Neuronové sítě za použití evolučních algoritmů pro učící proces

Neural Networks Trained by Evolutionary Algorithms

Bc. Lukáš Sedlák

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2009/2010

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Lukáš SEDLÁK**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Neuronové sítě za použití evolučních algoritmů pro učící proces**

Zásady pro vypracování:

Cílem diplomové práce je zkompletovat modul pro učení neuronových sítí evolučními algoritmy a srovnat jejich výkonnost, kvalitu a další parametry s klasickými metodami učení.

1. **Seznámení se s neuronovými sítěmi, hlavně s Perceptronem a Sítěmi s dopředným šířením (Feedforward net).**
2. **Seznámení se s evolučními algoritmy.**
3. **Naprogramování modulu neuronových sítí s učícím algoritmem pomocí evolučních algoritmů v prostředí Mathematica.**
4. **Srovnání s klasickými metodami učení.**
5. **Závěr a vyhodnocení použité metody.**

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **ZELINKA, I., OPLATKOVÁ, Z., OŠMERA, P., ŠEDA, M., VČELAŘ, F. Evoluční výpočetní techniky – principy a aplikace. BEN – technická literatura, Praha, 2008, ISBN 80-7300-218-3.**
2. **ZELINKA, I. Umělá inteligence I. VUT Brno, ISBN 80-214-1163-5, 1998.**
3. **BOSE, N.K., LIANG, P. Neural Network Fundamentals with Graphs, Algorithms, and Applications, McGraw-Hill Series in Electrical and Computer Engineering. ISBN 0-07-006618-3, 1996.**
4. **FREEMAN, J. A., Simulating Neural Networks with Mathematica. Adison-Weslez Publishing Company, 1994, ISBN 0-201-56629-X.**
5. **OPLATKOVÁ, Z.: Metaevolution – Synthesis of Optimization Algorithms by means of Symbolic Regression and Evolutionary Algorithms, Lambert-Publishing, 2009, ISBN 978-8383-1808-0.**
6. **DAVIS L.: Handbook of Genetic Algorithms, International Thomson Computer Press, 1996, ISBN 1850328250.**
7. **MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J.: Umělá inteligence, Academia, 1993, ISBN 80-200-0496-3.**
8. **MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J.: Umělá inteligence 4., Academia, 2003, ISBN 80-200-1044-0.**

Vedoucí diplomové práce: **Ing. Zuzana Oplatková, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **19. února 2010**

Termín odevzdání diplomové práce: **8. června 2010**

Ve Zlíně dne 19. února 2010

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Tato práce se zabývá otázkou srovnání klasických metod učení neuronových sítí a metod využívajících evolučních algoritmů, jako například SOMA nebo diferenciální evoluce. Hlavním úkolem této práce bylo vytvořit funkční modul v prostředí Mathematica pro učení neuronových sítí s využitím evolučních algoritmů. Práce obsahuje jak teoretický základ pro danou problematiku, tak analýzu výsledků srovnání obou metod učení neuronových sítí.

Klíčová slova: SOMA, Diferenciální evoluce, neuronová síť, Perceptron, neuronová síť s dopředným šířením, Back propagation, Mathematica.

ABSTRACT

This paper deals with the question of comparison of classical methods used for training process of neural networks and methods using evolutionary algorithms such as SOMA or differential evolution. The main task of this work was to create a functional module in the Mathematica environment for learning neural networks using evolutionary algorithms. The work contains both the theoretical basis for the issue and analyze the results of comparison between the two methods of learning in neural networks.

Keywords: SOMA, Differential evolution, neural network, Perceptron, Feed forward neural network, Back propagation, Mathematica.

Poděkování

Děkuji vedoucímu diplomové práce Ing. Zuzaně Oplatkové, Ph. D. za pomoc a odborné vedení při tvorbě této práce.

Také děkuji své rodině, která mi po celou dobu studia poskytovala podporu a skvělé podmínky pro mé studium.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD

Neuronové sítě rozdělují vědeckou společnost na dva nesmiřitelné tábory. Jejich zastánci v nich spatřují možný směr vývoje optimalizačních algoritmů, které člověku pomohou posunout hranice lidského poznání. Na druhé straně jejich odpůrci se staví proti nim, když argumentují, že se jedná pouze o „hračky" vědců, které jsou schopné řešit jen omezenou množinu problémů. Čas ovšem ukazuje, že neuronové sítě se uplatňují ve stále větším měřítku. Hrají jednu z předních rolí na poli optimalizace, aproximace, filtrace anebo úloh zabývajících se predikcí. Spolu s vývojem neuronových sítí vznikaly algoritmy, pomocí kterých se neuronové sítě učily. Jako zástupce takových algoritmů je možné uvést jeden z nejpoužívanějších v dnešní době, a to algoritmus Back propagation pro sítě perceptronovského typu, nebo K – means pro sítě RBF. Tyto algoritmy vlastně hledaly optimální nastavení vah v neuronových sítích. Proto většinou obsahovaly nějaké konkrétní optimalizační techniky, např. gradientní metody optimalizace.

Mezi léty 1970 – 80 se podařilo vyvinout nový druh algoritmů. Jednalo se o evoluční algoritmy. Zmíněné algoritmy přinesly celou řadu nových možností. Díky nim bylo možné pracovat se všemi typy proměnných (reálná čísla, celá čísla, binární čísla). To pro stávající optimalizační algoritmy představovalo poměrně velký problém. Jelikož evoluční algoritmy jsou jako stvořené pro optimalizační úlohy, začalo se testovat jejich využití jak pro návrh optimální struktury, tak pro optimální nastavení vah neuronových sítí. Základem celého procesu optimalizace u těchto algoritmů je pouze důkladná znalost problematiky a schopnost sestavit vhodně účelovou funkci.

V této práci jsem se zaměřil na porovnání výkonnosti klasických způsobů učení neuronových sítí a učení pomocí evolučních algoritmů, konkrétně algoritmů SOMA (Samo Organizující – se Migrační Algoritmus) a diferenciální evoluce. Oba způsoby učení, jak klasické metody, tak pomocí evolučních algoritmů, jsem testoval na souborech dat, které právě pro tyto účely dala k dispozici Fakulta Informatiky University v Karlsruhe

v Německu.

I. TEORETICKÁ ČÁST

1 NEURONOVÉ SÍTĚ

Umělé informační systémy, v tomto případě neuronové sítě, jsou systémy, které napodobují činnost funkce nervových soustav a mozků živých organismů. Dělají to daleko dokonalejším způsobem, než v dnešní době běžně používaná výpočetní technika. S neuronovými sítěmi se začalo pracovat zhruba ve dvacátých letech 20. století. V té době se objevily první práce o modelech neuronu. Po přibližně dvaceti letech následovalo publikování prvního modelu umělého neuronu. Šlo o McCulloch – Pittsův perceptron [11]. Tento model je i přes svoji jednoduchost stále využíván i v dnešní době. Na základě právě tohoto modelu neuronu B. Widrow a jiní vytvořili první umělé neuronové sítě, které řešily praktické úlohy.

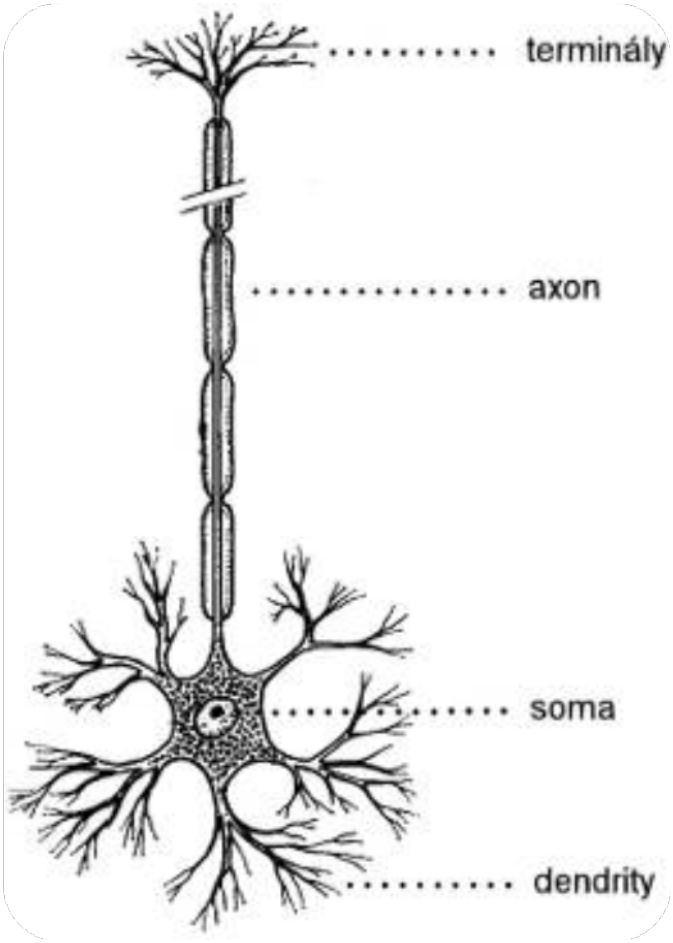
Neuronové sítě v porovnání s konvenčními počítači nepracují na základě pevně daného popisu jak problém řešit – algoritmu, ale využívají tzv. proces učení, ve kterém provádějí velké množství dílčích kroků současně a adaptují se na co nejlepší řešení daného problému. Díky tomu se neuronové sítě s úspěchem dají aplikovat na celo řadu velmi komplexních problémů, jako je rozpoznávání písma, obličejů nebo převod písma na mluvenou řeč. V poslední době se neuronové sítě využívají i v oblasti data miningu.

1.1 Biologický neuron

Při tvorbě formálního modelu neuronu vycházeli jejich tvůrci hlavně z poznatků fyziologů. Ti v živých organismech zkoumali neurony jako základní stavební jednotky nervové soustavy. Neurony jsou speciální buňky uzpůsobené pro co nejefektivnější zpracování, uchování a přenos informací. Všechny tyto neurony ale mají stejnou strukturu. Každý neuron se tedy skládá z těchto částí:

- Tělo neuronu s jádrem – soma
- Dendrit se synapsemi
- Axon
- Terminály axonu

Vše je názorně přiblíženo na obrázku (Obr. 1.)



Obr. 1. Biologický neuron – převzato z [11]

Tělo neuronu s jádrem, které se nazývá soma, tvoří jeho základ. K němu směřují a přenášejí informace dendrity. Slouží jako vstupní kanály pro neuron. Do jednoho neuronu směřuje a přenáší informace 10 až 100 tisíc dendritů. Jako výstupní kanál neuronu slouží axon. U této části je pozoruhodné zmínit jeho délku. Zpravidla přesahuje 1 m. Axon se obvykle na konci větví na terminály. Terminály ukončuje blanka, která se v některých místech dotýká výběžků jiných neuronů. Proces komunikace mezi jednotlivými neurony potom probíhá na základě chemických reakcí. Proto se mluví o tzv. chemické synapsi. Tyto synapse se díky tomu, jak celý proces funguje, berou nejen jako prostředky k mezineuronovému spojení, ale také jako prostředek paměti [11].

1.2 Umělá neuronová síť

Neuronovou síť je možné chápat jako transformaci T vstupního signálu X na výstupní signál

$$\vec{Y} = T(\vec{X}). \tag{1}$$

Podle [11] je také možné tento pojem definovat následovně:

Za neuronovou síť se obecně dá považovat taková struktura pro distribuci paralelního zpracování dat, která se skládá z jistého, obvykle velmi vysokého počtu vzájemně propojených výkonových prvků. Každý z nich může současně přijímat libovolný konečný počet různých vstupních dat. Na další výkonné prvky může předat libovolný konečný počet shodných informací o stavu svého jediného, avšak velmi rozvětveného výstupu. Každý výkonový prvek transformuje vstupní data na výstupní podle jisté přenosové funkce. Přitom se též může uplatnit obsah jeho lokální paměti.

Pokud tedy přijmeme jako fakt, že neuronovou síť je možné chápat jako transformaci vstupního signálů na výstupní signál, je důležité zmínit se o tzv. třináctém Hilbertově problému a jeho řešení. Celý tento problém se týká toho, zda je možné reprezentovat spojitou funkci n proměnných pomocí součtu a superpozice spojitých funkcí jedné proměnné. Hilbert se domníval, že kořeny rovnice

$$\mathbf{x}^7 + ax^3 + bx^2 + cx + 1 = 0 \tag{2}$$

nelze vyjádřit jako funkce koeficientů a, b, c konečným součtem spojitých funkcí pouze dvou proměnných. Tento problém vyřešil Kolmogorov svým teorémem.

1.2.1 Kolmogorův teorém

Tento teorém aplikovaný na neuronové sítě ve svém důsledku říká, že pro transformaci libovolné funkce *f* pomocí transformační funkce *T* je postačující, aby neuronová síť měla alespoň tři vrstvy s odpovídajícím počtem neuronů v jednotlivých vrstvách. Jinými slovy, pokud chceme vyjádřit jakoukoliv reálnou funkci *f*, která má *n* proměnný a je definována v *n* – rozměrné krychli o hranách (0,1), pak platí:

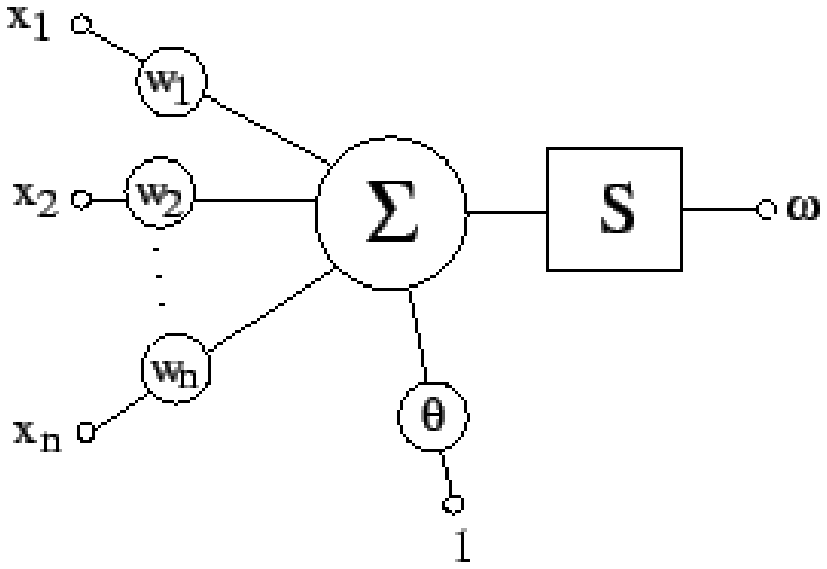
$$f(x1,\dots,xn) = \sum_{q=1}^{2n+1} \Psi_q \left(\sum_{p=1}^{n} \phi_{pq}(x_p) \right) \tag{3}$$

kde Ψ_q a ϕ_{pq} jsou spojité funkce jedné proměnné a $p = 1,\dots,n$ a $q = 1,\dots,(2n+1)$.

Pro danou funkci *f* jsou specifické pouze funkce Ψ_q, naproti tomu funkce ϕ_{pq} jsou pro dané *n* nezávislé. Později se také díky Hecht – Nielsonovi ukázalo, že je toto možné aplikovat i na funkce, které jsou definované ve vyšších prostorech.

1.2.2 Základní prvek umělé neuronové sítě – formální perceptron

Umělá neuronová síť je složená z jednotlivých prvků, které se nazývají formální neurony (Obr. 2). Tyto formální neurony se snaží napodobovat funkčnost biologických neuronů.



Obr. 2. Model formálního neuronu

Pro neurony perceptronového typu platí podle [9] pro zpracování dat tento vztah

$$y = S\left(\sum_{j=1}^{N} w_j x_j + \theta\right) \tag{4}$$

kde

- y je výstup neuronu, někdy se také označuje jako aktivita neuronu
- x_j je *j* – tý vstup neuronu, těch je celkově N
- w_j jsou *j* – té synaptické váhy
- S je přenosová funkce neuronu
- θ představuj prahovou hodnotu neuronu

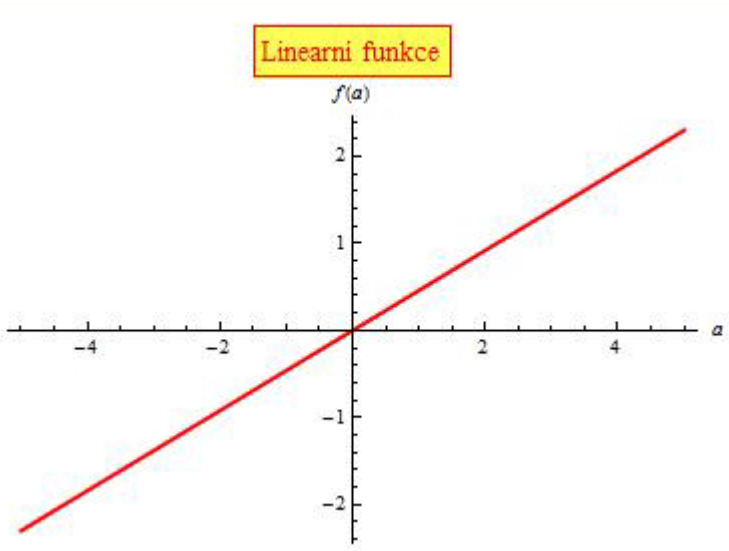
Neurony mohou pracovat jak s daty spojitého, tak binárního charakteru.

1.2.3 Typické přenosové funkce neuronu

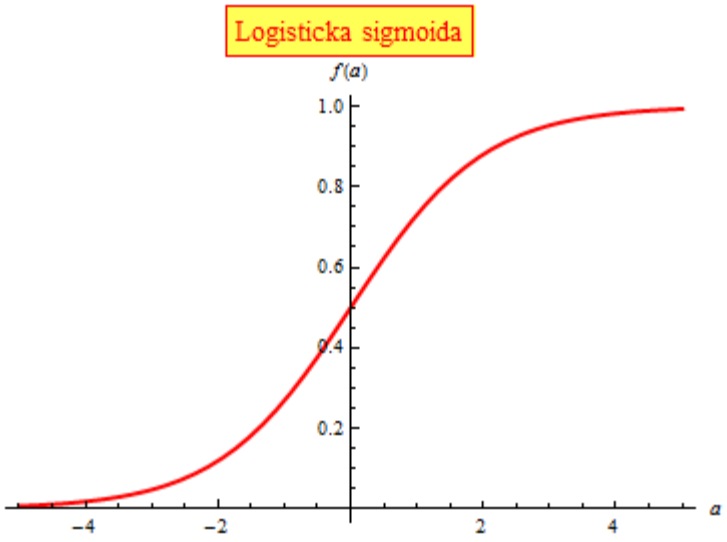
Pro činnost formálního neuronu hraje přenosová funkce (někdy také označovaná jako aktivační) velmi důležitou roli. Tato funkce určuje, zda bude neuron pracovat jako binární nebo spojitý prvek. Klasický neuron pracuje s funkcí dvouhodnotovou nebo Heavisideovou. Ve vícevrstvých sítích, pracujících se spojitými vstupními a výstupními daty, se velmi často používá funkce sigmoida.

Název	**Předpis**
Lineární funkce	$F(a) = a$
Logistická sigmoida	$F(a) = \frac{1}{1+e^{a}}$
Hyperbolická tangenta	$F(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$
Heavisideova funkce	$F(a) = \begin{cases} 1, & a > 0 \\ 0, & a \leq 0 \end{cases}$
Omezená funkce	$F(a) = 1\,;\ \forall\, a > 0\ jinak\ a = 0$

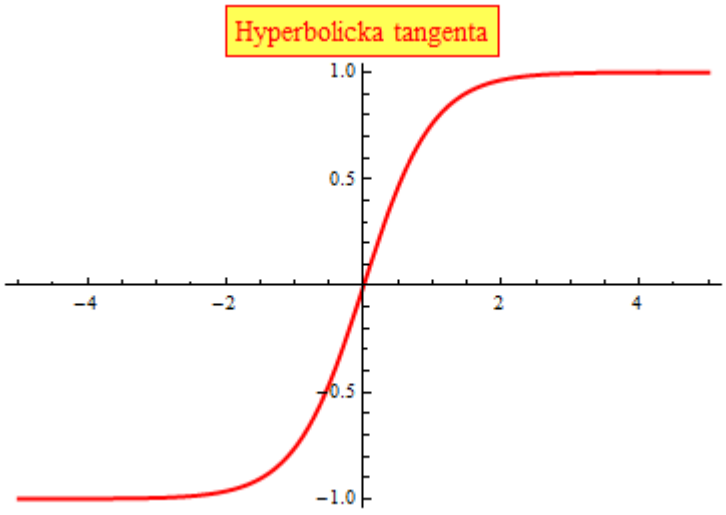
Tab. 1. Běžně používané aktivační funkce v neuronech



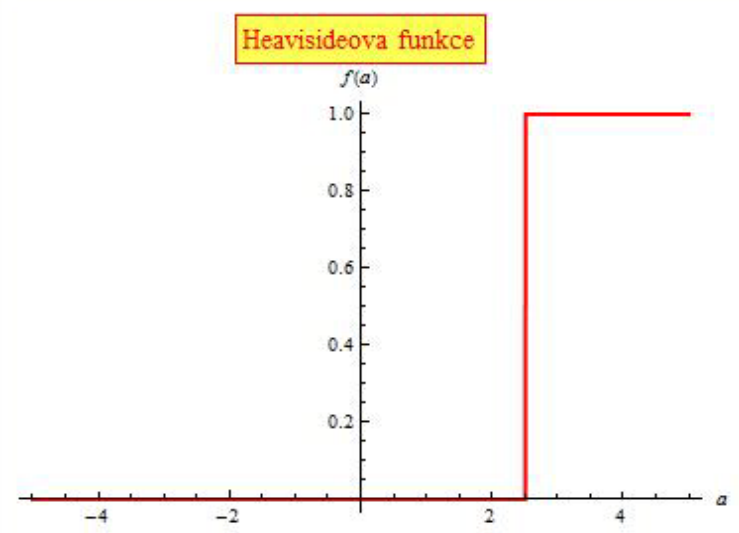
Obr. 3. Lineární funkce



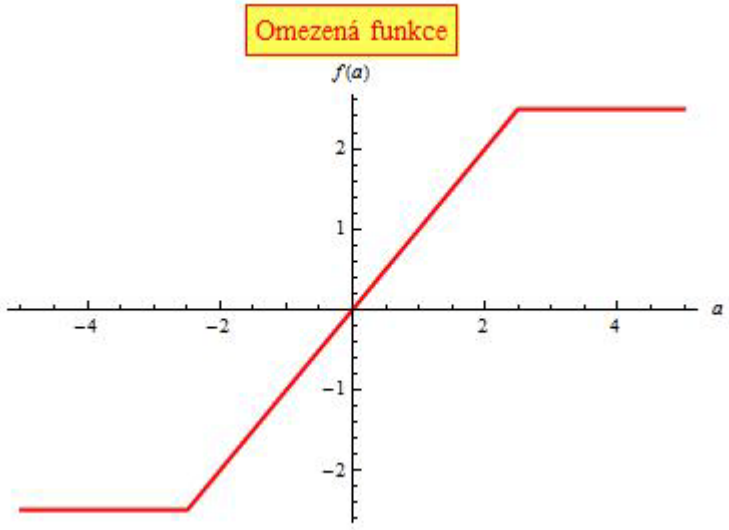
Obr. 4. Logistická sigmoida



Obr. 5. Hyperbolická tangenta



Obr. 6. Heavisideova funkce



Obr. 7. Omezená funkce

1.3 Pracovní fáze umělé neuronové sítě

Činnost umělé neuronové sítě je možné fakticky rozdělit do dvou fází. První fáze je adaptivní. Dochází k samotnému učení neuronové sítě. Tento proces je dynamický, v neuronové síti dochází ke změnám. Celá síť se adaptuje na řešení konkrétního problému. Během procesu učení se nastavují a mění hodnoty vah a prahů.

Druhou fází je proces vybavovací. Pracuje se opět se vstupními daty a sítí, která prošla adaptivní fází, tzn. že je naučená správně reagovat na tyto vstupní data.

1.3.1 Adaptivní fáze

Před započetím procesu učení se náhodně, nebo v případě podobnosti s dříve řešenými problémy, podobně nastaví váhy neuronové sítě. Na vstup se přivede množina dat, která se nazývá trénovací. Tato trénovací množina vytváří odezvu na výstupu neuronové sítě. Proces učení je ovlivněn tím, jaký typ sítě, respektive druh učení se využívá. Rozlišují se dva způsoby učení

- **Učení s učitelem.** - V případě, že se používá učení s učitelem, existuje nějaké kritérium, které určuje, jaká reakce na vstupní data je správná. Váhy neuronové sítě se nastavují podle toho, jak moc se liší výstupní reakce neuronové sítě od požadované (skutečné) hodnoty. Toto nastavování vah se děje ve zpětné vazbě. Modifikace vah se provádí podle zvoleného algoritmu, pomocí kterého se snižuje velikost chyby mezi požadovanou hodnotou a aktuální výstupní hodnotou neuronové sítě. Celý tento proces porovnávání a korekce vah se cyklicky opakuje. Po dostatečně velkém počtu cyklů učení je síť schopná správně reagovat na data, které se na vstupu neuronové sítě objeví. Jako příklad metod učení s učitelem je možné zmínit například posílené učení, stochastické učení nebo algoritmus zpětného šíření chyby.

- **Učení bez učitele.** - Pokud se využívá metoda učení bez učitele, pak neexistuje žádné kritérium, podle kterého by se porovnávaly odezvy neuronové sítě na data na vstupu. Neuronová síť v takovém případě vyhledává společné znaky nebo vzory ve vstupních datech. Je tedy možné pro takovou metodu učení použít termín samoorganizace, protože síť se sama organizuje pouze na základě výše uvedeného principu, tj. pouze na základě hledání stejných vzoru ve vstupních datech. Jako

příklady využití metody učení bez učitele je možné uvést Hebbovské učení nebo kompetici.

1.3.2 Vybavovací fáze

Druhou pracovní fází umělé neuronové sítě je fáze vybavování. Jak bylo uvedeno výše, tento proces následuje po adaptivní fázi. Z podstaty celého systému umělých neuronových sítí lze i zde nalézt podobnost funkcí lidského mozku, potažmo nervové soustavy. Stejně jako lidský mozek vytváří spojení mezi obličeji lidí a jejich jmény nebo mezi tvary a objekty, tak neuronové sítě také napodobují takové chování.

Neuronové sítě jsou opět rozděleny do dvou skupin, pokud jako kritérium pro jejich dělení využijeme to, jakým způsobem probíhá vybavovací fáze. Jedná se proces autoasociativní a proces heteroasociativní.

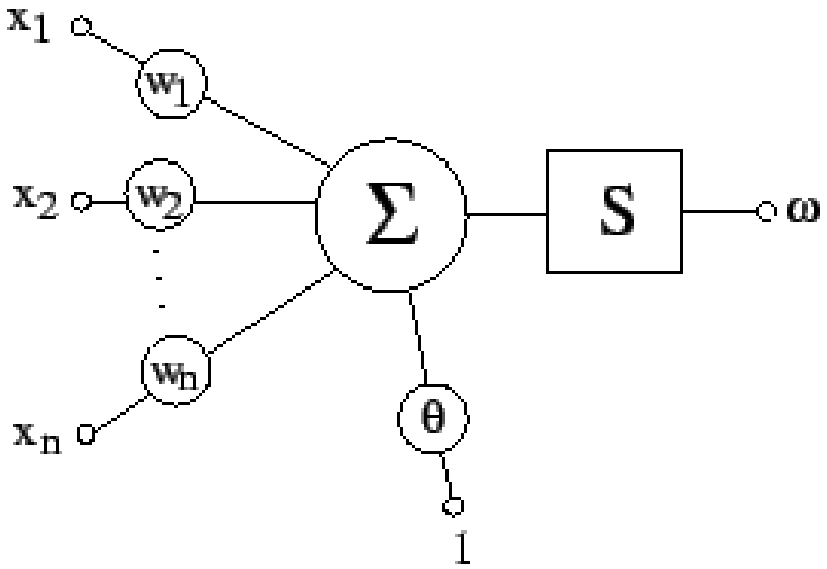
- **Autoasociativní vybavování. -** Tento model vybavování je založen na tom, že se z „paměti“ neuronové sítě vybavují různé vzory. U tohoto modelu vybavování se konkrétně jedná o vektory $\vec{X}_1,\dots,\vec{X}_n$. Právě díky tomuto hraje autoasociativní vybavování velký význam při zpracovávání nekompletních nebo vstupních vektorů. Jako příklad je možné uvést zpracování obrazových dat, které jsou nějakým způsobem zašuměna nebo jinak poškozena. Právě díky autoasociativnímu vybavování je umělá neuronová síť schopná přiřadit vstupu správný výstup, jinými slovy – je schopná ho správně klasifikovat. Velmi známou umělou neuronovou sítí využívající autoasociace je Hopfieldova síť, která je právě díky této vlastnosti schopna rekonstruovat i z poškozených vstupních dat originály.
- **Heteroasociativní vybavování. -** U tohoto druhu vybavování se pracuje s dvojicemi vzoru a obrazu, tedy páry vektorů $(\vec{X}_1,\vec{Y}_1)\dots,(\vec{X}_n,\vec{Y}_n)$, kde vektor $\vec{X}$ představuje vzor a vektor $\vec{Y}$ jeho obraz. Tento proces je možné přiblížit následujícím způsobem. Pokud člověk uvidí obraz moře a vzpomene si na dovolenou ve Španělsku, proběhla u něj heteroasociace. Tento proces je tedy možné popsat tak, že při vybavování nedochází k přiřazování vstupů k originálům ale asociativních vzorů. Zástupcem umělých neuronových sítí, které používají heteroasociativní vybavování je například síť BAM (Bidirectional Associatove Memory).

1.4 Neuronová síť typu Perceptron

Jako první tento druh sítě uveřejnili McCulloch a W. Pitts v roce 1943 [11]. Tento model neuronu, potažmo jednovrstvé neuronové sítě se stal základem pro další vědce, kteří díky jeho modifikaci a vylepšování přicházeli s dalšími a dalšími typy neuronových sítí. Tento typ sítě je možné popsat jako jednovrstvou síť bez skrytých vrstev. Pouze jednu vrstvu lze tedy učit. Tento model nastartoval vlnu zájmu o neuronové sítě a jejich aplikaci v praxi. Ovšem po publikování knihy M. Minského a S. Papperta Perceptrons, kde díky generalizaci jejich tvrzení o tom, že klasický perceptron má své hranice v realizaci logické funkce XOR i na jiné sítě, znamenal velký útlum zájmu a zpoždění vývoje. Díky Kolmogorově teorému se ovšem ukázalo, že funkci XOR samozřejmě lze realizovat neuronovými sítěmi. Ovšem ne jednovrstvou sítí typu Perceptron.

1.4.1 Klasický McCulloch – Pittsův Perceptron

Síť vytvořená pomocí těchto neuronů byla jedním ze základních kamenů rozvoje neuronových sítí. Dle názvu je jasné, že autory jsou McCulloch a W. Pitts. Pro tento neuron je typická tato stavba



Obr. 8. Model neuronu podle McCulloch – Pittse

kde

- y je výstup neuronu, někdy se také označuje jako aktivita neuronu
- x_j je *j* – tý vstup neuronu, těch je celkově N
- w_j jsou *j* – té synaptické váhy
- S je přenosová funkce neuronu
- θ představuj prahovou hodnotu neuronu

Klasický perceptron pracoval pouze s binární reprezentací vstupních a výstupních dat. Jako přenosová funkce se u klasického perceptronu tedy právě díky tomu, že pracoval s dvouhodnotovou reprezentací dat, používá dvouhodnotová nebo Heavisideova funkce. Pokud se pracuje s hodnotami 0 a 1, využívá se dvouhodnotová přenosová funkce. Ale u některých jiných typů sítí se pracuje s hodnotami -1 a 1 a proto se pracuje se znaménkovou funkcí.

1.4.2 Rosenblattův perceptron

Tato síť vznikla v roce 1957. Jeho autorem je F. Rosenblatt. Ten při tvorbě tohoto perceptronu vycházel s fyziologické stavby lidského oka. To je vlastně tvořeno maticí světlocitlivivých čidel. Ty jsou napojeny na speciální buňky, které umí rozeznávat určité typy vzorů. Buňky se nazývají démony. Tyto démony jsou spojeny s dalšími buňkami, které reagují jenom pokud dosáhne jejich podráždění určité úrovně vzruchu.

Proto Rosenblatt vytvořil síť kopírující fyziologickou strukturu stavby lidského oka. Tato síť je tedy třívrstvá. První vrstva je vstupní a funguje jako rozvětvení. Tato vrstva mapuje matici, tedy dvourozměrné pole čidel a převádí ho na vektor hodnot. Tato vrstva je spojena s prostřední vrstvou, která plní funkci detektorů příznaků. Spojení mezi detektory a vstupní vrstvou je náhodné a váhy jsou konstantní. Ve třetí, výstupní vrstvě jsou umístěny rozpoznávače příznaků. Tato vrstva je vlastně ta nejdůležitější. U této vrstvy se při trénování váhy na rozdíl od předchozích dvou vrstev váhy nastavují. Proto se Rosenblattova perceptronová síť klasifikuje jako jednovrstvá. Je možno říct, že Rosenblattův perceptron se od klasického McCulloch – Pittsova perceptronu liší tím, že kromě klasického perceptronu zahrnuje i učící algoritmus [9].

1.4.3 Minsky – Papert perceptron

Tento neuron, potažmo neuronová síť vychází stejně jako Rosenblattův perceptron z klasického McCulloch – Pittsova neuronu. Minsky s Papertem maximálně zjednodušili klasický perceptron a tím z něj vytvořili jednovrstvou síť. Předností tohoto perceptronu je relativní snadnost geometrické reprezentace adaptivního procesu.

Právě díky tomu, že autoři kladli důraz na co největší jednoduchost, vyvstala otázka řešení lineárně neseparovatelných problémů. Autoři se tím zabývali ve své publikaci Perceptrons. Díky tomu, že jejich jednovrstvá síť nebyla schopná řešit logickou funkci XOR, autoři tvrdili, že všechny neuronové sítě nejsou schopny tento problém úspěšně řešit. Díky tomu, že šlo o uznávané odborníky, začala se tato hypotéza obecně přijímat a tím se do jisté míry zastavil vývoj neuronových sítí jako vědního oboru.

1.4.4 ADELINE – Adaptive Linear Neuron

I přes útlum zájmu o perceptrony a sítě perceptronového typu pokračoval jejich výzkum a vývoj. Jedním z vědců, kteří pokračovali ve výzkumu, byl B. Widrow a tým lidí ze Stanford University. Jako většina jejich předchůdců, vycházeli z formálního modelu perceptronu McCulloch – Pittse [11]. Stejně jako v případě neuronové sítě popsané v předchozí kapitole 1.4.3, je odlišnost sítě v tom, že obsahuje učící algoritmus stejně jako neuronová síť s kterou přišli Minsky a Papert.

U klasického perceptronu je vstupem do učícího bloku dvouhodnotový výstup neuronu. V případě sítě ADELINE je vstupem učícího bloku vnitřní potenciál neuronu. Tento učící algoritmus je popsán podle [9] touto rovnicí

$$w_{ij}(t+1) = w_{ij}(t) + \eta e_j(t) x_i(t) \tag{5}$$

kde

- η je učící koeficient
- w_{ij} nastavované váhy neuronu
- e_j chyba perceptronu

Pro výpočet chyby perceptronu u sítě *ADELINE* se porovnává odchylka mezi vnitřním staven neuronu a jeho potenciálem. Toto je popsáno rovnicí

$$e_j = d_j(t) - s_j(t) \tag{6}$$

kde

- e_j je chyba perceptronu
- d_j skutečný vnitřní stav neuronu
- s_j vnitřní potenciál neuronu

Neuronová síť ADELINE tedy využívá pro adaptační proces algoritmu, kdy se minimalizuje Euklidova vzdálenost požadované odezvy a vnitřního potenciálu neuronu. Tato minimalizace spadá to gradientních metod.

1.5 Neuronové sítě s dopředným šířením

Model neuronové sítě založený na jednoznačně definovaném informačním toku ve struktuře sítě. Neexistují žádná spojení vedoucí z vyšší vrstvy neuronové sítě do vrstvy nižší. Nejsou definována ani spojení mezi prvky v jednotlivé vrstvě. Typicky se jedná o síť s jednou vstupní vrstvou, jednou skrytou vrstvou a jednou výstupní vrstvou. Toto je minimální architektura sítě s dopředným šířením.

Mezi nejznámější zástupce dopředných neuronových sítí patří neuronové sítě MADELINE, vícevrstvé neuronové sítě s algoritmem Back propagation (zpětné šíření chyby) a také sítě typu RBF.

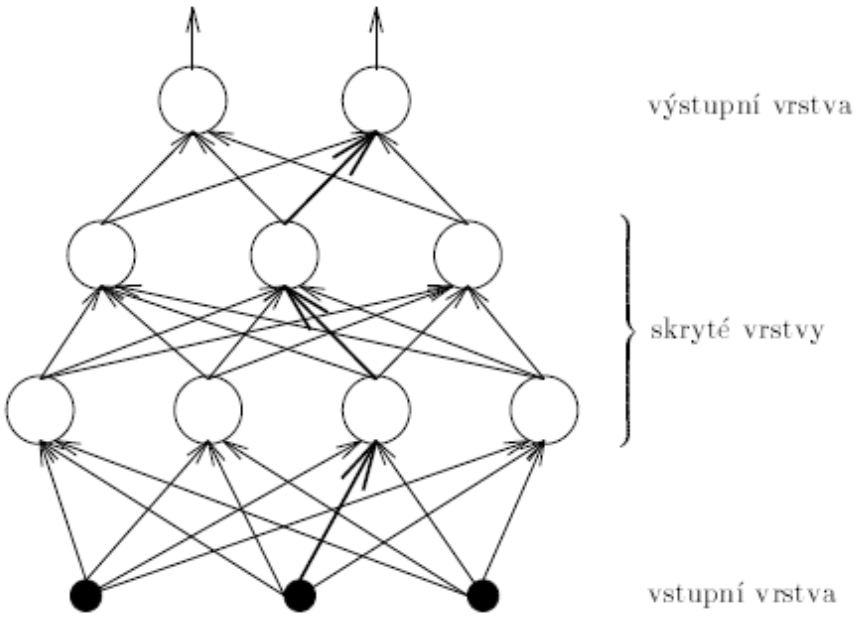
1.5.1 MADELINE

Tato síť je nejjednodušším zástupcem dopředných neuronových sítí. Jejím autorem je B. Widrow. Síť je vlastně zdokonalením sítě ADELINE. Vlastní název MADELIN*E* je zkratkou vytvořenou ze slov Multiple Adaptive Linear Element. Síť MADELINE je tedy velmi podobná síti ADELINE. Rozdíl je v její architektuře a také v principu učení. Pracuje pouze s binárními signály.

V adaptivní fázi se využívá principu majority. Jestliže je více jak polovina neuronů ve skryté vrstvě nastavena na hodnotu +1, pak je výstupem neuronové sítě +1. Není – li tomu tak, pak se na výstupu objeví hodnota -1. V průběhu vybavovací fáze dochází k adaptaci vah u těch neuronů ve skryté vrstvě, jejichž vnitřní potenciál se nejvíc blíží hodnotě 0. To se opakuje tak dlouho, až je na výstupu požadovaná odezva. Tato neuronová síť si pro jednodušší aplikace vystačí s jedním neuronem ve výstupní vrstvě. Pro složitější aplikace může být neuronů více.

1.5.2 Vícevrstvá neuronová síť s algoritmem Back propagation

U této neuronové sítě je její název odvozen od algoritmu učení, který se dá charakterizovat jako zpětné šíření chyby v neuronové síti. Tento druh sítí je v dnešní technické praxi jedním z nejvíce rozšířených. Právě vývoj těchto neuronových sítí znovu nastartoval zájem o neuronové sítě.



Obr. 9. Obecná struktura vícevrstvé dopředné neuronové sítě

Jak je patrné z obrázku (Obr. 9.) je obecně možné tento druh neuronových sítí klasifikovat jako vícevrstvou neuronovou síť, která je složena z McCulloch – Pitssových neuronů jako výkonných prvků. U těchto neuronů se používá jako aktivační funkce logistická sigmoida. To dovoluje pracoval se spojitými hodnotami na vstupu i výstupu. Neurony jsou v jednotlivých vrstvách kompletně propojeny. Funkcionalita sítě je celkově postavena na učícím algoritmu Back Propagation, který byl původně objeven P. Werbosem a následně znovu objeven a publikován v knize *Learning Internal Representation by Error Propagation* v roce 1986 autory Rumelhartem, Hintonem a Williamsem.

1.5.2.1 Popis algoritmu Back propagation

Algoritmus Back propagation je iterační proces. Jeho činnost je možné podle [9] popsat touto posloupností vykonávaných kroků

1. Inicializace sítě
2. Předložení vzoru
3. Srovnání
4. Zpětné šíření chyby a modifikace vah
5. Ukončení výběru vzorů z trénovací množiny
6. Ukončení adaptační fáze

- V kroku 1. nazvaném *inicializace* se nastavují váhy v neuronové síti na náhodné hodnoty.
- Krok 2., označený jako *předložení vzoru,* je možné chápat tak, že se na vstup sítě přivede vzor s trénovací množiny a přenese se přes všechny vrstvy neuronové sítě na výstup. To znamená, že se přepočítají všechny výstupy neuronů směrem od vstupu k výstupu. Pro tento proces platí tyto vztahy:

$$y = S\left(\sum_{j=1}^{N} w_j x_j + \theta\right) \tag{7}$$

$$S(\varphi) = \frac{1}{1 + e^{-\gamma\varphi}} \tag{8}$$

- Následuje fáze *srovnávání*, označená jako krok 3. Zde se počítá energie podle vztahu (9) a také chyba výstupní vrstvy podle vztahu (10)

$$E = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{y_i} - \mathbf{d_i})^2 \tag{9}$$

$$\delta_i^0 = (d_i - y_i^0)\gamma y_i^0(1 - y_i^0) \tag{10}$$

- V dalším kroku dojde k modifikaci vah a zpětnému šíření chyby. Pro všechny neurony ve vrstvě se vypočítá:

$$\Delta w_{ij}^{l}(t) = \eta \delta_i^l(t) y_i^{l-1}(t) + \alpha \Delta w_{ij}^{l}(t-1) \tag{11}$$

$$\Delta \Theta_{ij}^{l}(t) = \eta \delta_i^l(t) y_i^{l-1}(t) + \alpha \Delta \Theta_{ij}^{l}(t-1) \tag{12}$$

Poté podle vztahu (13) dojde ke zpětnému šíření chyby do vrstvy ležící blíže vstupu.

$$\delta_i^{h-1} = y_i^{h-1}(1 - y_i^{h-1}) \sum_{k+1}^{n} w_{ki}^{h} \delta_k^{h} \tag{13}$$

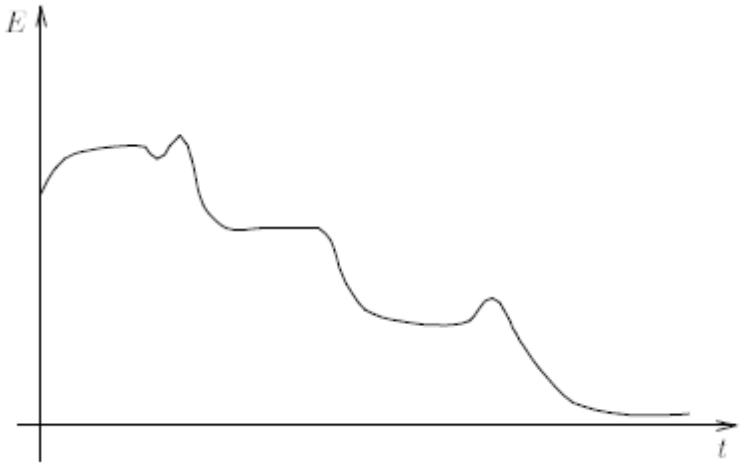
Posledním procesem vykonaným v kroku 4. je modifikace vah. To se děje podle vztahů (14) a (15)

$$w_{ij}^{l}(t+1) = w_{ij}^{l}(t) + \Delta w_{ij}^{l}(t) \tag{14}$$

$$\Theta_i^l(t+1) = \Theta_i^l(t) + \Delta \Theta_i^l(t) \tag{15}$$

Krok 4. se pakuje pro všechny vrstvy sítě od výstupní vrstvy($l = 0$) až po vstupní vrstvu. Pokud se zpracovává skrytá vrstva nejblíže vstupní vrstvě, nahradí se y_j^{l-1} v rovnici (11) za odpovídající vstupní hodnotu, tzn. za x_j.

- Ukončení výběru vzorů z testovací množiny testuje, zda byly předloženy všechny vzory. Pokud ne, vrací se algoritmus na krok 2. Jinak se pokračuje krokem 6.
- Pokud byla chyba za poslední epochu menší než v předchozí, ukončuje se celý chod algoritmu. Pokud se vyčerpal počet epoch, opět se ukončuje chod algoritmu. Jinak se algoritmus vrací na krok 2.

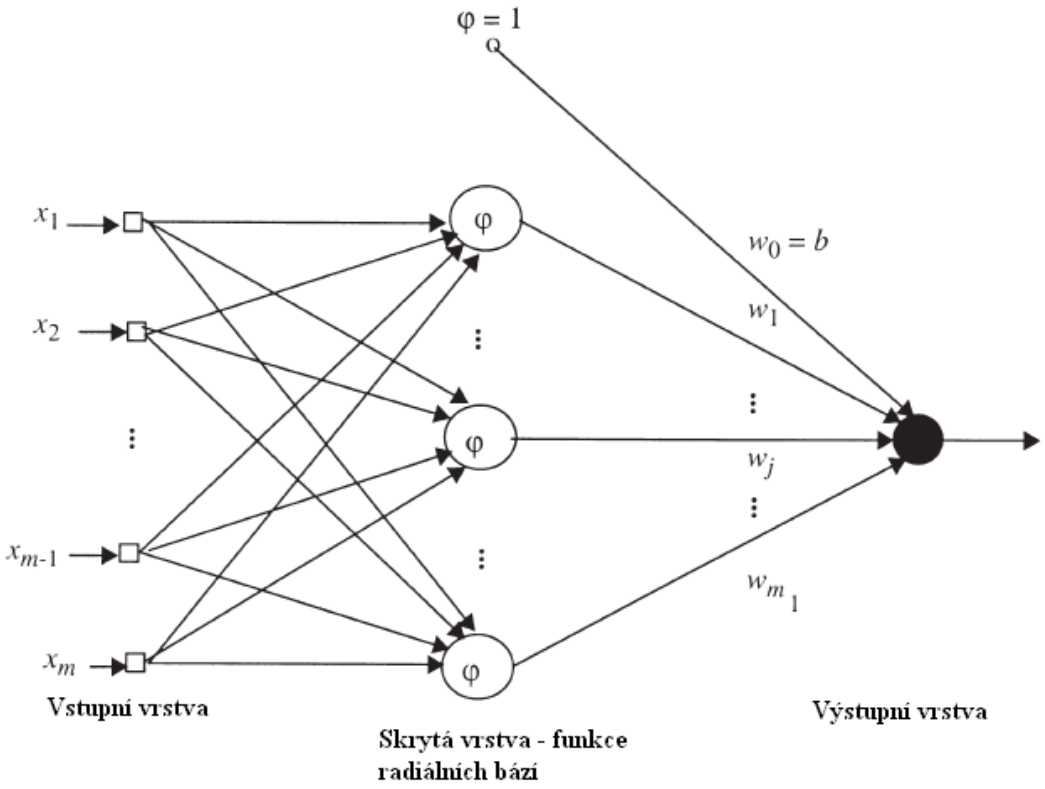


Obr. 10. Typický průběh chyby během učení pomocí Back propagation – převzato z [11]

Existuje několik variant tohoto algoritmu, které mají za cíl zlepšit jeho činnost a zrychlit proces učení. Jako zástupce modifikací algoritmu Back propagation je možné uvést Quickprop, Maxprop, Fast – Learning.

1.5.3 Neuronová síť RBF

Tento typ neuronové sítě je do jisté míry konkurentem vícevrstvých neuronových sítí perceptronovského typu. Neuronová síť RBF typicky obsahuje vstupní, výstupní a jednu skrytou vrstvu. Ve skryté vrstvě jsou obsaženy neurony typu RBF, ve výstupní vrstvě jsou neurony perceptronovského typu. Propojení mezi prvky v síti bývá úplné. Struktura sítě je na obrázku (Obr. 11.):



Obr. 11. Struktura RBF sítě

Tento typ sítí, využívá na rozdíl od perceptronovských u neuronů ve skryté vrstvě jako svou aktivační funkci Gaussovu funkci. Podle [9] jsou neurnony RBF reprezentovány pomocí následujících vztahů:

$$\varphi = \sqrt{\sum_{i=1}^{n} (x_i - c_i)^2} \tag{16}$$

$$y^* = e^{-\frac{\varphi^2}{\sigma^2}} \tag{17}$$

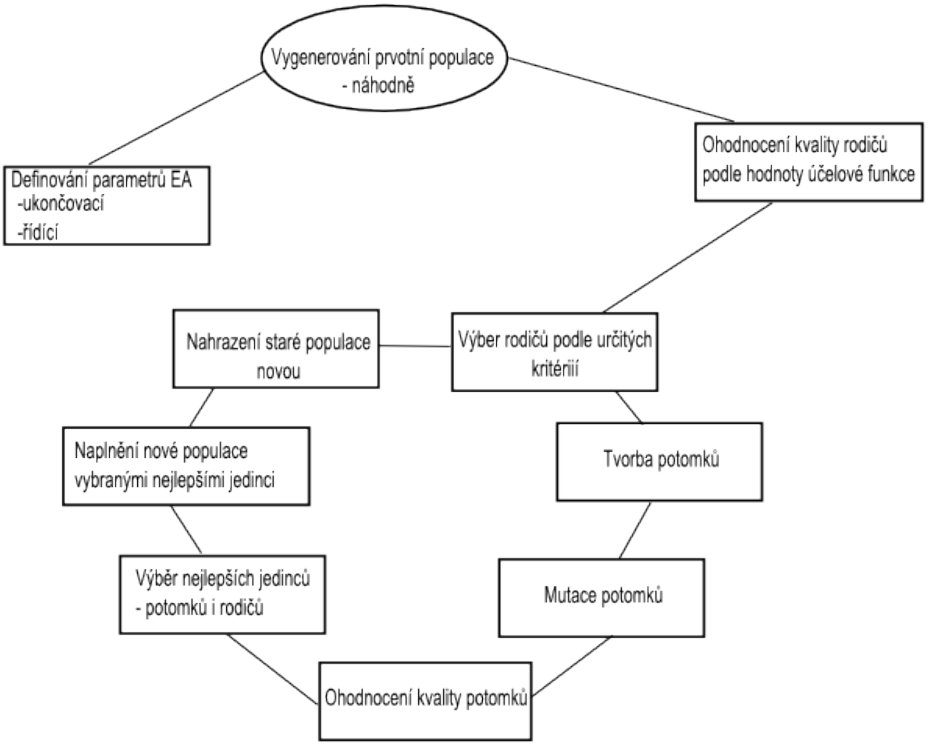
Popis chování RBF neuronu je rozložen na tyto dvě rovnice. První rovnice odpovídá výpočtu vnitřního potenciálu a druhá popisuje aktivační funkci neuronu. Vnitřní potenciál je reprezentován vzdáleností mezi vstupním vektorem a vektorem $\vec{c}$. Používá se Euklidovská metrika, díky níž jsou vstupní data rozložena pro konstantní hodnotu φ symetricky kolem jediného bodu. Vektor $\vec{c}$ se označuje jako prototyp. Reprezentuje určitou podmnožinu vstupních dat, která mám tvar shluku.

Proces učení u sítí RBF se skládá ve své podstatě ze dvou fází. V první fázi se zpracovávají jen vstupní data bez ohledu na vzory. Určí se vektor $\vec{c}$, tedy prototyp a parametr σ , určující strmost aktivační funkce (Gaussova funkce). V následující fázi učení se vypočítávají váhy výstupních neuronů. Jedním z používaných algoritmů pro učení sítí RBF je K – meas algoritmus nebo jeho adaptivní verze.

Tyto typy sítí se používají jak na klasifikační úlohy, tak na úlohy aproximačního typu.

2 EVOLUČNÍ ALGORITMY

Evoluční algoritmy hledají svou inspiraci v přírodě. První evoluční algoritmy čerpaly inspiraci z Darwinovi teorie evoluce a základů genetiky. První evoluční algoritmus, který vznikl, nese název genetický algoritmus. U evolučních algoritmů se běžně používá terminologie známá z biologie, jako například chromozom, genotyp, fenotyp a podobně. Myšlenkovou podstatou evolučních algoritmů je vlastně napodobování zušlechťování kvality populace během jeho generačního vývoje. Jedinci z populace – rodiče, tvoří potomky. Zkoumá se vhodnost potomků – stejně jako v reálném světě, kde se uplatní pouze jedinci nejlépe vybavení pro přežití v daném prostředí (při převedení do světa evolučních algoritmů je toto posuzováno z hlediska ohodnocení účelové funkce). Celý evoluční cyklus je možné znázornit takto:



Obr. 12. Cyklus evolučního algoritmu

Z obrázku (Obr. 12.) je patrné, že pro výběr jedinců, ať již rodičů nebo potomků, hraje zásadní roli jejich vhodnost. To znamená ohodnocení jejich účelové funkce. Příkladem může být aplikace evolučního algoritmu na hledání minima funkce. Tady je jasné, že

jedinci, jejichž hodnota účelové funkce bude velká, nebudou těmi pravými kandidáty na tvůrce nové populace. Při tvorbě nové populace hraje také významnou roli to, v jaké míře dochází k mutaci potomků. Tento jev se podepisuje velmi výrazně na kvalitě celé populace. Existují v zásadě dva různé přístupy k tomuto aspektu. Je možné využívat mutaci pouze v malé míře, ale s velkou pravděpodobností. Tím vznikají jedinci s jenom velmi mírně odlišnými vlastnostmi. Druhou variantou je aplikovat mutaci méně častěji, ale s větším vlivem na změnu parametrů. Obě varianty jsou ovšem mezními případy. Ve velké míře je nutné s nastavením míry a četnosti mutace experimentovat. Jeho nevhodným nastavením je možné celou populaci velmi snadno zničit.

Evoluční algoritmy jsou relativně nové, co se týká jejich stáří. Ovšem vznikla jich celá řada. Jako zástupce je možné uvést algoritmus SOMA, Diferenciální evoluce, Ant Colony Optimalization, Scatter Search, Particle Swarm, genetické programování a další. Tyto algoritmy našly řadu uplatnění. Podle [10] je jednou z alternativ uplatnění evolučních algoritmů vytvoření jakéhosi mostu mezi algoritmy, které jsou soběstačné a těmi, které potřebují pro svůj běh lidskou obsluhu. Jako nejperspektivnější z evolučních algoritmů se jeví genetické programování, které místo hledání řešení regresních funkcí hledá samotné regresní funkce a jejich parametry.

2.1 Populace

Pro oba níže popsané evoluční algoritmy je tento pojem velmi důležitý, jelikož jejich činnost je na práci s populací jedinců založená. Protože je tvorba populace pro oba stejná, je popsána samostatně v této kapitole. Populace je vlastně množina všech jedinců, kterou lze reprezentovat jako matici. Ve sloupcích jsou jedinci a řádky představují argumenty účelové funkce. Ke každému jedinci je také přidána hodnota účelové funkce. Ta říká, jak je jedinec vhodný pro účast na tvorbě nové populace [10].

Dříve než se populace vytváří, musí být nadefinován vzorový jedinec – tzv. Specimen. Ten určuje jak budou jedinci ve vytvořené populaci vypadat.Typicky obsahuje informaci o tom, o jaký typ proměnné půjde (celočíselné, reálné, diskrétní) a interval, který udává nejnižší a nejvyšší hodnotu, ve které se jednotlivé parametry jedince mohou pohybovat. Příkladem takové definice může být tento symbolický zápis {Real,{Dolni,Horni}}. Takto definovaný vzorový jedinec bude mít parametry reálného charakteru, které se budou pohybovat

v mezích Dolni a Horni. Správná definice hranic intervalu přispívá k odstranění nevhodných nalezených řešení (např. fyzikálně nerealizovatelných).

2.2 SOMA

Samo – Organizující se Migrační Algoritmus, SOMA existuje od roku 1999. Principiálně je založen na genetických algoritmech. Původní myšlenka, která vedla k jeho vytvoření, spočívá v napodobení chování skupiny inteligentních jedinců, kteří spolu kooperují při řešení společného problému, jako je například hledání zdroje potravy apod. [1].

Algoritmus SOMA je, jak bylo uvedeno výše, založen na principech genetických algoritmů. Ale jelikož je inspirován chováním inteligentních jedinců, je mírně odlišný od ostatních evolučních algoritmu. Hlavní odlišnost spočívá v tom, že nový jedinci (řešení) nevznikají křížením rodičů, ale pomocí kooperativního prohledání prostoru možných řešení. Samoorganizační vlastnost algoritmu je dána tím, že se jedinci za běhu algoritmu mohou spojovat do skupin, ty se mohou opětovně rozpadat, a tím se vlastně ovlivňují všichni jedinci sami navzájem. Tím se dosahuje toho, že si populace sama organizuje pohyb jedinců v prostoru řešení problému. Z toho je právě odvozeno slovo samo – organizující se v názvu algoritmu.

2.2.1 Princip algoritmu SOMA

Podle toho, co bylo uvedeno výše, SOMA napodobuje chování inteligentních jedinců, kteří spolu dohromady řeší nějaký problém. Tento přístup je možné nazvat jako konkurenčně - kooperační. Toto chování je v přírodě přítomné na každém kroku, ať už jde o smečku vlků, nebo včelí roj. Podstata je v tom, že spolu jedinci soutěží do té doby, než najdou nejlepší řešení – např. nejlepší zdroj potravy. Tyto jevy se dají popsat jako konkurenční fáze a kooperační fáze. V konkurenční fázi se každý jedinec snaží získat co nejlepší řešení – opět např. zdroj potravy, v kooperační fázi si jedinci předají informace a využijí poznatků nejúspěšnějšího jedince z předchozí konkurenční fáze.

2.2.2 Parametry algoritmu SOMA

Celý běh algoritmu je ovlivňován pomocí jeho parametrů. Ty určují jak kvalitní bude jeho běh. Je nutné mít určitou znalost o významu parametrů, protože ty se musí volit ještě před spuštěním algoritmu. Nevhodné nastavení vede k tomu, že výsledky algoritmu by mohly zůstat daleko za očekáváním.

Parametr	Doporučený rozsah	Druh parametru
PartLength	[1,5;5>]	Řídící parametr
Step	[0,11;PathLength]	Řídící parametr
PRT	[0;1]	Řídící parametr
D	Dané problémem	Počet argumentů úč. funkce
PopSize	[10;volí uživatel]	Řídící parametr
Migrace	[10;volí uživatel]	Ukončovací parametr
MinDiv	[volí uživatel]	Ukončovací parametr

Tab. 2. Parametry algoritmu SOMA – převzato z [1]

- **PathLength -** Tento parametr je důležitý pro pohyb jedince v prostoru možných řešení. Parametr říká, v jaké vzdálenosti se jedinec zastaví od vedoucího jedince v daném migračním kole – tzv. leadera. Pokud je nastaven tento parametr na hodnotu 1, jedinec se zastaví na pozici leadera, při nastavení tohoto parametru na hodnotu 2 dochází k tomu, že se jedinec zastaví na pozici stejně vzdálené od leadera, jaká byla jeho vzdálenosti k leaderovi v době startu jedince. Pokud se parametr nastaví na čísla menší než 1, dojde k degeneraci algoritmu, protože bude nalezené extrémy budou pouze lokální. Jako dostatečná se podle [1] jeví hodnota 3.

- **Step -** Parametr Step určuje, s jakou jemností se bude prohledávat plocha možných řešení. Při znalosti prohledávaného prostoru, při vědomí toho, že se jedná o jednoduchou funkci, je možné pro zrychlení chodu algoritmu zvolit hodnotu parametru Step relativně velkou. Pokud se ale jedná o složitější prostor řešení – multimodální funkci s velkým počtem lokálních extrémů, je vhodné zvolit parametr Step menší, čímž se zabezpečí to, že plocha řešení se bude prohledávat důkladněji. Je vhodné zvolit hodnotu parametru tak, aby nebyla celočíselným násobkem vzdálenosti mezi leaderem a aktuálním jedince. Pokud by tomu tak bylo, mohlo by to ovlivnit chod algoritmu negativním způsobem. Mohlo by dojít k tomu, že jedinci by byli vtaženi do lokálního extrému. Proto je vždy lepší volit hodnoty jako například 0.11 než 0.1.

- **PRT -** Tento parametr reprezentuje tzv. pertubaci. Tedy proces, podobný mutaci u ostatních evolučních algoritmu. Technicky vzato se jedná o stejný princip, ale název je jiný diferenciaci terminologie. Pertubace je jedním z nejdůležitějších parametrů algoritmu a s jeho nastavením stojí a padá jeho výkonnost. Nastavením tohoto parametru je možné algoritmus SOMA degradovat na obyčejný deterministický algoritmu, který zvládne vyhledávat pouze lokální extrémy. Tento jev nastane, pokud nastavíme parametr PRT na hodnotu 1. To vyplývá z rovnice, která platí pro výpočet pertubačního vektoru.

$$\vec{r} = \vec{r_0} + \vec{m}\, t\, PRT\vec{V}ector \qquad (18)$$

  Optimální hodnota nastavení tohoto parametru je někde okolo 0,1. Pokud je prostor řešení tvořen jednoduchou funkcí, je opět možné pro zrychlení chodu algoritmu nastavit PRT na hodnotu od 0,7 – 1,0.

- **D –** tento parametr reprezentuje dimenzi konkrétního problému řešeného pomocí SOMA algoritmu. Jinými slovy udává počet argumentů optimalizované účelové funkce. Jako parametr je dám samou podstatou problému a jeho změna je možná jen tehdy, pokud by se celý problém přeformuloval a došlo by ke změně počtu argumentů účelové funkce.

- **PopSize** – parametr PopSize určuje, kolik jedinců bude mít populace, která se bude podílet na běhu SOMA algoritmu. Velikost populace by měla odpovídat dimenzi. Pro komplikované problémy se volí hodnota parametru jako 0,2 – 0,5 násobek parametru D. Je možné tento parametr nastavit přímo na hodnotu parametru D. Pro optimální běh celého algoritmu by neměla hodnota tohoto parametru klesnout pod 10. Při volbě PopSize na hodnotu 2 opět dojde k degradaci algoritmu na obyčejný deterministický algoritmus.

- **Migrace -** migrace vyjadřuje, kolik proběhne cyklů SOMA, ve kterých dojde k přeorganizování jedinců. Tento parametr je jedním z ukončovacích a je ekvivalentem k parametru Generace z jiných evolučních algoritmů.

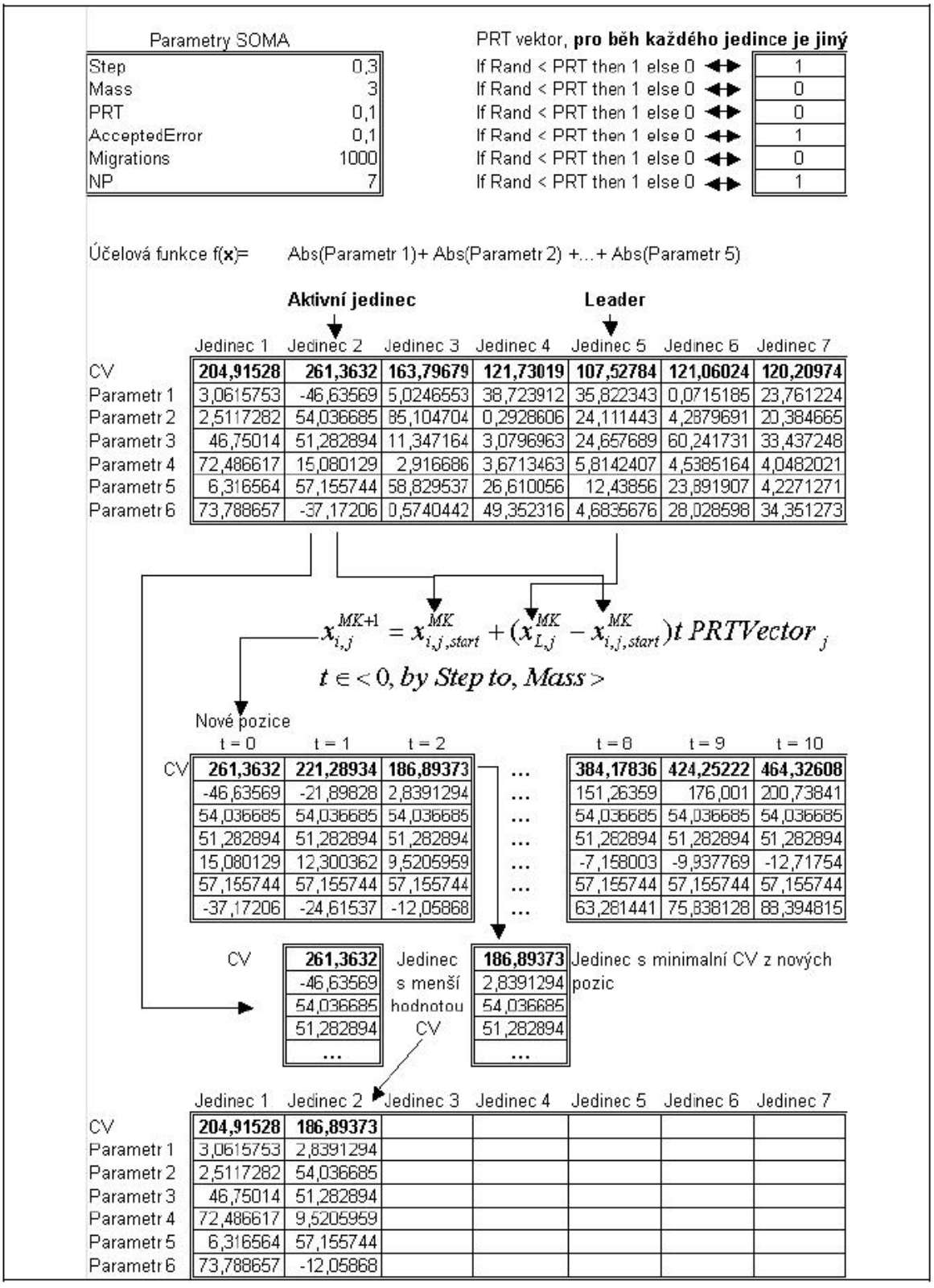
- **MinDiv** – Další z ukončovacích parametrů SOMA. Jeho hodnota udává, jakou tolerujeme odchylku mezi nejlepším a nejhorším jedincem v populaci. Tímto parametrem můžeme zkrátit běh celého algoritmu. Ovšem jeho špatným nastavením může dojít k destrukci celého algoritmu. Protože pokud nastavíme

parametr příliš velký, dojde v prvním migračním kole ke splnění podmínky rozdílu mezi hodnotou nejlepšího a nejhoršího jedince a celý algoritmus se zastaví, aniž by došlo k nalezení globálního extrému. Jeho nastavení na malou nebo zápornou hodnotu nehraje žádnou zásadní roli. Dojde maximálně k tomu, že se vykoná plný počet migrací.

2.2.3 Fáze algoritmu SOMA

Algoritmus SOMA je podle [1] možné popsat pomocí několika fází, které se opakují, dokud nejsou naplněny ukončovací parametry:

1. **Nastavení parametrů.** – V počáteční fázi se nastaví všechny parametry algoritmu, které byly zmíněny v předešlé kapitole(2.2.2). Dále se také definuje Specimen, tedy vzorový jedinec, podle kterého se vytvoří populace. Také se definuje účelová funkce, pro kterou budeme hledat optimální řešení.
2. **Vytvoření nové populace.** – Nová populace se vytváří pomocí generátoru náhodných čísel. Podle Specimena se náhodně generují hodnoty pro jednotlivé parametry jedinců. Tyto náhodně generované hodnoty jsou vlastně parametry účelové funkce.
3. **Migrační kola.** – Každé migrační kolo začíná výběrem *leadera* – jedince s nejlepší hodnotou účelové funkce. Následně se všichni jedinci začnou pohybovat směrem k *leaderovi.* To, jak probíhá pohyb je ovlivněno pomocí parametrů nastavených v bodě 1. Po ukončení pohybu se jedinci vracejí na pozici s nejlepší hodnotou účelové funkce nalezenou během jejich pohybu k leaderovi. V této fázi se také uplatňuje pertubační vektor. Ten ovlivňuj pohyb jedinců. Jedná se o tzv. *pertubaci,* Tato vlastnost algoritmu zvyšuje jeho robustnost díky tomu, že pohyb je vlastně prováděn po daleko větší hyperploše.
4. **Testování, zda byli naplněny ukončovací parametry.** – Testuje se, jestli je splněn parametr *MinDiv*, popřípadě zda bylo dosaženo hodnoty parametru *Migration.* Pokud nejsou tyto požadavky splněny, vrací se běh algoritmu na krok 3.
5. **Ukončení algoritmu.** – V této fázi algoritmus vrátí nalezené řešení.



Obr. 13. Jedno migrační kolo algoritmu SOMA, převzato z [1]

2.2.4 Strategie SOMA

Pro algoritmus SOMA existuje několik strategií ve smyslu verzí. Jednotlivé verze jsou podle [10] plně porovnatelné ve výkonnosti nalezení globálního extrému. Tyto jednotlivé strategie se liší převážně svým přístupem k pohybu jedinců po dané hyperploše.

1. **SOMA AllToOne.** – Tato strategie je vlastně popsána v předchozí kapitole. Všichni jedinci se v migračním kole pohybují směrem k *leaderovi.*
2. **SOMA AllToAll.** – Strategie AllToAll je založena na tom, že neexistuje žádný leader a všichni jedinci se pohybují směrem ke všem jedincům. Výpočetně náročnější varianta SOMA, ale s větší pravděpodobností nalezení globálního extrému.
3. **SOMA AllToAllAdaptive.** – Podobná strategie jako SOMA AllToAll. Rozdíl je v tom, že se jedinec se posunuje na nejlepší pozici už během jedné migrace k ostatním a ne až po skončení migrací ke všem ostatním.
4. **SOMA AllToOneRand.** – Tato strategie má podobnost ze SOMA AllToOne. Je zde také *leader*. Ten se ovšem nevybírá podle ohodnocení účelové funkce ale náhodně.
5. **SOMA Clusters.** – Tato modifikace je použitelná se všemi předchozími strategiemi. Její podstata leží v tom, že se jedinci rozdělí do svazků(clusters) a v těchto svazcích probíhá algoritmus SOMA. Svazky se mohou rozpadat nebo spojovat.

2.3 Diferenciální evoluce

Diferenciální evoluce má svůj původ v simulovaném žíhání, které vyvinul Kenneth V. Price. Díky tomu, že se Kenneth V. Price rozhodl spolupracovat s Dr. Rainierem Stormem na řešení složitějších problémů, začalo se jeho genetické žíhání měnit. Z binární reprezentace přešel na dekadickou a proto zaměňoval logické operace za vektorové. Tím se stal ze simulovaného žíhání nástroj vhodný pro numerickou optimalizaci.

Po aplikování výše uvedených změn Keneth Price objevil tzv. diferenciální mutace. Diferenciální mutace znamená přičtení diference dvou náhodně vybraných vektorů k vektoru třetímu. Později se přidaly metody selekce z genetického žíhání a tak vznikla první verze diferenciální evoluce

2.3.1 Parametry diferenciální evoluce

Stejně jako u všech ostatních evolučních algoritmů, je kvalita diferenciální evoluce kruciálně závislá na nastavení parametrů algoritmu. U diferenciální evoluce se konkrétně jedná o *velikost populace* označované jako ***NP***, *mutační konstantu* ***F***, *práh křížení* ***CR*** a *počet kol šlechtění populace,* označovaný jako ***Generations.*** Podrobnější vysvětlení jednotlivých parametrů a doporučení hodnot pro jejich nastavení je uvedeno v následující tabulce.

Parametr	**Doporučený rozsah**	**Druh parametru**
D	Dané problémem	Počet argumentů úč. funkce
NP	10D	Velikost populace
F	0,3 – 0,9	Mutační konstanta
CR	0,8 – 0,9	Práh křížení
Generations	Volí uživatel	Počet kol šlechtění populace

Tab. 3. Parametry diferenciální evoluce – převzato z [1]

- **Velikost populace NP.** - Tento parametr udává, kolik jedinců bude obsaženo v populaci. Nastavení tohoto parametru částečně vychází ze znalosti algoritmu. Jako doporučená hodnota se uvádí 10ti násobek dimenze problému. Nastavením tohoto parametru je možné ovlivnit nejen rychlost běhu, ale zásadně i kvalitu chodu algoritmu. Pokud zvolím nízkou hodnotu tohoto parametru, bude menší výběr při tvorbě populace. Pokud bude ovšem populace velká, zvětší se čas vynaložený na tvorbu nové populace. Jak je uvedeno v tabulce (Tab. 3.), doporučenou hodnotou je desetinásobek dimenze problému. Pro vysoce multimodální funkce je však dobré uvažovat o hodnotě 100 D. Jak je patrné, je nastavení tohoto parametru velmi variabilní. Jedinou hranicí je ovšem nastavení parametru NP na hodnotu větší než 4. Pokud se nastaví menší hodnota, přestává algoritmus pracovat.

- **Mutační konstanta F. –** Mutační konstanta je řídicím parametrem algoritmu. Hraje důležitou roli při tvorbě nových jedinců, jelikož se uplatňuje při tvorbě diferenciačního váhovaného vektoru.

- **Práh křížení CR.** – Práh křížení je dalším řídícím parametrem. Tímto parametrem ovlivňujeme, do jaké míry probíhá křížení mezi jedinci v populaci. Při nastavení parametru CR na hodnotu nula dochází k tomu, že se mutace nedostane do zkušebního jedince a díky tomu bude čistou kopií aktuálního rodiče. Z toho vyplývá, že se proces evoluce zastavil. Pokud naopak nastavíme hodnotu CR na 1, bude se zkušební jedinec vytvářet pouze ze tří náhodně vybraných rodičů, a tím se celý algoritmus degraduje na náhodné prohledávání. Proto je vhodné, aby parametr těchto hodnot nikdy nenabýval. Jeho nastavení také souvisí se separabilitou řešené funkce. Pokud je funkce separabilní, pak by se práh křížení měl volit jako hodnota blízká 0 a naopak, pokud jde o funkci, která není separabilní, měl by uživatel volit hodnotu prahu křížení blízkou 1.

- **Parametr Generations.** – Udává počet evolučních cyklů, tedy počet generací, během kterých se populace vyvíjí. Tento parametr slouží jako ukončovací. Po proběhnutí všech generací celý běh algoritmu skončí.

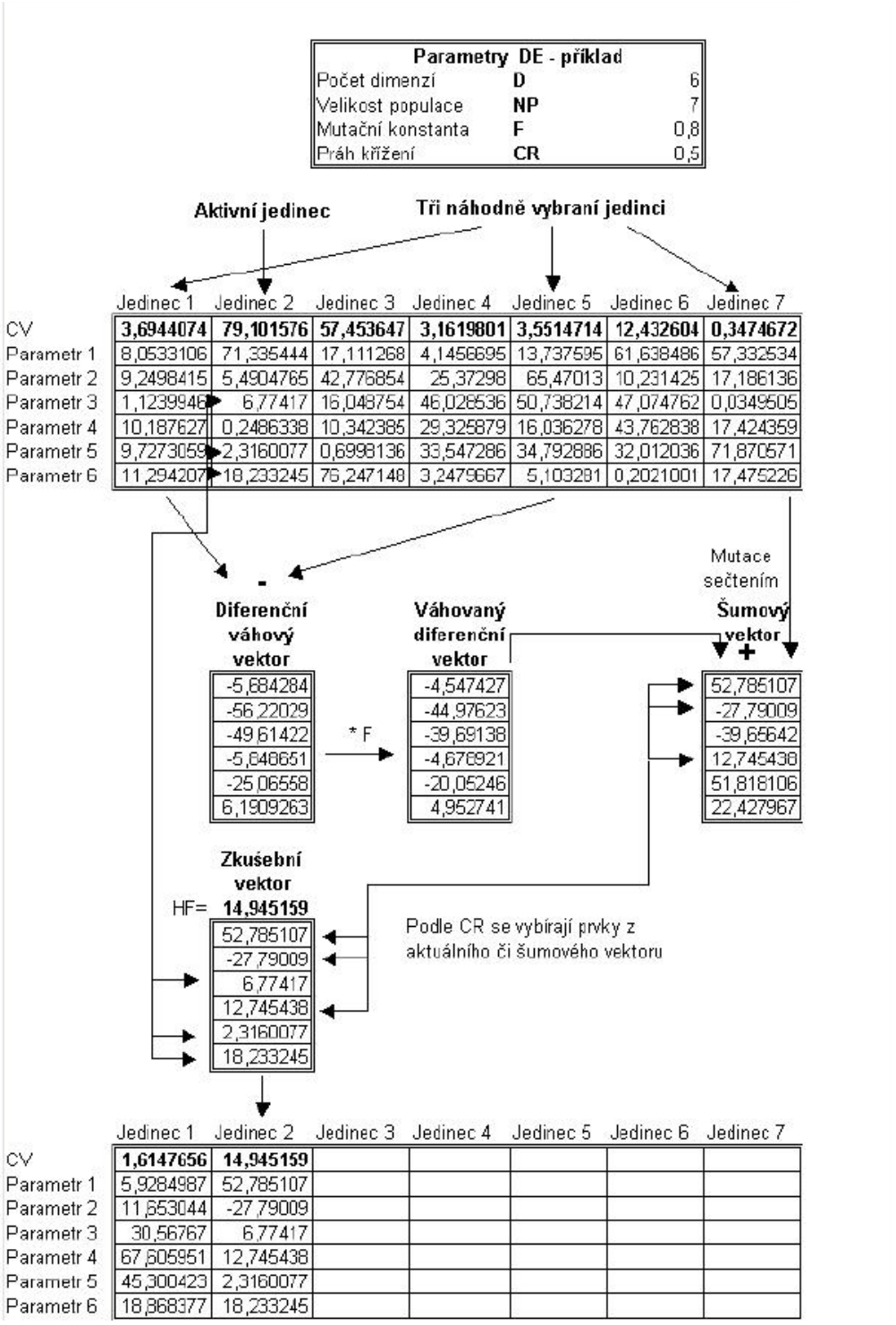
2.3.2 Fáze algoritmu diferenciální evoluce

Celý běh algoritmu [10] je možné brát jako posloupnost generací, jejichž cílem je vyšlechtit co nejlepší populaci z hlediska účelové funkce.

1. **Definování parametrů.** – Před započetím běhu celého algoritmu se provede nastavení parametrů popsaných výše. Nadefinují se hodnoty F, CR, NP a Generations. Nezbytné je také definování prototypu jedince. Označuje se jako tzv. Specimen, který udává, v jakém oboru čísel se budou jedinci použití pro běh algoritmu pohybovat.
2. **Vytvoření populace.** – Dalším nezbytným krokem je také tvorba populace. Ta se vytváří podle prototypového jedince – Specimena. Můžeme tedy říct, že populaci tvoří množina vektorů.
3. **Cyklus generací.** – V jedné generaci se vybírá jeden jedinec za druhým až do konce populace. Následuje proces generací. Pro každého jedince se provede evoluční cyklus.
4. **Evoluční cyklus.** – Tento cyklus sestává z několika činností. Zjednodušeně řečeno je možné říct, že pro každého jedince proběhne mutace a křížení. Výsledný jedinec

se porovnává s hodnotou účelové funkce a do nové populace se přebírají jen jedinci s lepšími vlastnostmi. Proces mutace je jako u ostatních evolučních algoritmů tím nejdůležitějším. U diferenciální evoluce se pro mutaci využívá 4 rodičů. Pro každého jedince se náhodně vyberou tři různí jedinci z populace. Z nich se vytvoří tzv. šumový vektor, který je mutací kombinace výše zmíněných tří rodičů. Mutace je provedena tak, že se první dva náhodně vybrané vektory – rodiče od sebe odečtou. Vznikne diferenční vektor. Ten se vynásobí mutační konstantou ***F*.** Vektor, který dostaneme jako výsledek předešlé operace, se přičte ke zbývajícímu, třetímu rodiči. Po mutaci následuje proces křížení. To je na rozdíl od dalších evolučních algoritmů specifikum diferenciální evoluce. Proces křížení lze tedy popsat jako kombinace nepoužitého čtvrtého jedince a šumového vektoru. Tím se vytvoří nový jedinec populace. Nazývá se zkušební jedinec – vektor. Zde se uplatňuje práh křížení ***CR*.** Porovnávají se odpovídající parametry čtvrtého jedince a šumového vektoru tak, že pomocí srovnání náhodně generovaného čísla a hodnoty v CR se přebírá parametr buď z čtvrtého jedince, nebo z šumového vektoru.

5. **Test na splnění ukončovacích parametrů.** – Po proběhnutí celé jedné generace se testuje, jestli se nenaplnily ukončovací parametry. Jelikož je diferenciální evoluce ukončena na základě parametru Generations, tak se testuje, zda proběhl uživatelem nastavený počet generací.



Obr. 14. Jeden evoluční cyklus diferenciální evoluce, převzato z [1].

II. PRAKTICKÁ ČÁST

3 SROVNÁNÍ KVALITY UČÍCÍHO PROCESU NEURONOVÝCH SÍTÍ POMOCÍ KLASICKÝCH METOD A EVOLUČNÍCH ALGORITMŮ

V praktické části diplomové práce jsem se zabýval srovnáním kvality procesu učení neuronových sítí z hlediska klasifikace. Pro testování jsem hledal vhodná data, na kterých bych mohl provést zamýšlené srovnání. Využil jsem souboru dat od L. Prechelta z Fakulty Informatiky University v Karlsruhe, dostupného na této adrese: http://digbib.ubka.uni-karlsruhe.de/volltexte/39794. Tento soubor dat obsahuje 15 různý problémů z 12 různých oblastí. Každý soubor dat pro jednotlivý problém obsahuje reálná data. Ta jsou připravena ve formátu, který lze přímo využít pro učení neuronových sítí. Součástí celého souboru dat je i dokument obsahující výsledky testů, které byly provedeny přímo na Universitě v Karlsruhe. Jsou v něm obsaženy také informace o tom, jak testování probíhalo a co se vyhodnocovalo. Záměrem autora tohoto souboru dat bylo poskytnout zájemcům, kteří by chtěli testovat svoje sítě platformu, na které by to mohli provést.

V této práci jsem ovšem srovnávání s výsledky uvedenými ve zmíněném dokumentu neprovedl. Využil jsem toolboxu Neural Networks v programu Matematica 5.2 pro vytvoření vlastních referenčních výsledků. Ze zmíněných 15 problémů jsem si zvolil 4, a to problém pojmenovaný jako Cancer, Card, Diabetes a Horse. Každý problém obsahuje tři různě permutované soubory dat pro testování. První z nich jsem použil pro učící proces a na dalších dvou jsem provedl testování. Následně jsem vyhodnocoval chybu, s kterou byla naučená síť schopná klasifikovat předložené testovací soubory dat.

Pro testování v toolboxu Neural Networks jsem zvolil Feedforward Network, tedy vícevrstvou síť s učícím algoritmem Back propagation. Pro testování efektivity učení s využitím evolučních algoritmů, konkrétně SOMA a diferenciální evoluce, jsem použil síť s jednou skrytou vrstvou.

3.1 Použité neuronové sítě v toolboxu Neural

Pro nalezení výsledků, které bych mohl použít jako referenci pro srovnání procesu učení za pomoci evolučních algoritmů, jsem využil v toolboxu Neural Network v prostředí Mathematica 5.2 síť Feed forward network, tedy dopřednou síť s jednou skrytou vrstvou. Počet neuronů ve skryté vrstvě jsem určil podle vztahu(19):

$$N_{skrytá} = \sqrt{N_{vstup} * N_{vystup}} \tag{19}$$

kde

- $N_{skrytá}$ – počet neuronů ve skryté vrstvě (zaokrouhleno na celé číslo)
- N_{vstup} – počet neuronů ve vstupní vrstvě
- N_{vystup} – počet neuronů ve výstupní vrstvě

Jako aktivační funkci ve výstupní vrstvě jsou zvolil lineární funkci. Učení probíhá pomocí algoritmu Back propagation. Pro jeho běh jsem zvolil 50 iterací.

3.2 Zvolené evoluční algoritmy

Pro tuto práci jsem použil algoritmus SOMA a diferenciální evoluci. Pomocí nich se hledá optimální nastavení vah neuronové sítě. Bylo tedy nutné vytvořit účelovou funkci která by odpovídala soustavě optimalizovaných vah.

3.2.1 SOMA

Byla použita strategie AllToOne algoritmu SOMA. Pro tento algoritmus byly použity tyto konkrétní hodnoty parametrů:

	Problém			
Parametr	**Cancer**	**Card**	**Diabetes**	**Horse**
PartLength	3.0	3.0	3.0	3.0
Step	0.11	0.11	0.11	0.11
PRT	0.3	0.3	0.3	0.3
D	50	542	46	809
PopSize	60	60	60	60
Migrace	10	12	12	12
MinDiv	0.1	0.1	0.1	0.1

Tab. 4. Použité hodnoty parametrů algoritmu SOMA při testování

3.2.2 Diferenciální evoluce

Použitá verze diferenciální evoluce se označuje jako Rand1Bin. Pro testování byly použity tyto hodnoty parametrů:

	Problém			
Parametr	**Cancer**	**Card**	**Diabetes**	**Horse**
D	50	542	46	809
NP	60	60	60	60
F	0.8	0.8	0.8	0.8
CR	0.8	0.8	0.8	0.8
Generations	250	300	300	250

Tab. 5. Použité hodnoty parametrů diferenciální evoluce při testování

3.3 Tvorba účelové funkce

Pro korektní činnost evolučních algoritmů bylo nutné z vah neuronové sítě vytvořit účelovou funkci. Následně by SOMA a diferenciální evoluce hledaly optimální nastavení vah tak, aby nalezly jedince s nejlepší hodnotou účelové funkce. To znamená jedince s minimální hodnotou, protože hodnota účelové funkce zde představuje chybu sítě. Pro vytvoření této účelové funkce byl použit tento kód (Obr.15):

```
CostFunction[vahy_] :=
 Module[{}, bestjedinec = vahy; w2 = vyplodvahy1[vahy, pocetvstupu, skryte, pocetvystupu];
  neuronkavystup = DejHodnotyVystup[vstupy, w2, neuronfce];
  cv = Total[Table[Total[Abs[neuronkavystup[[i]] - vystupy[[i]]]], {i, Length[vystupy]}]];
  jedinecAbort = {cv, vahy}; pocitadlo++; If[cv < chyba, Abort[]];
  Return[cv]]
```

Obr. 15. Konstrukce účelové funkce

3.4 Testované problémy

3.4.1 Soubor dat Cancer

Tento problém se zabývá testováním diagnostikování rakoviny prsu. Původně byl vytvořen Dr. Williamem H. Wolberem z University of Wisconsin Hospital v Madisonu. Tento testovací soubor dat obsahuje 9 vstupů, 2 výstupy a celkově 699 vzorků. Všechny vstupy jsou ve spojité podobě. Na základě informací mikroskopického vyšetření buněk se testuje, zda je předložený vzorek benigní nebo maligní.

3.4.2 Soubor dat Card

Další testovací soubor odráží problém přijmutí nebo odmítnutí kreditní karty bankou nebo podobnou institucí. Vstupní hodnoty odráží reálná data z existujících kreditních karet. Výstupní data vypovídají o tom, zda banka kartu příjme nebo ne. Jednotlivá vstupní data nejsou popsána z důvodu utajení. Celý soubor dat obsahuje 51 vstupů, 2 výstupy a celkově 690 vzorků. Vstupní parametry obsahují mix spojitých parametrů s velkou hodnotou a malou hodnotou. Soubor je také zatížen chybějícími vstupními hodnotami u 5% vzorků. Tyto chybějící vzorky byly nahrazeny konstantní hodnotou při vytváření testovacího souboru.

3.4.3 Soubor dat Diabetes

Soubor dat Diabetes je založen na výzkumu diagnózy indiánů kmene Pima. Na základě jejich osobních dat (věk, počet těhotenství) a výsledků medicínských testů (např. krevní tlak, BMI, výsledek krevního testu atd.) došlo k rozhodnutí, zda je diagnostikován diabetes nebo je výsledek negativní. Testovací soubor obsahuje 8 vstupů, 2 výstupy a 768 vzorků. Všechny vstupní hodnoty jsou spojité.

3.4.4 Soubor dat Horse

Tentokrát se soubor dat zabývá odhadem vývoje zdravotního stavu koní majících koliku. Odhaduje se, zda kůň přežije, umře, nebo bude muset podstoupit eutanázii. Zmíněný soubor dat obsahuje 54 vstupů, 3 výstupy a celkem 364 vzorků. Tento soubor je zatížen velkým počtem chybějících hodnot. Ty byly nahrazeny implicitně při tvorbě testovacího souboru.

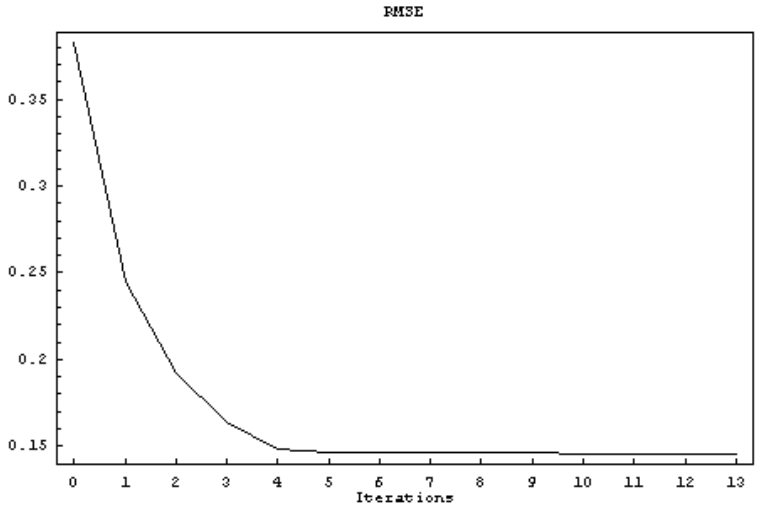
3.5 Výsledky učení

Jako výsledek učení jsem uvažoval chybu, respektive úspěšnost vybavovací fáze neuronové sítě pro jednotlivé třídy, a celkovou chybu. Testovací množiny představovaly permutované trénovací množiny.

3.5.1 Soubor dat Cancer

1. **Testování neuronové sítě naučené v toolboxu Neural Networks v prostředí Mathematika 5.2**

Znázornění vývoje střední kvadratické chyby na počtu iterací:



Obr. 16. Vývoj RMSE během iterací – problém Cancer

Nejlepší dosažené výsledky úspěšnosti neuronové sítě při klasifikaci:

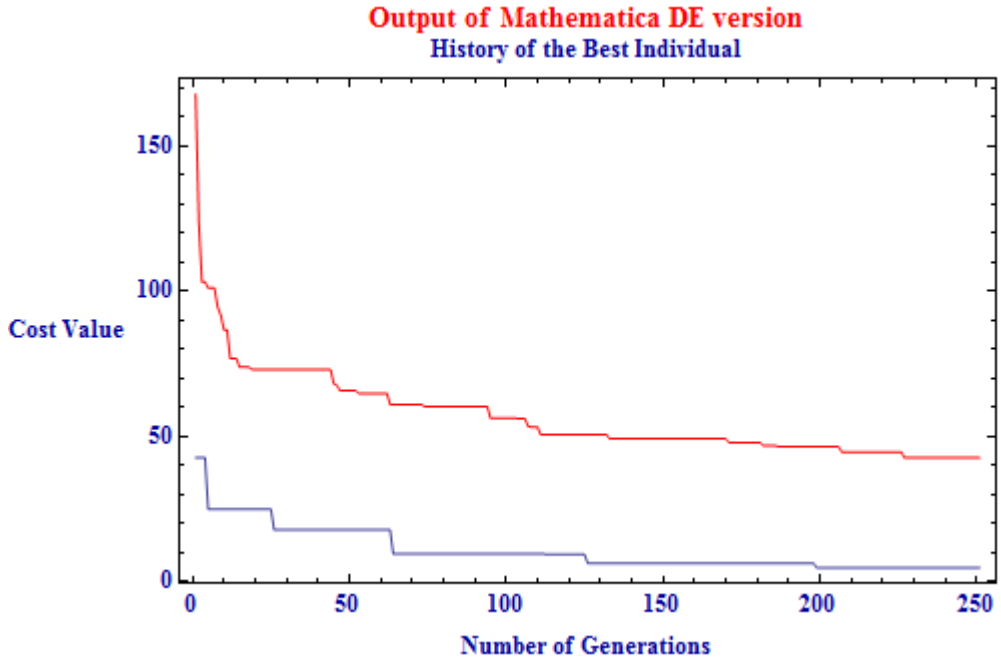
	Třída 1 [%]	Třída 2 [%]	Celková [%]
Cancer2	100	100	100
Cancer3	100	100	100

Tab. 6. Procentuální úspěšnost sítě pří klasifikaci Cancer – učení pomocí toolboxu Neural Network

2. Testování neuronové sítě naučené pomocí diferenciální evoluce

Bylo provedeno 5 experimentů. Vybral jsem ten s nejlepšími výsledky.

Zobrazení vývoje nejlepšího (modrý průběh) a nejhoršího (červený průběh) jedince v závislosti na počtu generací pro experiment s nejlepšími dosaženými výsledky (Obr. 17.):



Obr. 17. Historie nejlepšího a nejhoršího jedince – Cancer (DE)

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

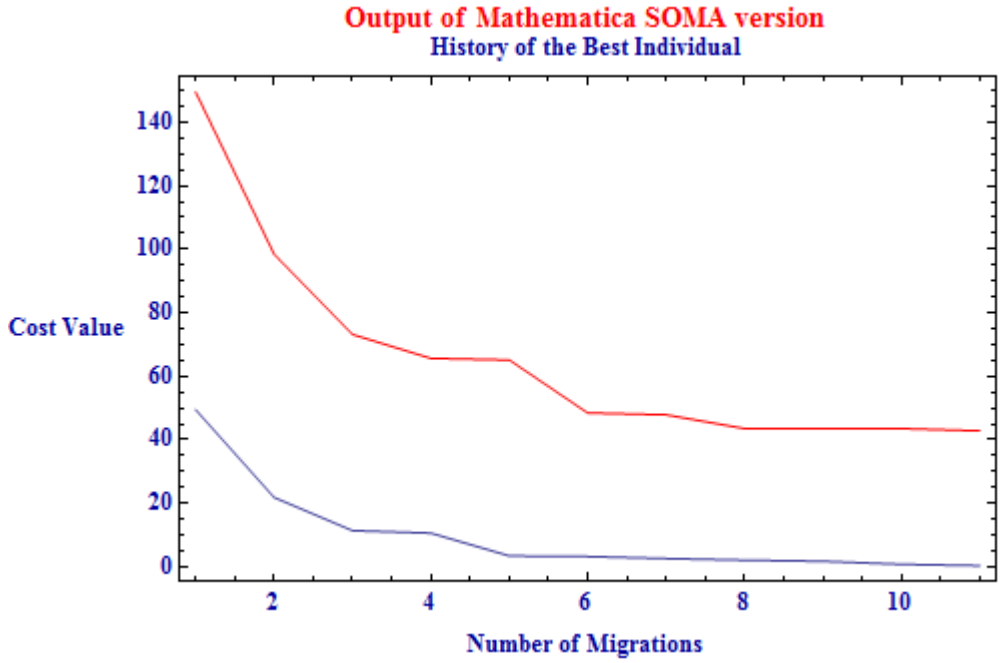
	Třída 1 [%]	Třída 2 [%]	Celková [%]
Cancer2	97.93	96.94	97.28
Cancer3	97.93	96.93	97.28

Tab. 7. Procentuální úspěšnost sítě při klasifikaci Cancer – učení pomocí DE

3. Testování neuronové sítě naučené pomocí algoritmu SOMA

Bylo provedeno 5 experimentů. Vybral jsem ten s nejlepšími výsledky.

Zobrazení vývoje nejlepšího (modrý průběh) a nejhoršího (červený průběh) jedince v závislosti na počtu generací pro experiment s nejlepšími dosaženými výsledky (Obr. 18.):



Obr. 18. Historie nejlepšího a nejhoršího jedince – Cancer (SOMA)

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

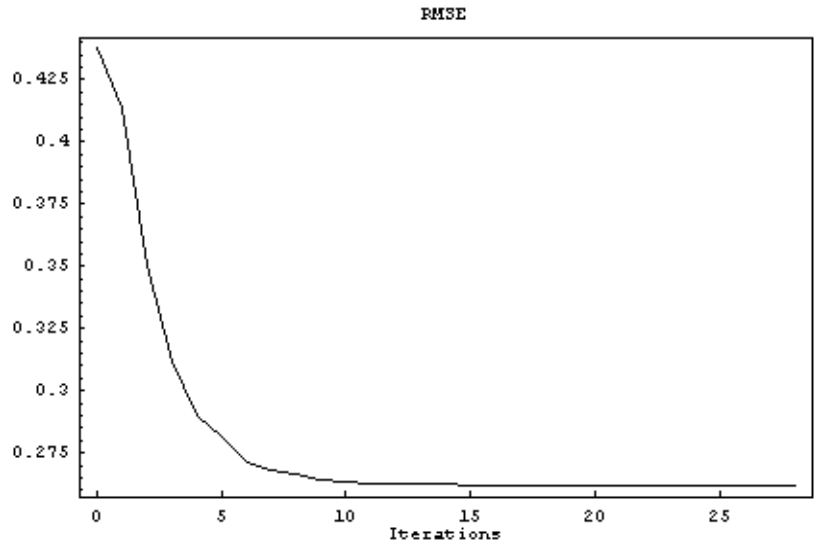
	Třída 1 [%]	Třída 2 [%]	Celková [%]
Cancer2	97.51	98.91	98.42
Cancer3	97.51	98.91	98.42

Tab. 8. Procentuální úspěšnost sítě při klasifikaci Cancer – učení pomocí SOMA

3.5.2 Soubor dat Card

1. **Testování neuronové sítě naučené v toolboxu Neural Networks v prostředí Mathematika 5.2**

Znázornění vývoje střední kvadratické chyby na počtu iterací:



Obr. 19. Vývoj RMSE během iterací – problém Card

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

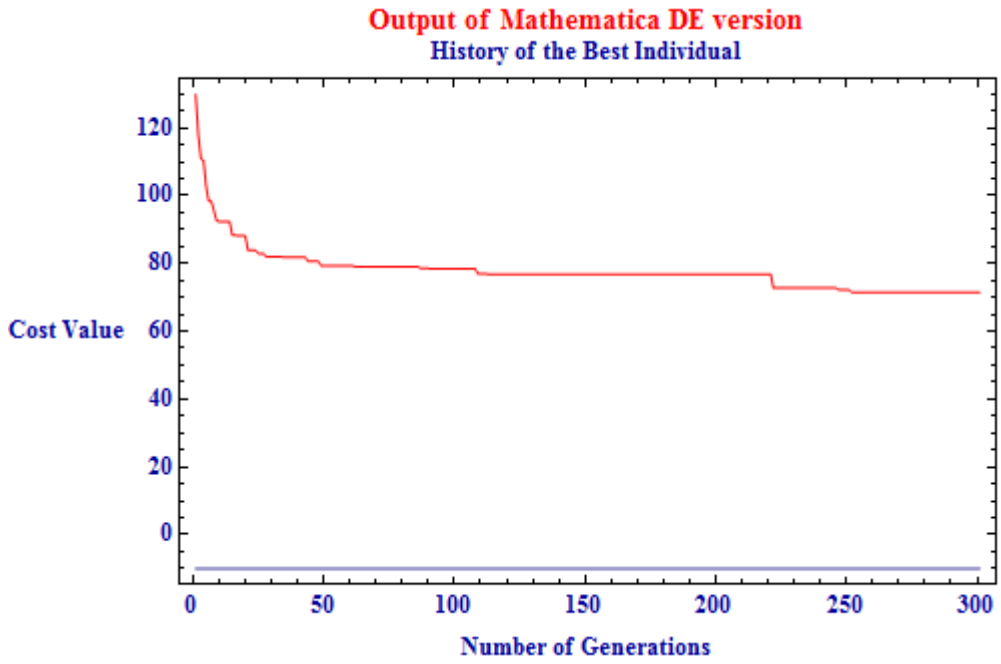
	Třída 1 [%]	Třída 2 [%]	Celková [%]
Card2	100	100	100
Card3	100	100	100

Tab. 9. Procentuální úspěšnost sítě při klasifikaci Card – učení pomocí toolboxu Neural Network

2. **Testování neuronové sítě naučené pomocí diferenciální evoluce**

Bylo provedeno 5 experimentů. Vybral jsem ten s nejlepšími výsledky.

Zobrazení vývoje nejlepšího (modrý průběh) a nejhoršího (červený průběh) jedince v závislosti na počtu generací pro experiment s nejlepšími dosaženými výsledky (Obr. 20.):



Obr. 20. Historie nejlepšího a nejhoršího jedince – Card (DE)

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

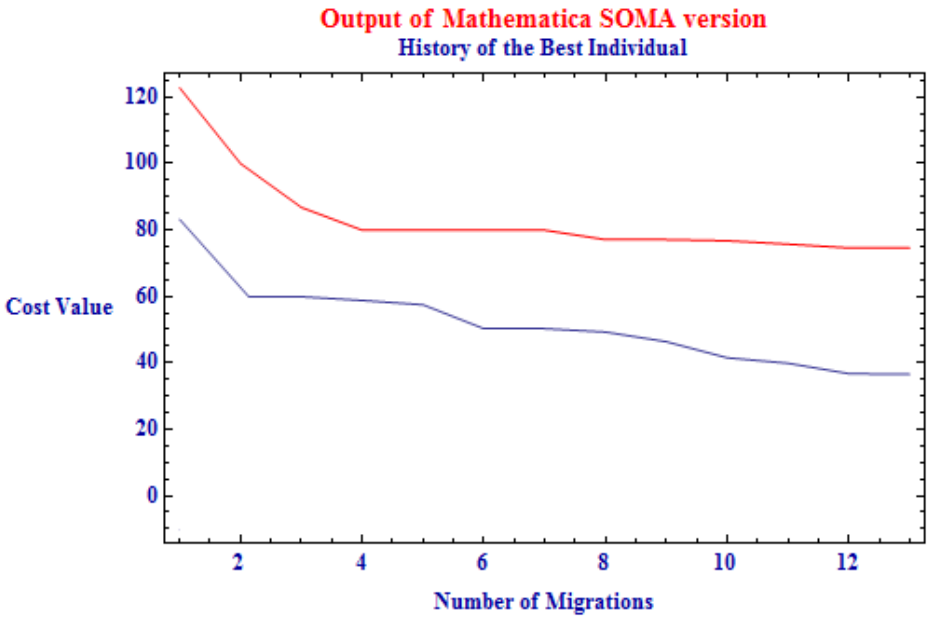
	Třída 1 [%]	Třída 2 [%]	Celková [%]
Card2	96.44	67.44	83.00
Card3	96.61	78.05	89.00

Tab. 10. Procentuální úspěšnost sítě při klasifikaci Card – učení pomocí DE

3. Testování neuronové sítě naučené pomocí algoritmu SOMA

Bylo provedeno 5 experimentů. Vybral jsem ten s nejlepšími výsledky.

Zobrazení vývoje nejlepšího (modrý průběh) a nejhoršího (červený průběh) jedince v závislosti na počtu generací pro experiment s nejlepšími dosaženými výsledky (Obr. 21.):



Obr. 21. Historie nejlepšího a nejhoršího jedince – Card (SOMA)

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

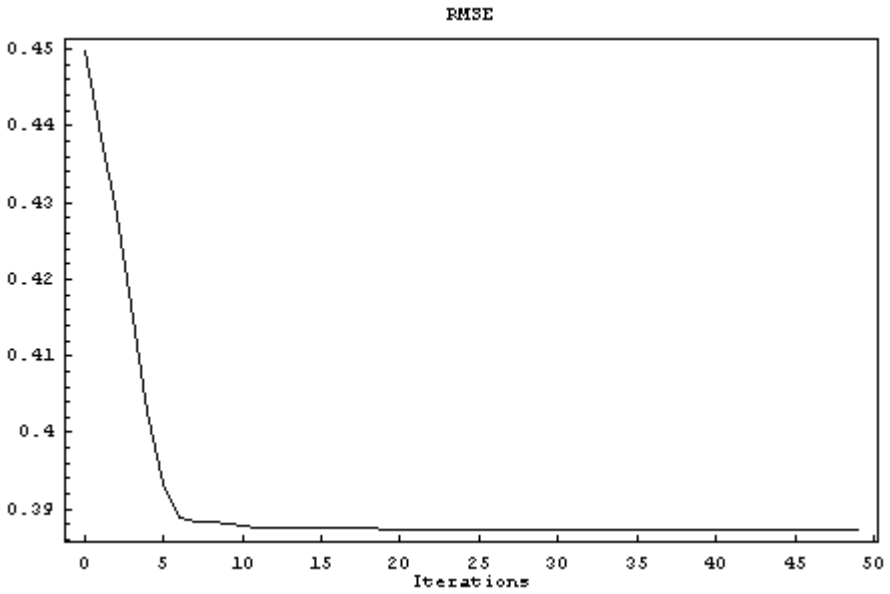
	Třída 1 [%]	Třída 2 [%]	Celková [%]
Card2	89.30	89.57	98.40
Card3	89.30	89.58	98.42

Tab. 11. Procentuální úspěšnost sítě při klasifikaci Card – učení pomocí SOMA

3.5.3 Soubor dat Diabetes

1. **Testování neuronové sítě naučené v toolboxu Neural Networks v prostředí Mathematika 5.2**

Znázornění vývoje střední kvadratické chyby na počtu iterací:



Obr. 22. Vývoj RMSE během iterací – problém Diabetes

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

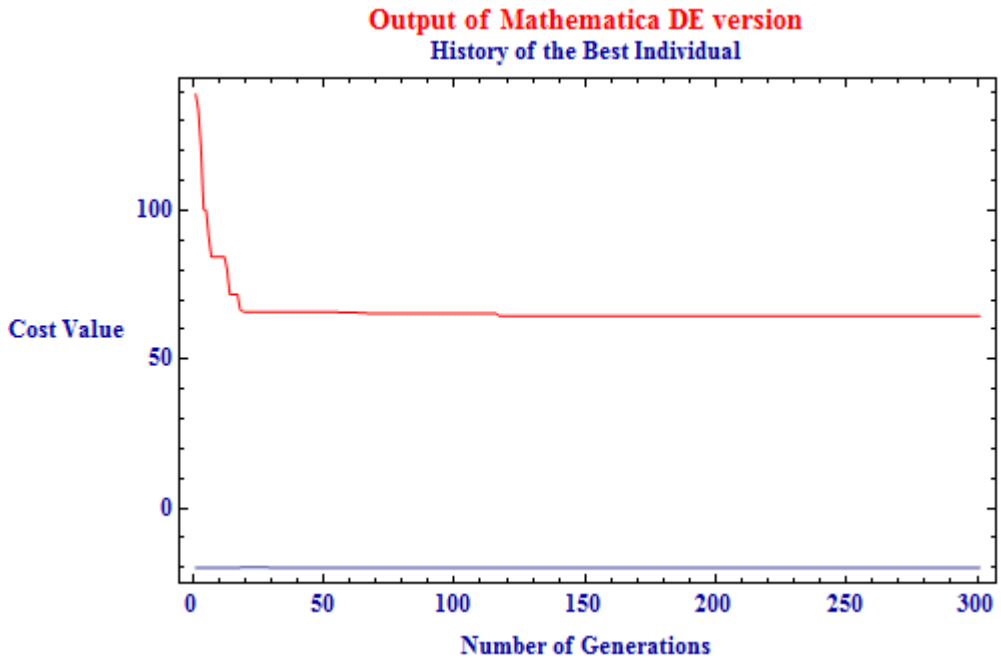
	Třída 1 [%]	Třída 2 [%]	Celková [%]
Diabetes2	100	100	100
Diabetes3	100	100	100

Tab. 12. Procentuální úspěšnost sítě při klasifikaci Diabetes – učení pomocí toolboxu Neural Network

2. Testování neuronové sítě naučené v pomocí diferenciální evoluce

Bylo provedeno 5 experimentů. Vybral jsem ten s nejlepšími výsledky.

Zobrazení vývoje nejlepšího (modrý průběh) a nejhoršího jedince (červený průběh) v závislosti na počtu generací pro experiment s nejlepšími dosaženými výsledky (Obr. 23.):



Obr. 23. Historie nejlepšího a nejhoršího jedince – Diabetes (DE)

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

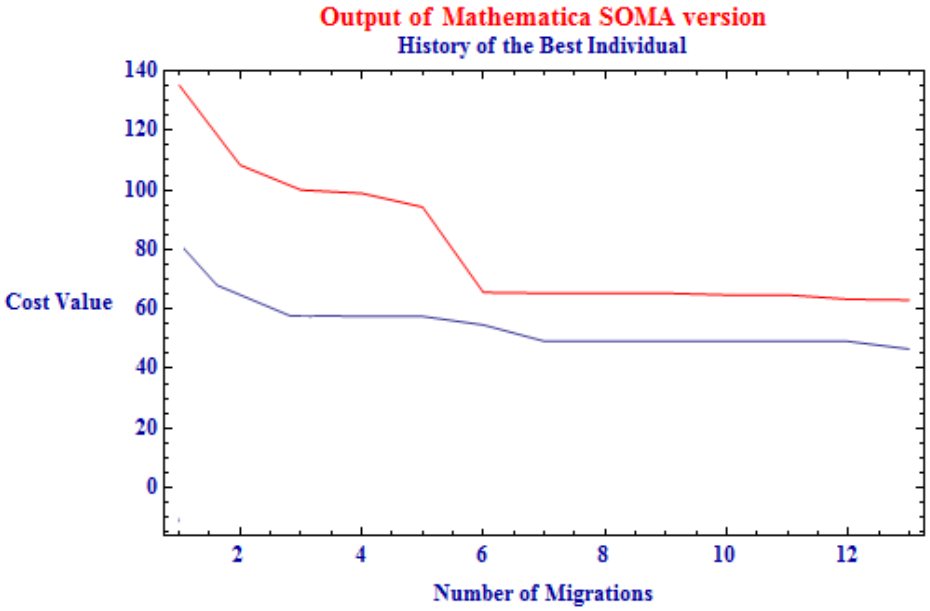
	Třída 1 [%]	Třída 2 [%]	Celková [%]
Diabetes2	93.60	78.73	88.41
Diabetes3	93.60	78.73	88.41

Tab. 13. Procentuální úspěšnost sítě při klasifikaci Diabetes – učení pomocí DE

3. Testování neuronové sítě naučené v pomocí algoritmu SOMA

Bylo provedeno 5 experimentů. Vybral jsem ten s nejlepšími výsledky.

Zobrazení vývoje nejlepšího (modrý průběh) a nejhoršího (červený průběh) jedince v závislosti na počtu generací pro experiment s nejlepšími dosaženými výsledky (Obr. 24.):



Obr. 24. Historie nejlepšího a nejhoršího jedince – Diabetes (SOMA)

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

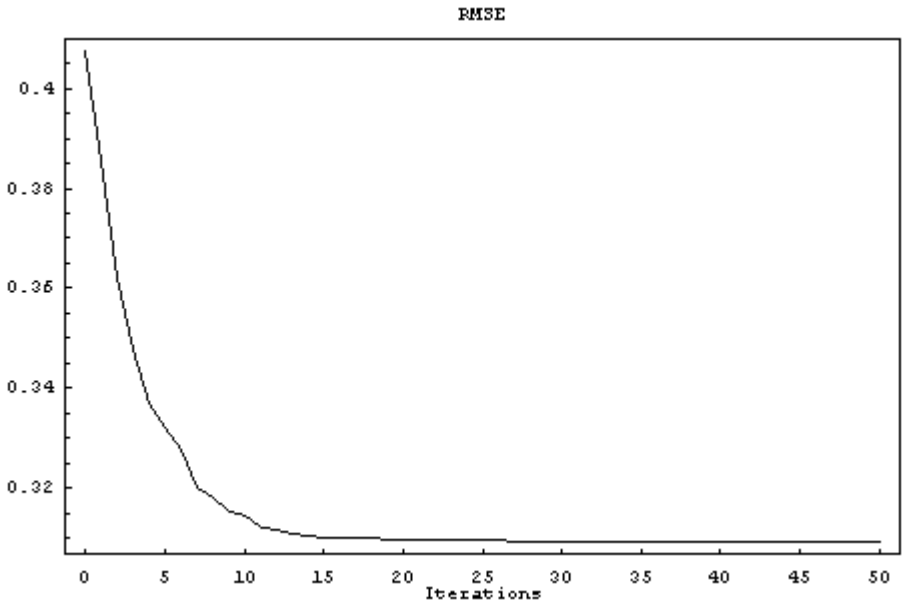
	Třída 1 [%]	Třída 2 [%]	Celková [%]
Diabetes2	94.60	88.43	92.44
Diabetes3	94.60	88.43	92.44

Tab. 14. Procentuální úspěšnost sítě při klasifikaci Diabetes – učení pomocí SOMA

3.5.4 Soubor dat Horse

1. **Testování neuronové sítě naučené v toolboxu Neural Networks v prostředí Mathematika 5.2**

Znázornění vývoje střední kvadratické chyby na počtu iterací:



Obr. 25. Vývoj RMSE během iterací – problém Horse

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

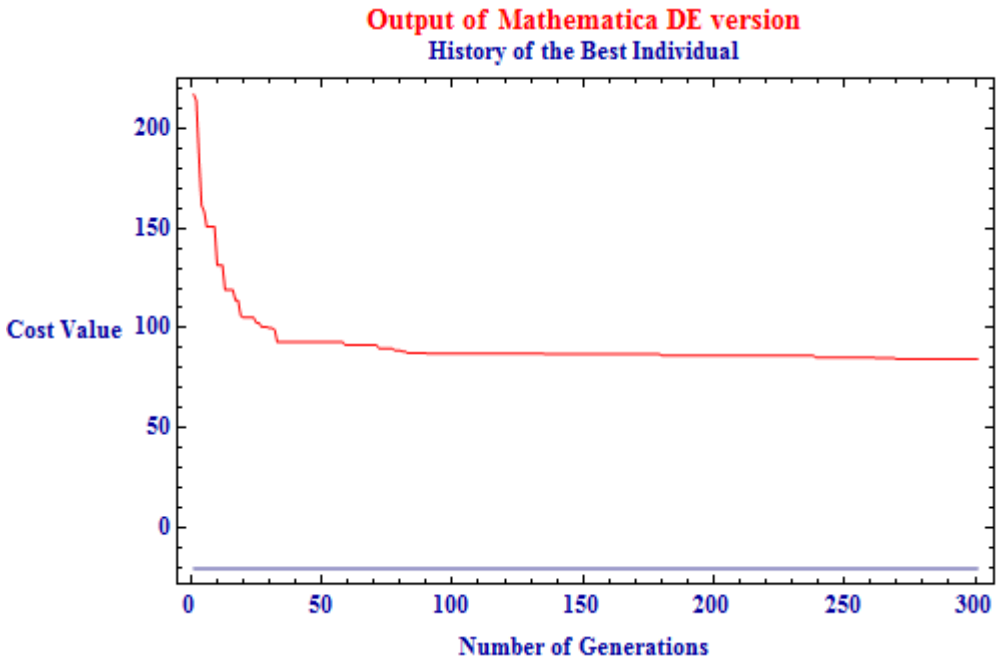
	Třída 1 [%]	Třída 2 [%]	Třída 3 [%]	Celková [%]
Horse2	76.92	88.64	96.43	91.76
Horse3	76.92	88.64	96.43	91.76

Tab. 15. Procentuální úspěšnost sítě při klasifikaci Horse – učení pomocí toolboxu Neural Network

2. Testování neuronové sítě naučené v pomocí diferenciální evoluce

Bylo provedeno 5 experimentů. Vybral jsem ten s nejlepšími výsledky

Zobrazení vývoje nejlepšího (modrý průběh) a nejhoršího (červený průběh) jedince v závislosti na počtu generací pro experiment s nejlepšími dosaženými výsledky (Obr. 26.):



Obr. 26. Historie nejlepšího a nejhoršího jedince – Horse (DE)

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

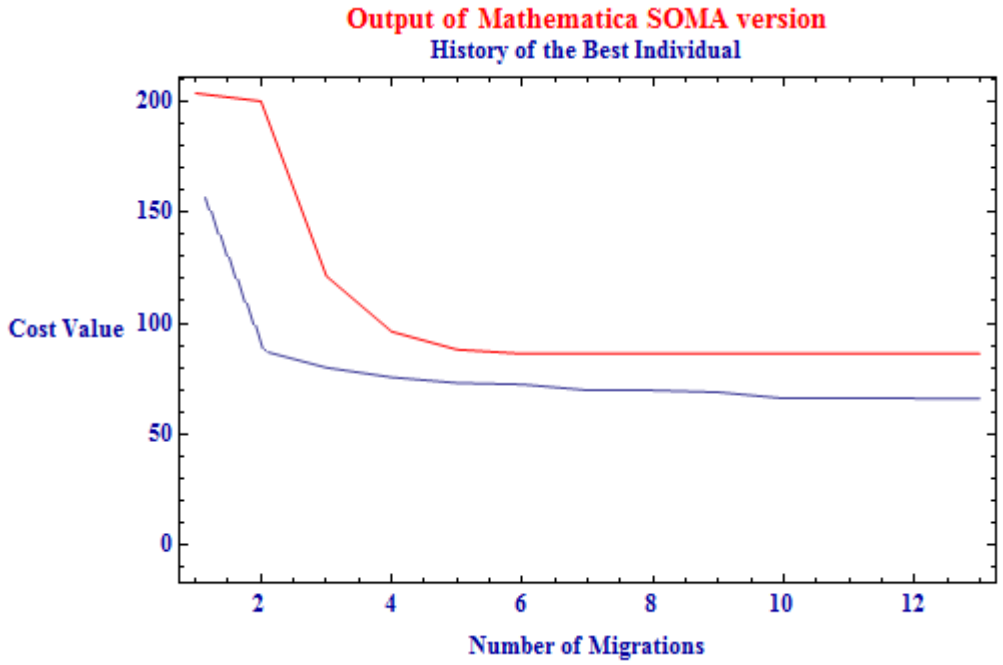
	Třída 1 [%]	Třída 2 [%]	Třída 3 [%]	Celková [%]
Horse2	76.92	67.05	85.71	79.45
Horse3	76.92	67.05	85.71	79.45

Tab. 16. Procentuální úspěšnost sítě při klasifikaci Horse – učení pomocí DE

3. Testování neuronové sítě naučené v pomocí algoritmu SOMA

Bylo provedeno 5 experimentů. Vybral jsem ten s nejlepšími výsledky

Zobrazení vývoje nejlepšího (modrý průběh) a nejhoršího (červený průběh) jedince v závislosti na počtu generací pro experiment s nejlepšími dosaženými výsledky (Obr. 27.):



Obr. 27 Historie nejlepšího a nejhoršího jedince – Horse (SOMA)

Nejlepší dosažené výsledky neuronové sítě při klasifikaci:

	Třída 1 [%]	Třída 2 [%]	Třída 3 [%]	Celková [%]
Horse2	40.38	42.05	73.21	60.99
Horse3	40.38	42.05	73.21	60.99

Tab. 17. Procentuální úspěšnost sítě při klasifikaci Horse – učení pomocí SOMA

3.6 Diskuze výsledků

3.6.1 Problém Cancer

Testovací soubor se klasifikoval do dvou tříd. Třída 1 představovala benigní nález a Třída 2 představovala maligní nález mikroskopického vyšetření buněk. Celková úspěšnost představuje celkovou chybu sítě pro všechny třídy. V kapitole **3.5.1** jsou uvedeny nejlepší dosažené výsledky klasifikace pro síť naučenou klasickou metodou (Feedforward net s Back propagation algoritmem) a pro síť naučenou pomocí diferenciální evoluce a algoritmu SOMA.

Výsledky dosažené pomocí dopředné sítě z toolbox Neural Network Mathematica 5.2 byly v každém pokusu totožné, tzn. 100% úspěšná klasifikace testovaných souborů dat. Výsledky úspěšnosti klasifikace sítě naučené pomocí evolučních algoritmů kolísaly v řádu jednotek procent. Doba běhu algoritmu pro nalezení optimální kombinace vah se u diferenciální evoluce pohybovala okolo 6 hodin. U algoritmu SOMA byla doba potřebná k nalezení vah podobná.

3.6.2 Problém Card

Testovací soubor se opět klasifikoval do dvou tříd. Třída 1 představovala odmítnutí kreditní karty a Třída 2 představovala příjmutí kreditní karty bankou nebo podobnou institucí. Celková úspěšnost představuje celkovou chybu sítě pro všechny třídy. V kapitole **3.5.2** jsou uvedeny nejlepší dosažené výsledky klasifikace pro síť naučenou klasickou metodou (Feedforward net s Back propagation algoritmem) a pro síť naučenou pomocí diferenciální evoluce a algoritmu SOMA.

Výsledky dosažené pomocí dopředné sítě z toolbox Neural Network Mathematica 5.2 byly stejně jako u předchozího testovacího problému v každém pokusu totožné, tzn. 100% úspěšná klasifikace testovaných souborů dat. Výsledky úspěšnosti klasifikace sítě naučené pomocí evolučních algoritmů kolísaly v řádu jedné desítky procent. Zřejmě byl počet agrumentů účelové funkce příliš velký na to, aby se v reálně rozumné době nalezlo optimální nastavení vah. Proto zřejmě docházelo ke kolísání výsledků. Doba běhu algoritmu pro nalezení optimální kombinace vah se u diferenciální evoluce pohybovala okolo více než 24 hodin pro jeden běh. U algoritmu SOMA byla doba potřebná k nalezení vah ještě o něco delší.

3.6.3 Problém Diabetes

Soubor se také klasifikoval do dvou tříd. Třída 1 představovala negativní diagnózu diabetes a Třída 2 pozitivní diagnózu diabetes u zkoumaných subjektů. Celková úspěšnost představuje celkovou chybu sítě pro všechny třídy. V kapitole **3.5.3** jsou uvedeny nejlepší dosažené výsledky klasifikace pro síť naučenou klasickou metodou (Feedforward net s Back propagation algoritmem) a pro síť naučenou pomocí diferenciální evoluce a algoritmu SOMA.

Výsledky dosažené pomocí dopředné sítě z toolbox Neural Network Mathematica 5.2 byly v každém pokusu totožné, tzn. 100% úspěšná klasifikace testovaných souborů dat. Výsledky úspěšnosti klasifikace sítě naučené pomocí evolučních algoritmů lišily až o 12% oproti nejlepším dosaženým výsledkům. Opět je zřejmé, že velký počet argumentů účelové funkce působil na schopnost nalézt optimální řešení. Doba běhu algoritmu pro nalezení optimální kombinace vah se u diferenciální evoluce pohybovala okolo 24 hodin. Stejně tak doba běhu algoritmu SOMA byla 24 hodin pro jeden běh.

3.6.4 Problém Horse

Poslední testovací soubor se klasifikoval do 3 tříd. Třída 1 představovala koně, kteří museli být utraceni, Třída 2 představovala koně, kteří uhynuli a Třída 3 představovala koně, kteří přežili. Celková úspěšnost představuje celkovou chybu sítě pro všechny třídy. V kapitole **3.5.4** jsou uvedeny nejlepší dosažené výsledky klasifikace pro síť naučenou klasickou metodou (Feedforward net s Back propagation algoritmem) a pro síť naučenou pomocí diferenciální evoluce a algoritmu SOMA.

Výsledky úspěšnosti klasifikace dosažené pomocí dopředné sítě z toolbox Neural Network Mathematica 5.2 se lišily v pokusech maximálně o 1% - 2%. Výsledky úspěšnosti klasifikace sítě naučené pomocí evolučních algoritmů se od nejlepšího dosaženého výsledku lišily o více než 14% procent. Tedy stejně jako v předchozích případech, i zde byl pozorovatelný vliv velkého počtu argumentů účelové funkce na nalezení optimálního řešení. Doba běhu algoritmu pro nalezení optimální kombinace vah se u diferenciální pohybovala okolo 48 hodin. U algoritmu SOMA byla doba potřebná k nalezení vah velmi podobná.

ZÁVĚR

Cílem této diplomové práce bylo porovnat, jaký vliv budou mít evoluční algoritmy na kvalitu učícího procesu neuronových sítí. Byly provedeny simulace jak pomocí toolboxu Neural Network (klasická metoda učení) v prostředí Mathematica, tak i v prostředí Mathematica 7.0 (evoluční algoritmy v učícím procesu). S použitými testovacími problémy si Feedforward Network (vícevrstvá dopředná síť s algoritmem Back propagation) z toolboxu Neural Network poradila bez větších problémů. U souborů dat Cancer, Card a Diabetes pracovala síť po naučení se 100% úspěšností a u posledního souboru dat Horse se pohybovala celková úspěšnost kolem 91%. Naproti tomu při použití evolučních algoritmů tomu tak nebylo. Algoritmus SOMA dosahoval lepších výsledků než defirenciální evoluce. Úspěšnost sítě naučené pomocí algoritmu SOMA se pohybovala mezi 90 – 98% v případě souborů dat Cancer, Card a Diabetes. Diferenciální evoluce dosahovala mírně horších výsledků. Pouze u posledního testovacího souboru Horse si vedla diferenciální evoluce o poznání lépe než SOMA. Pracovala s úspěšností vyšší asi o 20%.

Z uvedených výsledků pro jednotlivé testy je vidět, že srovnání pro evoluční algoritmy nedopadlo nejlépe. Tento pozorovaný fakt ještě ovšem nemusí znamenat, že evoluční algoritmy nejsou pro tento typ problému vhodné. Navrhnutá konstrukce účelové funkce, která je pro činnost evolučních algoritmů zásadní, zřejmě není nejvhodnější protože při současném tvaru se u některých problému dostávám až na 800 vah pro optimalizací u diferenciální evoluce. To je časově velmi náročné a také neefektivní. Celé testování probíhalo v prostředí Mathematica 5.2, resp. Mathematica 7.0, což značně limitovalo provádění testů, jelikož se ukázaly jako časově velmi náročné. To se týkalo testování evolučních algoritmů. Ale to byl pouze logický důsledek toho, jak byla formulována účelová funkce. Naproti tomu v toolboxu Neural Network v prostředí Mathematica 5.2 probíhalo testování velmi rychle. To je způsobeno tím, že neuronová síť přenastavuje svoje váhy při každém průchodu vektoru z trénovací množiny.

Diplomová práce by mohla sloužit jako podklad pro další zájemce o tuto problematiku, kteří by mohli vyzkoušet různé konstrukce účelových funkcí. Například nalezení nějakého vhodného rozdělení účelové funkce na dílčí části, které by se zpracovávaly samostatně. Dále by bylo možné experimentovat se strukturou neuronových sítí použitých pro testování.

ZÁVĚR V ANGLIČTINĚ

The aim of this diploma thesis was to compare effect on quality of training process neural networks using evolutionary algorithms. There were performed tests in Mathematica 5.2 enviroment(clasic methods of training) and in Mathematica 7.0 enviroment(training via evolutionary algorithms).Feed forward net(multilayer feedforward net with Back propagation algorithm) had very good results. For test problems Cancer, Card and Diabetes the net worked with 100% efficacy after the end of training process, for the last test problem called Horse was the total efficacy around 91%. On the other hand while I used evolutionary algorithm the results were not so good.Algorithm SOMA was reaching better results than differential evolution. The efficacy of neural network trained via SOMA algorithm was between 90 - 98% for test problems Cancer, Card and Diabetes.Differential evolution was reaching a similar but worse results. Only for the last test problem neural network trained via differential evolution reached better efficacy than neural network trained via SOMA. Network trained via SOMA was reaching results about near 20% worser.

From results written above for test problems is clear, that comparison for evolutionary algorithms did not go very well. But this observed fact does not mean that evolutionary algorithms are not suitable for this type of problem. Designed construction of cost function, which is critical for properly work of evolutionary algorithms, is probably not the most suitable. Because with current form of cost function there is up to 800 parametres for optimalization for Differencial evolution. This is very time consuming and very ineffective.All the tests were performed in Mathematica 5.2, or Mathematica 7 enviroment which was also very limiting for performing the tests, because the time needed for one test was huge. This was mainly the question of testing nets trained via evolutionary algorithms. But that was only the logical conclusion of formulation of cost function. Testing performed in toolbox Neural Network was very quick because weights of neural networks were changed for every pass of vector from training set through the network.

This diploma thesis could be used as a base for other people who are interest in this problematic who could try to find some different construction of cost function. Possibly by splitting cost function for smaller instances on which could evolutionary algoriths run separately. Or they could do some experiments with architecture of used neural networks.

SEZNAM POUŽITÉ LITERATURY

[1] ZELINKA, I., OPLATKOVÁ, Z., OŠMERA, P., ŠEDA, M., VČELAŘ, F.: Evoluční výpočetní techniky – principy a aplikace. BEN – technická literatura, Praha, 2008, ISBN 80-7300-218-3.

[2] ZELINKA, I.: Umělá inteligence I., VUT Brno, ISBN 80-214-1163,1998.

[3] BOSE, N.K., LIANG, P.: Neural Network Fundamentals with Graphs, Algorithms, and Applications, McGraw-Hill Series in Electrical and Computer Engineering. ISBN 80-7300-218-3, 1996.

[4] FREEMAN, J.A.: Simulating Neural Networks with Mathematica. Adison – Weslez Publishing Company, 1994, ISBN 0-201-56629-X.

[5] OPLATKOVÁ, Z.: Metaevolution – Synthesis of Optimization Algorithms by means of Symbolic Regression and Evolutionary Algorithms, Lambert-Publishing, 2009, ISBN 978-8383-1808-0.

[6] DAVIS, L.: Handbook of Genetic Algorithms, International Thomson Computer Press, 1996, ISBN 1850328250.

[7] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J.: Umělá inteligence, Academia, 1993, ISBN 80-200-0469-3.

[8] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J.: Umělá inteligence 4, Academia, 2003, ISBN 80-200-1044-0.

[9] ŠNOREK, M.: Neuronové sítě a neuropočítače, ČVUT Praha, 2004, ISBN 80-01-02549-7.

[10] ZELINKA, I.: Umělá inteligence v problémech globální optimalizace, BEN – technická literatura, Praha, 2002, ISBN 80-7300-069-5.

[11] ŠÍMA, J., NERUDA, R.: Teoretické otázky neuronových sítí. Matfyzpress, Praha, 1996.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SOMA	Samo – Organizující se Migrační Algoritmus
DE	Diferenciální evoluce
Back propagation	Algoritmus zpětného šíření chyby
RMSE	Střední kvadratická chyba(Root Mean Square Error)
RBF	Funkce radiálních bází(Radial basis function)
XOR	Výhradní součet(exclusive OR)
CD	Záznamové médium CR – ROM
EA	Evoluční algoritmus

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: ADRESÁŘOVÁ STRUKTURA PŘILOŽENÉHO CD

\diplomova práce

- Obsahuje diplomovou práci v el. podobě.

\zdrojeve kody

\Cancer

- Obsahuje testovací soubory Cancer1.txt, Cancer2.txt, Cancer3.txt a notebooky s učením pomocí klasické metody v toolboxu Neural Network a také pomocí SOMA a diferenciální evoluce.

\Card

- Obsahuje testovací soubory Card1.txt, Card2.txt, Card3.txt a notebooky s učením pomocí klasické metody v toolboxu Neural Network a také pomocí SOMA a diferenciální evoluce.

\Diabetes

- Obsahuje testovací soubory Diabetes1.txt, Diabetes2.txt, Diabetes3.txt a notebooky s učením pomocí klasické metody v toolboxu Neural Network a také pomocí SOMA a diferenciální evoluce.

\Horse

- Obsahuje testovací soubory Horse1.txt, Horse2.txt, Horse3.txt a notebooky s učením pomocí klasické metody v toolboxu Neural Network a také pomocí SOMA a diferenciální evoluce.