}{2}\vec{y}_*^T E''(\vec{w})\vec{y}_* + \frac{1}{2}\vec{y}_*^T E''(\vec{w})(\vec{y} - \vec{y}_*) \\
&\quad + \frac{1}{2}(\vec{y} - \vec{y}_*)^T E''(\vec{w})\vec{y}_* + \frac{1}{2}(\vec{y} - \vec{y}_*)^T E''(\vec{w})(\vec{y} - \vec{y}_*) \qquad \textit{(Vz. 5.13)} \\
&= E_{\vec{w}}(\vec{y}_*) + (\vec{y} - \vec{y}_*)^T(E''(\vec{w})\vec{y}_* + E'(\vec{w})) + \frac{1}{2}(\vec{y} - \vec{y}_*)^T E''(\vec{w})(\vec{y} - \vec{y}_*) \\
&= E_{\vec{w}}(\vec{y}_*) + \frac{1}{2}(\vec{y} - \vec{y}_*)^T E''(\vec{w})(\vec{y} - \vec{y}_*)
\end{aligned}$$

Využitím (Vz. 5.13) a faktu, že $\vec{y}_*$ je minimum, dostáváme, že $\frac{1}{2}(\vec{y} - \vec{y}_*)^T E''(\vec{w})(\vec{y} - \vec{y}_*) > 0$ pro každé $\vec{y}$, proto $E''(\vec{w})$ musí být pozitivně definitní.

Body $\vec{y}_{k+1} = \vec{y}_k + \alpha_k \vec{p}_k$ získané během *N* iteračních kroků k dosažení $\vec{y}_*$ tvoří ve skutečnosti minima pro $E_{\vec{w}}(\vec{y})$ omezená na *k-rovinu* π_k [3]: $\vec{y} = \vec{y}_1 + \alpha_1\vec{p}_1 + \ldots + \alpha_k\vec{p}_k$. Jak rekurzivně volit tyto body vysvětluje následující následující věta. Důkaz můžeme najít v [4].

Věta 1**:** *Nechť* $\vec{p}_1,\ldots,\vec{p}_k$ *je konjugovaný systém a* $\vec{y}_1$ *váhový vektor. Nechť vektory* $\vec{y}_2,\ldots,\vec{y}_{N+1}$ *jsou definovány rekurzivně jako* $\vec{y}_{k+1}=\vec{y}_k+\alpha_k\vec{p}_k$, *kde*

$$\alpha_k=\frac{\mu_k}{\delta_k},\quad \mu_k=-\vec{p}_k^{\,T}E'_{\vec{w}}(\vec{y}_k),\quad \delta_k=\vec{p}_k^{\,T}E''(\vec{w})\vec{p}_k.$$

Potom $\vec{y}_{k+1}$ *minimalizuje* $E_{\vec{w}}$ *omezené na k-rovinu* π_k *danou vektory* $\vec{y}_1$ *a* $\vec{p}_1,\ldots,\vec{p}_k$.

Nyní můžeme formulovat obecný algoritmus konjugovaných gradientů, který zaručuje nalezení globálního minima v nejvýše *N* iteracích. $\vec{p}_1$ se na začátku nastaví proti směru největšího gradientu $-E'_{\vec{w}}(\vec{y}_1)$. $\vec{p}_{k+1}$ jsou určeny rekurzivně jako lineární kombinace aktuálního směru adaptace a $-E'_{\vec{w}}(\vec{y}_{k+1})$ a předchozího směru adaptace $\vec{p}_k$, tedy $\vec{p}_{k+1}$ jsou voleny jako ortogonální projekce $-E'_{\vec{w}}(\vec{y}_{k+1})$ na *(N-k)-rovinu* π_{N-k} konjugovanou k π_k.

Tento postup vychází z následující věty, jejíž důkaz můžeme najít v [4].

Věta 2**:** *Nechť* $\vec{y}_1$ *je váhový vektor a* $\vec{p}_1,\vec{r}_1$ *odpovídají zápornému gradientu* $-E'_{\vec{w}}(\vec{y}_1)$. *Definujme body* $\vec{p}_{k+1}$ *rekurzivně jako* $\vec{p}_{k+1}=\vec{r}_{k+1}+\beta_k\vec{p}_k$, *kde*

$$\vec{r}_{k+1}=-E'_{\vec{w}}(\vec{y}_{k+1}),\quad \beta_k=\frac{\left|\vec{r}_{k+1}\right|^2-\vec{r}_{k+1}\vec{r}_k}{\vec{p}_k^{\,T}\vec{r}_k}$$

a $\vec{y}_{k+1}$ *je bod určený pomocí (Věta 1). Potom* $\vec{p}_{k+1}$ *odpovídá zápornému gradientu* $E_{\vec{w}}$ *omezenému na (N-k)-rovinu* π_{N-k} *konjugovanou k* π_k, *která je určená vektory* $\vec{y}_1$ *a* $\vec{p}_1,\ldots,\vec{p}_k$.

Pro každou iteraci algoritmu je nutno určit a uložit Hessovskou matici $E''(\vec{w}_k)$, což zvyšuje složitost algoritmu. Proto člen $E''(\vec{w}_k)\vec{p}_k$ z *(Věta 1)* odhadneme pomocí následujícího vztahu

$$\vec{s}_k=E''(\vec{w}_k)\vec{p}_k\approx\frac{E'(\vec{w}_k+\sigma_k\vec{p}_k)-E'(\vec{w}_k)}{\sigma_k}\quad ;0<\sigma_k<<1. \qquad \textit{(Vz. 5.14)}$$

Kvadratická aproximace (Vz. 5.9) způsobuje, že algoritmus dává spolehlivé výsledky pouze pro funkce s pozitivně definitní Hessovskou maticí na celém váhovém prostoru, což nemusí být vždy splněno. Řešením tohoto problému je modifikace algoritmu s použitím Leven-Marquardtovy metody. Základní ideou je zavést řídící parametr λ_k, který v každé

iteraci zajistí, že $E''(w_k)$ bude pozitivně definitní . Použijeme tedy následující modifikaci (Vz. 5.14)

$$\vec{s}_k = \frac{E'(\vec{w}_k + \sigma_k \vec{p}_k) - E'(\vec{w}_k)}{\sigma_k} + \lambda_k \vec{p}_k \qquad \textit{(Vz. 5.15)}$$

a pro každou iteraci upravíme λ_k podle znaménka δ_k, které určuje, zda je $E''(\vec{w}_k)$ pozitivně definitní. Pokud $\delta_k \le 0$, potom je λ_k zvýšeno a $\vec{s}_k$ musí být odhadnut znovu. Přejmenujeme-li nové $\vec{s}_k$ jako $\bar{\vec{s}}_k$ a zvýšené λ_k jako $\bar{\lambda}_k$, potom $\bar{\vec{s}}_k$ odpovídá

$$\bar{\vec{s}}_k = \vec{s}_k + (\bar{\lambda}_k - \lambda_k)\vec{p}_k \qquad \textit{(Vz. 5.16)}$$

V Leven-Marquardtově aproximaci je pro snížení λ_k použit konstantní faktor, protože algoritmus sám neobsahuje informaci o tom, o kolik musí být λ_k zvýšeno. Tuto informaci je však možné odhadnout následujícím způsobem

$$\bar{\delta}_k = \vec{p}_k^T \bar{\vec{s}}_k = \vec{p}_k^T (\vec{s}_k + (\bar{\lambda}_k - \lambda_k)\vec{p}_k) = \delta_k + (\bar{\lambda}_k - \lambda_k)\|\vec{p}_k\|^2 > 0 \quad \Rightarrow$$
$$\bar{\lambda}_k > \lambda_k - \frac{\delta_k}{\|\vec{p}_k\|^2} \qquad \textit{(Vz. 5.17)}$$

Vidíme, že volba $\bar{\lambda}_k$ není přesně definována a proto ji můžeme zvolit takto

$$\bar{\lambda}_k = 2\left(\lambda_k - \frac{\delta_k}{\|\vec{p}_k\|^2}\right) \qquad \textit{(Vz. 5.18)}$$

Potom

$$\bar{\delta}_k = \delta_k + (\bar{\lambda}_k - \lambda_k)|\vec{p}_k|^2 = \delta_k + \left(2\lambda_k - 2\frac{\delta_k}{\|\vec{p}_k\|^2} - \lambda_k\right)\|\vec{p}_k\|^2$$
$$= -\delta_k + \lambda_k \|\vec{p}_k\|^2 > 0 \qquad \textit{(Vz. 5.19)}$$

Velikost kroku je zvolena jako

$$\alpha_k = \frac{\mu_k}{\delta_k} = \frac{\mu_k}{\vec{p}_k^T \vec{s}_k + \lambda_k \|\vec{p}_k\|^2} \qquad \textit{(Vz. 5.20)}$$

Protože λ_k upravují Hessovskou matici uměle, může být kvadratická aproximace (Vz. 5.9) v některých bodech $\Re^N$ špatná. Pro vyřešení tohoto problému zavedeme do algoritmu nový člen Δ_k, který vyjadřuje míru přesnosti dané aproximace.

$$\Delta_k = \frac{E(\vec{w}_k) - E(\vec{w}_k + \alpha_k \vec{p}_k)}{E(\vec{w}_k) - E_{\vec{w}}(\alpha_k \vec{p}_k)} = \frac{2\delta_k \left[E(\vec{w}_k) - E(\vec{w}_k + \alpha_k \vec{p}_k)\right]}{\mu_k^2} \qquad \textit{(Vz. 5.21)}$$

Zavedeme následující pravidlo, které upravuje λ_k podle míry přesnosti aproximace.

$$\begin{aligned} &\textit{Pokud} \quad \Delta_k > 0{,}75 \quad \textit{potom} \quad \lambda_k = \frac{1}{4}\lambda_k \\ &\textit{Pokud} \quad \Delta_k < 0{,}25 \quad \textit{potom} \quad \lambda_k = \lambda_k + \frac{\delta_k (1-\Delta_k)}{\|\vec{p}_k\|^2} \end{aligned} \qquad \textit{(Vz. 5.22)}$$

5.3 Teoretické srovnání s algoritmem zpětného šíření

Z popisu algoritmu zpětného šíření lze vyhodnotit, že pro každou iteraci je nutné určit skutečné výstupy pro daný trénovací vzor (příp. vzory), poté vypočíst hodnotu účelové funkce a zpětným procházením sítě určit hodnoty gradientů, pomocí nichž lze aktualizovat hodnoty vah a prahů. Tato operace se dá provést s výpočetní složitostí $O(N^2)$, kde N je počet vah a prahů, neboť účelová funkce může být určena jedním průchodem celé sítě směrem od vstupní vrstvy k výstupní a zpětným průchodem jsou postupně vypočteny hodnoty gradientů, pomocí nichž se zároveň aktualizují hodnoty vah a prahů.

V případě algoritmu SCG jedna iterace obsahuje jedno volání účelové funkce a dvě volání pro výpočet gradientů, což dává výslednou výpočetní složitost $O(3N^2)$. Při implementaci může být tato složitost snížena na $O(2N^2)$, neboť vypočtení účelové funkce je provedeno již při výpočtu gradientů, což nám ve výsledné podobě dává v průměru dvakrát větší výpočetní složitost, než v případě algoritmu zpětného šíření.

Z hlediska rychlosti konvergence je situace naprosto odlišná. Z testů, které provedli Johansson, Dowla a Goodman vyplývá, že algoritmus SCG konverguje při aplikaci na vybraných problémech řádově rychleji [5]. Testování implementovaných algoritmů tyto výsledky potvrzuje (7.7).

5.4 Algoritmus SCG

Algoritmus je převzat z [3]. Značení vychází z notace (5.1).

I. Zvol váhový vektor $\vec{w}_1$ a skaláry $0 < \sigma \leq 10^{-4}$, $0 < \lambda_1 \leq 10^{-6}$, $\hat{\lambda}_1 = 0$.

Nastav $p_1 = r_1 = -E'(\vec{w}_1)$, *k=1, uspech = true.*

II. Pokud *uspech = true* potom:

$$\begin{aligned} \sigma_k &= \frac{\sigma}{\|\vec{p}_k\|} \\ \vec{s}_k &= \frac{E'(\vec{w}_k + \sigma_k \vec{p}_k) - E'(\vec{w}_k)}{\sigma_k} \\ \delta_k &= \vec{p}_k^{\,T} \vec{s}_k \end{aligned}$$

III. Uprav δ_k:

$$\delta_k = \delta_k + (\lambda_k - \hat{\lambda}_k)\|\vec{p}_k\|^2$$

IV. Pokud $\delta_k \le 0$ potom vytvoř pozitivně definitní Hessovskou matici:

$$\begin{aligned} \hat{\lambda}_k &= 2\left(\lambda_k - \frac{\delta_k}{\|\vec{p}_k\|^2}\right) \\ \delta_k &= -\delta_k + \lambda_k \|\vec{p}_k\|^2 \\ \lambda_k &= \hat{\lambda}_k \end{aligned}$$

V. Urči velikost kroků:

$$\begin{aligned} \mu_k &= \vec{p}_k^{\,T} \vec{r}_k \\ \alpha_k &= \frac{\mu_k}{\delta_k} \end{aligned}$$

VI. Vypočítej porovnávací parametr:

$$\Delta_k = \frac{2\delta_k [E(\vec{w}_k) - E(\vec{w}_k + \alpha_k \vec{p}_k)]}{\mu_k^2}$$

VII. Pokud $\Delta_k \ge 0$ potom je možné snížit hodnotu účelové funkce:

$$\begin{aligned} \vec{w}_{k+1} &= \vec{w}_k + \alpha_k \vec{p}_k \\ \vec{r}_{k+1} &= -E'(\vec{w}_{k+1}) \\ \hat{\lambda}_k &= 0 \\ &uspech = true \end{aligned}$$

a) Pokud *k mod N = 0* potom restartuj algoritmus $p_{k+1} = r_{k+1}$,

jinak zvol nový směr adaptace:

$$\begin{aligned} \beta_k &= \frac{\|\vec{r}_{k+1}\|^2 - \vec{r}_{k+1}\vec{r}_k}{\mu_k} \\ \vec{p}_{k+1} &= \vec{r}_{k+1} + \beta_k \vec{p}_k \end{aligned}$$

b) Pokud $\Delta_k \geq 0.75$ potom redukuj řídící parametr $\lambda_k = \frac{1}{4}\lambda_k$.

jinak není možné snížit hodnotu účelové funkce: $\widehat{\lambda}_k = \lambda_k$, *uspech* = *false.*

VIII. Pokud $\Delta_k < 0.25$ zvyš řídící parametr: $\lambda_k = \lambda_k + \frac{\delta_k (1 - \Delta_k)}{\left\| \vec{p}_k \right\|^2}$.

IX. Pokud směr největšího poklesu $\vec{r}_k \neq 0$ potom nastav *k* = *k+1* a jdi na **II.**,

jinak skonči a vrať $\vec{w}_{k+1}$ jako požadované minimum.

6 Implementace

V současné době existuje mnoho nekomerčních aplikací a frameworků pro práci s neuronovými sítěmi, které dokáží simulovat práci většiny známých algoritmů pro adaptaci prahů a vah, nicméně v mnoha případech se jedná o vyvíjené betaverze sloužící pouze pro akademický výzkum. Můžeme zmínit např. systém *WEKA* vyvíjený novozélandskou univerzitou Waikato, aplikační framework *ENCOG*, vyvíjený pod vedením Jeffa Heatona, případně i zásuvný modul Neural Network Toolbox pro MATLAB. Tyto nástroje nebývají vždy plně přístupné veřejnosti, mohou být poměrně náročné z uživatelského hlediska, o hardwarových či softwarových požadavcích nemluvě. Z těchto důvodů tedy nemusí být nejvhodnějšími prostředky pro úvod do problematiky neuronových sítí.

6.1 Cíle implementace a volba programovacího jazyka

Cílem této implementace bylo vytvořit názorný výukový software pro usnadnění pochopení základní problematiky neuronových sítí společně s algoritmy zpětného síření a škálovaných konjugovaných gradientů, který bude sloužit jak pedagogickým účelům na akademické půdě, tak studentům samotným. Implementace tedy není původně určena pro akademický výzkum a tudíž vývoj nebyl omezen rychlostními a složitostními testy. Důraz byl kladen především na názornost a jednoduchost.

Kvůli přenositelnosti a jednoduchosti celé aplikace bylo nutné vytvořit platformě nezávislou a hardwarově nenáročnou aplikaci, která nebude určena pouze akademické elitě v oboru.

Při volbě programovacího jazyka bylo nutné vybírat z objektově orientovaných jazyků, aby bylo možné efektivně implementovat obecné rozhraní vrstevnatých neuronových sítě, což společně s podmínkou přenositelnosti omezilo výběr pouze na jazyk *Java*, který se i díky svým vestavěným grafickým funkcím a knihovnám ukázal být vhodným kandidátem. Při výběru byl zvažován i jazyk C++, příp. C#, který byl zavržen především kvůli nesnadné přenositelnosti a softwarovým požadavkům.

6.2 Externí knihovny

Pro implementaci grafického uživatelského rozhraní byla použita knihovna uživatelských prvků *Swing* na platformě *Java* vyvíjená firmou Sun Microsystems od roku 1997, která poskytuje aplikační rozhraní pro tvorbu a obsluhu klasického grafického uživatelského rozhraní. Jelikož *Swing* je vestavěnou součástí jazyka *Java*, aplikace tím neztrácí svou nezávislost na externím software.

Při implementaci rozhraní byla využita objektově orientovaná knihovna *JOONE (Java Object Oriented Neural Engine) [14]* vyvinutá týmem Joone, který je od roku 2007 neaktivní. Knihovna je zveřejněna pod licencí *GPL (General Public License)*, oddíl *Core Engine*, který byl využit při implementaci softwaru *NeDro*, dokonce pod licencí *LGPL(Lesser General Public License)*. Využitý zdrojový kód knihovny *JOONE* je kompilován společně s celým projektem *NeDro*, což umožňuje běh celé aplikace bez nutnosti instalace externích knihoven a jiných prvků. Původní zdrojové kódy a popis knihovny *Core Engine* společně s programátorskou dokumentací k této knihovně nalezneme [14].

6.3 Knihovna JOONE

Volba efektivní externí knihovny pro implementaci rozhraní neuronových sítí se ukázala jako stěžejní pro následný vývoj prostředí pro vizualizaci struktury vrstevnatých neuronových sítí. Pro tento účel bylo nezbytné zvolit nekomerční knihovny s veřejnou licencí, která bude později rozšířena o grafické prostředí. Výše zmíněné požadavky splňovaly pouze 3 knihovny, a sice JOONE, vyvíjená Joone týmem do roku 2007, dále ENCOG, vyvíjená pod vedením Jeffa Heatona, a konečně Neuroph, vyvíjená akademickým týmem univerzity v Bělehradě. Na základě uspokojivé rychlosti a dostupnosti zdrojového kódu byla vybrána knihovna JOONE, která se později ukázala jako velice nevhodná. Knihovna JOONE je v současné době zastaralá a velice obtížná na použití. Rozšíření o další algoritmy je nesnadné, efektivně je implementován pouze algoritmus zpětného šíření chyby, který ale v rychlostním srovnání s knihovnou ENCOG také selhává (7.8).

Mezi přednosti můžeme řadit licenci LGPL *(Lesser General Public License)*, která dovoluje použít a modifikovat kód v plném rozsahu pouze se zmínkou o užití.

6.4 Systémové požadavky

Aplikace je hardwarově i softwarově nenáročná, pro své korektní spuštění je potřeba mít na počítači nainstalovánu *JVM (Java Virtual Machine)[15]*. Verze *JVM* jsou dnes dostupné pro většinu nejrozšířenějších operačních systémů.

6.5 Využití

Mezi hlavní funkce aplikace patří vytváření instancí vrstevnatých neuronových sítí, jejichž váhy a prahy mohou být adaptovány pomocí algoritmů zpětného šíření a škálovaných konjugovaných gradientů.

Instanci neuronové sítě lze zobrazit v grafickém editoru, což nám umožní sledovat aktuální váhy a prahy neuronové sítě, poslední výstupy každého neuronu a typy přenosových funkcí. Pro každý trénovací vstupní vzor aplikace umožňuje zobrazit aktuální výstup a chybu každého neuronu sítě, která je vyjádřena zápornou parciální derivací *E* podle dané váhy - uvažujeme i nadále prahy neuronů jako váhy vycházející z fiktivního neuronu s konstantním výstupem *1*. Grafický editor tedy umožňuje sledovat důležité parametry neuronové sítě společně s jednoduchou velikostně modifikovatelnou vizualizací. Teno vizualizační prostředek lze považovat za stěžejní nástroj aplikace.

Další důležitým nástrojem aplikace je adaptace vah a prahů dle zvoleného algoritmu společně s odpovídajícím výstupem v podobě globálních chyb v průběhu učení, konečných vah a prahů neuronové sítě, případně grafu globální chyby v závislosti na iteraci. Výstup učení sítě může být směrován jak do panelu aplikace, tak do externího souboru.

Mezi další důležité nástroje aplikace řadíme rozpoznávání vstupního vzoru předloženého neuronové síti nebo textový editor pro rychlou úpravu trénovacích vzorů a dalších vstupních prvků.

6.6 Rozšiřitelnost implementace

Mezi důležité předpoklady efektivní implementace umělých neuronových sítí řadíme rozšiřitelnost implementace o další algoritmy, příp. metody. Pro snadnou modifikaci stávajících algoritmů je nutným předpokladem objektově orientované rozhraní neuronových sítí, které umožní v každém okamžiku přistupovat k vybraným parametrům neuronové sítě, případně je měnit. Dalším důležitým předpokladem je co nejnižší výpočetní rychlost, která zaručí její využitelnost i na rozsáhlých datech a problémech.

Rychlost a rozšiřitelnost jsou mnohdy navzájem kontraproduktivními cíli, neboť pro dosažení nízké výpočetní rychlosti je nutno efektivně využít všechny dostupné nástroje a prostředky daného hardwaru, především pak převést výpočet neuronové sítě do paralelního zpracování dílčích procesů. Pokud využijeme více či méně paralelního přístupu k výpočtu neuronové sítě, mnohdy se ochuzujeme o možnost modifikace a přístupu k datům v každém okamžiku probíhajícího výpočtu. Tento problém můžeme vyktnout mnoha stávajícícm systémům implementujícícm rozhraní neuronových sítí, které jsou sice výpočetně velice rychlé, avšak nedovolují uživateli získávat či modifikovat vybrané parametry v průběhu výpočtu. Díky knihovně JOONE se podařilo ve výsledné aplikaci NeDro zachovat jak uspokojivou rychlost, tak i částečnou rozšiřitelnost, přestože volba této knihovny se neukázala jako vhodná.

6.7 Vstupní data

Instance neuronových sítí společně s informacemi o jednotlivých neuronech ukládá i historii učení sítě - pokud nějaká byla, což společně s možnostmi ukládání a načítání instancí z externích souborů přípony *.snet* umožňuje vytvářet modelové prezentační příklady. Pro zapsání instance neuronové sítě do souboru je využito vestavěných metod programovacího jazyka *Java*, tudíž struktura souboru záleží na použité verzi jazyka *Java*.

Pro uchovávání trénovacích vstupních vzorů a jiných datových prvků slouží soubory s příponou *.dat*, tyto soubory mají přesnou strukturu, která je popsána níže.

6.8 Struktura trénovacích vzorů v souboru s příponou *.dat*

Datové prvky souboru jsou rozděleny maticově do řádků a sloupců, kde každý jednotlivý záznam je oddělen některým z povolených znaků. Povolené znaky jsou bílé znaky (mezera, tabulátor, nový řádek aj.) a středník. Jako záznam se považuje desetinné, příp. celé číslo. Dodržení této struktury je důležité, neboť aplikace načítá pouze zvolený počet řádků, které odpovídají počtu vstupních vzorů, v nich potom vstupní a výstupní složky trénovacího vzoru načítá podle zvoleného indexu sloupce.

6.9 Implementace algoritmu zpětného šíření

Při implementování algoritmu zpětného šíření bylo využito implementovaných tříd rozhraní *JOONE*. Algoritmus zpětného šíření je implementován ve dvou variantách, a sice tzv. *online* a *offline* varianta. *Online* varianta vykonává iteraci algoritmu vždy po určení skutečného výstupu každého jednotlivého vzoru z trénovací množiny, zatímco *offline* varianta spustí iteraci až po určení skutečných výstupů všech vzorů z trénovací množiny. Pro variantu online slouží třída *org.joone.engine.BasicLearner* a pro variantu offline třída *org.joone.engine.BatchLearner*. Obě tyto třídy jsou součástí implementovaného rozhraní *JOONE*.

6.10 Implementace algoritmu škálovaných konjugovaných gradientů

Knihovna *Core Engine* nabízí díky svému návrhu možnosti rozšíření v podobě nových algoritmů s využitím základního rozhraní knihovny. Při rozšíření knihovny o algoritmus SCG bylo rozšířeno základní rozhraní *org.joone.engine.AbstractLearner* v třídu *org.joone.engine.BatchSCGLearner* , která s využitím pomocných tříd *SCGUtils, SCGExtender a SCGGradientExtender* ve jmenném prostoru *org.joone.engine.extenders* a *org.joone.engine.SCGGradientLearner* implementují algoritmus SCG. Při vývoji bylo rozšířeno rozhraní *org.joone.engine.NeuralNetListener* o metodu *patternReaded()*, díky které se celá knihovna stává přístupnější pro implementaci online algoritmů. Podrobněji je tato problematika probrána v programátorské dokumentaci na přiloženém CD.

6.11 Implementace grafického uživatelského rozhraní

Jak již bylo zmíněno, pro implementaci grafického uživatelského rozhraní (dále jen GUI) bylo využito grafických prvků knihovny *Swing*, vyvíjené firmou Sun Microsystems. Aplikace byla rozdělena do hlavního a několika vedlejších oken tak, aby byl kladen důraz především na snadnost obsluhy aplikace a jednoduchou modifikaci vstupních dat. Hlavní okno obsahuje informativní výpis nejdůležitějších údajů o aktuálně načtené neuronové síti, obsahuje výstupové informace o procesu učení, případně rozpoznávání předloženého vzoru.

Hlavní okno reaguje na volby uživatele a za tímto účelem spouští vedlejší aplikační okna. Pro podrobnější popis a návod na obsluhu vedlejších oken poslouží uživatelská dokumentace přiložená na CD.

6.12 Dokumentace a uživatelská podpora

Zdrojové kódy aplikace *NeDro* splňují konvenci *javadoc*, což umožňuje generovat programátorskou dokumentaci přímo ze zdrojových kódů programu. Tato dokumentace ve formátu *HTML* je přiložena na CD společně s aplikací a obsahuje popisy nejdůležitějších programátorských struktur, které jsou využity v rámci projektu *NeDro*.

Uživatelská dokumentace popisuje obsluhu celé aplikace z uživatelského hlediska a vysvětluje propojení jednotlivých nástrojů v rámci aplikace *NeDro*. Je rovněž obsažena na přiloženém CD ve formátu *.pdf* .

Pro potřeby rozšíření softwaru mezi veřejnost byl zapůjčen prostor na adrese http://server.drobas.info/michal/NeDro/ kde lze volně stáhnout tento software včetně uživatelské a programátorské dokumentace a zdrojových kódů pod licencí LGPL*(Lesser General Public License)*. Tímto děkuji panu inženýru Martinu Drobnému za poskytnutí potřebného prostoru na serveru.

7 Testování implementace

Následující kapitola se věnuje testování implementace z hlediska korektnosti a rychlosti.

7.1 Srovnávací implementace

Bylo nutné vybrat z dostupných jiných implementací takovou, která při testovém srovnání s implemetací NeDro ověří její správnost na obou implementovaných algoritmech. Při vybírání srovnávací implemetace bylo přihlédnuto k faktu, že volba musí padnout na nekomerční, veřejně dostupnou implementaci, tudíž nebylo možné uvažovat výpočetně velmi rychlou a akademicky známou variantu MATLAB se zásuvným modulem Neural Network Toolbox, ani jiné komerční systémy, které se vzhledem k ceně staly nežádoucími. Dalším důležitým požadavkem pro výběr implementace byla dostupnost zdrojového kódu, což se ukázalo jako nezbytné pro synchronní spuštění výpočtu obou srovnávaných implementací.

Mezi vhodné varianty byly zařazeny pouze implementace Neuroph a ENCOG, obě implementované na platformě Java. Po důkladném zvážení bylo rozhodnuto ve prospěch implementace ENCOG, neboť je výpočetně řádově rychlejší a zdrojově dostupná pod licencí LGPL (*Lesser General Public License*).

7.2 Parametry testovacího počítače

Testy byly prováděny na notebooku značky Fujitsu-Siemens, model Lifebook S6410. Parametry hardware jsou shrnuty v následující tabulce.

Výrobce	FUJITSU SIEMENS
Model	S6410
Procesor	Intel Core™2 Duo Processor T8300 (2,40GHz, 3MB L2 cache, 800MHz)
Chipset	Intel GM965 Express Chipset
Systémová paměť	2,00 GB RAM
Architektura systému	32-bit
Podpora multi-threading	Ano
Počet jader	2

Tabulka 7.1 – Hardwarové parametry testovacího stroje

7.3 Testovací problémy

Algoritmy byly testovány na 2 sadách dat (8), obě sady jsou součástí přiloženého CD.

První sadou je *IRIS*. Jedná se o botanickou sadu dat kosatců. Původní sada dat je zveřejněna [16], obsahuje 150 vzorů, každý z nich obsahuje 4 vstupní a 1 výstupní parametr. Vstupní parametry odpovídají postupně délce a sířce kališních a korunních lístků dané květiny v centimetrech, výstupní parametr určuje druh kosatce.

Pro testování této sady dat byla zkonstruována 3-vrstvá neuronová síť se 4 vstupními, 6 skrytými a 3 výstupními neurony. Parametr učení pro tuto sadu byl nastaven na 0,01 a moment učení na 0,8.

Výstupní parametr původní sady byl pro účely testování vhodně vyjádřen 3 binárními parametry, které odpovídaly 3 výstupním neuronůn vytvořené neuronové sítě. Pro hodnotu *Iris Setosa* byly výstupní parametry určeny jako *1*, *0* a *0*, hodnota *Iris Versicolour* odpovídala *0*, *1* a *0* a konečně hodnota *Iris Virginica 0*, *0* a *1*.

Při procesu učení byla využita kompletní škála 150 vzorů, velikost trénovací množiny byla stanovena na 80% dostupných vzorů, tedy 120 vzorů, validační množina obsahovala zbylých 30 vzorů.

Druhou testovací sadou je *4-bit Binary Addition*, tedy sada pro 4-bitové binární sčítání. Jedná se o generovanou sadu vzorů s 12 binárními parametry, které vhodných způsobem reprezentují operaci sčítání pro 4-bitová binární čísla. První 4 binární parametry odpovídají postupně 4 bitům prvního operandu pro operaci binárního sčítání a další 4 parametry odpovídají stejným způsobem druhému operandu. Poslední 4 parametry jsou obdobným způsobem vyhrazeny pro výsledné 4-bitové binární číslo. Sada obsahuje kompletní škálu 128 vzorů, tedy všechny možné kombinace vstupních operandů pro operaci binárního sčítání.

Pro testování této sady dat byla zkonstruována 3-vrstvá neuronová síť se 8 vstupními, 8 skrytými a 4 výstupními neurony. Parametr učení pro tuto sadu byl nastaven taktéž na 0,01 a moment učení na 0,8.

Byla využita kompletní škála 128 vzorů, velikost trénovací množiny byla stanovena na 87,5% dostupných vzorů, tedy 112 vzorů, validační množina obsahovala zbylých 16 vzorů.

7.4 Testovací kritéria

U obou implementací byl testován jak algoritmus zpětného šíření, tak i algoritmus škálovaných gradientů pro všechny výše zmíněné sady dat. Pro každou sadu bylo zkonstruováno 30 testovaných neuronových sítí s náhodně generovanými vahami a prahy v intervalu (-0,2 ; 0,2), testovací a validační množiny byly pro každou testovanou neuronovou síť náhodně vygenerovány tak, aby měly dimenze příslušné zvolené sadě dat. U takto vytvořených neuronových sítí byl spuštěn proces učení pro danou implementaci a algoritmus. Dle pravidel strategie učení (3.4), proces učení byl ukončen ve chvíli, kdy křivka účelové funkce na validační množině dosáhla svého globálního minima. Pokud byl průběh křivky účelové funkce rostoucí, proces učení byl ukončen pro počet iterací 5000 u datové sady IRIS, resp. 15000 u datové sady 4-bit Binary Addition. U prováděných testů byl zaznamenán konečný počet iterací algoritmu a globální chyba na testovací i validační množině.

Implementace byly srovnávány i z rychlostního hlediska, a sice na datové sadě IRIS. Pro tento účel bylo obdobně zkonstruováno 50 testovaných neuronových sítí. U vytvořených neuronových sítí byl spuštěn proces učení pro danou implementaci a algoritmus tak, aby byl ukončen při dosažení počtu iterací 10000. Zaznamenán byl čas, po který byla neuronová síť v procesu učení.

7.5 Přesnost výpočtu

Jelikož pro každou testovanou neuronovou síť byly spuštěny 2 testy shodného algoritmu na odlišných implementacích, výsledné globální chyby jak na testovací, tak na validační množině dat se až na drobné odchylky shodují. Tyto odchylky mohou být vysvětleny nepřesností v důsledku zaokrouhlování. Obě implementace totiž používají datový atribut *double* pro uchování instance desetinného čísla, který je v programovacím jazyce *Java* omezen na 20 platných číslic. Použití stejných matematických operací v odlišném pořadí může způsobit, že výsledné parametry neuronové sítě při průchodu iterací nemusí být shodné. Pro vysoký počet matematických operací se může stát tato nepřesnost ve výsledku zřetelná.

7.6 Modifikace výpočtu implementace NeDro

Aby byly dosaženy stejné výsledky při testovém srovnání obou implementací, je nutným předpokladem, aby byly algoritmy implementovány shodně. Při výpočtech bylo zjištěno, že knihovna ENCOG při výpočtu gradientů neuronové sítě používá normalizaci. Vypočtené gradienty neuronové sítě násobí normalizačním faktorem

$$-\frac{2|y_i|}{|w|} \qquad \textit{(Vz. 7.1)}$$

kde $|y_i|$ je počet výstupních neuronů a $|w|$ je dimenze váhového vektoru (5.1).

Pro korektní srovnání bylo nutno použít tuto normalizaci i v implementaci NeDro. Rovněž bylo nutno modifikovat výpočet globální chyby (Vz. 3.1), neboť knihovna ENCOG používá chybu typu MSE *(Mean Squared Error)* dělenou počtem výstupních neuronů

$$E = \frac{1}{|y_i||P|}\sum_d\sum_j (y_{d,j} - n_{d,j})^2 \qquad \textit{(Vz. 7.2)}$$

kde *d* indexuje předkládané vzory z trénovací množiny *P*, *j* indexuje výstupní neurony, *y* představuje skutečný výstup příslušného neuronu a *n* jeho požadovaný výstup.

Pro statistické výstupy a srovnání byla použita zadefinovaná chyba (Vz. 3.1).

7.7 Test korektnosti

V následující tabulce jsou shrnuty průměrné výsledky z 30 výše popsaných testů (7.4), které ověřují korektnost výpočtu implementace NeDro ve srovnání s implementací ENCOG. Kompletní výsledky je možné nalézt v příloze (9). Tabulka obsahuje průměrný počet iterací pro daný algoritmus implementace NeDro.

	Počet iterací	
	BP	**SCG**
IRIS	2099,37	124,60
BA4	7627,17	349,50

Tabulka 7.2 – Průměrný počet iterací z 30 výše definovaných testů (7.4) pro algoritmy zpětného šíření s momentem (BP) a algoritmus konjugovaných škálovaných gradientů (SCG) implementace NeDro.

V další tabulce jsou zveřejněny odchylky počtu iterací a výsledné hodnoty účelové funkce na trénovací i validační množině pro implementace NeDro a ENCOG. Pro vyjádření odchylky hodnoty účelové funkce používáme součtu absolutních hodnot rozdílu výsledné hodnoty účelové funkce na trénovací a validační množině u obou implementací.

Průměrné odchylky	
Počet iterací	**Hodnota účelové funkce**
0,35	1,0138E-05
0,67	2,7705E-04

Tabulka 7.3 Průměrné odchylky počtu iterací a hodnoty účelové funkce na trénovací i validační množině pro implementace NeDro a ENCOG.

7.8 Rychlostní srovnání

V následující tabulce srovnáme obě implementace z rychlostního hlediska. Pro tento účel bylo zkonstruováno 50 výše popsaných testů (7.4), na kterých byla měřena délka výpočtu obou algoritmů pro 10000 iterací. Kompletní výsledky lze nalézt v příloze (9.3).

	ENCOG	***NeDro***	***Zrychlení***
BP	3309,92	35649,56	10,77
SCG	6179,30	126127,80	20,41

Tabulka 7.4 Rychlostní srovnání implementace NeDro a ENCOG na datové sadě IRIS pro 10000 iterací. BP značí algoritmus zpětného šíření s momentem, SCG algoritmus škálovaných konjugovaných gradientů.

Můžeme si všimnout, že implementace NeDro je na všech testovacích sadách pro oba algoritmy pomalejší než ENCOG. Tento fakt je způsoben především tím, že již vypočtení gradientů sítě v implementaci JOONE, kterou NeDro rozšiřuje, je řádově pomalejší než v konkurenčním ENCOG, což se následně projeví i v implementaci algoritmu konjugovaných škálovaných gradientů, kde je v jedné iteraci použit výpočet gradientů dokonce dvakrát. Tento rychlostní rozdíl je způsoben zejména návrhem rozhraní vrstevnatých neuronových sítí, jednak také zastaralostí knihovny *JOONE* - vyvíjena do roku 2007. Problematika výběru

knihovny JOONE byla probrána (6.3), výkonnostní srovnání knihoven JOONE a ENCOG je popsáno v článku [17].

7.9 Vyhodnocení testů

S přihlédnutím ke zveřejněným výsledkům testů je vhodné připomenout důležitý implementační cíl, software NeDro nebyl vyvíjen za účelem rychlostních testů. Implementace vznikla za účelem snadnějšího studia umělých neuronových sítí a implementovaných algoritmů, což podporuje schopností vizualizace struktury vrstevnatých neuronových sítí. Mezi dalšími argumenty můžeme zmínit i fakt, že srovnávací implementaci ENCOG řadíme mezi nejrychlejší volně dostupné implementace rozhraní neuronových sítí [17].

8 Příloha – datové sady

Následující datové sady byly použity při testování implementace.

8.1 Datová sada IRIS

Název: Databáze kosatců

Zdroj: Newman, D.J. & Hettich, S. & Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine,CA: University of California, Department of Information and Computer Science.

Základní popis:

150 vzorů, 3 výstupní třídy, každá obsahující 50 vzorů; lineárně neseparabilní. Každý vzor obsahuje 4 popisné číselné atributy.

8.2 Datová sada 4-bit Binary Addition

Název: 4-bitové binární sčítání

Zdroj: Generovaná sada.

Základní popis:

128 vzorů. Každý vzor obsahuje 12 binárních atributů, které vhodně charakterizují operaci binárního sčítání pro 4-bitová binární čísla. První 4 binární parametry odpovídají postupně 4 bitům prvního operandu pro operaci binárního sčítání a další 4 parametry odpovídají stejným způsobem druhému operandu. Poslední 4 parametry jsou obdobným způsobem vyhrazeny pro výsledné 4-bitové binární číslo. Sada obsahuje kompletní škálu všech kombinací vstupních operandů.

9 Příloha – výsledky testování

V této kapitole jsou uvedeny kompletní výsledky testování implementovaných algoritmů. Výsledky slouží jako příloha k (7). Tyto data jsou také zveřejněna na přiloženém na CD.

9.1 Testování algoritmů na datové sadě IRIS

Následující tabulka obsahuje výsledky obou implementovaných algoritmů na datové sadě IRIS pro implementaci NeDro. Celkem bylo provedeno 30 testů, výsledky jsou určeny jako příloha k pojednání (7.7). Tabulka pro každý jednotlivý test ukazuje výslednou hodnotu účelové funkce pro trénovací i validační množinu.

Algoritmus zpětného šíření s momentem			***Algoritmus škálovaných konjugovaných gradientů***		
Počet iterací	***Hodnota účelové funkce na trénovací množině***	***Hodnota účelové funkce na validační množině***	***Počet iterací***	***Hodnota účelové funkce na trénovací množině***	***Hodnota účelové funkce na validační množině***
2807	0,007885787	0,017532908	168	0,023595865	0,041780973
136	0,011037857	0,049176405	32	0,013143808	0,113786129
324	0,010629743	0,019962931	36	0,047507067	0,060184741
318	0,008708545	0,029566880	49	0,023620383	0,081963982
1697	0,009981856	0,008501406	75	0,042855510	0,023910274
349	0,012985955	0,008097191	32	0,036868879	0,054780788
791	0,009579840	0,015054046	32	0,009488140	0,160714207
147	0,009720108	0,049829170	28	0,054079295	0,056234297
363	0,012185515	0,012169644	104	0,026756026	0,035156072
1503	0,010574190	0,002808597	153	0,023502851	0,003761906
4211	0,010592780	0,001225574	238	0,002841589	0,068498311
795	0,005864630	0,028046660	104	0,031764631	0,032225750
5000	0,009116268	0,003308079	181	0,015375056	0,065736437
533	0,010822579	0,011198417	73	0,026362029	0,044806425
184	0,012924490	0,031175471	65	0,029074278	0,045361957
240	0,012862295	0,018879623	28	0,044726449	0,064262014
737	0,009524111	0,014981716	287	0,016666667	0,000000004
5000	0,009578070	0,000267928	57	0,039226684	0,023276028
5000	0,007459500	0,000152494	67	0,027537285	0,036332498
715	0,008122591	0,027902887	406	0,016634940	0,000038485
1573	0,010397487	0,003153956	51	0,013815230	0,103264869
5000	0,009662921	0,003795436	36	0,050156639	0,078066237
5000	0,008965620	0,002960640	493	0,017510310	0,000009524
805	0,009702940	0,014203760	40	0,036754406	0,083417262
5000	0,008107343	0,002738339	64	0,017629736	0,087417194
881	0,004231359	0,036099476	37	0,045481950	0,091618611
5000	0,007846985	0,003243323	86	0,023014812	0,032672677
2793	0,009768806	0,000336368	279	0,017025205	0,003184960

1079	0,009508381	0,013415870	39	0,042731426	0,039073404
5000	0,006858537	0,000037403	398	0,018760682	0,000012231

Tabulka 9.1 – Průměrné výsledky implementace NeDro pro oba implementované algoritmy na datové sadě IRIS.

9.2 Testování algoritmů na datové sadě 4-bit Binary Addition

Následující tabulka obsahuje výsledky obou implementovaných algoritmů na datové sadě 4-bit Binary Addition pro implementaci NeDro. Celkem bylo provedeno 30 testů, výsledky jsou určeny jako příloha k pojednání (7.7). Tabulka pro každý jednotlivý test ukazuje výslednou výslednou hodnotu účelové funkce pro trénovací i validační množinu.

Algoritmus zpětného šíření s momentem			***Algoritmus škálovaných konjugovaných gradientů***		
Počet iterací	***Hodnota účelové funkce na trénovací množině***	***Hodnota účelové funkce na validační množině***	***Počet iterací***	***Hodnota účelové funkce na trénovací množině***	***Hodnota účelové funkce na validační množině***
5463	0,041440316	0,287143817	605	0,134741227	0,450782678
10539	0,039134679	0,003876758	184	0,111346250	0,582456426
6667	0,158514510	0,372250824	168	0,240955509	0,409088394
5316	0,184161570	0,258125348	292	0,150568282	0,445701223
11454	0,060238012	0,336923928	490	0,109828228	0,341892879
5147	0,287810731	0,551098138	277	0,130762129	0,147069218
7353	0,092313102	0,345847013	138	0,183650768	0,480023224
11639	0,091894988	0,258713201	180	0,298645435	0,852430124
3316	0,157971975	0,208165656	182	0,132002037	0,376576451
9890	0,054871688	0,021735500	741	0,114496481	0,472594975
5976	0,171318736	0,526008951	107	0,148454131	0,275437866
15000	0,039976150	0,208421450	792	0,098773801	0,467309424
13432	0,120466371	0,141810031	529	0,057901273	0,447623925
7202	0,155846025	0,592446825	411	0,108900821	0,409689856
7638	0,037552376	0,113112389	62	0,538485612	0,773560332
3098	0,075371150	0,106168066	657	0,142364789	0,359405379
2690	0,262154185	0,452700570	494	0,160718450	0,226818011
2052	0,139956335	0,548659273	306	0,100147452	0,327040922
15000	0,021918717	0,383673615	69	0,499663355	0,736421806
15000	0,040039123	0,035182931	194	0,279648561	0,633972445
2144	0,328009614	0,410965428	405	0,072306380	0,377260865
15000	0,029012122	0,136683029	771	0,145976999	0,374708057
6839	0,026402077	0,028205783	30	0,712167034	0,804807904
2623	0,216042717	0,581702727	557	0,267172526	0,620726020
15000	0,073537192	0,037845830	244	0,058950813	0,369975258
4851	0,056863032	0,071387701	400	0,249608968	0,297766504
4269	0,155040924	0,328141403	239	0,229764481	0,486344953
3894	0,185992259	0,567257137	206	0,282457757	0,545176030

8648	0,144969681	0,188289254	343	0,101342762	0,402856163
1670	0,245346958	0,382034325	412	0,206729313	0,797570309

Tabulka 9.2 – Průměrné výsledky implementace NeDro pro oba implementované algoritmy na datové sadě 4-bit Binary Addition.

9.3 Rychlostní testování na datové sadě IRIS

Následující tabulka obsahuje rychlostní výsledky obou implementovaných algoritmů na datové sadě IRIS. Zveřejněny jsou výsledky jak pro implementaci NeDro, tak i ENCOG. Celkem bylo provedeno 50 testů, výsledky jsou určeny jako příloha k pojednání (7.8). Tabulka pro každý algoritmus a implementaci ukazuje čas, po který byla neuronová síť v procesu učení. Zveřejněn je rovněž poměr zrychlení implementace ENCOG ku implementaci NeDro u obou implementovaných algoritmů.

Algoritmus zpětného šíření s momentem			***Algoritmus škálovaných kojugovaých gradientů***		
ENCOG	***NeDro***	***Zrychlení***	***ENCOG***	***NeDro***	***Zrychlení***
2839	28969	10,20	5570	116704	20,95
3447	31185	9,05	6615	116174	17,56
3494	33946	9,72	5570	124519	22,36
3308	36286	10,97	6957	124971	17,96
3042	35069	11,53	5538	127842	23,08
3790	36489	9,63	6490	126532	19,50
3027	35536	11,74	5600	125424	22,40
3759	36145	9,62	6271	127125	20,27
3026	36380	12,02	5928	124738	21,04
3759	35834	9,53	6381	125690	19,70
3026	36410	12,03	6802	128263	18,86
3026	35537	11,74	5585	129434	23,18
3026	37955	12,54	6381	129325	20,27
3556	35241	9,91	5523	128279	23,23
3339	36363	10,89	6942	126517	18,22
2995	35287	11,78	5554	127187	22,90
3182	36130	11,35	6974	123849	17,76
3510	35412	10,09	5616	125456	22,34
3151	36223	11,50	6927	125456	18,11
3728	35084	9,41	5554	124566	22,43
2995	36130	12,06	6818	125845	18,46
3073	35131	11,43	5460	128919	23,61
3511	36410	10,37	6988	125486	17,96
2933	35553	12,12	5351	126532	23,65
3729	36208	9,71	6833	125658	18,39
3042	35662	11,72	6084	128326	21,09
3448	35912	10,42	5553	130276	23,46
3588	36645	10,21	6910	126517	18,31
2995	35225	11,76	5569	126704	22,75

3635	36442	10,03	6817	124815	18,31
3120	35443	11,36	5445	124629	22,89
3463	36410	10,51	6911	125705	18,19
2995	35443	11,83	5928	129309	21,81
3729	35912	9,63	5943	127514	21,46
3026	36582	12,09	6646	126111	18,98
3682	35256	9,58	5569	126656	22,74
3026	36473	12,05	6412	123583	19,27
3759	35412	9,42	5991	125066	20,88
2995	36208	12,09	6599	126766	19,21
3260	35053	10,75	5585	129028	23,10
3651	36426	9,98	6942	126734	18,26
2948	35552	12,06	5897	128154	21,73
3728	35958	9,65	5928	127312	21,48
3073	36519	11,88	6194	124753	20,14
3760	35646	9,48	5912	126423	21,38
3011	36270	12,05	6895	127436	18,48
3213	35007	10,90	5554	127967	23,04
3323	36395	10,95	7004	125362	17,90
3432	35397	10,31	5554	124847	22,48
3323	36317	10,93	6895	125908	18,26

Tabulka 9.3 – Rychlostní srovnání implementace NeDro a ENCOG pro oba implementované algoritmy na datové sadě IRIS pro 10000 iterací.

10 Použité zdroje

10.1 Literatura

[1] ROJAS, R.: *Neural Networks: A Systematic Introduction*. Springer-Verlag, 1996.

[2] BISHOP, C.: *Neural Networks for Pattern Recognition*. Oxford University Press, 1996.

[3] MOELLER, M.: *A scaled conjugate gradient algorithm for fast supervised learning*. *Neural Networks*, ročník 6, 1993: s. 525–533.

[4] HESTENES, M.: *Conjugate Direction Methods in Optimalization*. Springer-Verlag, 1980.

[5] JOHANSSON, E.M., DOWLA, F.U., GOODMAN, D.M.: *Backpropagation Learning for Multilayer Feed-Forward Neural Networks Using the Conjugate Gradient Method*. Int. J. Neural Syst.,1991: s. 291-301.

[6] MCCULLOCH, W., WALTER P.: *A Logical Calculus of Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics, ročník 5**,** 1943: s. 115–133.

[7] ROSENBLATT F.: *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.* American Psychological Association, svazek 65/6, 1958: s.386-408.

[8] BOUČKA J.: *Neuronové sítě pro predikci časových řad*. Diplomová práce ČVUT FEL, 2008.

[9] WEIGEND, A.S., RUMELHART, D.E., HUBERMAN, B.A.. *Generalization by weight elimination with application to forecasting*. Advances in Neural Information Processing Systems 3, 1991: s.875-882.

[10] GNECCO, G., SANGUINETI, M.. *Regularization Techniques and Suboptimal Solutions to Optimization Problems in Learning from Data.* Neural Computation, č. 22(3), 2010: s.793-829.

[11] DU, K.-L.. *Clustering: A neural network approach*. Neural Networks, č. 23(1), 2010: s.89-107.

[12] BRYSON, A. E., HO, Y.-Ch.. (1969). *Applied optimal control: optimization, estimation, and control*. Blaisdell Publishing Company, 1969: s.481.

[13] TOLLENAERE, T.. *SuperSAB: Fast adaptive back propagation with good scaling properties.* Neural Networks, č. 3(5), 1990: s.561-573.

10.2 Elektronické zdroje

[14] Framework JOONE, http://www.jooneworld.com

[15] Java Virtual Machine, http://www.java.com/en/download/index.jsp

[16] IRIS dataset, http://archive.ics.uci.edu/ml/datasets/Iris

[17] Článek o rychlostním porovnání systémů JOONE, ENCOG a Neuroph, http://www.codeproject.com/KB/recipes/benchmark-neuroph-encog.aspx

Závěr

Klasické vrstevnaté neuronové sítě typu zpětného šíření jsou schopny se velmi dobře naučit i dosti náročné úlohy, proto je stále řadíme mezi nejpoužívanější modely neuronových sítí. Právě díky této vlastnosti se těší velikému zájmu z řad odborné veřejnosti, což má za následek vznik mnoha různých knihoven a systémů, které implementují rozhraní neuronových sítí spolu s výběrem algoritmů pro adaptaci vah a prahů. Systémy, které se věnují problematice neuronových sítí se však často chovají jako černé skříňky, které jsou optimalizovány na výkon. Přestože je v současné době efektivita více než žádoucí, tento přístup neprospívá samotnému studiu neuronových sítí. Z tohoto důvodu vznikla v rámci této práce uživatelsky nenáročná aplikace, která usnadňuje studium základní problematiky neuronových sítí a popsaných algoritmů. Při návrhu aplikace byl kladen důraz především na jednoduchost a názornost, což bylo podpořeno nástrojem pro vizualizaci struktury vrstevnatých neuronových sítí.

Pro vývoj aplikace bylo vhodné vybrat externí knihovnu, která efektivně implementuje rozhraní neuronových sítí, neboť vlastní vývoj by byl časově náročný. Volba efektivní externí knihovny pro implementaci rozhraní neuronových sítí se ukázala jako stěžejní pro následnou implementaci vybraných algoritmů a vývoj prostředí pro vizualizaci struktury vrstevnatých neuronových sítí. Pro tento účel bylo nezbytné uvažovat pouze nekomerční knihovny s veřejnou licencí, které mohou být doplněny o vybrané algoritmy a grafické prostředí. Výše zmíněné požadavky splňovaly pouze 3 knihovny, a sice JOONE, vyvíjená Joone týmem do roku 2007, dále ENCOG, vyvíjená pod vedením Jeffa Heatona, a konečně Neuroph, vyvíjená akademickým týmem univerzity v Bělehradě. Na základě uspokojivé rychlosti a dostupnosti zdrojového kódu byla vybrána knihovna JOONE, která se později ukázala jako poměrně nevhodná pro další rozšíření a rychlostní porovnání s konkurečními knihovnami odhalilo neefektivitu návrhu [16].

Kvůli implementaci grafického uživatelského prostředí bylo nutno rozšířit knihovnu zhruba o třetinový podíl původní velikosti zdrojového kódu, přičemž implementace vybraných algoritmů si vynutila rozšíření obecného rozhraní o metody a třídy, díky kterým se celá knihovna stala přístupnější pro implementaci online algoritmů a algoritmů. Rychlostní porovnání s konkurenční implementací ENCOG prováděné v rámci testování ukázalo, že rozhraní JOONE je při výpočtu gradientů neuronové sítě řádově pomalejší, což ovlivnilo výslednou rychlost obou implementovaných algoritmů.

Vzhledem k zmíněným nedostatkům knihovny JOONE, pro případné rozšíření aplikace o další algoritmy a grafické nástroje pro vizualizaci je vhodné volit jinou knihovnu, případně výrazněji upravit stávající rozhraní, které je rychlostně nevyhovující.