

GUI prostředí pro neuronové sítě v softwaru Mathematica

GUI environment for artificial neural networks in
Mathematica software

Bc. Jiří Zatloukal



ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jiří ZATLOUKAL**
Osobní číslo: **A09496**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **GUI prostředí pro umělé neuronové sítě v prostředí Mathematica**

Zásady pro vypracování:

1. Seznamte se s umělými neuronovými sítěmi.
2. Seznamte se s možnostmi GUI prostředí pro software Mathematica.
3. Zpracujte jednotlivé typy neuronových sítí za použití GUI v sw Mathematica.
4. Otestujte na vybraných typech problémů.
5. Připravte data pro demonstrační účely ve výuce.
6. Zpracujte závěr.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. ŠNOREK M., JIŘINA M.: Neuronové sítě a neuropočítače, ČVUT, 1996, ISBN 80-01-01455-X.
2. BÍLA J.: Umělá inteligence a neuronové sítě v aplikacích, ČVUT, 1996, ISBN 80-01-01275-1.
3. ZELINKA I.: Umělá inteligence I, VUT Brno, 1998, ISBN 80-214-1163-5.
4. BOSE N.K., LIANG P.: Neural Network Fundamentals with Graphs, Algorithms, and Applications, McGraw-Hill Series in Electrical and Computer Engineering, 1996, ISBN 0-07-006618-3.
5. FREEMAN J. A.: Simulating Neural Networks with Mathematica, Addison-Wesley Publishing Company, 1994, ISBN 0-201-56629-X.
6. NOVÁK M., FABER J., KUFUDAKI O.: Neuronové sítě a informační systémy živých organismů, Grada, 1993, ISBN 80-58424-95-9.
7. The Mathematica Book, manuál softwaru Mathematica.

Vedoucí diplomové práce:

Ing. Zuzana Oplatková, Ph.D.

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

24. února 2011

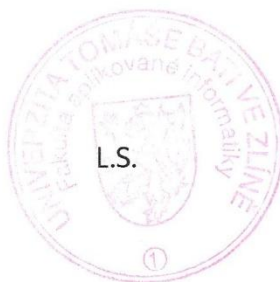
Termín odevzdání diplomové práce:

18. května 2011

Ve Zlíně dne 24. února 2011

prof. Ing. Vladimír Vašek, CSc.

děkan



doc. Mgr. Roman Jašek, Ph.D.

ředitel ústavu

ABSTRAKT

Cílem práce je představit vypracovaný toolbox pro neuronové sítě a pro něj vytvořené GUI v prostředí softwaru Mathematica a seznámit s jeho možnostmi. Balíček obsahuje několik typů neuronových sítí, které lze interaktivně ovládat z prostředí GUI a zároveň využívá potenciál výpočetního jádra a vizualizačních možností softwaru Mathematica. Aplikace obsahuje jednak připravená data pro typické aplikace, ale také lze data do trénovací množiny vložit uživatelem. Sítě používají různé učící algoritmy a jednotlivé fáze učení lze v GUI aplikaci demonstrovat v nejrůznějších grafech či tabulkách. Toolbox je možné využít v rámci výuky umělých neuronových sítí, ale také pro vědecké výpočty využívající výhod intuitivní aplikace a výpočetního výkonu softwaru Mathematica.

Klíčová slova:

GUI, Mathematica, umělá neuronová síť

ABSTRACT

The aim of this thesis is to introduce a created toolbox for neural networks and its GUI implemented in Mathematica software and to present its possibilities. The package contains several types of neural networks, which can be controlled by using of GUI environment. It also employs full potential of computational core and visualization possibilities of Mathematica software. The application contains prepared data for typical applications but it is also possible to insert training data by user. The neural networks use various training algorithms and individual phases of training can be displayed in graphs and tables. The toolbox can be used in neural networks during education in tutorials and lectures but also for scientific computation. Users can profit from advantages of intuitive application and computational performance of Mathematica software.

Keywords:

GUI, Mathematica, artificial neural network

Na tomto místě bych chtěl poděkovat vedoucí mé diplomové práce paní Ing. Zuzaně Oplatkové, Ph.D. za rady, odborné vedení a ochotu, které se mi dostalo během tvorby této práce. V neposlední řadě bych chtěl poděkovat také celé rodině za podporu a trpělivost, kterou pro mě měli.

Bez jejich přispění by tato práce nemohla vzniknout.

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.....	9
I TEORETICKÁ ČÁST.....	10
1 UMĚLÉ NEURONOVÉ SÍTĚ	11
1.1 ZÁKLADNÍ PRINCIP	11
1.2 HISTORIE NEURONOVÝCH SÍTÍ	11
1.3 MATEMATICKÝ MODEL NEURONU	12
1.4 PŘENOSOVÉ FUNKCE UMĚLÝCH NEURONŮ.....	13
1.4.1 Logistická sigmoida	14
1.4.2 Skoková funkce binární.....	14
1.4.3 Skoková funkce bipolární	15
1.4.4 Hyperbolický tangens.....	15
1.4.5 Lineární funkce	16
1.5 DĚLENÍ NEURONOVÝCH SÍTÍ	16
1.6 PERCEPTRON	16
1.6.1 Algoritmus Backpropagation	19
1.7 ADALINE	20
1.8 HOPFIELDOVA SÍŤ.....	20
1.8.1 Diskrétní Hopfieldova síť.....	20
1.8.1.1 Algoritmus učení pro binární hodnoty.....	21
1.8.1.2 Algoritmus učení pro bipolární hodnoty.....	22
1.8.2 Energetická funkce.....	22
1.8.3 Spojitá Hopfieldova síť	23
1.8.4 Kapacita Hopfieldovy sítě.....	24
1.9 DVOUSMĚRNÁ ASOCIATIVNÍ PAMĚŤ	24
1.9.1 Diskrétní BAM.....	25
1.9.2 Spojitá BAM	26
1.9.3 Kapacita sítě BAM	26
1.10 KOHONENOVA SÍŤ	26
2 EVOLUČNÍ ALGORITMY	29
2.1 ZÁKLADNÍ PRINCIP	29
2.2 HISTORIE EVOLUČNÍCH VÝPOČTŮ	29
2.3 ÚČELOVÁ FUNKCE.....	30
2.4 PENALIZACE	30
2.5 PRÁCE S CELOČÍSELNÝMI A DISKRÉTNÍMI HODNOTAMI	31
2.6 DIFERENCIÁLNÍ EVOLUCE (DE).....	31
2.6.1 Řídící parametry Diferenciální evoluce	31
2.6.2 Mutace a výpočet šumového vektoru.....	32
2.6.3 Křížení.....	33
2.6.4 Popis vlastních výpočtů během Diferenciální evoluce	33
2.6.5 Varianty Diferenciální evoluce	34
2.6.6 Závislost Diferenciální evoluce na jejích parametrech	34
2.6.7 Stagnace	34

2.7	SAMOORGANIZUJÍCÍ SE MIGRAČNÍ ALGORITMUS (SOMA)	34
2.7.1	Řídící parametry algoritmu SOMA.....	35
2.7.2	Mutace a tvorba perturbačního vektoru	36
2.7.3	Křížení.....	36
2.7.4	Popis vlastních výpočtů během optimalizace pomocí algoritmu SOMA	37
2.7.5	Varianty algoritmu SOMA.....	37
2.7.6	Závislost algoritmu SOMA na jejích parametrech.....	38
3	WOLFRAM MATHEMATICA	39
3.1	MOŽNOSTI SOFTWARE MATHEMATICA V OBLASTI GUI.....	40
3.1.1	Balíček GUIKit	40
3.1.1.1	Obecné vlastnosti objektů balíčku GUIKit	41
3.1.1.2	Práce s parametry objektů při běhu GUI	41
3.1.1.3	Button.....	42
3.1.1.4	Radiobutton.....	42
3.1.1.5	CheckBox.....	42
3.1.1.6	ComboBox	42
3.1.1.7	TextField.....	43
3.1.1.8	TextArea	43
3.1.1.9	Slider.....	43
3.1.1.10	MathPanel	43
3.1.1.11	IndexedImagePanel	44
3.1.1.12	OpenFileDialog	44
3.1.1.13	PopupMenu	45
II	PRAKTICKÁ ČÁST	46
4	TOOLBOX PRO NEURONOVÉ SÍTĚ	47
4.1	OBEČNÝ POPIS TOOLBOXU.....	47
4.2	UPŘESŇUJÍCÍ POPIS JEDNOTLIVÝCH ČÁSTÍ GUI	48
4.2.1	Adaline + Perceptron.....	48
4.2.2	Síť BAM.....	50
4.2.3	Hopfieldova síť	51
4.2.4	Učení ANN pomocí evolučních algoritmů	51
4.2.5	Samoorganizující se mapa (SOM) - Kohonenova síť	52
4.2.6	Učení ANN pomocí algoritmu backpropagation	53
	ZÁVĚR	55
	ZÁVĚR V ANGLIČTINĚ.....	56
	SEZNAM POUŽITÉ LITERATURY.....	57
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	58
	SEZNAM OBRÁZKŮ	60
	SEZNAM TABULEK.....	61
	SEZNAM PŘÍLOH.....	62

ÚVOD

Žijeme v době velkého zájmu o výpočetní technologie, které jsou využívány pro stále širší spektrum problémů a aplikací. Jednou z oblastí informatiky je i umělá inteligence. Tato práce se v této oblasti vědy zabývá hlavně umělými neuronovými sítěmi a z části i evolučními algoritmy. První aplikace neuronových sítí jsou datovány již v 60. letech minulého století. Za tu dobu prošly místy trnitým vývojem a dnes se těší zájmu odborníků po celém světě.

Umělé neuronové sítě jsou inspirovány biologickými nervovými systémy a způsob jejich činnosti je do jisté míry identický. Rozsah jejich použití spadá do nepřeberného množství oblastí. Úspěšně se používají například v úlohách klasifikace dat, aproximace a predikce průběhu systémů, identifikaci či filtraci signálů, optimalizaci atd. S výhodou je lze aplikovat u výpočetně složitých problémů, kde jsou klasické postupy časově velmi náročné a v některých případech dokonce neproveditelné.

Další oblastí umělé inteligence, kterou se tato práce z části také zabývá, jsou evoluční algoritmy a jejich použití jako metody učení pro umělou neuronovou síť. Evoluční algoritmy jako takové mají dnes hojné využití převážně v oblasti optimalizace a podobně jako umělé neuronové sítě jsou také inspirovány jevy v přírodě.

V praktické části této práce jsou demonstrovány některé základní aplikace umělých neuronových sítí a možnost jejich učení pomocí evolučních algoritmů. Všechny tyto demonstrativní ukázky jsou plnohodnotně ovladatelné pomocí GUI vytvořeném v prostředí Mathematica.

I. TEORETICKÁ ČÁST

1 UMĚLÉ NEURONOVÉ SÍTĚ

1.1 Základní princip

Umělé neuronové sítě jsou v podstatě matematické modely vycházející z biologických nervových systémů. Přejímají od nich vlastnosti jako je schopnost učit se, zobecňovat získané zkušenosti a vhodně je aplikovat na předložené problémy, zároveň s nimi přebírají ale i jejich nedostatky, jako je například jev zvaný „memory switch“ [1]. Dobře naučená neuronová síť je většinou schopná rychle reagovat na vstupní data a přizpůsobit se i datům, které se více nebo méně odlišují od prvků v trénovací množině. Jednotlivé druhy neuronových sítí mají ovšem odlišné délky doby učení (závisí samozřejmě i na konkrétním problému na jaký chceme síť naučit) a mírně nejistý výsledek. Obecně neexistuje přesný postup, jehož aplikací dosáhneme vždy stejně uspokojivého výsledku – tzn. ke každé síti a každému problému je třeba přistupovat individuálně.

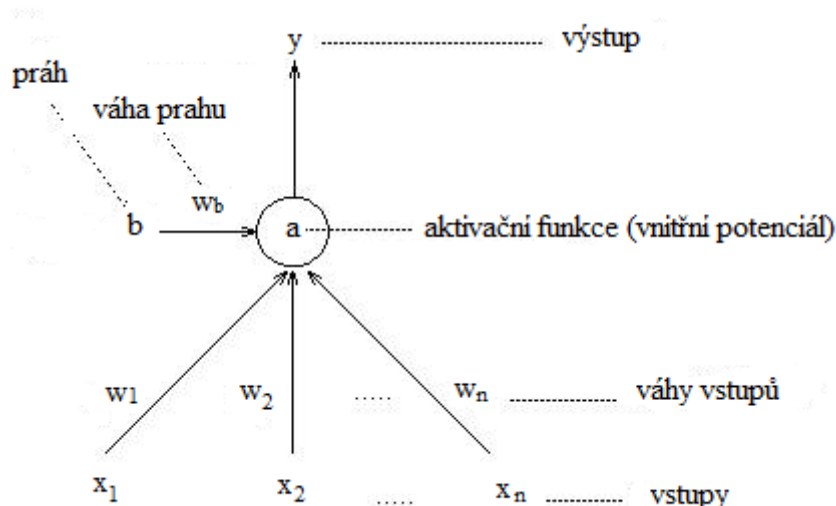
1.2 Historie neuronových sítí

Za první práci zabývající se problémem umělých neuronů se všeobecně považuje studie McCullocha a Walltera Pittse, která byla uveřejněna roku 1943 a popisovala tvorbu jednoduchého umělého matematického neuronu, který je základním prvkem neuronové sítě. V souvislosti s tímto ukázali jednoduché modely neuronových sítí, které byly schopné řešit základní matematické a logické funkce. Poté v roce 1949 vyšla kniha Donalda Hebba s názvem „The Organization of Behavior “ ve které představil pravidlo učení pro jednotlivé spoje mezi neurony v síti. Zásadní objev přišel ovšem až v roce 1957, kdy Frank Rosenblatt představil model umělého neuronu zvaný perceptron. Pro tento model také vypracoval algoritmus učení, který se opíral o matematický důkaz, že pro danou trénovací množinu nezávisle na počátečním nastavení vah nalezneme v konečném počtu kroků jejich odpovídající nastavení (pokud existuje). Nedlouho po tomto objevu vyvinul Bernard Widrow za spolupráce svých studentů nový typ umělého neuronu, který pojmenoval ADALINE (ADaptive LINear Element). Tento model využíval velice účinný učicí postup, který se dodnes využívá. Poté ovšem začal bohužel vývoj neuronových sítí víceméně stagnovat a těžkou ránu mu zasadili pánové Minsky a Papert svou knihou s názvem „Perceptrons“, ve které jako hlavní argument proti výhodám perceptronu využili tvrzení, že jeden perceptron nedokáže sám vyřešit logickou funkci XOR – tudíž poukázali na neschopnost jednoho neuronu typu perceptron řešit lineárně neseparabilní problémy. Po

tomto nastalo období, kdy se o neuronové sítě zajímali vědci jen ojediněle. Druhý dech této oblasti výzkumu vdechlo až v roce 1983 založení americké grantové agentury jménem DARPA (Defense Advanced Research Projects Agency) zásluhou programového manažera Ira Skurnicka. Tato organizace začala finančně podporovat výzkum v oblasti neurovýpočtů a na tento trend posléze navázaly další organizace. Další velkou zásluhu na obnovení výzkumu neuronových sítí měl celosvětově uznávaný fyzik John Hopfield, který v letech 1982 a 1984 publikoval články, v nichž poukázal na souvislosti fyzikálních modelů magnetických materiálů s některými modely neuronových sítí. Poté v roce 1986 publikovala takzvaná „PDP skupina“ (Paralell Distributed Procesing Group) článek o algoritmu zpětného šíření chyby (backpropagation) pro vícevrstvou neuronovou síť, čímž vyřešili problém, který se v 60. letech jevil Minskému a Papertovi jako nepřekonatelná překážka pro další využití neuronových sítí. Jak se ukázalo později, byl tento algoritmus objeven již několika jinými vědci v období nezájmu o neuronové sítě, ale právě kvůli výše zmíněnému se jim nedostalo odpovídající pozornosti. Jedním z prvních praktických využití algoritmu backpropagation byl známý systém NETalk vyvinutý Terencem Sejnowskim a Charlesem Rosenbergem, který úspěšně převáděl anglický psaný text na mluvené slovo. V roce 1987 se poté konala v San Diegu první velká konference zaměřená na oblast umělých neuronových sítí IEEE International Conference on Neural Networks a byla založena mezinárodní organizace INNS (International Neural Network Society), která dále podporovala výzkum v oblasti neuronových sítí. Od tohoto data začaly známé univerzity hojně zakládat výzkumné ústavy a výukové programy zabývající se problematikou neurovýpočtů [1].

1.3 Matematický model neuronu

Jak již bylo výše zmíněno umělý matematický neuron je inspirován neuronem biologickým a jakousi jeho jednodušší verzí. Struktura takového umělého neuronu je znázorněna na následujícím obrázku.



Obrázek 1, zobrazení matematického modelu neuronu

Neuronová jednotka má sérii vstupů (u biologického neuronu jsou to dendrity) „ x_1 “ až „ x_n “ a každý z těchto vstupů má pak svoji vlastní váhu, která určuje „propustnost“ příslušného vstupu. Vnitřní potenciál neuronu je v následujícím vztahu (1).

$$a = \sum_{i=1}^n (x_i w_i + b w_b) \quad (1)$$

Vlastní výstup z neuronu je poté dán přenosovou funkcí.

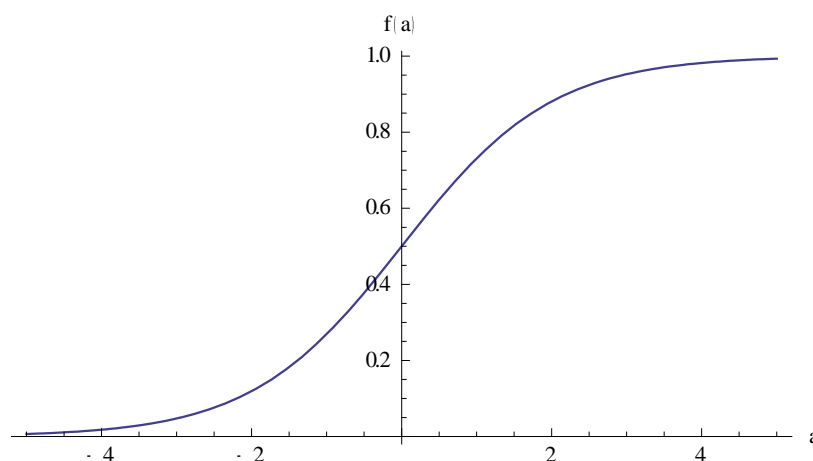
1.4 Přenosové funkce umělých neuronů

Přenosových funkcí existuje více typů. Použití konkrétní přenosové funkce závisí na druhu neuronové sítě, se kterou pracujeme a také na povaze problému, který její pomocí chceme řešit. Jak už bylo výše naznačeno vnitřní potenciál umělého neuronu je v podstatě vstupem do přenosové funkce, která poté definuje konečnou podobu výstupního signálu z neuronu. Níže jsou uvedeny přenosové funkce, které byly využity v této práci.

1.4.1 Logistická sigmoida

$$f(a) = \frac{1}{1 + e^{-\lambda a}}$$

(2)



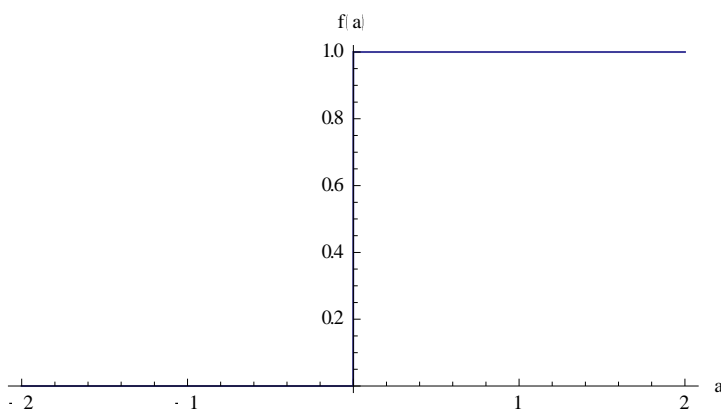
Obrázek 2, průběh Logistické sigmoidy pro $\lambda=1$

1.4.2 Skoková funkce binární

$$f(a) = 1 \quad \forall a \geq b$$

$$f(a) = 0 \quad \forall a < b$$

(3)



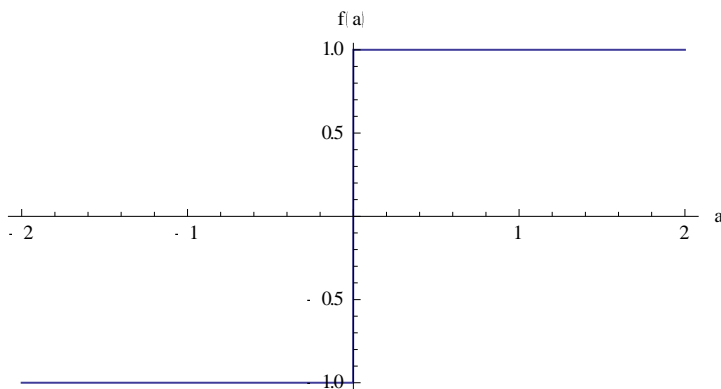
Obrázek 3, průběh Skokové funkce binární pro $b=0$

1.4.3 Skoková funkce bipolární

$$f_{(a)} = 1 \quad \forall a \geq b$$

$$f_{(a)} = -1 \quad \forall a < b$$

(4)



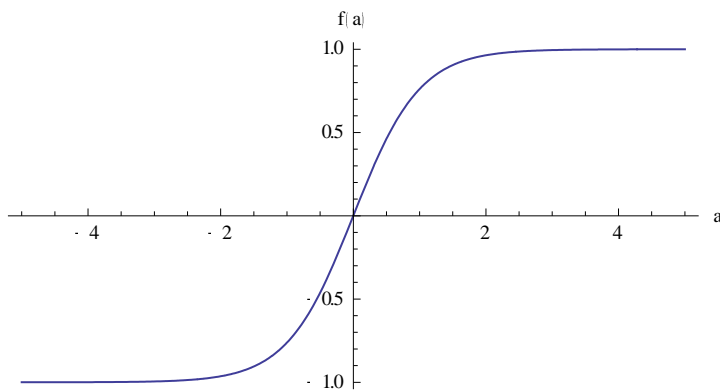
Obrázek 4, průběh Skokové funkce bipolární pro $b=0$

1.4.4 Hyperbolický tangens

$$f_{(a)} = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

[2]

(5)

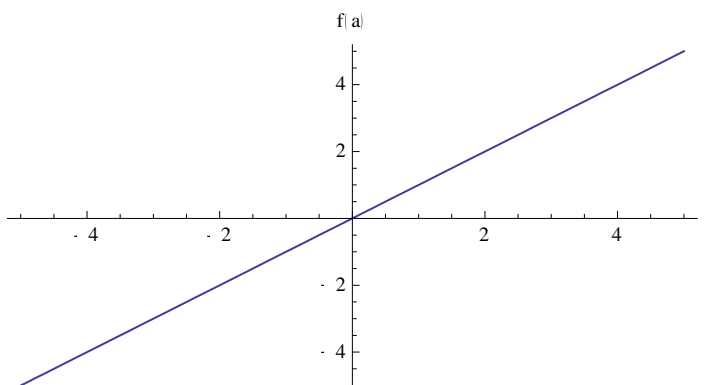


Obrázek 5, průběh funkce Hyperbolický tangens

1.4.5 Lineární funkce

$$f(a) = ka$$

(6)



Obrázek 6, průběh Lineární funkce pro $k=1$

1.5 Dělení neuronových sítí

Umělých neuronových sítí existuje celá řada a dají se dělit podle různých kritérií.

- a) Podle počtu vrstev: jednovrstvé, vícevrstvé
- b) Podle stylu učení: deterministické, stochastické
- c) Podle typu učícího algoritmu: s učitelem, bez učitele

1.6 Perceptron

Neuronová síť Perceptron byla představena Frankem Rosenblattem v roce 1957. Šlo o první úspěšný pokus vytvořit neuronovou síť schopnou adaptovat se na konkrétní problém. Model této sítě se skládá ze dvou vrstev. První vrstva neuronů je nazývána vstupní vrstvou a nemá přenastavitelné váhy. Váhy se mění až u neuronů ve výstupní vrstvě. Vstupní vrstvu lze teoreticky zanedbat a přenášet vstupní data přímo na neurony s měnitelnými vahami. V takovémto případě bychom dostali jednovrstvý Perceptron. Neurony s měnitelnými vahami mají ještě jeden vstup navíc, který zastupuje práh neuronu. Tento vstup je obvykle nastaven na hodnotu 1. Jako přenosová funkce je v této síti použita funkce skoková a to jak verze binární tak i bipolární. Jako učící algoritmus lze použít např. metodu fixních přírůstků.

Pokud: $w(k) * y(k) \leq 0$, pak: $w(k + 1) = w(k) + c * y(k)$

Pokud: $w(k) * y(k) > 0$, pak: $w(k + 1) = w(k)$

(7)

$c \dots$ koeficient učení

Koeficient učení se během učícího procesu může měnit na základě absolutní nebo zlomkové korekce.

Absolutní korekce:

Při použití této korekce musí platit následující vztah

$$w(k + 1) * y(k) = w(k) * y(k) + w(k) * cy(k) * y(k) > 0$$

(8)

Pak

$$c > \frac{|w(k) * y(k)|}{\|y(k)\|^2}$$

(9)

Zlomková korekce:

Při použití této korekce musí platit následující vztah

$$|w(k + 1) * y(k) - w(k) * y(k)| = \lambda |w(k) * y(k)|$$

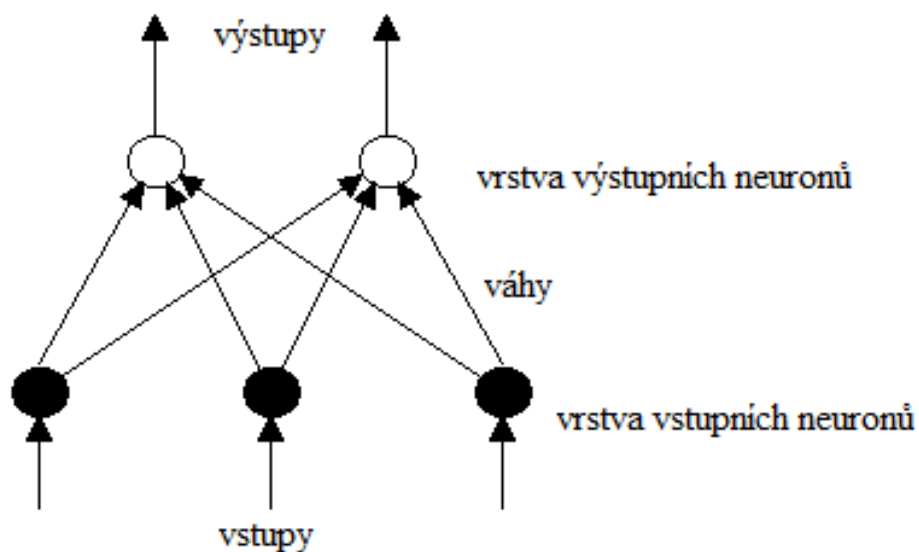
(10)

Pak

$$c > \lambda \frac{|w(k) * y(k)|}{\|y(k)\|^2}$$

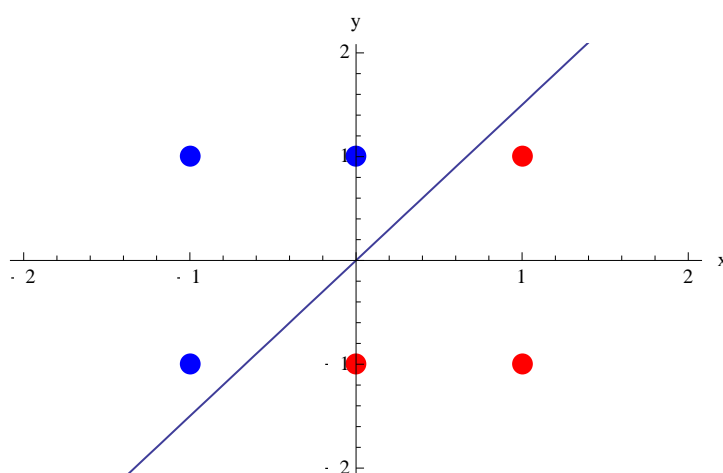
[2]

(11)

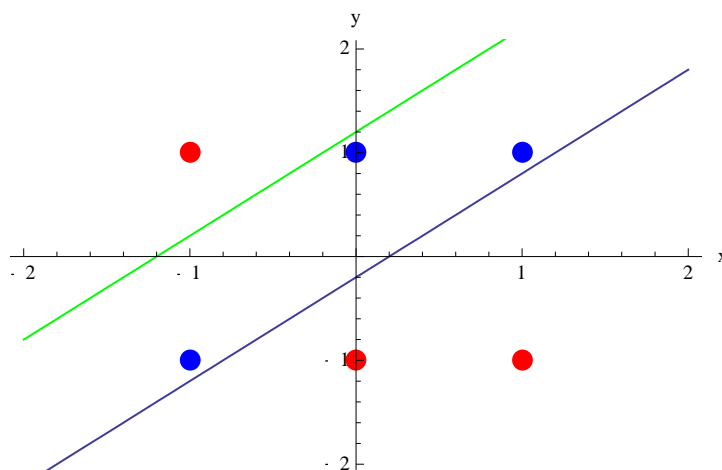


Obrázek 7, ukázka topologie sítě Perceptron

Jak už bylo zmíněno v kapitole 1.2, jednovrstvý Perceptron dokáže řešit pouze lineárně separabilní problémy. V případě, že chceme aplikovat Perceptronovou síť na lineárně neseparabilní problém, musíme použít minimálně dvě vrstvy, ve kterých mají neurony nastavitelné váhy. K naučení takovéto sítě slouží mimo jiné algoritmus Backpropagation.



Obrázek 8, příklad lineárně separabilního problému



Obrázek 9, příklad lineárně neseparabilního problému

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Tabulka 1, lineárně neseparabilní problém XOR

1.6.1 Algoritmus Backpropagation

Tento algoritmus je nejznámější a nejpoužívanějším učícím postupem pro optimalizaci vah vícevrstvé neuronové sítě. Při aktivizační fázi se signál v síti šíří klasicky od ze vstupní vrstvy, přes skryté vrstvy až na výstup. Poté se při adaptační fázi vypočítá odchylka δ^0 pro každý výstupní neuron podle následujícího vztahu.

$$\delta_j^0 = (y_j^0 - d_j^0) * y_j^0 * (1 - y_j^0)$$

(12)

d... předpokládaná hodnota výstupu

Z této se následně určí přírůstky vah jednotlivých vstupů každého neuronu.

$$\Delta w_{ij}^0(t) = \eta * \delta_j^0(t) * y_i^1 + \mu * \Delta w_{ij}^0(t-1) \quad (13)$$

η ... koeficient učení

μ ... momentum

Pro nižší vrstvy se jednotlivé odchylky δ vypočítají jako

$$\delta_j^l = y_j^l * (1 - y_j^l) * \sum_{k=1}^n (w_{kj}^{l-1} * \delta_k^{l-1}) \quad (14)$$

Přičtením přírůstků vah k jednotlivým vahám získáme váhy nové. Celý tento postup se poté opakuje pro každý následující prvek z trénovací množiny, čímž proběhne jedna učící epocha. Po proběhnutí daného počtu epoch získáme naučenou síť. [1]

1.7 Adaline

Neuronový výpočetní prvek Adaline byl přestaven profesorem Bernardem Widrowem v roce 1960. Na jeho vývoji se podíleli i někteří jeho studenti. Tato neuronová jednotka používá k optimalizaci vah následující vztah.

$$w(k+1) = w(k) + \eta(d - a) * x \quad (15)$$

η ... koeficient učení

a ... hodnota aktivační funkce neuronu

d ... předpokládaná hodnota výstupu

x ... hodnota vstupu do neuronu

[2]

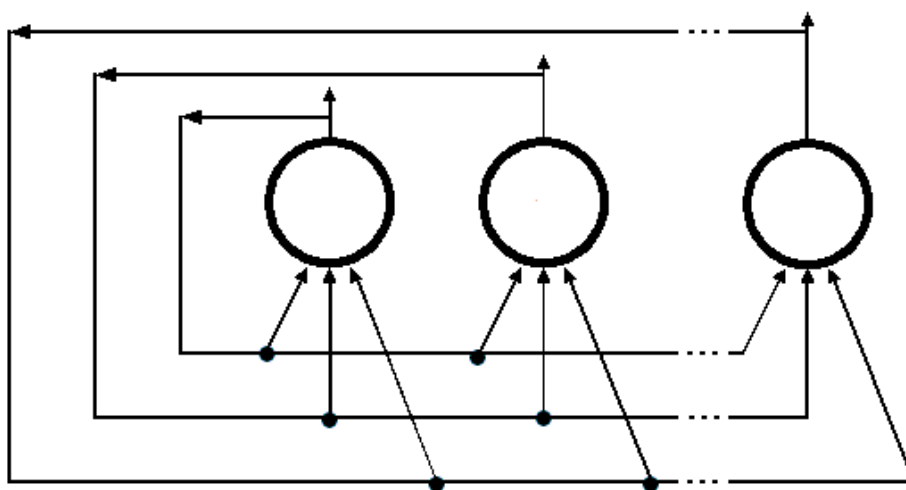
1.8 Hopfieldova síť

1.8.1 Diskrétní Hopfieldova síť

Tento typ neuronové sítě byl vytvořen fyzikem Johnem Hopfieldem. Jedná se o neuronovou síť, která se chová jako autoasociativní paměť. To v praxi znamená, že síť

dokáže přiřazovat k originálním vzorům, na které je naučena vzory jim podobné. Tato vlastnost má využití například při rozpoznávání nebo opravě zašuměného textu nebo jiným grafických vzorů. Tato síť se řadí mezi jednovrstvé sítě a každý neuron je propojen se všemi ostatními neurony v síti, ale ne samy se sebou. Všechny tyto spoje mezi jednotlivými neurony jsou ohodnoceny vahami. Zvláštností této sítě je fakt, že neurony nemají vstup pro práh.

Trénovací množina je opět tvořena sadou prvků, které jsou síti postupně předkládány ve formě vektorů. Při učení této sítě se opět hledá takové nastavení vah, pro které síť reaguje požadovaným způsobem na všechny prvky z trénovací množiny a k prvkům, které se od těch z trénovací množiny liší, vrací nejpodobnější vzory. Vlastní učení je založeno na využití energetické funkce, které se svázaná s neuronovou sítí podobně, jak to bývá obvyklé u fyzikálních systémů. Na začátku učicího procesu jsou všechny neurony v síti inicializovány binárními $\{0,1\}$ nebo bipolárními $\{-1,1\}$ hodnotami. Vzhledem k faktu, že jsou všechny neurony v síti vzájemně propojeny, začnou se tyto mezi sebou ovlivňovat. To v praxi znamená, že některé neurony začnou „posilovat“ vliv ostatních neuronů, zatímco jiné se budou pokoušet o přesný opak. Tyto akce postupně vyústí v kompromis a můžeme říci, že síť se nachází ve stabilním stavu.



Obrázek 10, ukázka topologie Hopfieldovy sítě

1.8.1.1 Algoritmus učení pro binární hodnoty

Požadované chování sítě je specifikováno trénovací množinou, ve které se nachází P trénovacích vzorů $s(p)$, $p=1, \dots, P$, z nichž je každý z nich zadán vektorem binárních hodnot.

$$s(p) = (s_1(p), \dots, s_i(p), \dots, s_n(p)) \quad (16)$$

Potom je váhová matice dána následujícím vztahem

$$w_{ij} = \sum_p [(2 * s_i(p) - 1) * (2 * s_j(p) - 1)] \quad (17)$$

pro $i \neq j$ a $w_{ii} = 0$.

1.8.1.2 Algoritmus učení pro bipolární hodnoty

Stejně jako u předchozího algoritmu je požadovaná funkce sítě určena trénovací množinou P vzorů $s(p)$, $p=1, \dots, P$, s tím rozdílem, že každý prvek trénovací množiny je zadán jako vektor bipolárních hodnot.

$$s(p) = (s_1(p), \dots, s_i(p), \dots, s_n(p)) \quad (18)$$

Potom je váhová matice dána následujícím vztahem

$$w_{ij} = \sum_p [s_i(p) * s_j(p)] \quad (19)$$

pro $i \neq j$ a $w_{ii} = 0$. [2]

1.8.2 Energetická funkce

Tato funkce byla definována Johnem Hopfieldem v analogii s fyzikálními jevy. Tato funkce přiřazuje každému stavu sítě jeho potenciální energii. Jedná se tedy o funkci, která je zdola ohraničená a zároveň je nerostoucí pro daný stav systému (množinu aktivací všech neuronů). Pokud je tedy tato funkce nalezena, tak síť bude konvergovat ke stabilní množině aktivací neuronů v daném čase. Energetická funkce pro diskrétní Hopfieldovu síť je definována následujícím vztahem.

$$E = -0,5 \sum_{i \neq j} \sum_j y_i y_j w_{ij} - \sum_i x_i y_i + \sum_i \theta_i y_i \quad (20)$$

Pokud se tedy změni aktivace sítě o Δy_i pak se hodnota energetické sítě změní podle následujícího vztahu.

$$\Delta E = - \left[\sum_j y_j w_{ij} + x_i - \theta_i \right] \Delta y_i \quad (21)$$

Z toho plyne, že stavy sítě, které mají nejnížší hodnotu energetické funkce, mají největší stabilitu. Rozdíl oproti učení pomocí algoritmu Backpropagation je hlavně ve faktu, že délka trvání adaptace Hopfieldovy sítě závisí pouze na počtu prvků v trénovací množině. Při učení Hopfieldovy sítě samovolně vznikají takzvané nepravé vzory (fantomy), které neodpovídají žádnému prvku z trénovací množiny. Existují varianty Hopfieldovy sítě, jejichž použitím je možné takto vzniklé fantomy odstranit. [2]

1.8.3 Spojitá Hopfieldova síť

U spojitě Hopfieldovy sítě je vývoj reálného stavu v aktivním režimu dán nejen spojitou funkcí vnitřního potenciálu, ale také i spojitou funkcí času. Aktivní dynamika sítě je tedy popsána diferenciální rovnicí, jejíž řešení nelze explicitně vyjádřit. Spojitá hopfieldova síť je v podstatě modifikací diskrétní Hopfieldovy sítě.

Funkce energie je tedy definována vztahem.

$$E = -0,5 \sum_{i=1}^n \sum_{j=1}^n w_{ij} v_i v_j + \sum_i \theta_i v_i \quad (22)$$

Potom bude tato síť konvergovat ke stabilní konfiguraci, když:

$$\frac{d}{dt} E \leq 0 \quad (23)$$

Aktivity všech neuronů v čase jsou poté určeny následujícím vztahem.

$$\frac{d}{dt} u_i = - \frac{\partial E}{\partial v_i} = - \sum_{j=1}^n w_{ij} v_j - \theta_i \quad (24)$$

v_i ... aktivace neuronu

[2]

Spojitou Hopfieldovu síť lze využít jako autoasociativní paměť nebo k řešení optimalizačních problémů při známých omezujících podmínkách.

1.8.4 Kapacita Hopfieldovy sítě

John Hopfield experimentálně zjistil, že počet binárních vzorů, které může síť uchovat a opětovně vyvolat je přibližně roven následujícímu vztahu.

$$P \approx 0,15n \quad (25)$$

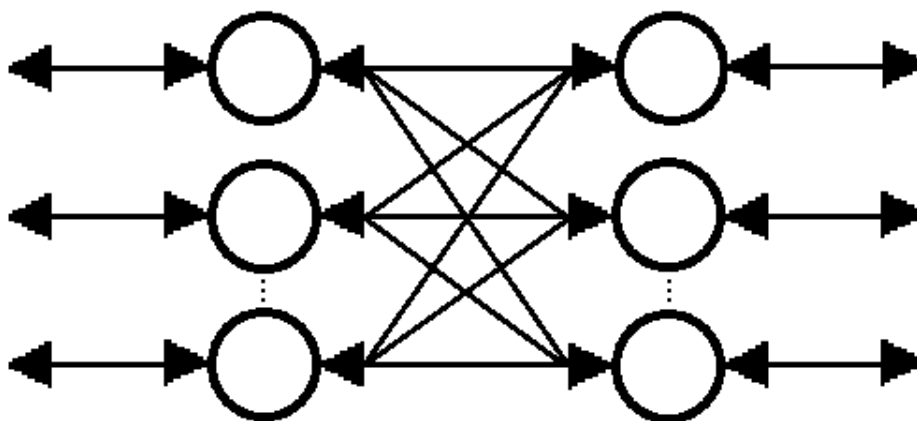
n ... počet neuronů v síti

Pro bipolární verzi sítě je to obdobný vztah.

$$P \approx \frac{n}{2 \log_2 n} \quad (26)$$

1.9 Dvousměrná asociativní paměť

Dvousměrná asociativní paměť (BAM) je v podstatě modifikovaná Hopfieldova síť, která má schopnost heteroasociace. Tato síť dokáže ke vstupnímu vzoru asociovat jiný vzor což znamená, že vytváří páry asociovaných vzorů. Celá struktura sítě je dána dvěma vrstvami neuronů, z nichž je každý propojen dvousměrným spojením se všemi neurony v protější vrstvě. Tyto neurony si mezi sebou vzájemně posílají signál tak dlouho, až nastane rovnovážný stav. Podobně jako u Hopfieldovy sítě existují dva základní typy této umělé neuronové sítě, a to diskretní BAM a spojitá BAM.



Obrázek 11, ukázka topologie sítě BAM

1.9.1 Diskrétní BAM

Tento druh sítě BAM má dvě formy. První z nich je bipolární BAM a druhou je binární BAM. Binární a bipolární BAM jsou si velice podobné, tudíž u každé lze vytvořit hodnoty jednotlivých vah pomocí sumace hodnot aktivačních funkcí neuronů pro odpovídající si páry vektorů v trénovací množině. Matici vah lze tedy vytvořit pomocí následujícího vztahu.

Množina párových vektorů $s(t) : t(p) ; p=1, \dots, P$

$$s(p) = (x_1, \dots, x_i, \dots, x_n)$$

$$t(p) = ((y_1, \dots, y_j, \dots, y_n)) \quad (27)$$

Pro binární data

$$w_{ij} = \sum_p [(2s_i(p) - 1)(2t_j(p) - 1)] \quad (28)$$

Pro bipolární data

$$w_{ij} = \sum_p [s_i(p)t_j(p)] \quad (29)$$

Hodnotu výstupu neuronu lze určit podle následujících pravidel.

Pro binární (bipolární) data

$$\text{pokud } \sum x_i w_{ij} > 0 \text{ tak } y_j = 1$$

$$\text{pokud } \sum x_i w_{ij} < 0 \text{ tak } y_j = 0 \text{ } (-1)$$

$$\text{pokud } \sum x_i w_{ij} = 0 \text{ tak } y_j = y_j$$

$$\text{pokud } \sum y_j w_{ij} > 0 \text{ tak } x_i = 1$$

$$\text{pokud } \sum y_j w_{ij} < 0 \text{ tak } x_i = 0 \text{ } (-1)$$

$$\text{pokud } \sum y_j w_{ij} = 0 \text{ tak } y_j = x_i \quad (30)$$

1.9.2 Spojitá BAM

Tato umělá neuronová síť transformuje na výstup hladký spojitý signál z intervalu (0,1) za použití logistické sigmoidy jako přenosové funkce. Hodnoty vah sítě lze určit následujícím způsobem.

$$w_{ij} = \sum_p [(2s_i(p) - 1)(2t_j(p) - 1)] \quad (31)$$

Hodnota výstupu se určí podle vztahu (32).

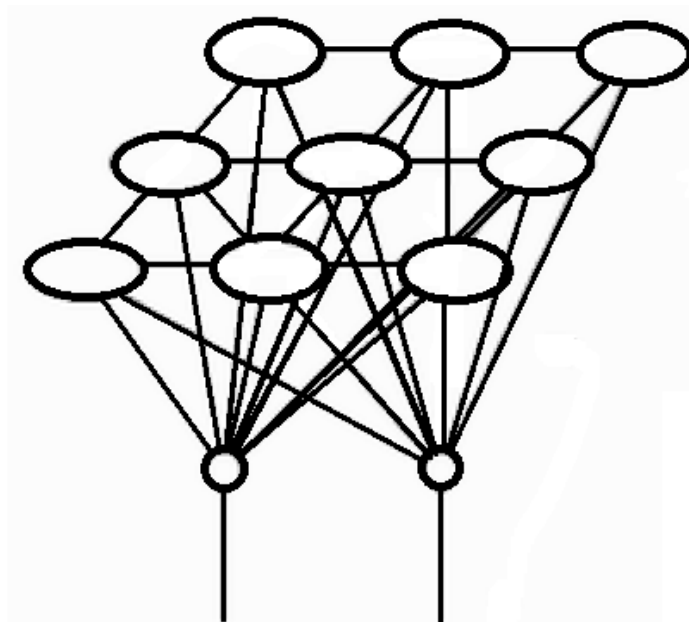
$$y_j = b_j + \sum x_i w_{ij} \quad (32)$$

1.9.3 Kapacita sítě BAM

Kapacita sítě BAM je její největší nevýhodou, protože je velmi malá. Bylo zjištěno, že nejnižší počet neuronů v libovolné vrstvě sítě se rovná nejvyššímu počtu vzorů, na které lze síť BAM naučit. [2]

1.10 Kohonenova síť

Kohonenova síť patří do skupiny umělých neuronových sítí, které se nazývají samoorganizující se mapy (Self-Organizing Maps; SOM). Síť jako taková patří do skupiny sítí, u nichž probíhá učení bez učitele. K naučení jí tedy stačí pouze dostatečně velká trénovací množina jednotlivých datových vzorů, které mají nějakou společnou vlastnost nebo naopak, mezi nimiž jsou velké odlišnosti. Podle těchto vlastností síť sama při učícím procesu vytváří jednotlivé klasifikační třídy a sama do nich zařazuje prvky z trénovací množiny. Počet těchto tříd se během učení postupně zvyšuje až do chvíle, kdy všechny vzory z trénovací množiny mohou být jednoznačně zařazeny do některé z klasifikačních tříd. Délka vlastního učení je dána konečným počtem iterací, ve kterých učení probíhá. Máme-li tedy sadu neznámých dat, které potřebujeme roztrždit podle jejich vlastností je Kohonenova síť tou správnou volbou.



Obrázek 12, ukázka topologie Kohonenovy sítě

Síť jako taková se skládá z jedné vrstvy neuronů, která může být uspořádána buďto v jedné řadě jako vektor nebo častěji jako dvourozměrná mřížka. Vlastní proces učení lze v podstatě chápat jako postupné přizpůsobování neuronů na konkrétní prvky z trénovací množiny, čímž během učení vzniknou v síti shluky neuronů, které „odpovídají“ konkrétním typům tříd prvků z trénovací množiny. Před začátkem vlastního učení jsou jednotlivé váhy náhodně nastaveny na hodnoty z intervalu $<0,1>$ a také velikost okolí. Okolními neurony se myslí sousední neurony na mřížce neuronové sítě. Existuje více typů okolí např. kruhové, čtvercové, obdélníkové atd.

V momentě, kdy je síti předložen vzor z trénovací množiny, tak jsou podle vztahu (33) vypočítány vzdálenosti (míra odlišnosti) jednotlivých neuronů od předloženého prvku.

$$d_j = \sum_{i=0}^{N-1} [x_i(t) - w_{ij}(t)]^2 \quad (33)$$

Dále je vybrán neuron, který má nejmenší vzdálenost (odlišnost) d_j od aktuálního prvku z trénovací množiny. Tomuto neuronu budou následně adaptovány váhy a zároveň budou i upraveny váhy ostatních neuronů v okolí nalezeného neuronu. Adaptace vah probíhá podle vztahu (34). Čím dále budou okolní neurony od nalezeného neuronu, tím méně budou jejich váhy ovlivněny. Navíc během učícího procesu je velikost okolí

postupně snižována. Prvky z trénovací množiny se vybírají náhodně. O délce učení rozhoduje zvolený počet iterací učícího procesu.

$$w_{ij}(t + 1) = w_{ij}(t) + \eta(t)g(x)[x_i(t) - w_{ij}(t)] \quad (34)$$

Funkce $g(x)$ ovlivňuje míru úpravy vah okolních neuronů a η zastupuje učící koeficient, jehož velikost se také během učení snižuje.

Vlastní testování naučené sítě probíhá předložením vzoru, poté vypočtením vzdálenosti podle (33) a váhy nalezeného neuronu představují odpověď sítě (např. mohou identifikovat konkrétní třídu prvku při klasifikační úloze) [2].

2 EVOLUČNÍ ALGORITMY

2.1 Základní princip

Evoluční algoritmy jsou v podstatě numerické výpočetní algoritmy, které jsou podobně jako umělé neuronové sítě inspirovány biologickým vzorem. V tomto případě se jedná o Darwinovu a Mendelovu teorii evoluce. Hlavní myšlenka této teorie spočívá v tom, že rodiče předávají svou genetickou informaci svým potomkům, kterým následně uvolní místo v prostoru, ve kterém existují. Podle evoluční teorie se jednotlivé druhy vyvíjejí tak, že z rodičů jsou ploženi potomci a tito potomci jsou při svém vzniku ovlivněni mutací. Rodiče a potomci nevhodní pro aktuální životní prostředí cyklicky vymírají, aby uvolnili místo vhodnějším jedincům, ze kterých se pak stanou rodiče.

Pokud se na výše zmíněný text podíváme z pohledu optimalizace řešení určitého problému (řekněme hledání minima složité funkce), tak životním prostorem jednotlivých jedinců je právě plocha, tvořená touto funkcí a vhodnost jedince je určena pomocí jeho polohy na této ploše.

2.2 Historie evolučních výpočtů

Počátky evolučních algoritmů jsou často datovány do první poloviny 70. let minulého století, kdy byly objeveny první genetické algoritmy. Jako počátek evolučních výpočtů však lze uvést i polovinu 60. let, kdy se poprvé podařilo úspěšně aplikovat takzvané evoluční strategie. Jako duchovní otcové evolučních algoritmů se dají uvést matematik A. M. Turing, N. A. Barricelli a další. Již v dobách těchto myslitelů byly definovány principy evolučních algoritmů, ovšem nebyly fyzicky realizovány z důvodu nedostatku výkonné výpočetní techniky.

První reálné experimenty s evolučními postupy jsou datovány do roku 1953, kdy N. A. Barricelli provedl první simulovanou evoluci na počítači. V roce 1962 svoji metodu ještě zdokonalil a podařilo se mu úspěšně realizovat evoluční experiment s 500 osmibitovými řetězci, které reprezentovaly jednoduché instrukce. V dalších letech poté zájem o evoluční výpočty postupně stoupal a v současné době se evoluční algoritmy těší zájmu odborníků po celém světě.[3]

2.3 Účelová funkce

Definice takzvané účelové funkce je kritická pro celý evoluční algoritmus. Je definována na základě problému, který chceme řešit pomocí evolučního postupu. Dosazením parametrů jedince do účelové funkce získáme informaci o kvalitě tohoto jedince v populaci a na základě této hodnoty je poté rozhodnuto o jeho vhodnosti do dalšího evolučního cyklu nebo jako rodiče pro tvorbu potomků. Nesprávně definovaná účelová funkce může způsobit neúspěch pokusu o vyřešení daného problému a naopak.

2.4 Penalizace

V reálném světě jsme omezeni řadou fyzikálních a logických zákonů, které definují možnosti nastavení parametrů daných jedinců v populaci evolučního algoritmu. Například je nemožné vyrobit strojní součástku o záporné tloušťce a podobně. Je nutné si uvědomit, že poloha jedince na ploše účelové funkce (a z té plynoucí kvalita jedince) je dána hodnotami jeho argumentů a existence tzv. zakázaných oblastí, představujících nerealizovatelné řešení, klade omezení na pohyb tohoto jedince v prostoru, tudíž toto omezení je kladeno na vlastní hodnoty argumentů daného jedince. Existují dva přístupy, kterými je možné takové omezení realizovat.

První možností je tzv. soft-constraints. V tomto případě je u jedince, který se ocitne mimo přípustnou oblast řešení, upravena hodnota účelové funkce takovým způsobem, aby byl proti zbytku populace výrazně znevýhodněn. To zapříčiní fakt, že takový jedinec se v následujícím evolučním cyklu neobjeví. Celé toto řešení si lze představit jako lokální deformace plochy účelové funkce v oblastech nedovolených hodnot argumentů jednotlivých jedinců

Druhým způsobem je tzv. hard-constraints. Aplikováním této metody jsou z prostoru všech možných řešení vyjmuty ty oblasti, které představují nerealizovatelné řešení. Důsledkem tohoto je tento prostor rozdělen na menší prostory, po kterých se jedinec může pohybovat. Toto řešení má ovšem nevýhodu v tom, že při mnohonásobném rozdělení prostoru mají jedinci tendenci kupit se na hranicích jednotlivých částí prostoru a snižuje se pravděpodobnost nalezení globálního optima. Proto je lépe sáhnout raději po metodě soft-constraints.

2.5 Práce s celočíselnými a diskrétními hodnotami

Evoluční algoritmy jsou schopny pracovat jak s celočíselnými, tak s reálnými a diskrétními hodnotami jednotlivých argumentů. Ošetření pro práci s takovými hodnotami argumentů jsou následující.

V prvním případě se údaj zaokrouhlí přímo v populaci a v rámci hodnot argumentů jedince se pracuje s celočíselnými hodnotami. V případě druhé metody se zaokrouhlení provede až těsně před dosazením do účelové funkce. Tento rozdíl má ovšem zásadní význam na průběh a kvalitu celého evolučního procesu. V prvním případě je totiž prostor možných řešení procházen po celočíselných krocích, což má nepříznivý vliv na kvalitu prohledávání prostoru. V druhém případě je prostor prohledávám po jemnějších krocích a parametry jedinců jsou uchovávány jako reálná čísla. Až při ověřování kvality těchto jedinců jsou hodnoty jejich parametrů zaokrouhleny.

Pro práci s diskrétními hodnotami pak lze použít podobný postup. Pro množinu diskrétních hodnot je vytvořen celočíselný index, který nahradí danou hodnotu parametru. Evoluční algoritmus poté pracuje s tímto indexem jako s normálním celočíselným parametrem. Až při dosazení do účelové funkce je tato celočíselná hodnota použita jako index do množiny diskrétních hodnot, ze které se poté do účelové funkce dosadí adekvátní hodnota. Množina diskrétních hodnot by měla být seřazena vzestupně aby platil následující vztah.

$$x_k^{(D)} < x_{k+1}^{(D)}, \quad k = 1, \dots, l \quad (35)$$

2.6 Diferenciální evoluce (DE)

Diferenciální evoluci poprvé použili Ken Price a Rainer Storm. Princip diferenciální evoluce spočívá v generování takzvaného zkušebního řešení přičtením difference dvou náhodně zvolených vektorů (jedinců z populace) ke třetímu jedinci z populace. Vlastní činnost a efektivita celého algoritmu je stejně jako u jiných evolučních algoritmů závislá na nastavení jeho řídicích parametrů.

2.6.1 Řídicí parametry Diferenciální evoluce

- $CR \in (0,1)$. Jedná se o takzvaný práh křížení. Pokud by byl práh křížení nastaven na 0 tak by byl výsledný jedinec identickou kopií čtvrtého rodiče, tudíž by

neprobíhala evoluce. Naopak kdyby měl tento parametr hodnotu 1, tak by byl výsledný potomek složen pouze ze tří náhodně vybraných rodičů a diferenciální evoluce by spíše připomínala náhodné prohledávání. Je tedy vhodné, aby tento parametr nenabýval svých krajních hodnot.

- **D** – dimenze problému. Tento parametr přímo závisí na konkrétní formulaci problému, který chceme pomocí diferenciální evoluce řešit. Udává počet argumentů účelové funkce (počet parametrů jednotlivých jedinců v populaci). Pokud chceme změnit hodnotu tohoto parametru, je nutné problém formulovat jiným způsobem.
- **NP** – velikost populace. Tento parametr přímo ovlivňuje počet jedinců v populaci evolučního algoritmu. Obecně se nastavuje v intervalu od $10 \cdot D$ do $100 \cdot D$, ale není to pevně dáno. Lze však s jistotou říci, že by počet jedinců v populaci neměl klesnout pod hodnotu 4 z důvodu nutnosti čtveřice rodičů k tvorbě nového potomka.
- $F \in (0,2)$ – mutační konstanta. Tento parametr ovlivňuje míru mutace při tvorbě nového potomka.
- **Generations** – Nastavením hodnoty tohoto parametru lze zvolit počet evolučních cyklů (generací), které proběhnou při vlastním evolučním procesu. Tento parametr by přirozeně měl být nastaven na hodnotu větší než nula.

2.6.2 Mutace a výpočet šumového vektoru

Mutace hraje u evolučních algoritmů důležitou úlohu. Diferenciální evoluce využívá pro tvorbu potomka (nového jedince) ne dva ale čtyři rodiče. Pro každého jedince jsou náhodně vybráni tři další jedinci (r_1, r_2, r_3) z populace, pomocí nichž je vytvořen takzvaný šumový vektor („ v “). Tento vektor představuje mutaci kombinace těchto tří jedinců. Nejprve je vypočten rozdíl jedinců r_1 a r_2 a tento rozdíl je poté vynásoben hodnotou mutační konstanty F . K tomuto výslednému vektoru je následně přičten vektor r_3 . [3]

$$v_j = x_{r_3,j}^G + F(x_{r_1,j}^G - x_{r_2,j}^G)$$

(36)

2.6.3 Křížení

V případě Diferenciální evoluce nastává křížba až po výpočtu šumového vektoru (36) a při tomto procesu je z aktuálně zvoleného jedince a k němu vypočítaného šumového vektoru následně vytvořen takzvaný zkušební vektor. Ten se tvoří pomocí cyklu, který pro každý prvek tohoto vektoru generuje náhodné číslo z intervalu $<0,1>$, které se poté porovnává s hodnotou parametru CR. Pokud je generované číslo menší než CR, tak je do zkušebního vektoru přepsán prvek z vektoru šumového a naopak.

2.6.4 Popis vlastních výpočtů během Diferenciální evoluce

Vlastním cílem diferenciální evoluce je vyšlechtit během evolučních cyklů (v tomto případě generací) takovou populaci jedinců, aby vzhledem k účelové funkci (dána formulací problému) zastupovali co možná nejvyšší řešení konkrétního problému. Celý proces evoluce probíhá tedy následujícím způsobem.

- 1) Stanovení parametrů – jedná se o nastavení řídicích parametrů Diferenciální evoluce a také o definování vzorového jedince (Specimen), který definuje fyzickou podobu všech jedinců v populaci. Udává počet parametrů jedinců a jejich datové typy. Dále je ještě třeba definovat vlastní účelovou funkci, která bude podávat informace o kvalitě jednotlivých jedinců.
- 2) Tvorba populace – v této fázi je podle vzorového jedince náhodně vygenerována množina jedinců, kteří se budou účastnit evolučního procesu. Každý jedinec obsahuje kromě výše zmíněných parametrů navíc ještě údaj o svojí kvalitě (vhodnosti), který se vypočítá pomocí účelové funkce.
- 3) Započítání cyklu generace – během každé generace je vykonáván cyklus, který zabezpečuje evoluční vývoj jednotlivých jedinců populace. Tento cyklus postupně vybírá jednoho jedince za druhým a pro každého vybraného jedince je vykonán následující evoluční cyklus.
- 4) Evoluční cyklus – v tomto cyklu je proveden výpočet šumového vektoru a jeho pomocí je poté vytvořen takzvaný vektor zkušební podle postupu v kapitole 2.6.3. K tomuto zkušebnímu vektoru je poté vypočítána jeho hodnota účelové funkce, která je poté porovnána s hodnotou účelové funkce aktuálního jedince. Podle toho, který jedinec má lepší hodnotu účelové funkce, ten jde do další populace.
- 5) Testování naplnění ukončovacích parametrů – V tomto kroku se testuje, zda již proběhl zvolený počet generací podle hodnoty v parametru Generations.

- 6) Vyhodnocení – během každé generace je uložena hodnota účelové funkce nejkvalitnějšího jedince. Sadu těchto hodnot lze poté vykreslit do grafu a snadno tak získat přehled o vývoji celého evolučního procesu.

2.6.5 Varianty Diferenciální evoluce

Výše popsaná Diferenciální evoluce je verze DERandBin1. Existují i další verze, které se liší způsobem výpočtu šumového vektoru. Tyto jsou velmi dobře popsány v[3].

2.6.6 Závislost Diferenciální evoluce na jejích parametrech

Experimentálně bylo zjištěno, že při rostoucích hodnotách parametrů F, CR a NP je Diferenciální evoluce schopna nalézat kvalitnější řešení. Nicméně existují situace, při kterých nastává jen nazývaný stagnace. Tento je popsán v následující kapitole.

2.6.7 Stagnace

Tento jev je nevýhodou algoritmu Diferenciální evoluce. Při tomto jevu již během evoluce nedochází k dalšímu zlepšování hodnoty účelové funkce, když ještě nebyl nalezen globální extrém. Tento jev většinou nastane z následujících důvodů.

- Proces optimalizace zavedl populaci do oblasti lokálního extrému účelové funkce.
- Populace ztratila svou diverzibilitu (rozdílnost mezi jednotlivými jedinci).
- Proces optimalizace je pomalý nebo neprobíhá vůbec.

I při vyvarování se výše uvedených situací může ovšem Diferenciální evoluce stagnovat.

2.7 Samoorganizující se migrační algoritmus (SOMA)

Činnost algoritmu SOMA je, podobně jako DE, založena na vektorových operacích. Na rozdíl od jiných evolučních algoritmů ale nejsou fyzicky vytvářeni noví jedinci, ale dochází k jejich pohybu po ploše účelové funkce v evolučních cyklech nazývaných migrační kole (zkráceně migrace). Základní princip algoritmu SOMA lze přirovnat ke spolupráci dané skupiny jedinců za cílem nalezení cíle (globálního extrému na účelové funkci). Jedna z hlavních vlastností algoritmu SOMA je, že jednotliví jedinci se při hledání vzájemně ovlivňují, čímž dochází k formování a rozpadání se skupin jedinců na prohledávaném prostoru, přes který postupně putují.

2.7.1 Řídící parametry algoritmu SOMA

- $PathLength \in (1, 5>$. Tento parametr určuje, do jaké vzdálenosti za vedoucího jedince bude aktuální jedinec prohledávat daný prostor. Pokud by byl tento parametr nastaven na hodnotu menší než 1, došlo by k degradaci algoritmu, protože aktuální jedinec by přestal prohledávat prostor ještě před dosažením pozice vedoucího jedince. Pro hodnotu 1 by se aktuální jedinec zastavil přesně na pozici vedoucího jedince, což také není ideální z toho důvodu, že toto konkrétní místo je už prozkoumáno, tudíž není důvod jej testovat znovu. Obecně se tento parametr nastavuje na hodnotu 3.
- $Step \in (0.11, PathLength>$. Tímto parametrem lze definovat přesnost, s jakou bude putující jedinec prohledávat prostor po cestě k vedoucímu jedinci. Jinými slovy určuje délku kroku putujícího jedince. Pro efektivitu algoritmu SOMA se doporučuje nastavovat tento parametr tak, aby vzdálenost mezi aktuálním a vedoucím jedincem nebyla celočíselným násobkem parametru Step.
- $PRT \in <0, 1>$. Tento parametr značí takzvanou perturbaci. Podle hodnoty tohoto parametru je vytvořen takzvaný perturbační vektor (PRTVector), který ovlivňuje přesnost aktuálního jedince, s jakou bude následovat vedoucího jedince. Většinou se tento parametr nastavuje na hodnotu 0.1, protože se stoupající hodnotou toho parametru silně narůstá konvergence algoritmu SOMA k lokálním extrémům. Tento parametr představuje stochastickou složku algoritmu SOMA a při jeho nastavení na hodnotu 1 tato složka zaniká, protože aktuální jedinec putuje přímo k vedoucímu jedinci.
- D – dimenze problému. Tento parametr závisí na formulaci daného problému a je jej možné změnit pouze jeho jinou formulací. Udává počet argumentů účelové funkce (počet argumentů jedinců v populaci).
- $PopSize \in <10, \text{dáno uživatelem}>$. Tímto parametrem je možno nastavit velikost populace, jinými slovy počet jedinců, kteří budou prohledávat plochu účelové funkce. Z důvodu zachování výkonnosti algoritmu SOMA by hodnota tohoto parametru neměla klesnout pod 10. Pro velké hodnoty parametru D se nastavuje v rozsahu $0.2*D$ až $0.5*D$ (např. pro $D = 100$).
- $Migrace \in <10, \text{dáno uživatelem}>$. Tímto parametrem lze nastavit počet evolučních cyklů (migrací), které mají během evolučního procesu proběhnout.

- MinDiv – minimální diverzita. Tento parametr definuje jaký maximální rozdíl mezi kvalitou nejlepšího a nejhoršího jedince v populaci je povolen pro běh algoritmu. Jinak řečeno pokud se rozdíl kvality nejlepšího a nejhoršího jedince v populaci dostane pod mez, udávanou hodnotou parametru MinDiv, je evoluční proces ukončen.

2.7.2 Mutace a tvorba perturbačního vektoru

Úlohu mutace přebírá u algoritmu SOMA takzvaná perturbace. Její princip spočívá ve faktu, že směrový vektor, po kterém se pohybuje aktuální jedinec k jedinci vedoucímu, je rušen (odkloněn) podle perturbačního vektoru. Každý jedinec v populaci má svůj unikátní perturbační vektor. Tvorba perturbačního vektoru probíhá podle vztahu (37).

$$\text{pokud } rnd_j < PRT \text{ pak } PRTVector_j = 1 \text{ jinak } 0, j = 1, \dots, D \quad (37)$$

Pro každý argument jedince je tedy vygenerováno náhodné číslo z intervalu 0 až 1 a toto se poté porovnává s hodnotou parametru PRT. Pokud je hodnota náhodného čísla menší, tak se na aktuální pozici perturbačního vektoru zapíše hodnota 1, v opačném případě pak 0. Tento vektor poté hraje zásadní roli při operaci křížení.[3]

2.7.3 Křížení

Vlastní křížení u algoritmu SOMA probíhá odlišně, než u Diferenciální evoluce. Křížením v pravém slova smyslu je tu myšleno putování aktuálního jedince k jedinci vedoucímu, podle jeho směrového vektoru. Tento směrový vektor je ale ovlivněn perturbací. Vlastní putování (modifikace hodnot argumentů) jedince tedy probíhá podle vztahu (38).

$$x_{i,j}^{ML+1} = x_{i,j,start}^{ML} + (x_{L,j}^{ML} - x_{i,j,start}^{ML})t PRTVector_j, \text{ kde } t \in \langle 0, PathLength \rangle \quad (38)$$

Počet hodnot, kterých může parametr t nabýt, je dán poměrem $PathLength/Step$. Úlohu perturbace lze tedy ze vztahu (38) definovat jako stochastickou složku algoritmu SOMA. Argumenty jedince, které mají na jejich ekvivalentní pozici v perturbačním vektoru hodnotu 1, se během operace křížení měnit budou a naopak.[3]

2.7.4 Popis vlastních výpočtů během optimalizace pomocí algoritmu SOMA

Vlastním cílem algoritmu SOMA je vyšlechtit během evolučních cyklů (v tomto případě migrací) takovou populaci jedinců, aby vzhledem k účelové funkci (dána formulací problému) zastupovali co možná nejkvalitnější řešení konkrétního problému. Celý proces evoluce probíhá tedy následujícím způsobem.

- 1) Definice parametrů – před startem algoritmu SOMA je nutné nastavit hodnoty parametrů, popsanych v kapitole 2.7.1 a definovat vzorového jedince (Specimen), který určuje fyzickou podobu všech jedinců v populaci. Udává počet parametrů jedinců a jejich datové typy. Dále je ještě třeba definovat vlastní účelovou funkci, která bude podávat informace o kvalitě jednotlivých jedinců.
- 2) Tvorba populace – v této části je vytvořena počáteční populace jedinců pomocí generátoru náhodných čísel. Každý jedinec je tvořen podle podoby vzorového jedince.
- 3) Migrační kola – vlastní evoluční proces probíhá v cyklech nazvaných migrační kola. V každém tomto kole jsou oceněni všichni jedinci v populaci pomocí účelové funkce a je vybrán nejkvalitnější jedinec, který je zvolen za vedoucího (leadera). Poté se začnou ostatní jedinci pohybovat směrem k leaderovi po pevně daných krocích definovaných hodnotou parametru Step. V každém kroku je vypočítána hodnota jeho účelové funkce a její nejlepší hodnota je uložena. V krocích jedinec pokračuje až do chvíle, kdy dosáhne vzdálenosti udávané parametrem PathLength. Nová pozice jedince poté odpovídá místu, ve kterém byla hodnota jeho účelové funkce nejlepší. Tento postup je proveden pro všechny jedince v populaci.
- 4) Testování naplnění ukončovacích parametrů – v této fázi proběhne test, zda je rozdíl mezi hodnotou účelové funkce nejlepšího a nejhoršího jedince menší než velikost parametru MinDiv a také, zda již proběhl předem daný počet migračních kol definovaný parametrem Migrace. Pokud není žádná tato podmínka splněna, proběhne další migrační kolo.
- 5) Stop – nejlepší nalezené řešení je vypsáno a algoritmus je ukončen.

2.7.5 Varianty algoritmu SOMA

V současné době existuje pět verzí tohoto algoritmu, které se liší použitou strategií pro migraci jedinců. Výše popsáný algoritmus SOMA je typu AllToOne, což znamená, že

všichni jedinci v populaci migrují směrem k leaderovi. Další verze SOMA jsou popsány v [3].

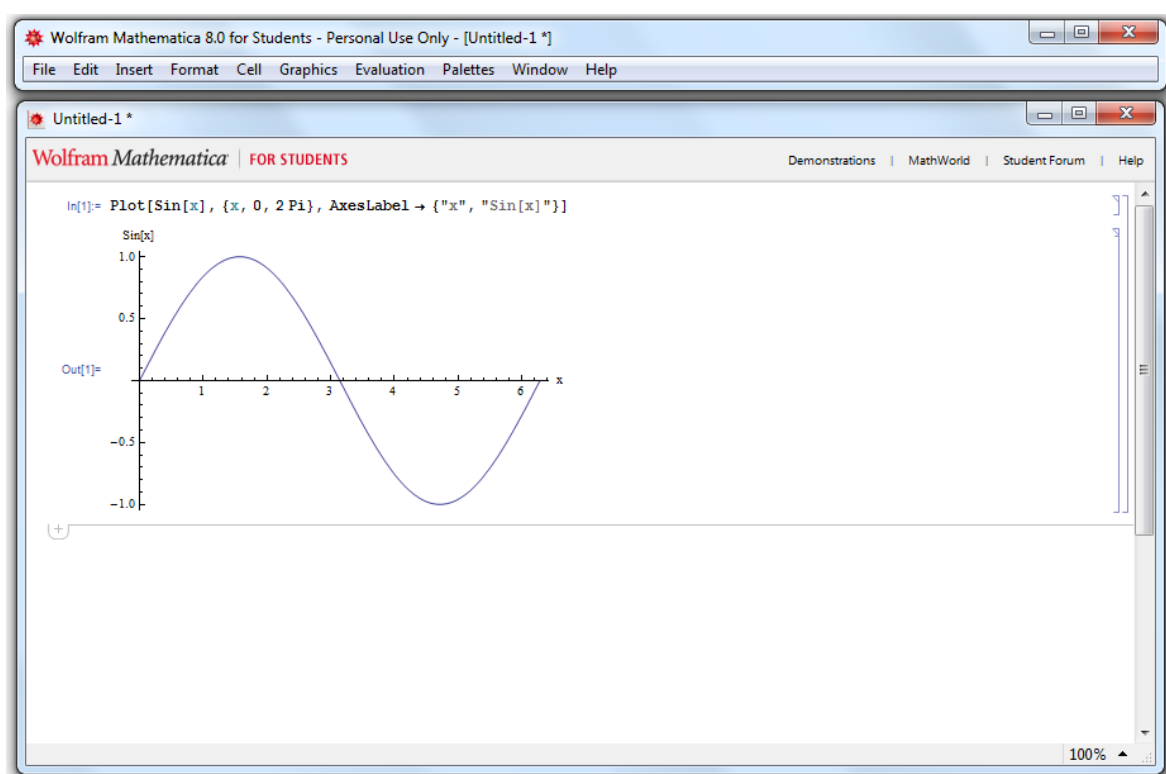
2.7.6 Závislost algoritmu SOMA na jejích parametrech

Bylo zjištěno, že pravděpodobnost nalezení globálního extrému pomocí algoritmu SOMA se zvyšuje s rostoucími hodnotami parametrů PathLength a Migrations, nebo když se snižuje hodnota parametru Step. Popřípadě lze tyto tři možnosti kombinovat.

3 WOLFRAM MATHEMATICA

Prostředí Mathematica vyvinuté společností Wolfram Research, Inc., představuje v současné době celosvětově rozšířený software, sloužící hlavně k provádění vědeckotechnických výpočtů a experimentů. Je vyhledáván a využíván odborníky hlavně díky vysoké přesnosti prováděných výpočtů a díky řadě jeho možností v oblasti vizualizace dat a postupů. Tento softwarový produkt je také velice dobře použitelný pro všechny možné typy statistických výpočtů a další řadu úloh.

Tento software lze ale také s výhodou využívat pro tvorbu výpočetních algoritmů hlavně díky velké spoustě výpočetních a dalších funkcí, které jsou připraveny pro přímé využití případným uživatelem. Vlastní programování v tomto prostředí probíhá v textovém rozhraní nazývaném notebook, které je velice podobné klasickým textovým editorům, jako je například poznámkový blok. Zvláštností tohoto prostředí ale je, že vlastní výstupy se rovněž vypisují do tohoto notebooku pod daný zdrojový kód.



Obrázek 13, grafická podoba prostředí Mathematica

Podrobná dokumentace k tomuto softwaru je dostupná přímo v jeho nápovědě a zároveň také na internetových stránkách společnosti Wolfram Research, Inc.

3.1 Možnosti softwaru Mathematica v oblasti GUI

V prostředí Mathematica lze vytvářet grafická uživatelská rozhraní využitím balíčku s názvem GUIKit. Tento balíček je běžnou součástí standardní distribuce softwaru Mathematica. GUIKit jako takový vychází z balíčku J/Link, který umožňuje uživatelům využívat třídy a funkce jazyka Java v prostředí Mathematica. GUIKit staví na balíčku J/Link a poskytuje možnost využití vyšší úrovně Mathematica syntaxí pro definování grafického uživatelského rozhraní. GUIKit tedy zjednodušuje tvorbu GUI za pomoci Mathematica syntaxí. Samotný balíček J/Link lze teoreticky také využít pro tvorbu GUI, ale tento způsob je značně pracnější než s použitím balíčku GUIKit. Navíc je zapotřebí velmi dobrá znalost jazyka Java. Za zmínku stojí ještě balíček s názvem Super Widget Package od Davida Baileyho, který sice není standardní součástí softwaru Mathematica, ale lze jej zdarma stáhnout z Internetu. Tento balíček do značné míry zjednodušuje tvorbu GUI v prostředí Mathematica hlavně tím, že práce s ním nevyžaduje žádné znalosti programovacího jazyka Java, na rozdíl od GUIKitu nebo J/Linku, kde je pro tvorbu plnohodnotného GUI znalost alespoň základů jazyka Java nezbytná [4].

3.1.1 Balíček GUIKit

Jak už bylo zmíněno v kapitole 3.1, balíček GUIKit je standardní součástí softwaru Mathematica a slouží pro tvorbu grafických uživatelských rozhraní (GUI) v tomto prostředí. Pro tvorbu a práci s jednoduchými objekty jako je například Button si lze víceméně vystačit se znalostmi programování v softwaru Mathematica s jeho nápovědou. Pro využití složitějších objektů jako je například ComboBox, pro tvorbu a práci s dialogy jako je například OpenFileDialog a také pro generování a odchyťování událostí a pro jejich obsluhu je již nutná znalost programovacího jazyka Java a také je třeba hledat dokumentaci a příklady mimo nápovědu Mathematica. Ta je totiž v tomto směru zpracována jen velmi stručně a velká část objektů a jejich možností zcela chybí. Níže je zpracována a popsána řada objektů z balíčku GUIKit. K těmto objektům jsou uvedeny i postupy pro práci s nimi. Základní příklady jednoduchých GUI zde uvedeny nejsou, protože je lze velmi jednoduše dohledat v nápovědě prostředí Mathematica. Tato práce je zaměřena hlavně na nezdokumentované části balíčku GUIKit.

3.1.1.1 Obecné vlastnosti objektů balíčku GUIKit

Objekt lze jednoduše definovat následující syntaxí.

```
Widget["typ_objektu", {"sada_parametrů_objektu"},  
"jméno_konkrétního_objektu_definované_uživatелеm"]
```

Za název jméno objektu je možné ještě doplnit funkci WidgetLayout, pomocí které lze definovat případné roztahování se objektu do stran (Stretching). Za zmínku stojí také parametr enabled, pomocí kterého lze daný objekt a interakci s ním povolit nebo zakázat. Poté tedy jednoduchý objekt může vypadat následovně.

```
Widget["Button", {"text"->"ok", "enabled"->True}, Name->"mojeTlacitkoOK",  
WidgetLayout -> {"Stretching" -> {True, False}}]
```

Výše uvedený kód tedy vytvoří objekt tlačítka, které na sobě bude mít text „ok“, a práce s ním bude povolena (půjde na něj klikat). Na tlačítka se bude lze odkazovat ve zdrojovém kódu pomocí názvu „mojeTlacitkoOK“ a v GUI se bude tlačítka roztahovat do prostoru v ose „x“. V ose „y“ bude mít základní velikost.

3.1.1.2 Práce s parametry objektů při běhu GUI

Pokud chceme změnit některý parametr u objektu v běžícím GUI, je nutné použít funkci SetPropertyValue. Řekněme tedy, že chceme u výše zmíněného tlačítka změnit jeho text z řetězce „ok“ na „start“. Použijeme tedy následující syntaxi.

```
SetPropertyValue[{"mojeTlacitkoOK", "text"}, "start"]
```

Pokud si tedy funkci SetPropertyValue blíže rozebereme tak zjistíme že.

```
SetPropertyValue[{"jméno_konkrétního_objektu_definované_uživatелеm", "název_parametru_který_chceme_měnit"}, "nova_hodnota_parametru"]
```

Je důležité si uvědomit, jaký typ proměnné používá parametr, který chceme měnit, abychom se například nepokoušeli nastavit parametr „enabled“ na „True“, ale abychom správně použili True.

Pokud chceme naopak zjistit hodnotu parametru konkrétního objektu v běžícím GUI, tak je nutné využít funkci PropertyValue. Zápis poté vypadá následovně.

```
PropertyValue[{"mojeTlacitkoOK", "text"}]
```

Dále jsou zdokumentovány objekty, které jsou využité v této práci.

3.1.1.3 Button

Pomocí tohoto objektu lze vytvořit standardní tlačítko pro potvrzování nabídek, spouštění skriptů a tak dále. Deklarace se provádí následovně.

```
Widget["Button", {"text" -> "start", }]
```

Když chceme přiřadit takovému tlačítku nějakou funkci, je nutné využít BindEvent.

```
Widget["Button", {"text" -> "start", BindEvent["action", Script[Print["akce"]]]}]
```

Všechn kód, který chceme vykonat, je tedy nutné vložit do Script.

3.1.1.4 Radiobutton

Klasický radiobutton pro volbu z možné nabídky možností.

```
{Widget["RadioButton", {"text" -> "a", "selected" -> True, BindEvent["action",  
Script[volba="a"]]}, Name-> "buttonA"],
```

```
Widget["RadioButton", {"text" -> "b", "selected" -> False, BindEvent["action",  
Script[volba="b"]]}, Name-> "buttonB"]},
```

```
Widget["ButtonGroup", {WidgetReference["buttonA"], WidgetReference["buttonB"]}]]
```

Výše uvedený kód vytvoří 2 radiobuttony které budou nastavovat hodnotu proměnné „volba“ na „a“ nebo „b“.

3.1.1.5 CheckBox

Jednoduchý objekt pro zaškrtnutí volby.

```
Widget["CheckBox", {"selected" -> True, BindEvent["Action", Script[ volba = 1 ]]}, Name-> "mujCheckBox"]
```

3.1.1.6 ComboBox

Objekt, ve kterém si lze ze seznamu vybrat požadovanou možnost.

```
Widget["ComboBox", {"items" -> Script[{"1", "2", "3", "4", "5"}], BindEvent["action",  
Script[volba=ToExpression[PropertyValue[{"mujComboBox", "selectedItem"}]]]},  
Name-> "mujComboBox"]
```

Výše uvedený kód vytvoří ComboBox, který nastaví proměnnou volba podle zvolené hodnoty ze seznamu.

3.1.1.7 TextField

Tento objekt slouží jako citovatelné textové pole pro vkládání údajů od uživatele nebo pro výpis dat z algoritmu.

```
Widget["TextField",{{"columns"->10, "editable"->True}, Name->"mojeTextovePole",
WidgetLayout -> {"Stretching" -> {True, False}}]
```

3.1.1.8 TextArea

Podobný objekt jako TextField, akorát lze nastavit i počet řádků. Parametr lineWrap slouží pro zalamování řádků.

```
Widget["TextArea", "rows"->6, "lineWrap"->True, Name->"mojeTextArea"]
```

3.1.1.9 Slider

Klasický posuvník pro nastavení hodnoty.

```
Widget["Slider",{{"minimum"->0,"maximum"->1000,"value"->0,
BindEvent["change",Script[cislo=ToExpression[PropertyValue[{"mujSlider","value"}]]]]
},Name->"mujSlider"]
```

Takto vytvořený Slider mění hodnotu proměnné „cislo“ od 0 do 1000.

3.1.1.10 MathPanel

Tento objekt slouží jako grafický výstup Mathematica syntaxí do GUI. Nedokáže ovšem korektně zobrazovat dynamické prvky jako je například Manipulate. Vše se zobrazí v podobě statického obrázku.

```
Widget["MathPanel",{{"preferredSize"->Widget["Dimension",{{"width"->50, "height"->50
}}},Name->"mujMathPanel",WidgetLayout->{"Stretching"->{False, False}}]
```

Výše uvedená syntaxe vytvoří MathPanel o rozměrech 50 x 50 pixelů. Pro vlastní úpravu jeho obsahu je možné využít následující funkci.

```
kresli[]:=Block[{$DisplayFunction=Identity},obsah=Plot[Sin[x],{x,0,2Pi}];
SetPropertyValue[{"mujMathPanel", "mathCommand"},ToString[obsah,InputForm]]];
```

Zavoláním funkce „kresli“ je poté v MathPanelu zobrazen graf funkce Sinus.

3.1.1.11 IndexedImagePanel

Tento objekt má podobu rastrové mřížky o volitelném rozměru. Klikáním na jednotlivé buňky této mřížky je poté možné měnit např. jejich barvu. Postup jak tohoto dosáhnout je uveden níže.

```
Widget["IndexedImagePanel",{    "preferredSize"->Widget["Dimension",{ "width"->130,
"height"->130}], "imageWidth"->3, "imageHeight"->3, "imagePixelSize"->10,
"imageColorMapSize"->2, "imageGrid"->True, "scaleImage"->True,
"preserveImageAspectRatio"->True, "imageColorComponents"-> Flatten[
  {{0,0,0},{255,255,255}}], "imagePixels"-> Flatten[ Table[1,{3},{3}]],
BindEvent["mouseClicked", Script[ mouseEventImage1 = WidgetReference["#"];
coordsImage1 = InvokeMethod[{"imagePanel","getImagePixelCoordinatesAt"},
mouseEventImage1]; If[(coordsImage1[[1]]>=0) &&( coordsImage1[[2]] >=0),
vImage1 = InvokeMethod[{"imagePanel","getImagePixelAt"}, mouseEventImage1];
InvokeMethod[{"imagePanel","setImagePixel"},coordsImage1,  If[vImage1  !=0,0,1],
True];;]], Name ->"imagePanel", WidgetLayout->{"Stretching"->{False, False}}]
```

Výše uvedený kód vytvoří mřížku 3 x 3 pixelů, ve které lze pomocí klikáním myši měnit barvu jednotlivých buněk z černé na bílou a naopak. Rozlišení této mřížky je dáno parametry imageWidth a imageHeight. Obsah celé mřížky je nastaven pomocí imagePixels a barevná paleta je definovaná parametrem imageColorComponents.

3.1.1.12 OpenFileDialog

OpenFileDialog lze využít pro načítání obsahu souborů ze spuštěného GUI. Dá se vyvolat následujícím kódem.

```
Widget["FileDialog",Name->"mOpenFileDialog"];
SetPropertyValue[{"mOpenFileDialog","multiSelectionEnabled"},False];
SetPropertyValue[{"mOpenFileDialog","fileSelectionMode"},PropertyValue[{"mOpenFile
Dialog","FILES_ONLY"}]];
pom=InvokeMethod[{"mOpenFileDialog","showOpenDialog"},Null];
```

```
If[pom===PropertyValue[{"mOpenFileDialog","APPROVE_OPTION"}],cesta=Property
Value[{PropertyValue[{"mOpenFileDialog","selectedFile"}],"path"}],Null];
```

V případě, že volba souboru proběhla korektně, tak je cesta k němu uložena v proměnné „cesta“.

3.1.1.13 PopupMenu

Toto vyskakovací menu se dá využít jako místní nabídka, kterou lze vyvolat nad řadou objektů. Velice užitečná je například u použití s objektem MathPanel.

```
Widget["PopupMenu",{
Widget["MenuItem",{text->"1",BindEvent["mousePressed", Script[volba = 1;]}],
Widget["MenuItem",{text->"2",BindEvent["mousePressed", Script[volba = 2;]}]},
Name->"mojePopupMenu"]
```

Výše uvedený kód však menu jen vytvoří, ale nezobrazí jej. Toto může být vyvoláno například kliknutím pravého tlačítka myši na konkrétní objekt. Jak toho docílit je ukázáno v následujícím kódu.

```
BindEvent[{"mujMathPanel","mousePressed"},
Script[
If[TrueQ[PropertyValue[{"#", "popupTrigger"}]],
InvokeMethod[{"mojePopupMenu","show"},PropertyValue[{"#", "component"}],Property
Value[{"#", "x"}],PropertyValue[{"#", "y"}]]],
BindEvent[{"mujMathPanel","mouseReleased"},
Script[
If[TrueQ[PropertyValue[{"#", "popupTrigger"}]],InvokeMethod[{"mojePopupMenu","sho
w"},PropertyValue[{"#", "component"}],PropertyValue[{"#", "x"}],PropertyValue[{"#", "y"
}]]]]];
```

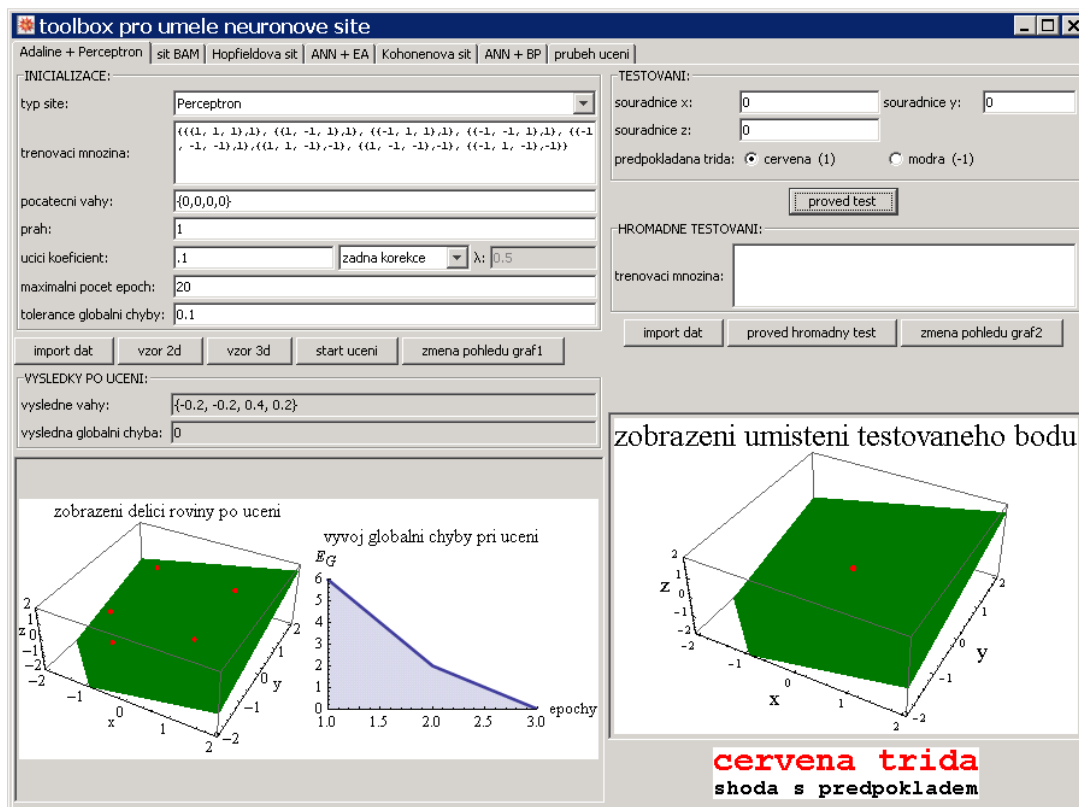
II. PRAKTICKÁ ČÁST

4 TOOLBOX PRO NEURONOVÉ SÍTĚ

V této části práce je popsán vytvořený toolbox pro umělé neuronové sítě. Jsou zde rozebrány jeho vlastnosti a také jsou následně jasně definovány jeho možnosti.

4.1 Obecný popis toolboxu

Vlastní toolbox obsahuje pět druhů neuronových sítí s různými učícími algoritmy, se kterými lze pracovat ve vytvořeném GUI. GUI se skládá z jednotlivých panelů (obrázek 14), mezi kterými se lze jednoduše přepínat. Každý panel je pro příslušný typ neuronové sítě. U každé sítě si může uživatel nastavit její specifické parametry a vyplnit trénovací množinu buď ručně, nebo importováním dat z textového souboru pomocí tlačítka „import dat“. Při plnění trénovací množiny je nutno dodržovat následující syntaxi. Jednotlivé prvky jsou zapsány jako vektory ve tvaru $\{\{\text{data prvku}\}, \text{předpoklad}\}$. Vytvořené prvky musí být také ohraničeny složenými závorkami jako klasický zápis vektoru v prostředí Mathematica, tedy $\{\{\{\text{data prvku1}\}, \text{předpoklad1}\}, \{\{\text{data prvku2}\}, \text{předpoklad2}\}\}$. Pokud je v GUI panelu neuronové sítě obsaženo i tlačítko „náhled“, lze jeho použitím získat grafickou podobu symbolů nebo barev v konkrétní množině prvků. Váhy jednotlivých vstupů neuronu musí být taktéž zapsány ve vektoru. Jakmile je vše nastaveno stiskem tlačítka „start učení“ započne proces optimalizace vah sítě. Po jeho dokončení jsou vypsány výsledné váhy a globální chyba, se kterou skončilo učení. Následně je možné síť testovat vyplněním trénovací množiny (opět ručně nebo importem ze souboru) a stiskem tlačítka „proved test“. Po dokončení testu jsou výsledky přehledně zobrazeny v GUI. V celém GUI se nevyskytuje diakritika z důvodu zpětné kompatibility zobrazení s prostředím Mathematica 7.



Obrázek 14, GUI prostředí pro vytvořený toolbox

4.2 Upřesňující popis jednotlivých částí GUI

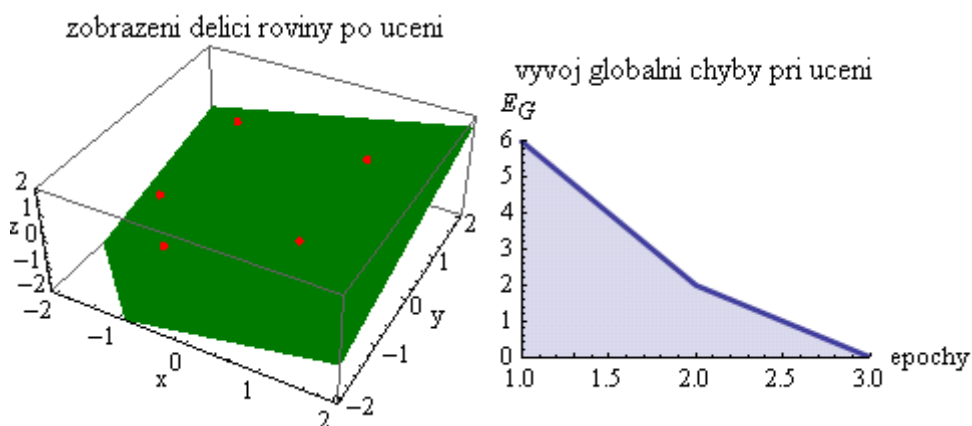
4.2.1 Adaline + Perceptron

Tento panel je zobrazen ihned po spuštění GUI a umožňuje práci s neuronovou sítí tvořenou jedním neuronem typu Perceptron nebo Adaline. Tato část GUI je primárně vytvořená pro klasifikační úlohy ve 2D a 3D prostoru, ale neuronová síť jako taková je samozřejmě schopná naučit se na jakkoliv rozměrná data ovšem s tím omezením, že nebudou zobrazeny grafy vyšších dimenzí. Jako nastavitelné parametry vystupují: typ sítě, trénovací množina, počáteční váhy, práh, učící koeficient (u toho lze také případně nastavit použití absolutní nebo zlomkové korekce), maximální počet epoch a tolerance globální chyby.

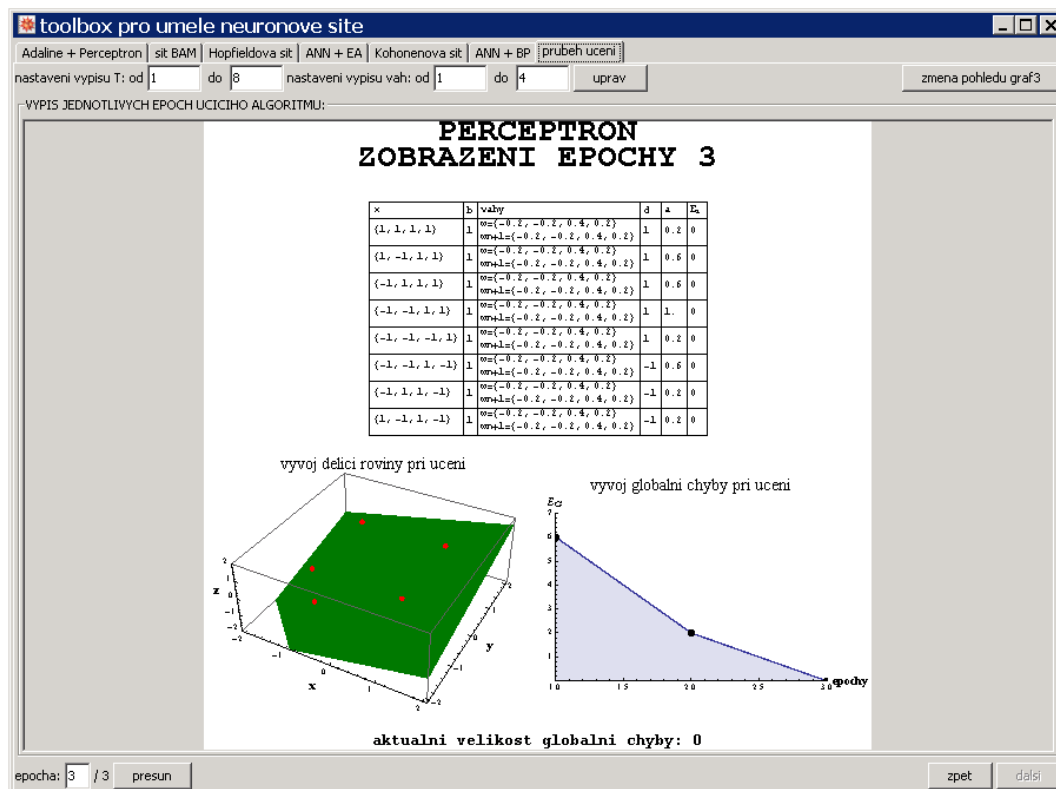
INITIALIZACE:	
typ neuronky:	Perceptron
trenovací množina:	$\{(1, 1), (1), \{(1, -1), 1\}, \{(0, -1), 1\}, \{(-1, -1), -1\}, \{(-1, 1), -1\}, \{(0, 1), -1\}\}$
pocateční vahy:	{0,0,0}
prah:	1
učicí koeficient:	.1 zadna korekce λ: 0.5
maximalní počet epoch:	20
tolerance globální chyby:	0.1

Obrázek 15, Nastavitelné parametry pro Adaline a Perceptron

Po naučení síť se zobrazí graf, který ukazuje dělicí hranici mezi třídami po učení a také graf zobrazující vývoj globální chyby během jednotlivých epoch učícího procesu (obrázek 16). Zároveň se na panelu s názvem „průběh učení“ zobrazí podrobný postup celého učícího procesu, který lze procházet epochu po epoše. Tímto způsobem lze sledovat vývoj globální chyby, dělicí hranice mezi třídami a jednotlivé přepočty vah k získání kompletního přehledu o všech operacích, které během učícího procesu proběhly a o změnách, které způsobily (obrázek 17).



Obrázek 16, Zobrazení dělicí hranice a vývoje globální chyby



Obrázek 17, Zobrazení průběhu učení sítě

Naučenou síť je možné poté testovat dvěma způsoby: způsobem popsáným v části 4.1. nebo prvek po prvku v níže zobrazené části GUI (obrázek 18).

TESTOVÁNÍ:

souradnice x: souradnice y:

souradnice z:

predpokladana trida: ☒ cervena (1) ☐ modra (-1)

Obrázek 18, Testování Perceptronu a Adaline prvek po prvku

U všech grafických výstupů této sítě (i v části „průběh učení“) lze jednotlivé „obrázky“ pomocí místní nabídky přibližovat nebo oddalovat, měnit natočení u 3D grafů atd.

4.2.2 Síť BAM

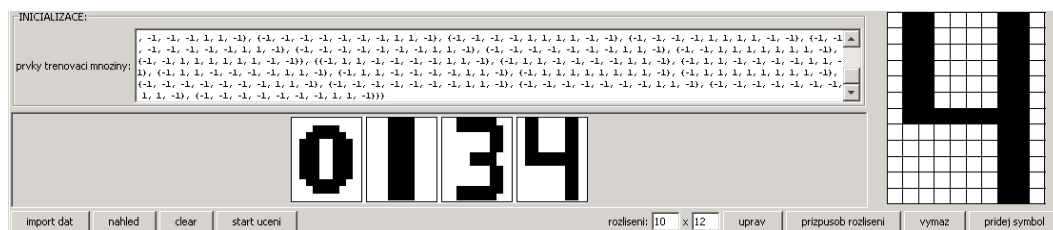
Druhý panel v pořadí obsahuje rozhraní pro práci s dvousměrnou asociativní pamětí (BAM) – konkrétně s její diskretní verzí. U této sítě lze nastavit obsah dvou vzájemně asociovaných vektorů, které slouží jako trénovací množina. První vektor je přitom využit pro uchování grafického vzoru, zatímco druhý je jeho asociovaným vektorem. Rozložení

pixelů grafického vzoru lze přitom v GUI jednoduše „naklikat“ a vložit do vektoru číslo jedna (toto takzvané „naklikávání“ symbolů je podrobněji popsáno v kapitole 4.2.3.).

Dále je tuto síť možno testovat předkládáním na její vstup vektorem z konkrétní dvojice a sledováním výstupu ze sítě. Toto testování funguje oběma směry.

4.2.3 Hopfieldova síť

Dalším typem sítě v GUI je Hopfieldova síť. Na jejím panelu se pracuje s obecnými funkcemi (tlačítka), které jsou popsány v kapitole 4.1. Je zde ale i speciální postup ukládání a přidávání prvků do trénovací nebo testovací množiny. Síť totiž pracuje s grafickými symboly, které lze vytvořit klikáním myši do připravené mřížky v GUI (obrázek 19).

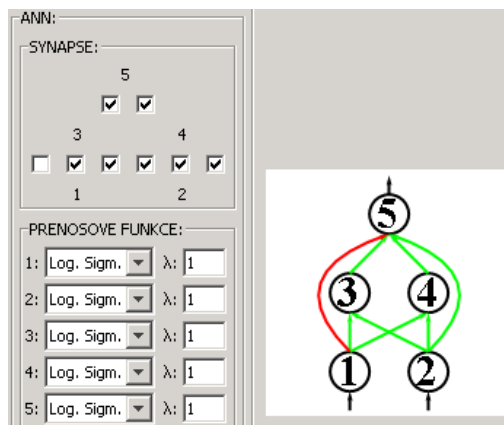


Obrázek 19, Nastavení Hopfieldovy sítě v GUI

V rámci testování jsou poté síti předkládány neznámé vzory (např. poškozené původní prvky) a ta k nim vrací vzory podle autoasociace.

4.2.4 Učení ANN pomocí evolučních algoritmů

V této části práce je věnována pozornost možnosti využití evolučních algoritmů jako učicích technik pro neuronové sítě. Základ tvoří neuronová síť s jednou skrytou vrstvou o dvou neuronech, dvou neuronech na vstupu a jedním na výstupu (obrázek 20). Je zde možno upravovat její topologii aktivováním nebo zakazováním jednotlivých spojení mezi neurony v síti. Zároveň je také možno zvlášť u každého neuronu nastavit jeho přenosovou funkci a u té její případný parametr. Výsledkem je řada možných návrhů sítí, které lze tímto postupem vytvořit a pro které by „klasické“ metody učení byly neúčinné nebo dokonce nerealizovatelné.



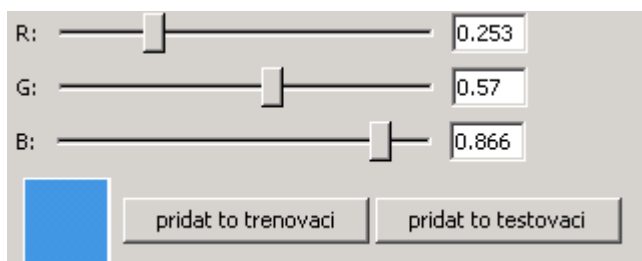
Obrázek 20, Editace topologie neuronové sítě

U výše navržené neuronové sítě lze poté zvolit konkrétní evoluční algoritmus (v současnosti DE nebo SOMA), který má být použitý pro optimalizaci jejích vah. U obou evolučních algoritmů lze samozřejmě v GUI nastavit jejich standardní parametry. Dále je síť nastavena a testována podle obecného postupu popsáno v 4.1.

4.2.5 Samoorganizující se mapa (SOM) - Kohonenova síť

GUI pro Kohonenovu síť je navrženo pro trénování sítě za účelem klasifikace barevných vzorů modelu RGB. Lze ji samozřejmě použít prakticky na jakýkoliv další typ problému, který je Kohonenova síť schopna řešit, ovšem grafický náhled sítě a prvků v trénovací množině nebude v GUI využitelný. Mimo obecného postupu pro tvorbu množiny prvků lze využít i část GUI určenou k „namíchání“ požadovaného barevného vzoru (obrázek 21) a ten poté přidat do testovací nebo trénovací množiny.

Při učení sítě je použito kruhové okolí, které se v průběhu učení postupně zmenšuje. Po naučení sítě jsou zobrazeny vytvořené shluky jednotlivých tříd barevných vzorů přímo do GUI.



Obrázek 21, Tvorba vlastních barevných vzorů



Obrázek 22, Náhled na vzory v trénovací množině



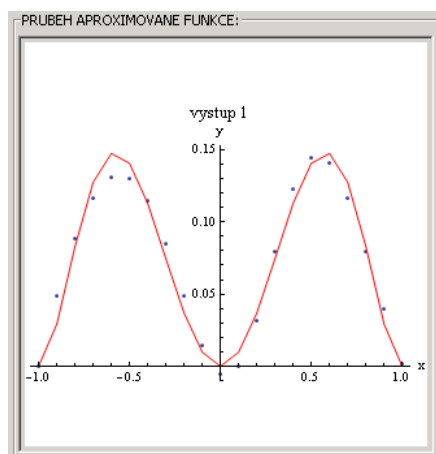
Obrázek 23, Zobrazení naučené Kohonenovy sítě

Naučenou síť lze poté testovat předkládáním barevných vzorů na její vstup a sledováním odpovědi sítě. Testování této sítě probíhá neadaptivně.

4.2.6 Učení ANN pomocí algoritmu backpropagation

Algoritmus Backpropagation je jedním ze standardních učících algoritmů pro vícevrstvé neuronové sítě a na záložce v GUI nazvané „ANN+BP“ s ním lze pracovat. Vlastní síť může být složena z jedné až čtyř vrstev s měnitelnými váhami – tudíž z nula až tří skrytých vrstev a každá tato vrstva může obsahovat libovolný počet neuronů. Návrh topologie sítě vkládá uživatel do GUI ve formě vektoru např. {3,2,1}, který udává, že síť se skládá ze 3 vrstev, kde ve vstupní části jsou 3 neurony a na výstupu je jeden neuron. Pro danou síť lze poté zvolit přenosovou funkci a její případný parametr. Přenosovou funkci výstupní části sítě lze volit jinou než u zbytku sítě. Například v síti bude používána přenosová funkce logistická sigmoida, zatímco ve výstupní části to bude funkce lineární.

V případě, že uživatel řeší problém aproximace průběhu funkce, je možné využít i graf, který porovnává shodu funkce generované neuronovou sítí s trénovací funkcí (obrázek 24).



Obrázek 24, Porovnání bodů trénovací množiny a bodů generovaných neuronovou sítí

Tuto síť a její GUI lze ovšem pohodlně využít také pro klasifikaci, predikci, filtraci atd.

ZÁVĚR

Výše představený toolbox umožňuje přímou práci s neuronovými sítěmi, které jsou v něm již naprogramovány. Nastavením jejich parametrů a vhodnou volbou trénovací množiny je možné je přizpůsobit k řešení problémů z oblasti aproximace funkcí, klasifikace dat, identifikace a filtrování signálů, optimalizace atd. Tento toolbox společně s GUI, které k němu bylo vypracováno, umožňuje jednoduchou a efektivní práci s uvedenými neuronovými sítěmi a zároveň lze využívat výpočetní jádro softwaru Mathematica spolu s jeho možnostmi a funkcemi v oblasti vizualizace. Z toho plyne, že lze celou tuto práci kvalitně využívat při výuce jak pro demonstrační účely, tak i pro vypracovávání případných úkolů samotnými studenty. Vzhledem k intuitivnímu a jednoduchému GUI není nutná znalost programování v prostředí Mathematica. Zároveň je však možné využít tuto aplikaci pro vědeckou práci a pro rychlé a jednoduché testování a analýzu chování jednotlivých neuronových sítí na konkrétních problémech.

ZÁVĚR V ANGLIČTINĚ

Presented toolbox allows direct work with neural networks which are its part. They can be modified to solve problems like approximation, classification, filtration, optimalization and identification by setting their parameters and by appropriate choice of their training set. This toolbox and its GUI allows simple and effective work with presented neural networks and also allows using computational core and visualization possibilities of Mathematica software. It follows that this entire work can be used for education and demonstrations and for production of tasks by students. Due to intuitional and simple GUI there is no need of knowledge of programming in Mathematica environment. It is also possible to use this application for scientific work and for quick and simple testing and for behavior analysis of neural networks on specific problems.

SEZNAM POUŽITÉ LITERATURY

- [1] Šíma J., Neruda R.: Teoretické otázky neuronových sítí, MATFYZPRESS, 1996, ISBN 8085863189
- [2] ZELINKA I.: Umělá inteligence I, VUT Brno, 1998, ISBN 8021411635
- [3] Zelinka, I., Oplatková, Z., Ošmera, P., Šeda, M., Včelař, F.: Evoluční výpočetní techniky - principy a aplikace, BEN - technická literatura, Ben - technická literatura, Praha, 2008, ISBN 8073002183
- [4] Wolfram Mathematica 8 Documentation Center
- [5] ŠNOREK M., JIŘINA M.: Neuronové sítě a neuropočítače, ČVUT, 1996, ISBN 800101455X
- [6] BÍLA J.: Umělá inteligence a neuronové sítě v aplikacích, ČVUT, 1996, ISBN 8001012751
- [7] BOSE N.K., LIANG P.: Neural Network Fundamentals with Graphs, Algorithms, and Applications, McGraw-Hill Series in Electrical and Computer Engineering, 1996, ISBN 0070066183
- [8] FREEMAN J. A.: Simulating Neural Networks with Mathematica, Addison-Wesley Publishing Company, 1994, ISBN 020156629X
- [9] NOVÁK M., FABER J., KUFUDAKI O.: Neuronové sítě a informační systémy živých organismů, Grada, 1993, ISBN 8058424959
- [10] Volná Eva, Neuronové sítě 2, 2008

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

a	Aktivační funkce neuronu
ADALINE	ADaptive LINEar Element
AllToOne	Název varianty algoritmu SOMA
ANN	Artificial Neural Network (Umělá neuronová síť)
b	Bias (práh neuronu)
BAM	Bidirectional Associative Memory (Dvousměrná asociativní paměť)
BP	BackPropagation
c	Učící koeficient
CR	Práh křížení
d	Předpokládaný výstup neuronu
d	Vzdálenost (odlišnost) neuronu od předloženého vzoru
D	Dimenze problému
DE	Differential Evolution (Diferenciální evoluce)
DERandBin1	Název varianty Diferenciální evoluce
E	Energetická funkce
F	Mutační konstanta
f(a)	Přenosová funkce neuronu
g(x)	Funkce udávající vliv neuronu na své okolí
GUI	Graphical User Interface (Grafické uživatelské rozhraní)
GUIKit	Název balíčku pro tvorbu GUI v softwaru Mathematica
J/Link	Název balíčku pro tvorbu GUI v softwaru Mathematica
MinDiv	Minimální diverzita
NP	Počet jedinců v populaci
P	Kapacita neuronové sítě

PathLength	Délka trasy migrujícího jedince
PopSize	Velikost populace
PRT	Perturbace
PRTVector	Perturbační vektor
RGB	Druh barevného modelu
SOM	Self-Organizing Maps (Samoorganizující se mapy)
SOMA	Self-Organizing Migration Algorithm (Samoorganizující se migrační algoritmus)
Step	Délka kroku migrujícího jedince
v	Šumový vektor
w	Váha vstupu neuronu
x	Vstup do neuronu
y	Výstup z neuronu
Δw	Přírůstek váhy
δ	Odchylka výstupu neuronu
μ	Momentum
η	Učící koeficient

SEZNAM OBRÁZKŮ

<i>Obrázek 1, zobrazení matematického modelu neuronu</i>	<i>13</i>
<i>Obrázek 2, průběh Logistické sigmoidy pro $\lambda=1$</i>	<i>14</i>
<i>Obrázek 3, průběh Skokové funkce binární pro $b=0$</i>	<i>14</i>
<i>Obrázek 4, průběh Skokové funkce bipolární pro $b=0$</i>	<i>15</i>
<i>Obrázek 5, průběh funkce Hyperbolický tangens</i>	<i>15</i>
<i>Obrázek 6, průběh Lineární funkce pro $k=1$</i>	<i>16</i>
<i>Obrázek 7, ukázka topologie sítě Perceptron</i>	<i>18</i>
<i>Obrázek 8, příklad lineárně separabilního problému</i>	<i>18</i>
<i>Obrázek 9, příklad lineárně neseperabilního problému</i>	<i>19</i>
<i>Obrázek 10, ukázka topologie Hopfieldovy sítě</i>	<i>21</i>
<i>Obrázek 11, ukázka topologie sítě BAM</i>	<i>24</i>
<i>Obrázek 12, ukázka topologie Kohonenovy sítě</i>	<i>27</i>
<i>Obrázek 13, grafická podoba prostředí Mathematica</i>	<i>39</i>
<i>Obrázek 14, GUI prostředí pro vytvořený toolbox</i>	<i>48</i>
<i>Obrázek 15, Nastavitelné parametry pro Adaline a Perceptron</i>	<i>49</i>
<i>Obrázek 16, Zobrazení dělicí hranice a vývoje globální chyby</i>	<i>49</i>
<i>Obrázek 17, Zobrazení průběhu učení sítě</i>	<i>50</i>
<i>Obrázek 18, Testování Perceptronu a Adaline prvek po prvku</i>	<i>50</i>
<i>Obrázek 19, Nastavení Hopfieldovy sítě v GUI</i>	<i>51</i>
<i>Obrázek 20, Editace topologie neuronové sítě</i>	<i>52</i>
<i>Obrázek 21, Tvorba vlastních barevných vzorů</i>	<i>52</i>
<i>Obrázek 22, Náhled na vzory v trénovací množině</i>	<i>53</i>
<i>Obrázek 23, Zobrazení naučené Kohonenovy sítě</i>	<i>53</i>
<i>Obrázek 24, Porovnání bodů trénovací množiny a bodů generovaných neuronovou sítí ...</i>	<i>54</i>

SEZNAM TABULEK

<i>Tabulka 1, lineárně neseparabilní problém XOR.....</i>	<i>19</i>
---	-----------

SEZNAM PŘÍLOH

Příloha P I: Manuál pro práci s vybranými částmi toolboxu a jeho GUI

Příloha P II: Elektronické přílohy a verze bakalářské práce na CD

PŘÍLOHA P I: MANUÁL PRO PRÁCI S VYBRANÝMI ČÁSTMI TOOLBOXU A JEHO GUI

1. Adaline + Perceptron

Inicializace

Trénovací množinu je nutné vyplnit formou vektoru, ve kterém představují jednotlivé vnitřní vektory prvky trénovací množiny.

Obecný zápis:

$\{\{\{argument1, argument2, \dots\}, předpokládáný_výstup\}\}$

Konkrétní příklad pro klasifikaci ve 3D:

$\{\{\{1, 1, 1\}, 1\}, \{\{1, -1, 1\}, 1\}, \dots\}$

Mimo manuální tvorbu trénovací množiny lze také využít import z textového souboru, který musí obsahovat pouze vektor trénovací množiny.

Dále je třeba definovat počáteční váhy včetně váhy prahu. Toto je opět nutné udělat pomocí vektoru.

Obecný zápis:

$\{váha_vstupu1, váha_vstupu2, \dots, váha_prahu\}$

Konkrétní příklad pro klasifikaci ve 3D:

$\{0,0,0,0\}$

Po dokončení učení lze manipulovat s grafy pomocí vyskakovacích menu (aktivují se pravým tlačítkem myši nad daným MathPanelem). V sekci “průběh učení” je možné detailně studovat učící proces.

Testování

Do příslušných textových polí je nutné vyplnit hodnoty jednotlivých souřadnic a dále zvolit předpokládanou třídu. Poté stačí jen stisknout tlačítko „proved test“.

Hromadné testování

Je nutné vyplnit testovací množinu stejným postupem jako u trénovací množiny. Poté stisknout tlačítko „proved hromadny test“.

2. Síť BAM

Inicializace

Trénovací množina se skládá z dvojice párových vektorů „vstupní vektor 1 (obrázek)“ a „vstupní vektor 2“. Pro vkládání grafických vzorů je určen vektor 1. Celý obrázek se poté zapíše ve formě vektoru, ve kterém zastupuje každý vnitřní vektor jeden řádek v obrázku. Hodnota 1 zastupuje černý pixel zatímco hodnota -1 pixel bílý.

Obecný zápis:

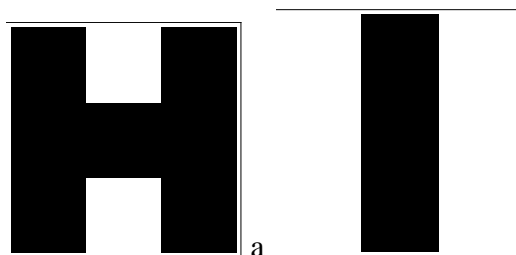
Vektor 1: $\{\{\{radek1_sloupec1, radek1_sloupec2, radek1_sloupec3, \dots\}, \{radek2_sloupec1, radek2_sloupec2, radek2_sloupec3, \dots\}, \{\dots\}\}, \{grafický_vzor2\}, \dots\}$

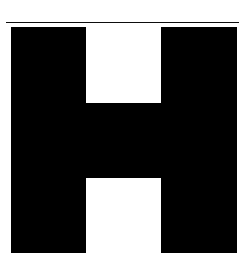
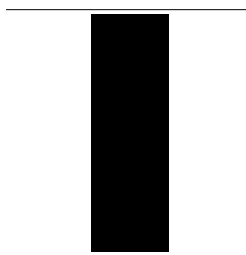
Vektor 2: $\{párový_vektor1, párový_vektor2, \dots\}$

Konkrétní příklad:

Vektor 1: $\{\{\{1,-1,1\}, \{1,1,1\}, \{1,-1,1\}\}, \{\{-1,1,-1\}, \{-1,1,-1\}, \{-1,1,-1\}\}\}$

Vektor 2: $\{\{-1,1\}, \{1,1\}\}$



Ve vektoru 1 jsou uloženy grafické vzory  a .

Kromě ručního plnění lze využít i panel pro tvorbu grafických vzorů, který je součástí GUI. Lze také importovat data z textového souboru, ve kterém musí být dodržen následující zápis.

$\{vektor1, vektor2\}$

Testování

Do vstupní části je nutné opět vložit sadu testovacích prvků ve formě vektoru a stiskem tlačítka „testuj“ proběhne test.

3. Hopfieldova síť

Inicializace

Trénovací množina je tvořena stejně jako vektor 1 u sítě BAM. Tlačítko „uprav rozlišení“ upraví rozlišení mřížky pro tvorbu grafických vzorů, podle rozlišení prvků v trénovací množině. Import dat očekává textový soubor, ve kterém je uložen vektor s jednotlivými grafickými vzory.

Testování

Zde stojí za zmínku, že do mřížky, která slouží pro tvorbu grafických vzorů pro testovací množinu, lze využít tlačítko „načti“. To způsobí, že do této mřížky bude načten prvek z trénovací množiny, jehož pořadí odpovídá číslu v textovém poli vedle nápisu „načti vzor číslo“. Tlačítko „načti horní vzor“ zkopíruje obsah horní mřížky do spodní. Tlačítka se šipkami umožňují nahrávat do dolní mřížky vzory z trénovací množiny jeden za druhým.

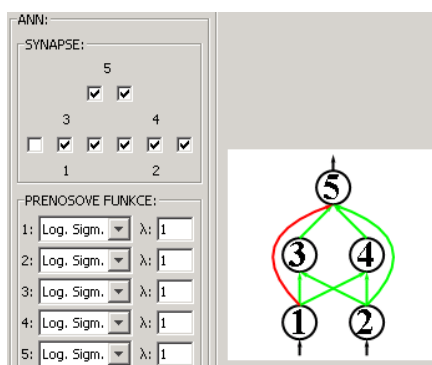
Po provedení testu lze zobrazovat jednotlivé výstupy zvolením čísla prvku v textovém poli u nápisu „zobraz výstup číslo“ a stiskem tlačítka „ok“.

4. ANN + EA

Inicializace

Trénovací množina se tvoří stejně jako u Adaline a Perceptronu. Stejně funguje i import dat.

Dále je třeba nastavit požadovanou topologii sítě a jednotlivé přenosové funkce. Jednotlivé synapse lze aktivovat nebo vypnout pomocí sady checkboxů. Jednotlivá čísla odpovídají číslům u neuronů v obrázku. Poté je ještě třeba nastavit požadované přenosové funkce a jejich parametry. Pro názornost je znovu uveden obrázek části GUI pro nastavení sítě.



Počáteční váhy mají opět formu vektoru, kde každý jeho vnitřní vektor představuje jednu vrstvu sítě. Pro vysvětlení je uveden příklad.

Obecný zápis:

$$\{\{\{vstup1, vstup2, \dots\}, \{váhy_neuronu2\}\}, \{\{ váhy_neuronu3\}, \{ váhy_neuronu4\}\}, \{\{váhy_neuronu5\}\}\}$$

Konkrétní příklad zápisu pro topologii na výše uvedeném obrázku, kde prvky trénovací množiny mají jeden argument:

$$\{\{\{1,1\}, \{1,1\}\}, \{\{1,1,1\}, \{1,1,1\}\}, \{\{1,1,1,1\}\}\}$$

Na zápis počátečních vah je nutné brát zřetel, aby odpovídaly počtu aktivních synapsí.

Testování

Zápis testovací množiny je stejný jako u hromadného testování Adaline a Perceptronu.

4. Kohonenova síť

Inicializace

Trénovací množina je tvořena vektory, které představují jednotlivé prvky. Pro klasifikaci barevných vzorů je tvořena vektory ve tvaru {R, G, B}. Vlastní zápis tedy vypadá následovně.

Obecný zápis:

$$\{\{Red1, Green1, Blue1\}, \{ Red2, Green2, Blue2\}, \{prvek2\}, \{prvek3\}, \dots\}$$

Konkrétní příklad:

$$\{\{1,0,0\}, \{0,1,0\}, \{0,0,1\}\}$$

Tento zápis je platný pro následující sadu barevných vzorů.



5. ANN + BP

Inicializace

Trénovací množina se opět zapisuje jako u Adaline a Perceptronu. Vlastní návrh topologie neuronové sítě probíhá zápisem vektoru, ve kterém jsou uvedeny počty neuronů v jednotlivých vrstvách počínaje vstupní. Pro jednu skrytou vrstvu o třech neuronech a výstupní vrstvu s jedním neuronem platí zápis $\{3,1\}$. Maximální počet zapsaných tímto vektorem je čtyři. Tudíž lze maximálně vytvořit síť o třech skrytých vrstvách a jedné výstupní $\{3,2,2,1\}$. Počty neuronů v jednotlivých vrstvách omezeny nejsou a závisí jen na úvaze uživatele.

Testování

Testovací množina se opět zapisuje stejně jako u hromadného testování Adaline a Perceptronu. V případě, že řešeným problémem je aproximace průběhu funkce lze zobrazit graf porovnání tlačítkem „zobraz prubeħ“. Tlačítka v podobě šipek v dolní části GUI slouží pro přepínání mezi jednotlivými výstupními neurony.