Toolbox pro neuronové sítě pro prostředí Mathematica

Toolbox for Neural Networks in the environment Mathematica

Bc. Martin Macháč

Diplomová práce
2009

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2008/2009

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin MACHÁČ**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Toolbox pro neuronové sítě pro prostředí Mathematica**

Zásady pro vypracování:

Cílem je vytvořit toolbox neuronových sítí pro prostředí Mathematica a výuku v předmětu Metody umělé inteligence.

1. **Seznamte se s neuronovými sítěmi, typy a jejich algoritmy učení.**
2. **Naprogramujte s prostředí Mathematica vybrané typy.**
3. **U naprogramovaných typů připravte ukázkové úlohy pro laboratoře v předmětu Metody umělé inteligence.**
4. **Připravte i možnost použití přes knihovny v prostředí Mathematica.**
5. **Závěr.**

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **Šnorek M., Jiřina M. 1996, Neuronové sítě a neuropočítače, ČVUT, ISBN 80-01-01455-X, 1996**
2. **Bíla J. 1996, Umělá inteligence a neuronové sítě v aplikacích, ČVUT, ISBN 80-01-01275-1, 1996**
3. **Zelinka I. 1998, Umělá inteligence I, VUT Brno, ISBN 80-214-1163-5, 1998**
4. **The Mathematica Book, manuál softwaru Mathematica**

Vedoucí diplomové práce: **Ing. Zuzana Oplatková, Ph.D.**
Ústav aplikované informatiky

Datum zadání diplomové práce: **20. února 2009**

Termín odevzdání diplomové práce: **27. května 2009**

Ve Zlíně dne 13. února 2009

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Tato diplomová práce se zabývá problematikou umělé inteligence, a to se zaměřením na umělé neuronové sítě. Cílem práce je představit vypracovaný programový balíček (toolbox) v prostředí Mathematica od společnosti Wolfram Research. Toolbox má sloužit jako učební pomůcka pro předmět Metody umělé inteligence vyučovaný na Fakultě aplikované informatiky Univerzity Tomáše Bati ve Zlíně.

V teoretické části práce je čtenář uveden do problematiky neuronových sítí, jejich historie, struktury, funkcionality a využití v běžném životě. Praktická část zahrnuje konkrétní vypracované sítě, jejich popis a ukázky použití. Jmenovitě se jedná o neuronovou síť Perceptron, Hopfieldovu síť a vícevrstvou síť s dopředným šířením signálu.

Klíčová slova:

umělá neuronová síť, trénovací množina, Perceptron, neuronová síť s dopřendým šířením signálu, Hopfieldova síť, Back-Propagation, Mathematica

ABSTRACT

This diploma thesis deals with issues of artificial intelligence, especially artificial neural networks are highlighted. Main objective of the thesis is to introduce program package (toolbox) evolved in the environment Mathematica from the Wolfram Research Company. The toolbox is instrumental towards the teaching in Methods of artificial intelligence which is taught at the Faculty of Applied Informatics in the Thomas Bata University in Zlín.

The theoretical part of this document refers about the main issues of artificial neural networks, their history, structure, functionality and applications in common life. The practical part includes particular evolved networks, their description and the examples of use. Concretely the Perceptron, Hopfield network and feed-forward network is explained.

Keywords:

artificial neural network, the training set, the Perceptron, feed-forward neural network, Hopfield network, Back-Propagation, Mathematica

Tímto bych chtěl poděkovat vedoucí diplomové práce Ing. Zuzaně Oplatkové, Ph.D., za cenné připomínky, trpělivost a čas, který mi věnovala při přípravě této práce.

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.

Ve Zlíně ……

Podpis diplomanta

OBSAH

ÚVOD

Pojem umělá inteligence (UI) můžeme zejména v dnešní, technologicky pokročilé době slyšet ze všech stran. Tento pojem ovšem není přesně vymezen. Můžeme si pod ním obecně představit obor informatiky, který se zabývá tvorbou mechanizmů vykazujících známky inteligentního chování. Jiná, přístupnější definice zní, že umělá inteligence se zabývá tím, jak počítačově řešit úlohy, které dnes zatím lépe zvládají lidé. Témata, kterými se dnes obor umělá inteligence obvykle zabývá, jsou výsledkem snahy o napodobení různých aspektů (nejen) lidského myšlení. Proto se v poslední době pod termínem umělá inteligence skrývají věci jako neuronové sítě, konečné automaty, algoritmy pro plánování a predikci, ale i multimediální operace, jako je zpracování přirozeného jazyka, obrazu nebo videa, řízení v počítačových hrách a podobně. Zajímavým odvětvím z oboru umělé inteligence jsou zejména umělé neuronové sítě (NS). Představují jedny z výpočetních modelů, jejichž vzorem je chování biologických struktur za účelem efektivnějších postupů ve výpočetních úlohách.

Důvod vzniku této práce spočívá v potřebě výpočetní pomůcky pro výuku předmětu magisterského studia Metody umělé inteligence vyučovaném na Fakultě aplikované informatiky Univerzity Tomáše Bati ve Zlíně. Náplní předmětu je naučit posluchače aplikovat metody neuronových sítí na různé typy reálných problémů – konkrétně se jedná o použití NS v oblasti klasifikace, asociace, aproximace dat a dalších. Podobná pomůcka vyhovující těmto požadavkům se již v předmětu používá, avšak existence nového a vlastního řešení přináší výhody z hlediska konkrétní přizpůsobitelnosti a případné rozšiřitelnosti tohoto nástroje do budoucna. Pomůcka je vypracována jako takzvaný toolbox (balíček) v prostředí Mathematica od společnosti Wolfram Research. Tento výkonný a robustní nástroj je vhodný nejen pro matematické operace, ale efektivně zvládá také práci s neuronovými sítěmi. Více bude o samotném prostředí Mathematica pojednáno dále v textu v samostatné kapitole.

I. TEORETICKÁ ČÁST

1 NEURONOVÉ SÍTĚ

Kolem umělých neuronových sítí je soustředěn velký zájem nejen odborné veřejnosti. Simulace těchto sítí dosahují překvapivě velmi slušné výsledky. Jak už bylo řečeno, umělé neuronové sítě (dále jen neuronové sítě) jsou zjednodušené matematické modely nervových systémů živých organismů. Dokazují tedy schopnost lidského myšlení, a to učit se. [1]

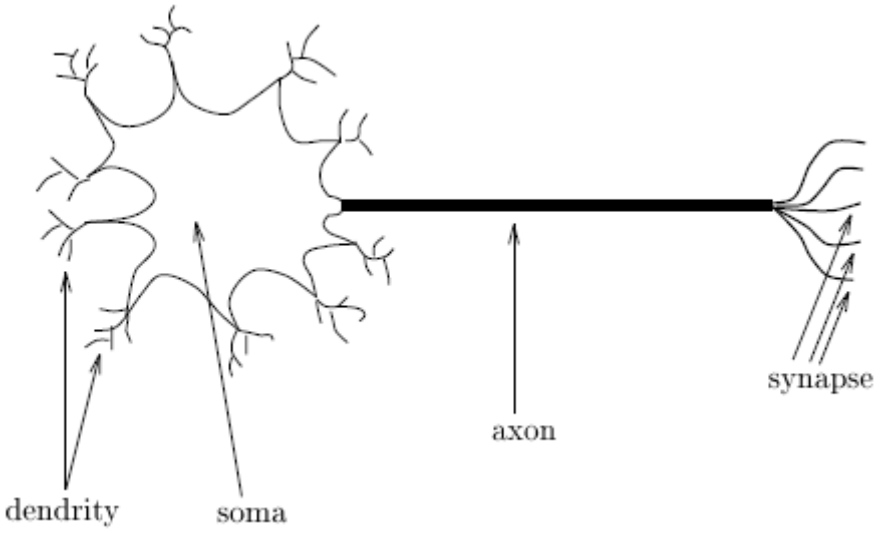
1.1 Nástin historie

Historie vzniku a stále pokračujícího vývoje neuronových sítí spadá do první poloviny 20. století, kdy byla publikována první práce o neuronech a jejich modelech Američanem W. S. McCullochem. Ve 40. letech tohoto století pak se svým studentem W. Pittsem vypracoval model neuronu, který je prakticky používán dodnes. Na základě těchto výsledků v roce 1958 F.Rosenblatt vytvořil první funkční perceptronovou síť. Tato síť měla ale určitou nevýhodu, a to že neuměla klasifikovat lineárně neseparabilní problémy. Díky tomuto faktu zájem o neuronové sítě na dlouhou dobu opadl.

Až v polovině 80. let došlo díky skalním příznivcům problematiky neuronových sítí k jejich pomyslné renesanci, a to když se objevil první model vícevrstvé neuronové sítě, který problém lineární neseparability odstranil. Vznikaly také další typy sítí, jako je Hopfieldova, Kohonenova, ART a další. [2]

1.2 Základní pojmy

Základním stavebním funkčním prvkem nervové soustavy je nervová buňka, tzv. neuron. Jen mozková kůra člověka je tvořena asi 13 až 15 miliardami neuronů, z nichž každý může být spojen s 5000 jinými neurony. Neurony jsou samostatné specializované buňky určené k přenosu, zpracování a uchování informací nutných pro realizaci životních funkcí organismu. Struktura neuronu je schematicky znázorněna na obrázku *Obr. 1*.



Obr. 1: Schematické znázornění biologického neuronu [3]

Neuron je přizpůsoben pro přenos signálů tak, že kromě vlastního těla, tzv. somatu, má i vstupní a výstupní přenosové kanály: dendrity a axon. Z axonu obvykle odbočuje řada větví, tzv. terminálů, zakončených blánou, která se převážně stýká s výběžky dendritů jiných neuronů. K přenosu informace pak slouží unikátní mezineuronové rozhraní, tzv. (chemická) synapse. Míra synaptické propustnosti je nositelem všech význačných informací během celého života organismu.

Šíření informace je umožněno tím, že soma i axon jsou obaleny membránou, která má schopnost za jistých okolností generovat elektrické impulsy. Tyto impulsy jsou z axonu přenášeny na dendrity jiných neuronů synaptickými branami, které svojí propustností určují intenzitu podráždění dalších neuronů. Takto podrážděné neurony při dosažení určité hraniční meze, tzv. prahu, samy generují impuls a zajišťují tak šíření příslušné informace. Po každém průchodu signálu se synaptická propustnost mění, což je předpokladem paměťové schopnosti neuronů. Také propojení neuronů prodělává během života organismu svůj vývoj. V průběhu učení se vytváří nové paměťové stopy nebo při zapomínání synaptické spoje přerušují. [4]

1.2.1 Umělý neuron

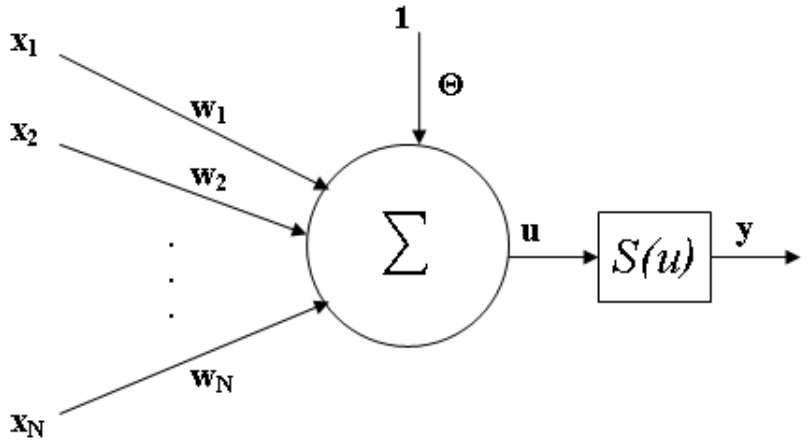
Základem matematického modelu neuronové sítě je umělý (formální) neuron, který získáme přeformulováním zjednodušené funkce neurofyzického neuronu do matematické řeči. Jeho struktura je schematicky znázorněna na obrázku *Obr. 2*.

Umělý neuron (dále jen neuron) má N obecně reálných vstupů $x_{1, \dots, } x_N$, které modelují dendrity. Vstupy jsou ohodnoceny obecně reálnými synaptickými váhami $w_1, \dots, w_N$, které určují jejich propustnost. Ve shodě s neurofyzickou motivací mohou být synaptické váhy záporné, čímž se vyjadřuje jejich inhibiční charakter.

Zvážená suma vstupních hodnot představuje vnitřní potenciál neuronu:

$$u = \sum_{i=1}^{n} w_i x_i \tag{1}$$

Hodnota vnitřního potenciálu u po dosažení tzv. prahové hodnoty Θ indukuje výstup (stav) neuronu y, který modeluje elektrický impuls axonu. Nelineární nárůst výstupní hodnoty $y = S(u)$ při dosažení prahové hodnoty potenciálu Θ je dán tzv. aktivační (přenosovou) funkcí S.



Obr. 2: Schematické znázornění umělého neuronu

Formální úpravou lze docílit toho, že funkce S bude mít nulový práh a vlastní práh neuronu budeme chápat jako váhu, tzv. bias dalšího formálního vstupu s konstantní jednotkovou hodnotou. [4]

1.2.2 Přenosová funkce neuronu

Výstup z neuronu můžeme tedy výsledně přepsat do tvaru

$$y = S\left(\sum_{i=1}^{n} w_i x_i + \Theta \right) \tag{2}$$

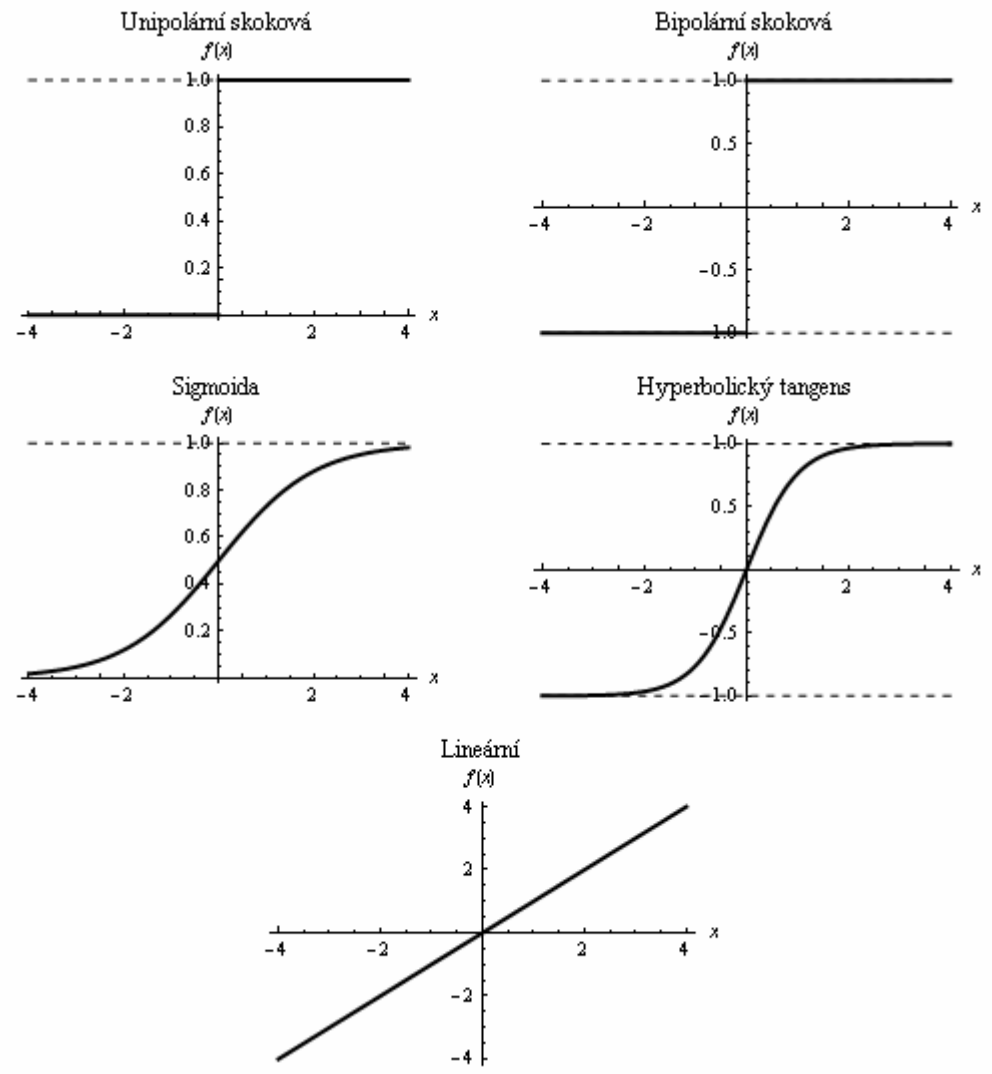
Přenosová funkce neuronu je funkce, která transformuje vstupní signál na signál výstupní v intervalech 0 až 1 a -1 až +1. Tato funkce může být skoková či spojitá a musí být monotónní - tzn. že přiřazení odezev výstupu na vstup je jednoznačné. Přenosová funkce je tedy velmi důležitým prvkem při práci neuronu. Pro správný chod neuronu a neuronových sítí je důležité, jakou přenosovou funkci zvolíme. Přenosová funkce udává, jaká bude odezva na výstupu na vstupní podnět. Jsou různé druhy funkcí u kterých obecně platí, že jejich hodnota má být v intervalu -1 až +1 a že mají být spojité nebo s nespojitostí prvního druhu (binární funkce 0-1). Volba funkce závisí na problému, který chceme řešit. Např. pokud chceme klasifikovat, zda je výrobek dobrý či špatný, pak nám stačí binární funkce. Pokud bychom použili funkci spojitou, pak musíme rozhodnout, jaká její hodnota (0.7, 0.8,...) znamená dobrý a jaká špatný. [2]

Nejpoužívanější typy přenosových funkcí jsou:

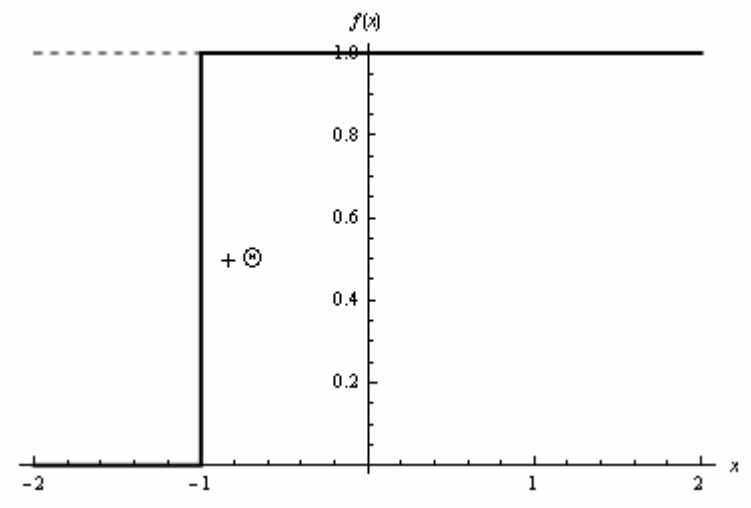
- skoková
- lineární
- logistická

Skoková přenosová funkce nabývá pouze 2 diskrétních hodnot, a to podle toho, zda je unipolární či bipolární. Unipolární binární skoková funkce nabývá hodnot 0 a 1, zatímco bipolární -1 a +1. Podobné je to i s logistickými přenosovými funkcemi. Výstupy těchto typů funkcí ovšem nabývají spojitých hodnot. Obor hodnot z intervalu (0, 1) má logistická sigmoida, hyperbolický tangens pak interval (-1, +1). Lineární přenosová funkce vykazuje taktéž spojitý výstup. Její transformace je zřejmá, tedy že tutéž hodnotu na vstupu dává i na výstupu. Grafické znázornění vyjmenovaných přenosových sítí zachycuje obrázek *Obr. 3*.

Na výstup přenosové funkce neuronu má velký vliv jeho práh Θ. Ten do jisté míry ovlivňuje citlivost neuronu na vstupní podněty. Pokud je práh nulový, výstup přenosové funkce nabývá standardních hodnot. Jestliže se ale práh nastaví na konkrétní hodnotu, projeví se to na výstupu neuronu jako posunutí. Tento jev je patrný na následujícím obrázku *Obr. 4*. Definováním prahu na jinou než nulovou hodnotu tak může výrazně ovlivnit chování neuronu a potažmo i celé sítě.



Obr. 3: Vybrané přenosové funkce

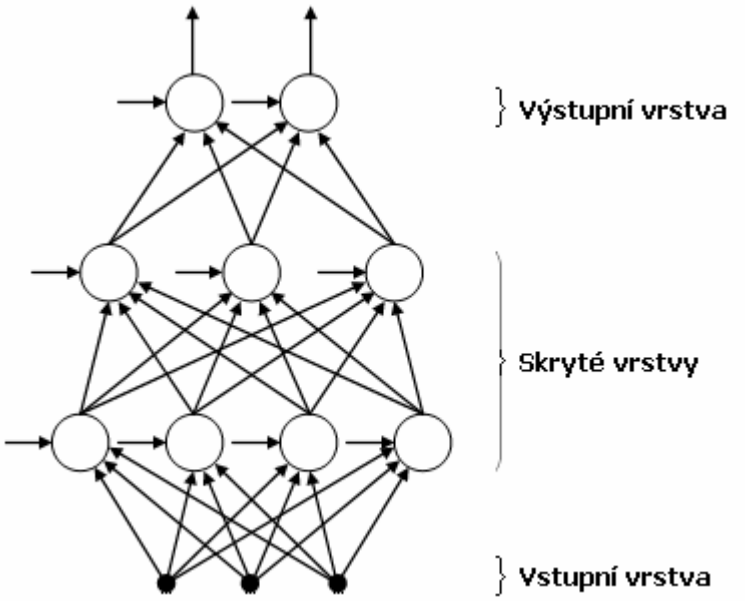


Obr. 4: Vliv zavedení prahu na výstup přenosové funkce

1.3 Struktura a funkcionalita sítí

Pro větší výpočetní sílu se neurony uspořádávají do sítí neuronů. Neurony se sdružují do struktur a tím vlastně vytvoří specificky provázanou síť. Tato síť se většinou skládá z oblastí s podobnou funkcionalitou, tzv. vrstev. Neuronová sít je složena z jednotlivých neuronů, které jsou vzájemně propojeny a to tak, že výstup jednoho neuronu je vstupem obecně libovolnému počtu neuronů v další vrstvě. Počet neuronů v jednotlivých vrstvách a princip jejich propojení určuje topologii sítě. [5] Pro topologii sítí platí obvykle pravidlo, že každý neuron bývá spojen s každým neuronem ve vyšší vrstvě. Existují ovšem i výjimky, jakou je například Hopfieldova síť (Obr), ve které je spojen každý neuron se všemi ostatními. Každý spoj je navíc ohodnocen vahami, které mohou nabývat různých hodnot a vyjadřují, jaký význam tento spoj má pro daný neuron. To ovšem neznamená, že spoj s malou vahou můžeme zanedbat, protože nevíme, jaký vliv má tento vstup na celkovou činnost sítě. [2]

U vícevrstvých sítí platí, že první vrstva je vždy větvící, což znamená, že neurony ve vstupní vrstvě pouze distribuují vstupní hodnoty do další vrstvy. [2] Vrstvy bezprostředně po vrstvě vstupní jsou nazývány skryté vrstvy, jichž bývá ve vícevrstvé síti hned několik. Za výstupní vrstvu bývá označována poslední vrstva sítě, která na svém výstupu de facto předkládá výstup celé sítě.



Obr. 5: Obecná struktura vícevrstvé neuronové sítě

Problém vhodného počtu vrstev u vícevrstvých sítí byl „vyřešen“ až ve druhé polovině dvacátého století. Vlastní funkce neuronové sítě je v podstatě transformační funkce, která přiřazuje jistému vstupnímu obrazu obraz výstupní. Důkaz matematického charakteru o takovéto funkci nebyl dlouho k dispozici. Kolmogorovův teorém o řešení třináctého Hilbertova problému aplikovaného na neuronové sítě vedl k poznatku, že k aproximaci libovolné funkce neuronovou sítí stačí, aby měla minimálně tři vrstvy s odpovídajícím počtem neuronů v každé vrstvě. Bohužel důkaz, ze kterého plyne výše zmíněný počet vrstev, nám nic neříká o počtu neuronů v těchto vrstvách, jejichž počet by byl z hlediska řešení daného problému optimální. [2]

1.4 Učení sítě

Učící schopnost neuronových sítí spočívá právě v možnosti měnit všechny váhy v síti podle vhodných algoritmů na rozdíl od sítí biologických, kde schopnost se učit založena na možnosti tvorby nových spojů mezi neurony. Fyzicky jsou tedy obě schopnosti se učit založeny na rozdílných principech, nicméně z hlediska logiky ne.

Algoritmus se obvykle dělí na dvě fáze a to na fázi *aktivační* (vybavovací) a *adaptační* (učící), které ke své činnosti potřebují trénovací množinu. Trénovací množina je skupina vektorů obsahující informace o daném problému pro učení. Pokud učíme síť s učitelem, pak jsou to dvojice vektorů vstup-výstup. Jestliže učíme síť bez učitele, pak trénovací množina obsahuje jen vstupní vektory. Pokud používáme jen fázi aktivační, pak mluvíme o vybavování. Tato fáze se používá samostatně jen tehdy, když je síť naučena. Cyklické střídání obou fází je vlastně učení.

Aktivační fáze je proces, při kterém se předložený vektor informací na vstup sítě přepočítá přes všechny spoje včetně jejich ohodnocení vahami až na výstup, kde se objeví odezva sítě na tento vektor ve formě výstupního vektoru. Při učení se tento vektor porovná s vektorem originálním (požadovaným, výstupním) a rozdíl mezi oběma vektory (lokální odchylka – chyba) se uloží do paměťové proměnné.

Adaptační fáze je proces, při kterém je minimalizována lokální chyba sítě tak, že se přepočítávají váhy jednotlivých spojů směrem z výstupu na vstup za účelem co největší podobnosti výstupní odezvy s originálním vektorem. Po té se opět opakuje aktivační fáze. Další získaný rozdíl (lokální odchylka) se přičte k předchozímu atd. Pokud se tímto

postupem projde celá trénovací množina, je hotová jedna epocha. Celé sumě odchylek za jednu epochu se říká globální odchylka – chyba. Pokud je globální odchylka menší než námi požadovaná chyba, pak proces učení skončí. Z výše popsaného je vidět, že proces učení není nic jiného, než přelévání informací ze vstupu na výstup a naopak.

To zda se síť naučí správným odezvám na dané podněty, závisí na více okolnostech, a to na množství vektorů a jejich velikosti, topologii sítě, odlišnosti charakteristických vlastností jednotlivých tříd, přípravě trénovací množiny, a jiných. [2]

1.4.1 Algoritmus Back-Propagation

Specifickým a v poslední době hojně používaným algoritmem co se týká učení neuronových sítí je tzv. Back-Propagation, což znamená zpětné šíření (propagace) chyby. Iterativní gradientní algoritmus učení, který minimalizuje čtverce chybové funkce.

Princip Back-Propagation:

Adaptační algoritmus zpětného šíření chyby je používán v přibližně 80% všech aplikací neuronových sítí. Samotný algoritmus obsahuje tři etapy: dopředné (feed-forward) šíření vstupního signálu tréninkového vzoru, zpětné šíření chyby a aktualizace váhových hodnot na spojeních.

Během dopředného šíření signálu obdrží každý neuron ve vstupní vrstvě vstupní signál a zprostředkuje jeho přenos ke všem neuronům vnitřní vrstvy. Každý neuron ve vnitřní vrstvě vypočítá svou aktivaci a pošle tento signál všem neuronům ve výstupní vrstvě. Každý neuron ve výstupní vrstvě vypočítá svou aktivaci, která odpovídá jeho skutečnému výstupu po předložení vstupního vzoru. V podstatě tímto způsobem získáme odezvu neuronové sítě na vstupní podnět daný excitací neuronů vstupní vrstvy.

Během adaptace neuronové sítě metodou Back-Propagation jsou srovnávány vypočítané aktivace s definovanými výstupními hodnotami pro každý neuron ve výstupní vrstvě a pro každý tréninkový vzor. Na základě tohoto srovnání je definována chyba neuronové sítě, pro kterou je vypočítán faktor δ, jež odpovídá části chyby, která se šíří zpětně z daného neuronu ke všem neuronům předcházející vrstvy majícím s tímto neuronem definované spojení.

Úprava váhových hodnot na spojeních mezi neurony konkrétní a vyšší vrstvy pak závisí na odpovídajícím faktoru δ a aktivacích neuronů ve dané vrstvě.[6]

Odvození algoritmu Back-Propagation:

K úspěšnému naučení neuronové sítě (v případě učení s učitelem) je potřeba tzv. trénovací množina. Každý vzor trénovací množiny popisuje jakým způsobem jsou excitovány neurony vstupní a výstupní vrstvy. Formálně můžeme za trénovací množinu *T* považovat množinu prvků (vzorů), které jsou definovány uspořádanými dvojicemi následujícím způsobem [6]

$$T = \{x[k], d[k]\}_{k=1}^{N} \tag{3}$$

kde *N* je počet prvků (dvojic), *x*[*k*] je *k*-tý vstupní vzor a *d*[*k*] je *k*-tý výstupní vzor. Odchylka mezi výstupní odezvou a požadovaným originálem je dána vztahem

$$E(k) = \frac{1}{2}\sum_{j=1}^{m}\left[y_k(k) - d_j(k)\right]^2 \tag{4}$$

kde $y_j(k)$ je odezva na vstupní *k*-tý vektor a $d_j(k)$ je požadovaný výstupní vzor. Chyba *E*(*k*) je chyba za jeden jediný vektor (*k*-tý) přes všechny jeho prvky (*j*=1 až *m*) a na její výpočet lze nahlížet jako na první krok zpětného chodu. Globální chyba (chyba za celou trénovací množinu, tedy za jednu epochu) je potom dána vztahem

$$E_T = \sum_{k=1}^{N} E(k) \tag{5}$$

kde *N* je počet dvojic vektorů v trénovací množině a bývá také označována jako energetická funkce. Jediný parametr (opomineme-li práh neuronu a strmost jeho přenosové funkce), který lze měnit, je váha každého spoje.

K tomu, abychom dobře rozuměli následujícím odvozením, si musíme určit obecnější popis jednotlivých proměnných. Pamatujme si, že w_{ij}^L je váha, která je v *L*-té vrstvě, počítáno od výstupní vrstvy dolů směrem ke vstupní. Tato váha spojuje *i*-tý neuron v této vrstvě s *j*-tým neuronem ve vrstvě *L*+1. Výstup z tohoto *i*-tého neuronu bude y_i^L a suma vstupů $\psi_i^L = \sum_m y_m^{L+1} w_m^L - \Theta_i^L$ kde Θ_i^L je práh neuronu *i* ve vrstvě *L*. Pro výstupní vrstvu tedy platí *L*=0, pro nejbližší nižší *L*=1 atd. Výše uvedené rovnice platí pro vrstvu *L*=0.

Uvažujme pouze jeden neuron. Ten, jak víme se skládá z ohodnocených vstupů, jejich sumy a přenosové funkce. Hodnotu prahu položíme rovnu 0. Vlastní závislost změny chyby na váze je dána vztahem

$$\frac{\partial E(k)}{\partial w_{ij}^0} \tag{6}$$

což nám určuje lokální gradient energetické funkce. Neuron se dá rozložit na tři části a to vstupy ohodnocené vahami, přenosovou funkcí a rozdílem mezi výstupem a požadovanou odezvou. Proměnná ψ zde zastupuje sumu všech vstupů, w_{ij} je příslušná váha daného spoje a y_i je výstupní hodnota přenosové funkce neuronu ve výstupní vrstvě

$$\frac{\partial E(k)}{\partial w_{ij}^0} = \frac{\partial E}{\partial y_i^0}\frac{\partial y_i^0}{\partial \psi_i^0}\frac{\partial \psi_i^0}{\partial w_{ij}^0} \tag{7}$$

Složením těchto mezivýsledků tak dostaneme rovnici (8) kde δ je chyba výstupního neuronu. Tato chyba, která závisí také na přenosové funkci neuronu, je počítána pro každý neuron výstupní vrstvy.

$$\frac{\partial E(k)}{\partial w_{ij}^0} = \frac{\partial E(k)}{\partial \psi_i} y_j^1 = \delta_i^0 y_j^1 \tag{8}$$

Po té, co jsme spočítali gradient chybové funkce, můžeme přistoupit k výpočtu přírůstku váhy daného spoje podle rovnice (9). Parametr *t* znamená obecně opravu vah v epoše $t\langle 0, \max.epoch\rangle$.

$$\Delta w_{ij}^0(t) = \eta\delta_i^0(t)y_j^1(t) + \mu\Delta w_{ij}^0(t-1) \tag{9}$$

kde μ je momentum (setrvačnost) a η parametr učení. Aktualizace vah se poté provede podle následujícího vztahu [2]

$$\Delta w_{ij}^0(t+1) = w_{ij}^0(t) + \Delta w_{ij}^0(t) \tag{10}$$

Výpočet nových vah pro další nižší vrstvy je podobný. Musíme si ale uvědomit, že ve vyšší vrstvě může být více neuronů, a proto se bude výstup skrytého neuronu větvit. Z tohoto důvodu je hodnota δ_i^L dána součtem příspěvků $w_{ki}^{L+1}\delta_k^{L+1}$ od všech neuronů vyšší vrstvy sítě. [7]

1.5 Dělení sítí

Neuronové sítě je možno členit podle mnoha různých kategorií. Pro pochopení problematiky týkající se vypracovaného toolboxu postačí představit základní aspekty, jako je počet vrstev, styl učení a algoritmus učení.

Podle počtu vrstev

Podle počtu vrstev dělíme neuronové sítě na sítě

- s jednou vrstvou
- s více vrstvami

Dělení podle počtu vrstev znamená, že rozlišujeme z kolika vrstev se daná síť skládá. Existují sítě s jednou vrstvou, se dvěma, třemi a více vrstvami. Sítě s jednou či dvěma vrstvami bývají většinou speciální sítě jako například Hopfieldova, Kohonenova či ART síť, které mají svůj speciální učící algoritmus a topologii, zatímco pro sítě se třemi a více vrstvami se obvykle používá klasická vícevrstvá síť s algoritmem Back-Propagation. [2]

Podle stylu učení

Podle stylu učení rozeznáváme sítě s učením

- deterministickým
- stochastickým

Styl učení v podstatě znamená, jak se přistupuje k zjištění vah sítě. V případě, že se jedná o zjištění výpočtem, pak mluvíme o deterministickém učení. Jestliže jsou však váhy získávány pomocí generátoru náhodných čísel, pak mluvíme o stochastickém stylu učení. Tento způsob získání vah sítě se obvykle používá jen při startu sítě. [2]

Podle algoritmu učení

Konečně podle typu algoritmu rozlišujeme neuronové sítě s učením

- s učitelem
- bez učitele

Učení neuronové sítě s učitelem znamená, že se síť snaží přizpůsobit svou odezvu na vstupní informace tak, aby se její momentální výstup co nejvíce podobal požadovanému originálu.

Učení bez učitele je proces, ve kterém síť vychází z informací, které jsou obsaženy ve vstupních vektorech. [2]

1.6 Využití neuronových sítí

Využití neuronových sítí je opravdu široké a nabývá čím dál tím více na významu. Ve všeobecnosti můžeme vyjmenovat následující oblasti využití neuronových sítí pro:

- problémy aproximace funkcí
- klasifikace do tříd, klasifikace situací
- řešení predikčních problémů a plánování
- problémy řízení procesů
- transformace a filtrace signálů
- asociační problémy
- optimalizační problémy
- simulace pamětí
- kompresi dat
- počítačové vidění
- rozpoznávání znaků a řeči
- expertní systémy
- robotiku a jiné

2 TYPY VYBRANÝCH NEURONOVÝCH SÍTÍ

Druhů neuronových sítí existuje velké množství. Některé z nich už zazněly výše v textu, jiné budou probrány dále. Cíleně byly pro toolbox vypracované sítě Perceptron, vícevrstvá síť s dopředným šířením signálu a Hopfieldova síť. Je to z důvodu požadavků osnovy předmětu, ve kterém se tyto sítě vyučují. Proto budou tyto konkrétní typy sítí podrobněji popsány v následujících kapitolách, a to v takovém tvaru a v takovém rozsahu, v jakém byly vypracovány.

2.1 Jednoduchý Perceptron

Perceptron sestává z jediného výkonného prvku modelovaného obvykle McCullochovým a Pittsovým modelem neuronu, který má nastavitelné váhové koeficienty a nastavitelný práh *Obr. 2*. Někteří autoři označují stejným názvem i celou síť takových prvků. Algoritmus vhodný k nastavení parametrů perceptronu publikoval poprvé F. RosenBlatt v roce 1958 a později v roce 1962. Rosenblatt dokázal následující větu:

Máme-li v n-rozměrném prostoru lineárně separabilní třídy objektů, pak lze v konečném počtu kroků učení (iterací optimalizačního algoritmu) nalézt vektor vah W perceptronu, který oddělí jednotlivé třídy bez ohledu na počáteční hodnotu těchto vah.

Perceptron s jedním výkonným prvkem umožňuje ovšem nanejvýše klasifikaci do dvou tříd. Zvětšíme-li však počet výkonných prvků pracujících v perceptronu a zvětšíme-li i počet jeho vrstev, je možno jím klasifikovat do více tříd. Tyto třídy již nemusí být lineárně separabilní, musí však být separabilní. [1]

Neuron má kromě klasických vstupů zpravidla ještě jeden vstup navíc, jehož hodnota je konstantně nastavena na 1. Tento vstup je zde přidán z důvodů nastavování prahu neuronu stejným způsobem, jako se nastavují váhy. Takto rozšířený jednotkový vstup do neuronu znamená rozšíření každého vstupního vektoru o jeden vstup s hodnotou 1, přičemž bývá nazýván rozšířený vektor (Augmented Vector). Po té se příslušné prahy položí rovny 0. Provedeme další úpravu, která se používá a bude použita při demonstraci učení perceptronu. Vzhledem k tomu, že výstupní neuron perceptronu může nabývat pouze dvou hodnot, lze vstupní vektory (vzory) přiřadit pouze do dvou tříd a to např. do T_+ a T_-.

Zmíněná úprava bývá označována jako „nastavený rozšířený vektor“ (Adjusted Augmented Vector) a spočívá v tom, že vektory třídy T_- bývají násobeny -1. Členy T_- třídy mají pak invertovaná znaménka. [2]

Pro výpočet nových vah sítě můžeme použít gradientní metodu a metodu fixních přírůstků, jejíž koeficient může být pevný či modifikován absolutní korekcí, zlomkovou korekcí. [2] Při učení perceptronu obsaženého v toolboxu byla použita metoda fixních přírůstků bez korekce. Při použití tohoto pravidla se nové váhy přepočítají podle vztahu

$$w(k+1) = w(k) + cy(k) \text{ pokud } w(k) \cdot y(k) \leq 0 \tag{11}$$

kde koeficient c ovlivňuje míru učení a nabývá celočíselné kladné hodnoty.

2.2 Vícevrstvá síť s dopředným šířením signálu

Vícevrstvá síť s dopředným šířením signálu je síť, kde se šíří signál v jednom směru od vstupu na výstup.

Spojením více samostatných neuronů tak, že výstup z neuronu i přivedeme opět jako vstup do neuronu i+1 získáváme strukturu, která sčítá rozhodovací schopnost samostatného neuronu. Všechny neurony uspořádáme do vrstev a propojíme je tak, že budou spojeny systémem každý s každým v rámci sousedních vrstev, ale neurony v rámci stejné vrstvy zůstanou nespojeny *Obr. 5*. První vrstva (vstupní), sama o sobě data nezpracovává, jenom zajišťuje distribuci mezi ostatní neurony sousední vrstvy. Klasifikace se odehrává zejména ve skryté vrstvě, kterých může být obecně libovolný počet, nicméně nejpoužívanější topologie sítě s algoritmem učení Back-Propagation má právě jednu. Výsledek je vyhodnocen výstupní vrstvou. Výsledky klasifikace pro aktuální data tedy získáváme z poslední, výstupní vrstvy. [8]

Jako algoritmus učení byl v toolboxu použit Back-Propagation. Back-Propagation je algoritmus, který byl vytvořen pro učení vícevrstvých neuronových sítí s učitelem. Tento algoritmus opravuje (nastavuje) váhy jednotlivých spojů zpětným chodem tak, aby jejich velikosti byly z hlediska řešeného problému pokud možno optimální – hledá se globální minimum chybové funkce. Nastavení vah tedy probíhá v opačném směru, než jakým se šíří vstupní informace.

Při každém porovnávání výstupní odezvy s požadovaným originálem se daný rozdíl uchová v paměťové proměnné a sumarizuje se s dalšími postupně získanými rozdíly. Takto získané číslo za celou trénovací množinu (epochu) se nazývá globální chyba. Tato globální chyba je po každé epoše kontrolována s chybou, kterou zadal uživatel a pokud je nižší, než chyba zadaná, pak je síť naučena a učení končí. Algoritmus tedy hledá globální minimum chybové funkce.

Algoritmus Back-Propagation, nastavuje váhy tak, aby se globální chyba co nejvíce přiblížila k globálnímu minimu chybové funkce. Vlastní globální chyba je tedy pozice na tzv. chybové ploše, která je dána všemi vahami a jejich možnými hodnotami. Vlastní chybová plocha je mnohdy komplikovaná plocha s mnoha lokálními minimy, přičemž „pozice neuronové sítě na této ploše“ je dána aktuálním stavem vah. Z toho je jasné, že pokud startujeme síť na nové učení s nově náhodně nastavenými vahami, jsme při každém startu pokaždé v jiném bodě chybové plochy. [2]

Bohužel existují i příklady, kdy se nám nepodaří síť naučit s dostatečně malou chybou. V takovém případě dosáhne energie určité hodnoty a nadále přestane klesat. Tomuto jevu se říká „uvíznutí v lokálním minimu“. Problém uváznutí v lokálním minimu je však možné řešit:

- vhodnou volbou parametrů η a μ (viz kapitola 1.4.1)
- vhodnou strategií výběru vzorů z trénovací množiny (např. náhodně)
- vhodným počátečním nastavením vah

Využití vícevrstvé neuronové sítě je rozmanité, nicméně pro účely výuky byl specifikován požadavek na aproximaci reálných funkcí a dat. Toto využití lze opodstatnit větou, kterou uvedl A. N. Kolmogorov:

Každou reálnou spojitou funkci mající n proměnných lze vyjádřit pomocí spojitých funkcí jedné proměnné.

Z toho lze aplikovat na neuronové sítě Kolmogorův teorém, který říká, že aby transformační funkcí umělé neuronové sítě bylo možno aproximovat libovolnou funkci *f*, stačí, aby příslušná síť měla alespoň 3 vrstvy s odpovídajícím počtem neuronů. Volba neuronů pevně určena není, stejně tak jako parametry sítě ovlivňující průběh jejího učení.

Neuronová síť aproximuje požadované zobrazení funkcí

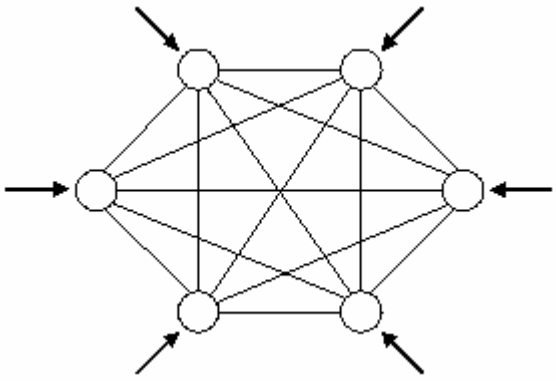
$$y = f(x, w, \Theta) \tag{12}$$

kde y je výstupní vektor, x je vstupní vektor, w je vektor všech vah a Θ vektor všech prahů. Funkce f je určena topologií sítě a typem neuronů. Během učení se tato funkce nemění, mění se pouze její parametry w a Θ . [7]

2.3 Hopfieldova síť

Na začátku osmdesátých let John Hopfield studoval autoasociativní sítě na nový model neuronové sítě, zda nemůže dojít k tomu, aby všechny neurony mezi sebou nemohly být propojeny. Hopfield zjistil, že správné použití energetické funkce má velmi důležitou roli na správné chování a vybavování neuronové sítě. V mnoha případech se dá říci, že Hopfieldova neuronová síť může sloužit jako optimalizační prostředek či jako klasifikátor nebo autoasociativní paměť. [1]

Neuronem Hopfieldovy sítě je opět klasický McCulloch-Pittsův perceptron. Skládá se z tolika neuronů, kolik má vstupů. Každý neuron je napojen na vstup sítě pouze jediným svým vstupem. Výstup každého neuronu se vede zpět na vstupy ostatních neuronů přes synaptické váhy, a tak se vytváří zpětnovazební smyčky. Výstup neuronu se však na vstup téhož neuronu nevede. Tímto uspořádáním vzniká symetrická síť a diagonálně symetrická matice vah. [7] Symetričnost matice je dána tím, že váhy, které jsou na spojích z jednoho neuronu do druhého, jsou v obou směrech stejné. Navíc jsou na diagonále matice hodnoty vah nulové, protože, jak už bylo zmíněno, nejsou na vstupy neuronů přiváděny jejich vlastní výstupy.



Obr. 6: Schéma Hopfieldovy sítě

Práh je u všech neuronů nulový a přenosová funkce se tedy neposouvá. Signály (ať už vstupní nebo výstupní) jsou binární nebo i spojité podle toho, k jakému účelu je síť použita. Nicméně jsou přenosové funkce všech neuronů totožné. Protože jsou výstupy z neuronů (přenosových funkcí) přiváděny znovu na vstupy dalších neuronů, odpovídá typ aktivační funkce typu vstupnímu signálu.

Učení Hopfieldovy sítě je v podstatě práce se čtvercovými maticemi, jejichž dimenze je rovna počtu vstupů. Výsledná matice vah se získá tak, že se sečtou jednotlivé čtvercové matice, kde každá matice představuje součin každého prvku s každým, jak znázorňuje následující rovnice

$$w_{ij} = \sum_{S=1}^{n} x_i^S x_j^S \text{ pro } i \neq j \text{, jinak } w_{ij} = 0 \tag{13}$$

Vlastní proces učení je obsažen ve dvou krocích:

1. nastavení vah podle rovnice (13)
2. opakuj bod 1 dokud nebudou použity všechny vstupní vzory, pak konec

Z výše popsaného algoritmu je vidět, že proces učení je jednorázový, což není případ např. sítí s dopředným šířením. Jinými slovy, Hopfieldova síť je naučena, jakmile jednou projde celou trénovací množinou.

Vybavování je proces, kdy Hopfieldova síť hledá originální vzor ke vzoru předloženému. Tento proces je na rozdíl od procesu učení iterační. To znamená, že proces vybavování se zastaví tehdy, kdy nedochází ke změně na výstupu jakéhokoliv neuronu. Vlastní algoritmus by se dal shrnout opět do dvou bodů :

1. Předložíme vstupní obraz (bod z oblasti přitažlivosti) a získáme odezvu na výstupech neuronů.
2. Opakujeme bod 1 s tím rozdílem, že hodnoty na výstupech použijeme jako nové vstupní hodnoty místo originálu. To opakujeme tak dlouho, až nedochází ke změnám na výstupech neuronů (bylo dosaženo stabilního stavu).

Matematicky řečeno, je vybavování v této síti hledáním vzoru, který má nejmenší Hammingovu vzdálenost od vstupního vzoru.

Hopfieldova síť má tři závažné nevýhody, a to malý počet uschovaných vzorů (pokud bude naučena na velký počet vzorů, pak její odezva může konvergovat k neznámému obrazci), velké nároky na paměť (matice vah odpovídá čtverci počtu vstupů) a nutnost volit vzory s co největší Hammingovou vzdáleností. Existuje pravidlo, podle kterého lze určit, na jaký maximální počet vzorů se neuronová síť může naučit. Toto pravidlo říká, že počet vzorů by měl být menší než 15% neuronů (vstupů) Hopfieldovy sítě. [2]

Výhodou je naopak to, že naučená Hopfieldova síť umí asociovat i inverzní data. Tzn. že po předložení inverzního vzoru Hopfieldově síti naučené na normální data dostaneme tento inverzní vzor i na výstupu.

3 PROSTŘEDÍ MATHEMATICA

K vypracování toolboxu bylo v zadání práce požadováno prostředí Mathematica od společnosti Wolfram Research. Software Mathematica paří do rodiny aplikací určených nejen pro matematické výpočty, jako je i například Maple nebo Matlab. Ve výuce na Fakultě aplikované informatiky se využívá již řadu let a tento nástroj je mezi studenty poměrně oblíbený. Je to zejména z důvodu uživatelsky přívětivého prostředí, které Mathematica k práci nabízí, a také z důvodu symbolického, logicky snadno zapamatovatelného jazyka. Díky stále rostoucí podpoře produktu a mnoha rozšířením je Mathematica také vhodná i pro práci s neuronovými sítěmi.

Toolbox byl konkrétně vypracován v prostředí Mathematica 7.0 for Students, který nově škola od letošního roku nabízí studentům k domácímu a nekomerčnímu použití.

3.1 Představení

Software Mathematica vyvíjí prakticky od počátku své existence americká společnost Wolfram Research (http://www.wolfram.com/). Společnost, nyní se sídlem v Champaign (Illinois, USA), založil v roce 1987 anglický matematik Stephen Wolfram. Mathematica byla zpočátku jediným produktem této firmy, avšak v průběhu času a se zvyšujícím se zájmem o tento produkt se společnost rozrůstala a začala vyvíjet i rozšiřující, sofistikovanější nástroje. Nyní už společnost nabízí kromě Mathematicy ve verzi 7.0 i programy pro webová rozhraní (webMathematica), rychlejší verze pro paralelní nasazení (gridMathematica) nebo nástroj pro spouštění aplikací vytvořených v Mathematice bez nutnosti jejího pořízení (Mathematica Player).

Mathematica nabízí také obsáhlou a kvalitní podporu pro její uživatele. K tomuto účelu slouží aplikace Documentation Center, která je dostupná i na webovém serveru společnosti Wolfram. K dispozici je i množství návodů a ukázkových projektů, díky kterým se může začátečník naučit s Mathematicou pracovat velmi rychle. Navíc má Mathematica tu výhodu, že pracuje kromě vlastních systémových knihoven i na modulárním principu, kdy je možné zakomponovat do programu rozšiřující balíčky (tzv. Add-Ons nebo toolboxy) podporující různé specifické problematiky. Tyto balíčky vydává v průběhu času jako doplňky sama společnost, ale také běžní uživatelé pracující s Mathematicou. Díky tomu je Mathematica mezi uživateli hojně rozšířená a oblíbená.

Samozřejmostí nejnovější verze Mathematicy je podpora všech nejrozšířenějších operačních systémů jako Microsoft Windows, Linux, Apple's Mac OS X i Sun's Solaris a to i s podporou 64-bitových systémů.

Nasazení Mathematicy je opravdu široké, od matematických a vědeckých problematik až po technické využití či dokonce aplikací v chemickém modelování. Také umožňuje vystupovat jako programovací jazyk vyšší úrovně, jazyk pro popis grafických objektů či jako systém pro psaní technických dokumentů a mnoho dalších.

3.2 Práce s prostředím

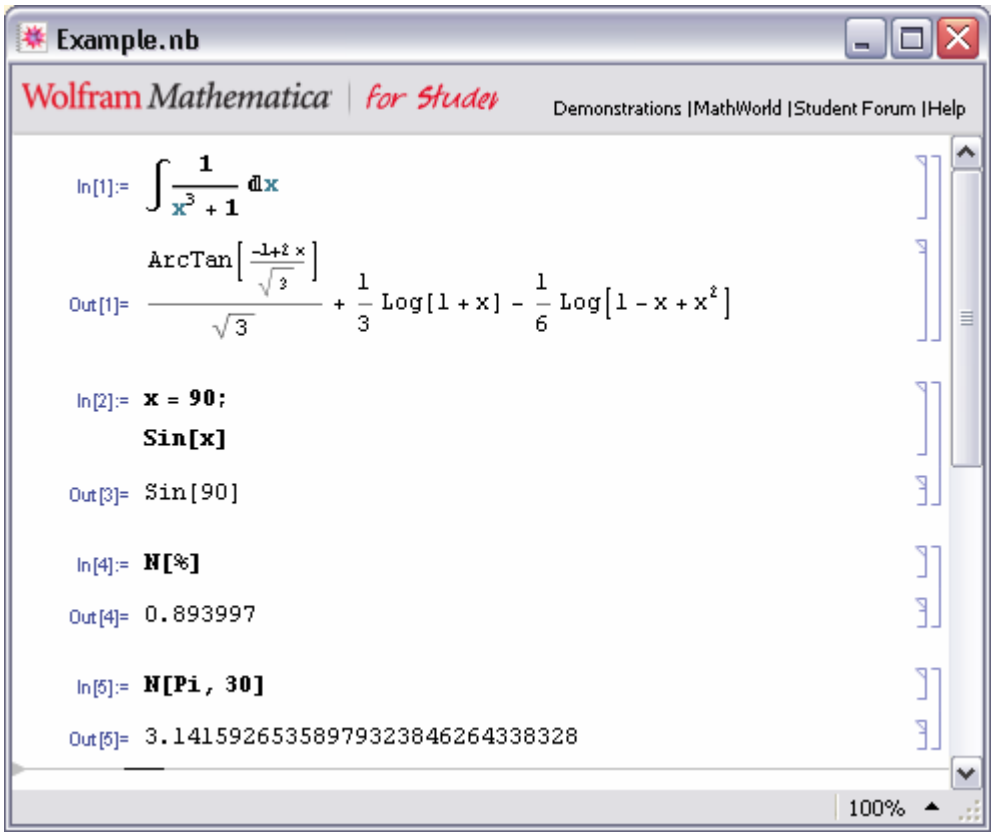
Práce s Mathematicou je intuitivní a jednoduchá. Prostředí používá dvou rozhraní – tzv. front-end pro styk s uživatelem a jádro (kernel) zajišťující na pozadí veškeré výpočty. Nejběžnější způsob práce v prostředí je práce v interaktivních dokumentech, tzv. noteboocích (ty tvoří soubory s příponou *.nb). Notebooky umožňují kombinovat vstupy a výstupy Mathematicy s textem, grafickými objekty a jinými formáty dat.

Pro začátek práce s Mathematicou stačí do notebooku napsat požadovaný výpočet (tj. vstup) a pro jeho vyhodnocení stlačit kombinaci kláves Shift+Enter (nebo numerický Enter). Kernel Mathematicy požadavek vyhodnotí a vrátí ve front-endu uživateli výsledek (výstup). Výstupy kernelu nemusí být ve standardní číselné podobě, protože Mathematica podporuje řadu formátů dat – např. texty, 2D grafy, 3D modely s možností rotace, obrázky, animace a podobně. Každá položka notebooku je považována za samostatnou a je zobrazována jako tzv. buňka (cell). Tyto buňky velmi zpřehledňují práci v programu, neboť mohou nabývat hierarchické struktury a shromažďovat či zabalovat (skrývat) požadovaný obsah notebooku například pro lepší přehlednost obsahu. K tomu přispívá i možnost stylování a formátování obsahu notebooku buďto samotným uživatelem nebo prostřednictvím předdefinovaných šablon.

Jazyk Mathematicy je symbolický, ovšem všechny příkazy a symboly se dají nahradit i jejich textovou formou. K tomu slouží vestavěné funkce a příkazy, kterých obsahuje Mathematica přes 2500. Jako důležitou poznámku je dobré zmínit, že program je citlivý na velikost písmen v textu a názvosloví (tzv. Case-Sensitive). Matematické operace a výpočetní maticové zvládá Mathematica bez sebemenších problémů a to s volitelnou přesností výsledků.

Pokud se vrátíme k používání funkcí Mathematicy, jejich používání je velmi intuitivní. Názvy funkcí jsou totiž většinou voleny podle jejich účelu. Argumenty funkce se zadávají do hranatých závorek a pomocí parametrů funkcí lze chování požadované funkce upravit podle vlastních potřeb. Možné je také vnořování funkcí. Pro lepší demonstraci funkčnosti a použití funkcí slouží rozsáhlá interaktivní nápověda s definicí, popisem a ukázkami použití dané funkce. Díky tomu může uživatel přesně zvolit funkci, která vyhovuje jeho požadavkům. Nápověda se vyvolává stisknutím tlačítka F1 a obsahuje mimo jiné také řadu podrobných tutoriálů a návodů, jak začít s programem Mathematica pracovat.

Mathematica umožňuje také práci s proměnnými (ve vyšších verzích i dynamickými) a podporuje příkazy standardního sekvenčního programování. Díky tomu může uživatel definovat a vytvářet vlastní funkce nebo programovat vlastní algoritmy rychle a efektivně. Také proto je Mathematica vynikajícím nástrojem nejen pro začátečníky, ale i pro pokročilé a náročně uživatele.

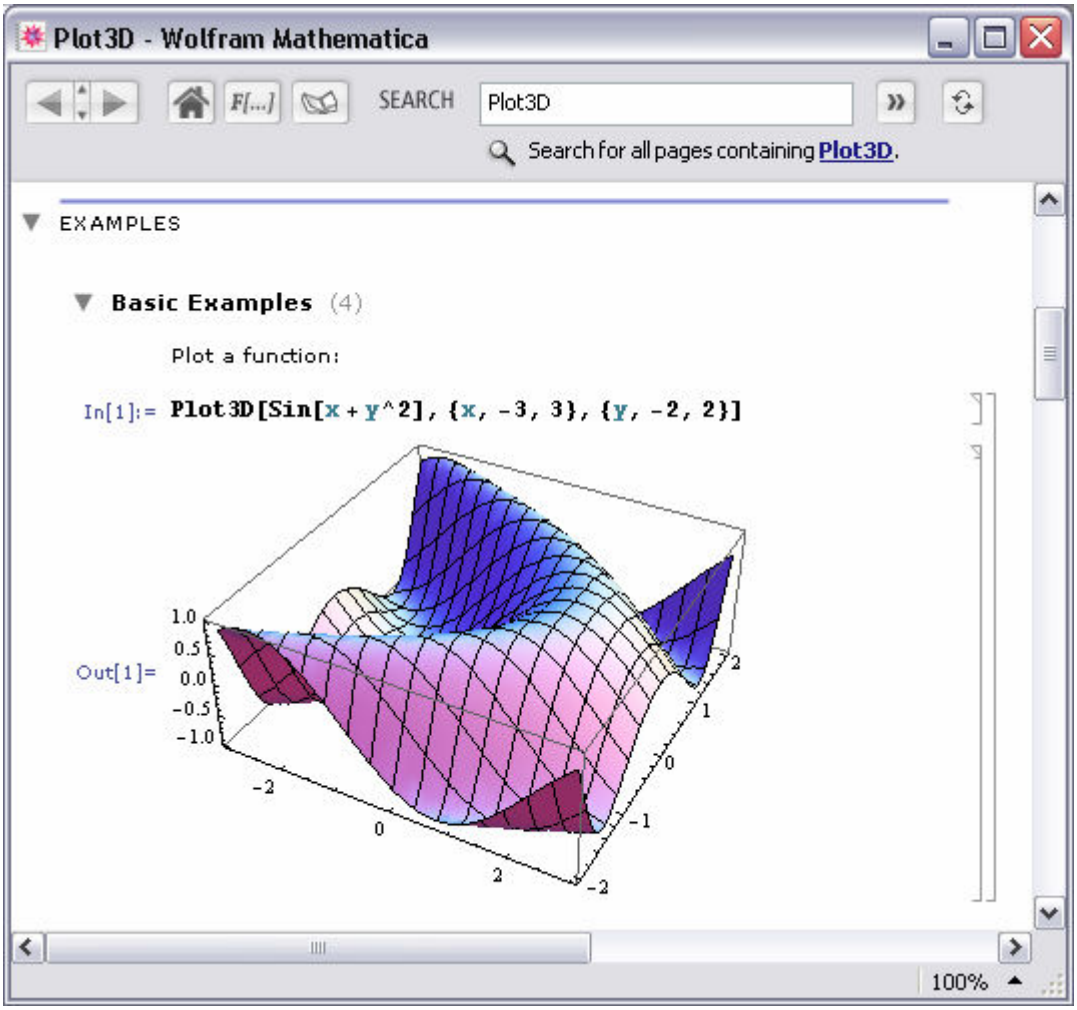


Obr. 7: Práce v prostředí Mathematica

V nejnovějších verzích nabízí Mathematica i dynamickou interaktivitu, díky čemuž lze vytvářet například modifikovatelné panely okamžitě reagující na změnu parametrů. Takový

výsledek lze poté otevřít ostatními uživateli pomocí Mathematica Playeru i bez nutnosti instalace prostředí Mathematica.

Závěrem je také nutné připomnět rozsáhlou podporu importu a exportu nejznámějších formátů dat. Zaručena je interaktivita nejen mezi Mathematicou a uživatelem, ale prostřednictvím protokolu MathLink komunikuje kernel přímo i s jinými rozhraními (Java, C++, MySQL, webové servery, ...).



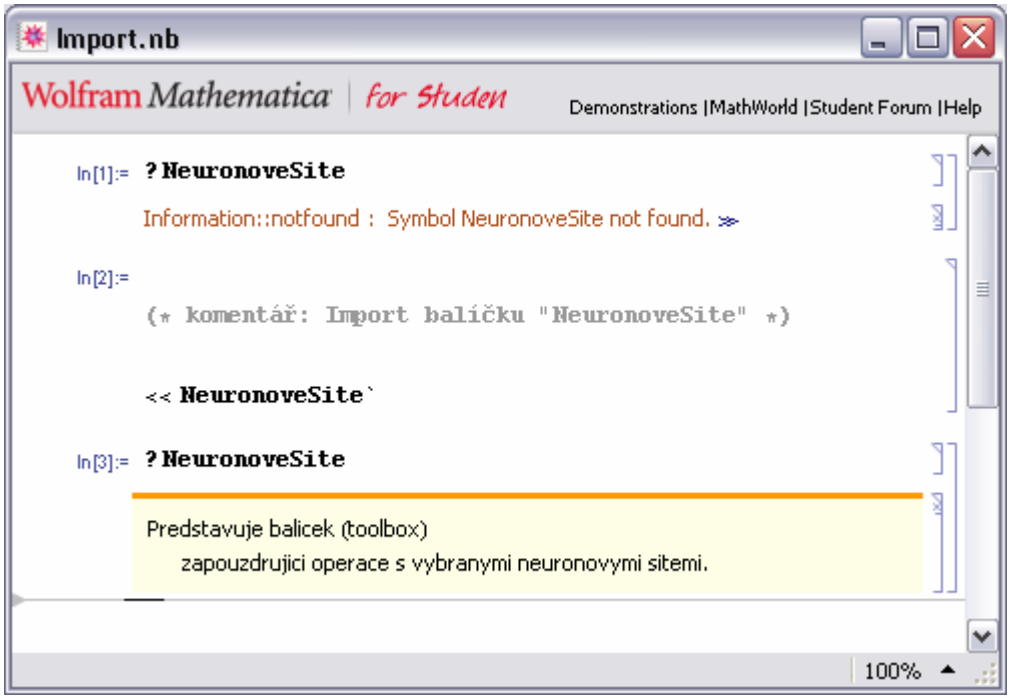
Obr. 8: Interaktivní nápověda Mathematicy s ukázkami použití

3.3 Rozšiřující balíčky (Toolboxy)

Mathematica se těší velké oblibě i díky rozšiřujícím balíčkům (toolboxům), které lze do programu importovat a obohacovat jej tak o další zajímavé funkce. Tyto balíčky, vytvořené samozřejmě v Mathematice, se se svým obsahem jednoduše nakopírují do adresářové struktury programu a před jejich použitím se připojí pomocí jediného příkazu. Poté je

Mathematica schopna využívat funkce a objekty z tohoto balíčku. Jak je tedy vidět, rozšíření Mathematicy o další funkce je snadné, rychlé a efektivní.

Definice balíčků jsou obsaženy v souborech s příponou *.m. Tím je zaručeno odlišení od klasických notebook souborů. Navíc může mít balíček definovanou i vlastní nápovědu či dokumentaci. Tato se po importu automaticky zobrazuje v nápovědě Mathematicy a je uživatelům tak rychle k dispozici. K tvorbě balíčků je možné přistupovat buďto manuálním způsobem, kdy tvůrce balíčku sepíše jeho obsah a také případnou nápovědu ručně a sám si poté otestuje a zajistí jeho funkčnost. Mnohem jednodušším způsobem je však použít vývojové prostředí Wolfram Workbench, které společnost Wolfram také nabízí a které je k tomuto účelu přímo určeno.



Obr. 9: Import balíčku do prostředí Mathematica

II. PRAKTICKÁ ČÁST

4 ROZBOR ÚKOLU

Cílem diplomové práce bylo vytvořit toolbox neuronových sítí pro prostředí Mathematica a výuku v předmětu Metody umělé inteligence. Požadavky na vypracování v přesném znění jsou uvedeny v kapitole 4.1 *Stanovené požadavky*. V kapitole 4.2 *Praktické řešení* je vysvětleno technologické řešení problému, metodika postupu práce a doplňující vysvětlivky či omezení, které z vypracování úkolu vyplývají.

4.1 Stanovené požadavky

Požadavky ze zadání diplomové práce byly následující:

- Seznamte se s neuronovými sítěmi, typy a jejich algoritmy učení.
- Naprogramujte s prostředí Mathematica vybrané typy.
- U naprogramovaných typů připravte ukázkové úlohy pro laboratoře v předmětu Metody umělé inteligence.
- Připravte i možnost použití přes knihovny v prostředí Mathematica.
- Závěr.

Po konzultaci s vedoucím diplomové práce byly s ohledem na probíranou látku v předmětu Metody umělé inteligence vybrány k zahrnutí do toolboxu tři typy neuronových sítí. Jmenovitě se jedná o klasický jednoduchý perceptron, vícevrstvou síť s dopředným šířením signálu s algoritmem učení Back-Propagation a Hopfieldovu síť. Všechny tyto sítě už byly podrobněji popsány výše v textu v teoretické části práce.

Jednoduchý perceptron byl vypracován za účelem demonstrování lineární klasifikace prvků do dvou tříd s názorným představením problému lineární separability prvků. Vícevrstvá neuronová síť už dokáže klasifikovat prvky i ve více třídách, nicméně tato síť se ve výuce představuje studentům zejména jako aproximátor funkcí (dat), a proto byla síť pro toolbox zpracována výhradně za účelem tohoto využití. Hopfieldova síť se zase pro své paměťové vlastnosti používá k autoasociaci prvků (znaků). Proto byla síť vypracovávána podle tohoto požadavku a její využití slouží k tomuto záměru.

4.2 Praktické řešení

Z hlediska zdrojového kódu bylo podle zadání použito prostředí Mathematica. Podobná pomůcka vyhovující požadavkům se již při výuce v předmětu používá, avšak existence nového a vlastního řešení přináší výhody z hlediska konkrétní přizpůsobitelnosti a případné rozšiřitelnosti tohoto nástroje do budoucna. Je však nutné dodat, že funkčnost nového toolboxu byla inspirována funkčností toolboxu již používaného, a to z toho důvodu, aby si studenti a potažmo i vyučující nemuseli zvykat na nový nástroj a na odlišnosti v jeho používání.

Po jazykové stránce byla při tvorbě toolboxu zvolena čeština. Mathematica podle údajů na stránkách výrobce uvádí češtinu jako jeden z podporovaných jazyků, nicméně z důvodu bezproblémové přenositelnosti toolboxu mezi jednotlivými platformami a vyhnutí se problémům s kódováním byl veškerý textový obsah psán bez diakritiky. Na čitelnosti a srozumitelnosti však text nijak výrazně neztratil.

Kromě samotných funkcí zprostředkovávajících práci s neuronovými sítěmi byla v Mathematice zpracována také dokumentace a stručná nápověda s praktickými ukázkami jejich použití. Použití nápovědy je velmi snadné, protože se při importu balíčku do Mathematicy její témata automaticky objeví v kontextu mezi ostatní dokumentací k rozšiřujícím toolboxům.

Výstupy funkcí jsou řešeny nejen klasicky návratovými hodnotami, ale většinou také grafickým způsobem, aby si mohl uživatel lépe představit danou problematiku. Vykreslování grafů při výstupu funkce, stejně tak jako i jiné možnosti, lze v případě nutnosti nastavením parametrů funkce jednoduše pozměnit či dokonce potlačit.

Protože jsou zdrojové kódy funkcí v součtu poměrně rozsáhlé, a protože se úseky kódů v některých případech opakují, bylo by nepraktické a obsahově náročné popisovat význam každého řádku kódu. Proto budou dále vysvětleny jen ty důležitější a stěžejní úseky. Všechny zdrojové kódy s komentáři jsou však k dispozici k nahlédnutí v příloze této diplomové práce. Z důvodu rozsahu byly kódy umístěny na přiložené CD. Jedná se především o zdrojové kódy jednotlivých balíčků neuronových sítí *NSPerceptron.m*, *NSDopredneSireni.m* a *NSHopfield.m*. Obsah těchto souborů není nijak specificky formátovaný, proto jej lze otevřít a prohlížet v libovolném textovém editoru.

5 VYPRACOVÁNÍ TOOLBOXU

V následující kapitole budou podrobně popsány jednotlivé typy vytvořených neuronových sítí. V úvodu bude popsáno, jakým způsobem byla síť navrhována a jaké funkce byly pro zajištění její práce nadefinovány. Zdrojové kódy daných funkcí budou poté podrobně rozebrány, aby byly vysvětleny všechny použité pojmy a metodiky. U každé sítě budou také v souladu s požadavky práce uvedeny ukázky jejich praktického použití a graficky znázorněny výstupy jednotlivých funkcí pro lepší představu o jejich činnosti.

5.1 Jednoduchý Perceptron

Síť Perceptron, vypracovaná v toolboxu, se skládá z jediného neuronu – perceptronu, který má jediný výstup *y*. Vstupů *x*, které jsou ohodnoceny vahami *w*, může být libovolné množství. Navíc vstupuje do neuronu i práh neuronu Θ ovlivňující jeho citlivost *Obr. 2*. Neuron vrací na výstupu binární hodnotu, což je hodnota 0 nebo 1. Aby bylo tohoto dosaženo, používá neuron přenosovou funkci ve tvaru unipolární skokové binární funkce, pomocí které převádí vstupní signál na výstup. Viz obrázek *Obr. 3*. Vstupní údaje zpracovává perceptron podle známého vztahu uvedeného v rovnici (2).

Jednoduchý perceptron se používá k lineární klasifikaci prvků. V procesu trénování se perceptron naučí vhodným nastavením vah na vstupech klasifikovat prvky, které odpovídají svým umístěním odlišným třídám. Po předložení jiného vstupního vektoru je poté schopen perceptron na výstupu vrátit příznak, patří-li prvek do první třídy či do druhé. Z toho je vidět, ze jednoduchý perceptron se skokovou binární přenosovou funkcí dokáže klasifikovat pouze prvky patřící do dvou tříd. Navíc je nezbytné, aby rozmístění prvků ve třídách bylo lineárně separabilní (tj. rozdělitelné pomyslnou přímkou). Pokud toto není zaručeno, perceptron nelze korektně naučit.

5.1.1 Funkce *VykresliPrvky*

Funkce *VykresliPrvky* zobrazuje v grafické podobě data reprezentovaná vektorem vstupních dat a jemu odpovídajícím vektorem tříd. Pomáhá tak vizualizovat rozmístění prvků ve třídách, které jsou primárně určeny k trénování perceptronu. Proto musí být počet tříd roven 2, protože jednoduchý perceptron se skokovou binární přenosovou funkcí dokáže klasifikovat prvky zařazené pouze ve 2 třídách.

Funkce nevrací žádná data, ale pouze objekt typu *Graphics* (graf). Prvky první třídy zobrazuje jako červené body, prvky třídy druhé pak jako modré body. V grafu pak lze přehledně rozeznat, jestli je rozmístění prvků do tříd lineárně separabilní či nikoliv. Viz obrázek *Obr. 10*.

Funkce *VykresliPrvky* nepoužívá žádné parametry.

Implementace

Důležité úseky zdrojového kódu:

```
1 VykresliPrvky[x_,y_]:=
2 Module[{pom,pom2,pom3=0,tridy={},pomVektor,rozdeleni},
3 ...
```

Funkce začíná její definicí a také definicí lokálních proměnných v případě potřeby spolu s jejich inicializačními hodnotami. Proměnná *x* zastupuje vstupní vektor, proměnná *y* zastupuje výstupní vektor (vektor tříd).

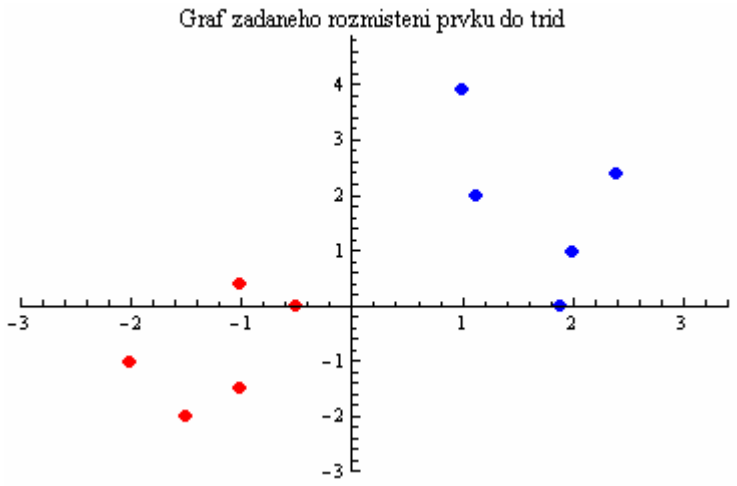
Hned v úvodu každé funkce je testována korektnost uživatelských dat z hlediska jejich správného typu, rozměru, délky, dimenze, omezení a podobně. Pokud nejsou data ve správném formátu, běh funkce se ukončí a vypíše se chybová hláška s upozorněním a s popisem očekávaných dat.

Data jsou následně rozdělena a upravena tak, aby odpovídala svým formátem funkci Mathematicy ListPlot pro vykreslení grafu.

```
4 ...
5 ListPlot[rozdeleni,PlotMarkers->{"\[FilledCircle]"},AxesOrigin->{0,
  0},PlotRangePadding->1,PlotStyle->{Red,Blue},PlotLabel->"Graf
  zadaneho rozmisteni prvku do trid"];
```

Pomocí této funkce se poté vykreslí data ve formě grafu s uvedeným rozmístěným prvků ve třídách, což je vlastně návratová hodnota funkce *VykresliPrvky*. Graf má nadefinované i vlastnosti, jako je jeho název, tvar a barva prvků, počátek souřadného systému a další parametry.

Ukázka výstupu



Obr. 10: Ukázka výstupu funkce VykresliPrvky

5.1.2 Funkce *Perceptron*

Perceptron nabývá dvou významů – jednak reprezentuje model perceptronu s určitým počtem vstupů a jediným výstupem. Tzn., že se jedná o datový typ, který v sobě uchovává data, což jsou váhy všech vstupů a také hodnotu prahu. Proměnnou tohoto typu vrací funkce *PerceptronTrenuj* používaná k učení sítě perceptron. Pokud se však jedná o funkci *Perceptron*, tak ta se využívá ke kontrole správné naučenosti konkrétního perceptronu. Funkce vyhodnotí data předložená naučenému perceptronu typu *Perceptron* a poté vrátí příslušné hodnoty na svém výstupu (0 nebo 1). Tím vlastně určí, do které ze dvou tříd daný prvek patří.

Funkce *Perceptron* používá následující parametry s výchozími hodnotami:

Tab. 1: Parametry funkce Perceptron

Název	Výchozí hodnota	Popis
UkazHranici	False	Určuje, zda se klasifikované prvky vykreslí v grafu spolu s hraniční přímkou, podle které perceptron klasifikuje prvky.

Implementace

Důležité úseky zdrojového kódu:

```
1 Options[Perceptron]={UkazHranici->False};
```

Před definicí funkce je třeba nastavit výchozí stav parametrů funkce uvedených v tabulce *Tab. 1*. V tomto případě je vykreslování hranice standardně deaktivováno.

```
2 Perceptron[{vahy_,prah_}][data_,priznaky:OptionsPattern[]]:=
```

Zde už je samotná definice funkce *Perceptron* následovaná opět definicí lokálních proměnných. Data v prvních hranatých závorkách jsou interní data proměnné typu *Perceptron* vrácená od funkce pro učení sítě. Uživatelsky zadaná data jsou položka *data* v druhých hranatých závorkách spolu s eventuálními parametry funkce zastoupenými položkou *priznaky*.

```
3 ...
4 ven=UnitStep[data.vahy+prah];
```

Po nezbytné kontrole korektnosti dat pokračuje kód funkce svou hlavní částí. Do proměnné *ven* se uloží finální výstup perceptronu. Ten je vypočítán klasicky podle rovnice (2), kdy je použita skoková unipolární přenosová funkce (příkaz UnitStep).

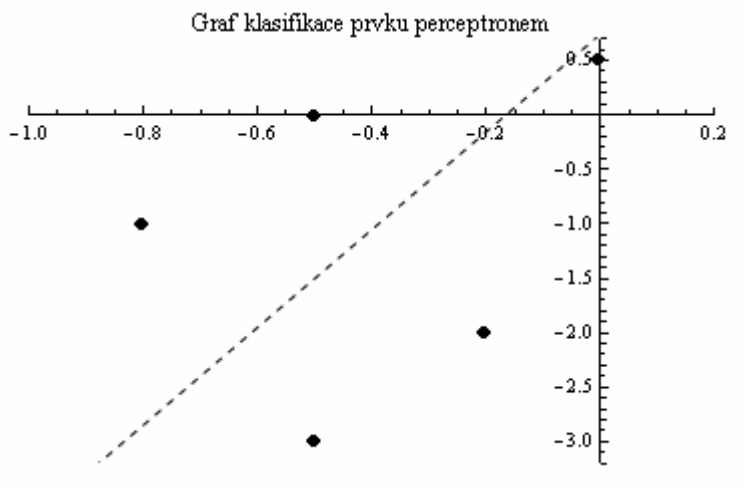
Poté se ve zdrojovém kódu zjišťuje, zda byl uživatelem zadán konkrétní parametr funkce *UkazHranici*. Pokud ano, tak jeho hodnotu uloží do zvláštní proměnné. Navíc pokud není parametr zadán ve správném tvaru, je vypsána chybová hláška a běh programu pokračuje dál se standardní hodnotou daného parametru.

```
5 ...
6 graf=Show[
7    ListPlot[data,AxesOrigin->{0,0},PlotRangePadding->0.2,
     PlotMarkers->{"\[FilledCircle]"},PlotStyle->{Black},PlotLabel->
     "Graf klasifikace prvku perceptronem"],
8    Plot[-vahy[[1]]/vahy[[2]]*h-prah/vahy[[2]],{h,Min[data]-1,
     Max[data]+1},PlotStyle->{Dashed,Black}];
9 ];
```

Do proměnné *graf* jsou uloženy 2 grafy, a to zobrazení rozmístění prvků obsažených v proměnné *data* a také graf hraniční přímky. Příkazem pro vykreslení Show se obsah proměnné *graf* vykreslí v Mathematice na obrazovku.

Na závěr funkce *Perceptron* se uvede proměnná *ven*, kterou bude funkce vracet na výstupu (návratová hodnota funkce).

Ukázka výstupu



Obr. 11: Grafický výstup funkce Perceptron

5.1.3 Funkce *PerceptronTrenuj*

Používá se k učení sítě perceptron. Zprostředkovává samotné hledání vah vstupů a prahu neuronu. Váhy jsou přepočítávány metodou fixních přírůstků. Funkce pracuje s daty trénovací množiny vstupů a výstupů. Vektor výstupů zastupuje třídy, do kterých spadají jednotlivé prvky definované ve vstupním vektoru. Počet tříd musí být roven 2 a navíc je pro správné naučení sítě nezbytné, aby bylo rozmístění prvků do tříd lineárně separabilní. K vizualizaci prvků ve třídách slouží již probíraná funkce *VykresliPrvky*.

Funkce vrací proměnnou typu *Perceptron*[$\{w, \Theta\}$], kde w je vektor nalezených vah vstupů perceptronu a Θ je nalezená hodnota prahu perceptronu.

Funkce *PerceptronTrenuj* používá následující parametry s výchozími hodnotami:

Tab. 2: Parametry funkce PerceptronTrenuj

Název	**Výchozí hodnota**	**Popis**
PocetEpoch	100	Určuje maximální počet epoch učení perceptronu.
UkazKlasifikaci	False	Určuje, zda se vykreslí sloupcové grafy se znázorněním zařazování prvků do tříd během učení.
UkazHranici	False	Určuje, zda se v grafu vykreslí hrance, podle které perceptron klasifikuje prvky do příslušných tříd.
PrirustekVah	1	Nastavuje hodnotu koeficientu, jež násobí váhy při jejich výpočtu metodou fixních přírůstků.

Implementace

Důležité úseky zdrojového kódu:

```
1 Options[PerceptronTrenuj]={PocetEpoch->100,UkazKlasifikaci->False,
  UkazHranici->False,PrirustekVah->1};
```

Nejprve opět definice parametrů funkce s jejich výchozími hodnotami. Parametry jsou uvedeny v tabulce *Tab. 2*.

```
2 PerceptronTrenuj[x_,y_,daneVahy_List:{},priznaky:OptionsPattern[]]:=
```

Definice funkce *PerceptronTrenuj*, kde proměnná *x* představuje vstupy a proměnná *y* výstupy z trénovací množiny. Proměnná *daneVahy* je nepovinná a obsahuje konkrétně definované inicializační váhy vstupů uživatelem. Pokud nejsou váhy uvedeny, nastaví se při inicializaci na náhodné hodnoty. Proměnná *priznaky* zahrnuje výčet všech parametrů, které jsou při volání funkce definovány.

Po běžném testu korektnosti dat předkládaných funkci *PerceptronTrenuj* pokračuje kód inicializační částí funkce.

```
3 ...
4 vahy=Table[Random[],{pom1},{1}];
```

Proces inicializace vah před spuštěním učení perceptronu – po získání správného rozměru vstupních dat *x* se váhy nastaví na náhodné hodnoty pomocí příkazu Random.

```
5 ...
6 If[Depth[daneVahy]!=Depth[x]||Length[daneVahy]!=pom1,
7   Message[PerceptronTrenuj::Vahy];Abort[],
8   vahy=daneVahy;
9 ];
```

V případě konkrétních vah zadaných uživatelem v proměnné *daneVahy* se však provede alternativní úsek kódu, který inicializační hodnoty vah přebírá (pokud je formát a rozměr dat v pořádku).

Poté, co se kontroluje zadání všech parametrů funkce, což se provádí obdobně jako v případě funkce *Perceptron*, se kontroluje počet tříd (musí být roven dvěma). Pokud je kontrola úspěšná, třídy se identifikují a data se podle těchto tříd rozkládají. Pro zjednodušení algoritmu učení je také nezbytné rozšíření vektoru tříd a vah a také úprava vektoru druhé třídy inverzí.

Nyní už může začít samotný proces učení.

```
10 ...
11 If[pracovni[[i,j]].vahy<=0,
12   vahy=vahy+prirustek*pracovni[[i,j]];
13   chyby++;
14 ];
```

Prochází se celkem 3 cykly, a to cyklus pro počet prvků ve třídě, cyklus pro celkový počet tříd a cyklus pro počet epoch učení. Proces učení spočívá díky úpravám pracovních vektorů (rozšířením a inverzí) pouze v testování součinu jednotlivých vstupů a jejich vah podle rovnice (11). Nová váha se v případě potřeby vypočítá přičtením dané váhy vynásobené koeficientem učení *c* obsaženém v proměnné *prirustek*. Také se registrují chybná zařazení prvků a další data pro pozdější využití zejména v grafech.

Tím vlastní učící proces končí.

```
15 ...
16 If[zmena===True,
17   Message[PerceptronTrenuj::Nenalezeno];
18   If[Length[Flatten[vahy]]==2,Print[VykresliPrvky[x,y]]];
19 ];
```

Vyčerpání počtu epoch učení při stále ještě nenaučeném perceptronu způsobí přerušení běhu programu za současného vypsání varovné hlášky (případně i s grafem zařazení prvků do tříd ve 2D prostoru pro vizuální kontrolu jejich rozmístění).

Následuje už jen výpis výsledků a konečná fáze funkce.

```
20 ...
21 graf1=ListPlot[Table[{i,vektorChyb[[i]]},{i,1,pom}],Joined->True,
   Filling->Axis,FillingStyle->Lighter[Yellow,0.8],AxesOrigin->{1,0},
   AxesLabel->{"Epochy","Chyby"},Ticks->{Range[0,pom,krok],
   Range[Min[vektorChyb],Max[vektorChyb]]},PlotLabel->"Zavislost chybne
   zarazenych prvku na epochach uceni"];
22
23 graf2=Show[
24   ListPlot[rozdelene,AxesOrigin->{0,0},PlotRangePadding->0.2,
     PlotMarkers->{"\[FilledCircle]"},PlotStyle->{Red,Blue},PlotLabel->
     "Graf nalezene hranice mezi tridami"],
25   Plot[-vahy[[1]]/vahy[[2]]*h-prah/vahy[[2]],{h,Min[rozdelene]-1,
     Max[rozdelene]+1},PlotStyle->{Dashed,Black}]
26 ];
27
```

```
28 graf3=BarChart[pom3,AxesLabel->{"Epochy","Prvky"},ChartLegends->
   {"Spravne zarazeno","Nespravne zarazeno"},ChartLabels->
   {Range[1,pom],None},ChartLayout->"Stacked",ChartStyle->
   {Lighter[Green,0.5],Lighter[Red,0.6]},PlotLabel->naz];
```

Pokud je uvedeno v parametrech funkce, vykreslí se také zvolené grafy – *graf1* obsahuje graf závislosti chybně zařazených prvků na epochách učení, *graf2* pak graf rozmístění prvků ve třídách spolu s hraniční přímkou a *graf3* obsahuje vývoj zařazování prvků do tříd.

```
29 If[pom1>1,
30   Print["Perceptron s ",pom1," vstupy byl uspesne naucen."],
31   Print["Perceptron s 1 vstupem byl uspesne naucen."]
32 ];
33
34 Print["\nVahy na vstupech pred procesem uceni: ",pom2,"\nFinalne
   nalezene vahy: ",vahy,"\nFinalne nalezeny prah: ",prah,"\nPocet
   epoch uceni: ",pom,"\n"];
```

Textový výstup funkce, který se vyobrazí po úspěšném naučení perceptronu.

```
35 vystupPerceptron=Perceptron[{vahy,prah}];
36 Return[vystupPerceptron];
```

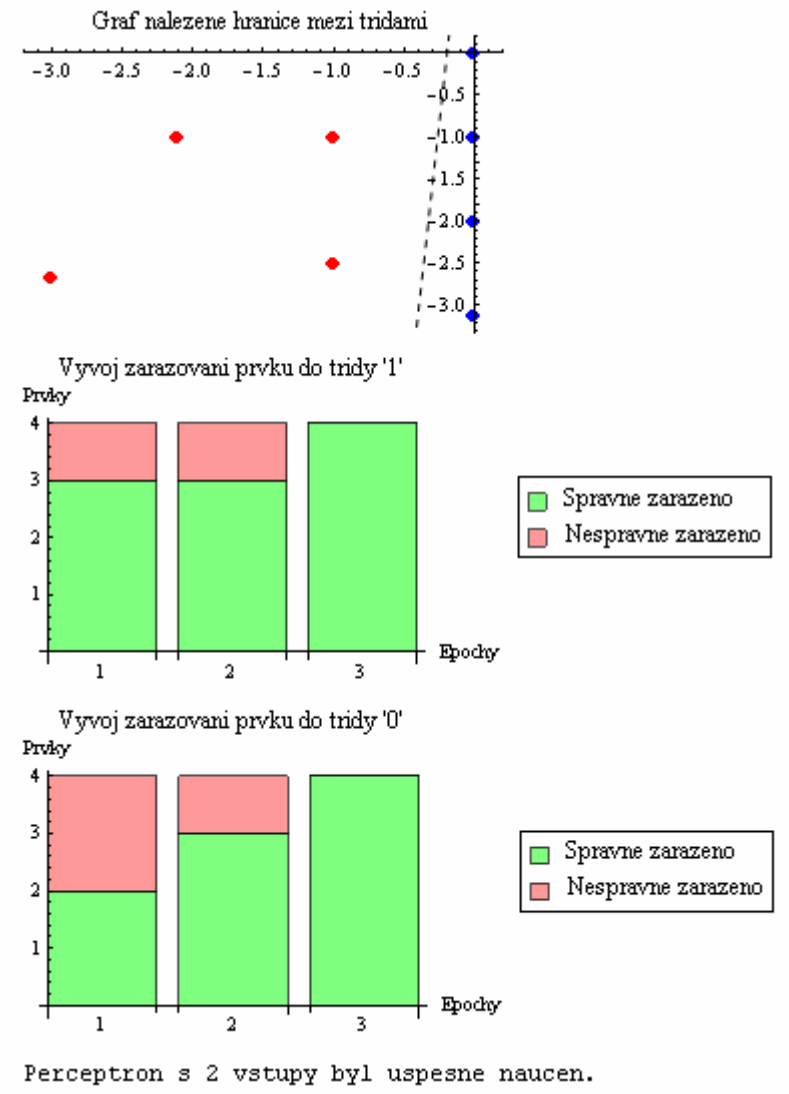
Návratová hodnota funkce *PerceptronTrenuj* je určena příkazem Return.

Ukázka výstupu


```
Perceptron s 3 vstupy byl uspesne naucen.

Vahy na vstupech pred procesem uceni: {{0.715617}, {0.381822}, {0.683855}}
Finalne nalezene vahy: {{-6.68438}, {4.62182}, {-2.07615}}
Finalne nalezeny prah: -3
Pocet epoch uceni: 17
```

Obr. 12: Výstup funkce PerceptronTrenuj

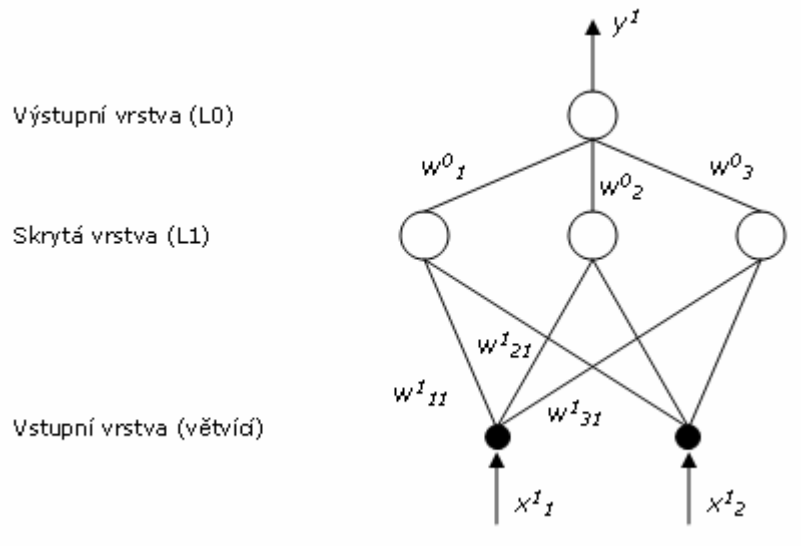


Obr. 13: Rozšiřující grafický výstup funkce PerceptronTrenuj

5.2 Vícevrstvá síť s dopředným šířením signálu

Síť s dopředným šířením signálu je vícevrstvá neuronová síť složená z neuronů perceptronovského typu. V tomto konkrétním případě se jedná o dvouvrstvou nerekurentní síť, tedy síť s jednou skrytou vrstvou a s výstupní vrstvou s jedním neuronem. Síť je tvořena libovolným počtem vstupů, jedním výstupem a jednou skrytou vrstvou s volitelným počtem neuronů. Propojení mezi vrstvami je úplné, tzn. že každý neuron dané vrstvy je spojen s každým neuronem vrstvy vyšší. Vstupní vrstva sítě pouze rozvětvuje signál na neurony v nejbližší vyšší vrstvě (skryté vrstvě). Tyto neurony, které jsou pro práci sítě nezbytné, transformují přicházející signál pomocí přenosové (aktivační) funkce jako klasický perceptron. Přenosové funkce neuronu v této vrstvě jsou logistické (sigmoidy) se

spojitým výstupem z intervalu (0, 1), zatímco přenosová funkce neuronu ve výstupní vrstvě je lineární. Je to z důvodu určení sítě zejména pro aproximaci dat z definičního oboru i mimo interval (0, 1). Zbývá také dodat, že v potaz jsou brány i prahy všech neuronů v síti.



Obr. 14: Obecná struktura vypracované neuronové sítě s dopředným šířením signálu

Struktura vícevrstvé sítě vyžaduje i specifický učící algoritmus. Zde byl zvolen nejpoužívanější algoritmus v sítích s dopředným šířením signálu, při kterém probíhá učení metodou zpětného šíření chyby (Back-Propagation). Jedná se o základní, nemodifikovanou verzi, nazývanou také jako delta pravidlo a patří mezi metody učení s učitelem. Z důvodu dosažení lepších výsledků při aplikaci sítě v aproximačních problémech byly do programu také zabudovány základní optimalizační techniky, jako je například zavedení setrvačnosti (momentum) a také adaptace prahů neuronů. Jako ukázka funkčnosti algoritmu Back-Propagation tato volba postačuje, pro přesnější výsledky by však bylo potřeba aplikovat další, výkonnější optimalizační algoritmy, jako například metodu Levenberg-Marquardt nebo Gauss-Newton.

Využití vícevrstvých sítí je velmi rozmanité. Protože je tento toolbox tvořen výhradně jako pomůcka pro výuku, byla funkčnost sítě koncipována speciálně pro požadovaný účel, což je aproximace funkcí a dat.

5.2.1 Funkce *VykresliBody*

Funkce *VykresliBody* vrací vizualizaci průběhu dat v 2D prostoru ve formě grafu. Tento graf nabývá 2 podob, a to podle počtu parametrů předkládaných funkci *VykresliBody*:

1. Funkce zobrazuje v grafické podobě data reprezentovaná vektorem vstupních dat a jemu odpovídajícím výstupním vektorem. Pomáhá tak vizualizovat rozmístění (průběh) dat, která jsou určena k trénování sítě s dopředným šířením signálu na aproximaci.
2. Funkce zobrazuje v grafické podobě kromě vstupních a výstupních dat trénovací množiny také vektor dat reprezentující primárně odezvy sítě na vstupní vektor. Pomáhá tak vizualizovat přesnost aproximace dat, na která se daná síť s dopředným šířením naučila.

Funkce *VykresliBody* vrací objekt typu *Graphics* (graf). Body zobrazené modrou barvou představují zadaná data trénovací množiny. Pokud je zadán i vektor odezev neuronové sítě, jsou aproximovaná data vyobrazena červenou barvou. Barevnost mezi jednotlivými body na společných vertikálách navíc určuje, zda jsou hodnoty jednotlivých dat po aproximaci menší (modrá) nebo větší (červená) než původní data.

Funkce *VykresliBody* nepoužívá žádné parametry.

Implementace

Důležité úseky zdrojového kódu:

```
1 VykresliBody[x_,y_]:=
2 ...
3 ven=Table[Flatten[{x[[i]],y[[i]]}],{i,1,Length[x]}];
4 ListPlot[ven,AxesOrigin->{0,0},AxesLabel->{"x","y"},PlotStyle->
  Directive[Blue,PointSize[0.01]]]
```

Výše uvedený kód představuje základní definici funkce *VykresliBody*, kde vektor *x* zastupuje x-ové souřadnice a vektor *y* y-ové souřadnice jednotlivých bodů ve 2D prostoru. Příkazem *Mathematicy* ListPlot se tyto body zobrazí v grafu jako plné modré kroužky.

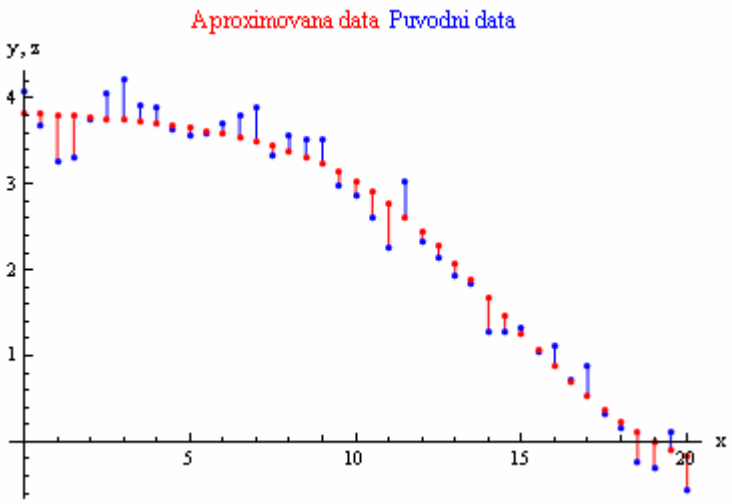
```
1 VykresliBody[x_,y_,z_]:=
2 ...
3 ven=Table[Flatten[{x[[i]],y[[i]]}],{i,1,Length[y]}];
```

```
4 ven2=Table[Flatten[{x[[i]],z[[i]]}],{i,1,Length[z]}];
5 ListPlot[{ven,ven2},AxesOrigin->{0,0},AxesLabel->{"x","y, z"},
  Filling->{1->{{2},{Red,Blue}}},PlotStyle->{Directive[Blue,
  PointSize[0.01]],Directive[Red,PointSize[0.01]]},PlotLabel->Style["
  Puvodni data ",Blue] Style[" Aproximovana data",Red]]
```

Zde je na druhou stranu rozšířená definice funkce *VykresliBody*, kde vektor *z* zastupuje výstupy naučené sítě (odezvy sítě na vektor *x*). Jednoduše lze říci, že se namísto jedné množiny bodů pro vykreslení v grafu vytvoří množiny dvě. Proměnná *ven* představuje body se souřadnicemi [*x*, *y*] a proměnná *ven2* se souřadnicemi [*x*, *z*]. Tyto proměnné pak reprezentují originální a aproximovaná data (barevně rozlišena).

Funkce *VykresliBody* nemá v obou variantách žádnou návratovou hodnotu, pouze vykresluje zadaná data (obdobně jako funkce *VykresliPrvky* u sítě Perceptron). Proto není v tomto případě požit příkaz Return.

Ukázka výstupu



Obr. 15: Ukázka jednoho z výstupů funkce VykresliBody

5.2.2 Funkce *DopredneSireni*

DopredneSireni nabývá obdobně jako v případě sítě perceptron dvou významů – Jednak reprezentuje model vícevrstvé neuronové sítě s dopředným šířením signálu. Síť obsahuje libovolný počet vstupů, jeden výstup a jednu skrytou vrstvu s volitelným počtem neuronů. Tzn., že se jedná o datový typ, který v sobě uchovává data, což jsou váhy spojů mezi vstupní a skrytou vrstvou, váhy spojů mezi skrytou a výstupní vrstvou a hodnoty prahů všech neuronů. V případě použití příkazu *DopredneSireni* jako funkce se jedná o kontrolu

správné naučenosti sítě. Funkce *DopredneSireni*[*x*] vrátí příslušné hodnoty z výstupu sítě, kde *x* jsou zadaná data testovací množiny přiváděná na vstup sítě.

Funkce *DopredneSireni* nepoužívá žádné parametry.

Implementace

Důležité úseky zdrojového kódu:

```
1 DopredneSireni[vahy_][data_]:=
```

Obdobná definice funkce (jako u perceptronu), která obsahuje dvojí data. Data *vahy* v prvních hranatých závorkách jsou interní data proměnné typu *DopredneSireni* vrácená od funkce pro učení sítě. Uživatelsky zadaná data jsou položka *data* v druhých hranatých závorkách. Proměnná pro zastupování parametrů funkce chybí, neboť funkce žádné parametry nepoužívá.

Po nezbytné kontrole korektnosti dat zdrojový kód pokračuje hlavním úsekem.

```
2 ...
3 pom=Inner[Times,data[[i]],vahy[[1]],Plus];
4 skryta=1/(1+E^(-(#1+#2)))&[pom,vahy[[3,1]]];
5 pom2=skryta.vahy[[2]]+vahy[[3,2,1]];
6
7 AppendTo[vystupni,pom2];
```

Hlavní úsek řeší vyhodnocování zvolených dat *data* neuronovou sítí. V cyklu přes všechny prvky těchto dat se postupně počítají vstupy pro přenosové funkce neuronů skryté vrstvy (*pom*), výstupy neuronů ve skryté vrstvě (*skryta*) a výstup neuronu ve výstupní vrstvě (*pom2*), což je zároveň odezva celé sítě. Jednotlivé odezvy jsou ukládány do vektoru *vystupni*, který je opět příkazem Return vracen jako návratová hodnota funkce. I ze samotného kódu je patrné, že přenosové funkce neuronů ve skryté vrstvě jsou sigmoidy, zatímco výstupní neuron přenosovou funkci nepoužívá (tj. vrací na výstupu to, co dostává na vstupu). V přeneseném významu tak lze říci, že tedy přenosovou funkci používá, a to konkrétně lineární.

5.2.3 Funkce *DopredneSireniTrenuj*

Funkce slouží k učení sítě s dopředným šířením signálu. Algoritmus učení je založen na metodě zpětného šíření chyby (viz kapitola 1.4.1 *Algoritmus Back-Propagation*), při

kterém se minimalizuje chyba sítě na výstupu. Tím se adaptují všechny váhy a prahy v síti. Protože se jedná o učení s učitelem, pracuje funkce s daty trénovací množiny (vstupů a výstupů). Touto množinou se definují data, která se pak síť pokusí aproximovat spojitou reálnou funkcí jedné proměnné.

Funkce vrací proměnnou typu *DopredneSireni*[*w*], kde *w* je vektor nalezených vah všech spojů v síti a také prahů všech neuronů ve skryté a výstupní vrstvě.

Funkce *DopredneSireniTrenuj* používá následující parametry s výchozími hodnotami:

Tab. 3: Parametry funkce DopredneSireniTrenuj

Název	**Výchozí hodnota**	**Popis**
PocetEpoch	100	Určuje maximální počet epoch učení.
MezChyby	0.1	Definuje globální chybu sítě, při jejíž dosažení se ukončí proces učení neuronové sítě.
DelkaKroku	0.1	Určuje koeficient učení η (viz rovnice (9)).
Setrvacnost	0	Definuje koeficient setrvačnosti μ (viz rovnice (9)). Hodnota 0 představuje žádnou setrvačnost.

Implementace

Důležité úseky zdrojového kódu:

```
1 Options[DopredneSireniTrenuj]={PocetEpoch->100,MezChyby->0.1,
  DelkaKroku->0.1,Setrvacnost->0};
```

Úvodem jsou definovány parametry funkce s jejich výchozími hodnotami. Popis všech parametrů je uveden v tabulce *Tab. 3*.

```
2 DopredneSireniTrenuj[x_,y_,neuronu_,priznaky:OptionsPattern[]]:=
```

Definice funkce *DopredneSireniTrenuj* sestává z dat a volitelných parametrů funkce v proměnné *priznaky*. Vstupy a výstupy trénovací množiny zastupují vektory *x* a *y*. Proměnnou *neuronu* se specifikuje počet neuronů ve skryté vrstvě ve formě celočíselné kladné hodnoty.

Po otestování uživatelem vkládaných dat na jejich správný formát a rozměr následuje také testování korektnosti zadaných parametrů funkce. To je prováděno stejným způsobem, jako u předešlých funkcí. Proto pokračuje kód až inicializací vah sítě.

```
3 vahy=RandomReal[UniformDistribution[{-0.2,0.2}],{pom1,neuronu}];
4 vahy=Append[{vahy},RandomReal[UniformDistribution[{-0.2,0.2}],
  {neuronu}]];
5 vahy=Append[vahy,{RandomReal[UniformDistribution[{-1,1}],
  {neuronu}],{RandomReal[{-1,1}]}}];
```

Váhy jsou zde nastaveny na své počáteční hodnoty. Tento krok je pro učení sítě důležitý a pro správnou inicializaci vah se používá řada specializovaných postupů. V tomto případě jsou váhy nastavovány na náhodné hodnoty z dostatečně malého intervalu. Nejdříve jsou vygenerovány váhy spojů mezi vstupní a skrytou vrstvou a poté spojů mezi vrstvou skrytou a výstupní. Na závěr je k vektorům vah připojen vektor prahů neuronů generovaný rovněž náhodně, ovšem z širšího intervalu (-1, 1).

```
6 ...
7 dynChyba=Dynamic[globCh];
8 dynEpocha=Dynamic[pom];
9 Print["Epocha: ",dynEpocha,"  Chyba site: ",dynChyba,"\n"];
```

V kódu je použita i ukázka práce s dynamickými proměnnými. Zmíněný úsek kódu například zobrazuje v průběhu učení sítě textový řetězec, jehož části, představované proměnnými *dynEpocha* a *dynChyba*, se mění v závislosti na změně dynamických proměnných, jež zastupují. Tím se uživateli během učení zobrazuje v reálném čase aktuální epocha a globální chyba sítě.

Nyní začíná vlastní proces učení sítě algoritmem Back-Propagation. Prochází přes dva cykly, a to cyklus počtu epoch učení a počtu prvků v trénovací množině. Průchod přes všechny prvky trénovací množiny má dvě fáze – aktivační a adaptační.

```
10 pom3=Inner[Times,do[[i]],vahy[[1]],Plus];
11 skryta=1/(1+E^(-(#1+#2)))&[pom3,vahy[[3,1]]];
12 vystupni=skryta.vahy[[2]]+vahy[[3,2,1]];
13
14 AppendTo[lokCh,0.5*(vystupni-od[[i]])^2];
```

Aktivační fáze zahrnuje průchod vstupu trénovací množiny přes síť až a její výstup. To prakticky odpovídá činnosti vybavování sítě jako u funkce *DopredneSireni*. Algoritmus je velmi podobný, jen s tím rozdílem, že se do vektoru *lokCh* neukládá daný výstup sítě, ale chyba výstupní odezvy ve srovnání s požadovaným výstupem trénovací množiny (viz rovnice (4)). Proměnná *vystupni* představuje právě aktuální výstup sítě a proměnná *od i*-tý požadovaný výstup.

Po aktivační fázi následuje adaptační fáze, což zahrnuje zpětné šíření chyby a modifikaci vah a prahů sítě.

```
15 grad=od[[i]]-vystupni;
16 gradVekt=skryta*(1-skryta)*grad*vahy[[2]];
17
18 dVstup=Outer[Times,do[[i]],(prirustek*gradVekt)]+alfa*d1stare;
19 dSkryta=prirustek*grad*skryta+alfa*d2stare;
20 dPrahySkryta=prirustek*gradVekt+alfa*d3stare;
21 dPrahVen=prirustek*grad+alfa*d4stare;
```

Podle vztahu (8) se vypočítají chyby všech neuronů sítě, kdy se při výpočtech postupuje směrem od výstupu sítě na vstup sítě. Tyto chyby představují proměnné *grad* (chyba výstupního neuronu) a *gradVektor* (chyby skrytých neuronů). Pomocí takto získaných chyb se poté v souladu s rovnicí (9) vypočítají změny jednotlivých vah a prahů sítě vedoucích k minimalizaci celkové chyby sítě. Ty jsou zastoupeny proměnnými s předponou *d-*.

```
22 vahy[[1]]=vahy[[1]]+dVstup;
23 vahy[[2]]=vahy[[2]]+dSkryta;
24 vahy[[3,1]]=vahy[[3,1]]+dPrahySkryta;
25 vahy[[3,2]]=vahy[[3,2]]+dPrahVen;
```

Na závěr adaptační fáze je definována finální změna vah a prahů, která nastává po každém předložení prvku z trénovací množiny. Celkový počet modifikací vah a prahů je dán součinem počtu epoch učení s počtem prvků trénovací množiny. Váhy a prahy se upravují podle vztahu (10) a postupuje se opět od nejnižší vrstvy do vrstev vyšších, tedy směrem od vstupu na výstup.

```
26 ...
27 globCh=Total[lokCh];
28 If[globCh<=mez,Break[]];
```

Po každém průběhu všech prvků trénovací množiny (tedy v každé epoše) se po procesu úprav vah počítá globální chyba sítě (viz vtah (5)) a porovnává se s tolerancí chyby zadané uživatelem. Pokud je globální chyba menší než tato mez, proces učení končí a síť je prohlášena z tohoto hlediska za naučenou.

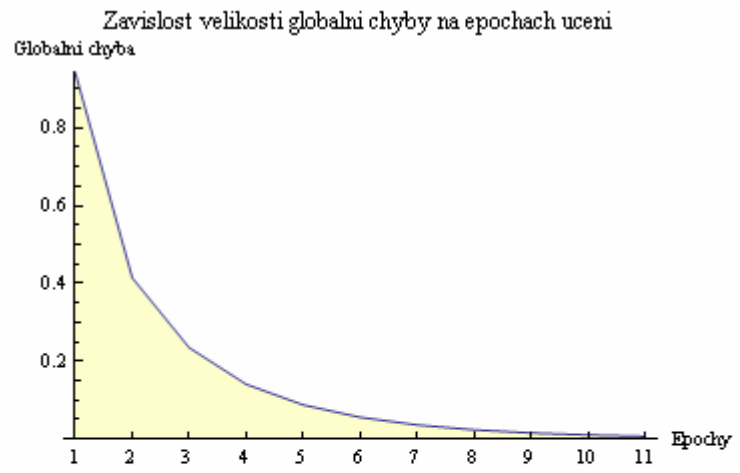
Po ukončení procesu učení sítě se už jen klasickým způsobem zobrazí graf závislosti velikosti globální chyby sítě na epochách učení, z kterého je dobře patrná případná

konvergence sítě k optimu. Nechybí samozřejmě ani textový výstup se souhrnnými informacemi o průběhu učení. Oba jmenované výstupy lze vidět na obrázku *Obr. 16*.

```
29 vystupDopredna=DopredneSireni[vahy];
30 Return[vystupDopredna];
```

Příkazem pro výstup se již známým způsobem definuje návratová hodnota funkce.

Ukázka výstupu




```
Neuronova sit s doprednym sirenim byla naucena.

Pocet epoch uceni: 11
Pocet vstupu site: 1
Pocet neuronu ve skryte vrstve: 5
Celkovy pocet vah site: 10

Tolerovana mez chyby site: 0.01
Finalni chyba site: 0.00728611
```

Obr. 16: Ukázka výstupu funkce DopredneSireniTrenuj

5.3 Hopfieldova síť

Neuronová síť Hopfield nese své pojmenování po Johnu Hopfieldovi, který se zabýval studiem autoasociativních neuronových sítí.

Struktura sítě je jednovrstvá a zpětnovazební a všechny neurony perceptronovského typu jsou vzájemně propojeny každý s každým *Obr. 6*. Počet neuronů sítě odpovídá počtu vstupů. Vstupní údaje zpracovává každý neuron jako klasický perceptron sumací součinu vah s odpovídajícími vstupy podle rovnice (1) a poté přepočítává prostřednictvím

přenosové funkce na výstupní údaje. Práh je totiž u všech neuronů pro jednoduchost nastaven pevně na hodnotu 0. Výstup sítě je dosažen až po posledním kroku vybavování, kdy se dosáhne stabilního stavu sítě - výstup sítě se pak skládá z aktuálních stavů všech neuronů.

Využití Hopfieldovy sítě spočívá v autoasociaci. Odpovědi sítě na předložený vzor je přímo nalezený vzor. Navíc může být předložený vzor poškozený a síť přesto vrátí původní nepoškozený vzor.

5.3.1 Funkce *VykresliZnaky*

Zobrazuje v grafické podobě data (znaky) určená pro předložení neuronové síti Hopfield k učení. Pomáhá tak vizualizovat znaky pro snadnější přehled uživatele. Znaky se zadávají jednotlivě, a to po řádcích, kde hodnota -1 odpovídá bílému bodu a +1 černému bodu. Zadávat je možno spojité i diskrétní hodnoty, funkce dokáže zobrazovat oba typy dat.

Funkce *VykresliZnaky* nemá žádnou návratovou hodnotu, pouze zobrazuje znaky zadané v předloženém vektoru. Funkce zobrazuje jak diskrétní, tak spojité hodnoty reprezentující znaky.

Funkce *VykresliZnaky* nepoužívá žádné parametry.

Implementace

Důležité úseky zdrojového kódu:

```
1 VykresliZnaky[vektor_List]:=
```

Definice funkce *VykresliZnaky* obsahuje jedinou vstupní proměnnou *vektor*. Jedná se o vektor dat, která představují znaky určené primárně pro učení Hopfieldovy sítě.

```
2 ...
3 pom=Count[#,_Integer,Infinity]&/@vektor;
4 pom1=Count[#,_Real,Infinity]&/@vektor;
5 pom=pom+pom1;
```

Ještě před testem korektnosti dat jsou spočítány všechny členy prvků vektoru *vektor* (celočíselné i reálné hodnoty). Podle těchto výsledků je kontrolována platnost vstupních dat a stejnoměrnost jejich složení.

```
6 If[pom1[[1]]>0,
7    ven=Row[ArrayPlot[#,Mesh->All,Frame->True,AspectRatio->1,
     ColorFunction->"GrayTones",ColorRules->{1.->White}]&/@-vektor],
8    ven=Row[ArrayPlot[#,Mesh->All,Frame->True,AspectRatio->1,
     ColorRules->{1->Black,-1->White}]&/@vektor];
9 ];
```

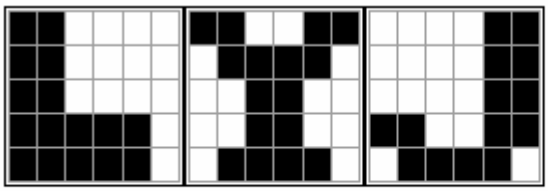
Následuje naplnění proměnné *ven* grafickou řadou znaků ve formě mřížek s jednotlivými body prvků vstupního vektoru (viz obrázek *Obr. 17*). Proměnná *ven* se vykresluje ve dvou alternativách určených typem dat. Pokud obsahuje vektor *pom1* alespoň jeden prvek, znamená to, že se zadaná data byla identifikována jako spojitá, jinak se jedná o data diskrétní (jedná se hlavně o upřesnění barevnosti vykreslovaných bodů).

Textový výstup funkce vypisuje počet vykreslovaných znaků a také důležitou minimální Hammingovu vzdálenost všech znaků. K tomu účelu byla sestavena funkce *MinHamming*, která má využití i v jiných funkcích týkající se práce s Hopfieldovou sítí.

```
1 MinHamming[vektor_]:=
2 ...
3 pom=Map[Flatten,vektor,{1}];
4 pom2=Permutations[pom,{2}];
5 pom3=HammingDistance[#[[1]],#[[2]]]&/@pom2;
6
7 Return[Min[pom3]];
```

Funkce *MinHamming* pracuje s konkrétním vektorem *vektor*, který obsahuje v číselné formě znaky, jejichž minimální Hammingovu vzdálenost funkce vrací. Pomocná proměnná *pom* pouze upravuje formát vstupních dat. Teprve potom se počítá vektor Hammingových vzdáleností (*pom3*) všech kombinací zadaných znaků (*pom2*). Funkce finálně vrací minimální hodnotu tohoto vektoru.

Ukázka výstupu



```
Zobrazeny 3 prvky s minimalni Hammingovou vzdalenosti 14.
```

Obr. 17: Ukázka výstupu funkce VyzkesliZnaky

5.3.2 Funkce *PoskodZnaky*

Funkce způsobuje poškození (zašumění) předložených znaků určených k asociaci Hopfieldovou sítí. Poškození spočívá v náhodné změně několika hodnot vybraných bodů ve znacích. Funkce může být nepovinně doplněna konkrétním intervalem pro rozsah rovnoměrného rozdělení používaného k ovlivnění míry zašumění znaků. Zadává se ve formě jednoduchého vektoru 2 prvků. Pokud není interval uveden, je použit výchozí interval {-0.1, 1}.

Funkce *PoskodZnaky* vrací data (znaky) ve formě vektoru hodnot, která jsou určena přímo pro Hopfieldovu síť. Hodnota -1 odpovídá zcela bílému bodu a +1 zcela černému bodu. Funkce zpracovává jak diskrétní, tak spojité hodnoty reprezentující znaky. To se volí při volání funkce parametrem *TypFunkce* (viz tabulka *Tab. 4*).

Funkce *PoskodZnaky* používá následující parametry s výchozími hodnotami:

Tab. 4: Parametry funkce PoskodZnaky

Název	**Výchozí hodnota**	**Popis**
TypFunkce	Diskretni	Určuje typ poškození zobrazovaných znaků. Možné volby jsou Spojita nebo Diskretni.

Implementace

Důležité úseky zdrojového kódu:

```
1 Options[PoskodZnaky]={TypFunkce->Diskretni};
```

Inicializace parametru *TypFunkce* jeho výchozí hodnotou. Viz tabulka *Tab. 4*.

```
2 PoskodZnaky[vektor_List,interval_List:{},priznaky:OptionsPattern[]
  ]:=
```

Definice funkce *PoskodZnaky*, která pracuje se dvěma proměnnými a eventuálním výčtem parametrů. Proměnná *vektor* představuje vstupní data a proměnná *interval* rozmezí intervalu rovnoměrného rozdělení používaného k ovlivnění míry zašumění znaků.

Funkce má podobnou strukturu jako funkce *VykresliZnaky*. V potaz je však brána míra zašumění dat před jejich vykreslením.

```
3 Switch[Length[interval],
4    0,pomInterval={-0.1,1},
5    2,pomInterval=interval,
6    _,Message[PoskodZnaky::Options];Abort[];
7 ];
```

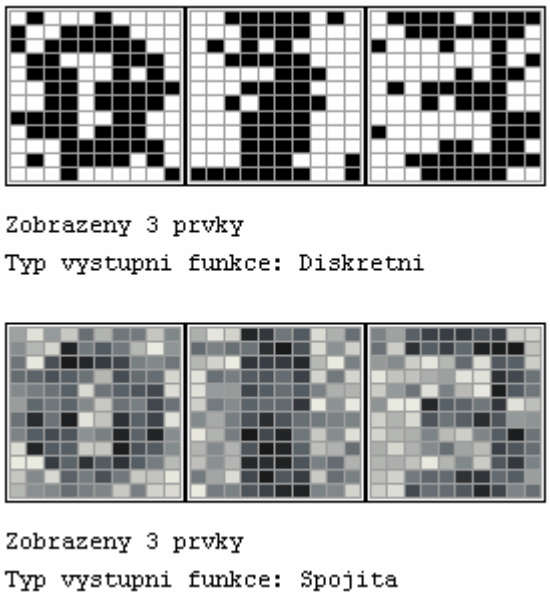
Při kontrole délky intervalu *interval* se upřesňuje jeho konkrétní hodnota. V případě jeho přímého definování uživatelem je tato hodnota převzata, pokud ovšem není interval zadán, je nastavena výchozí hodnota {-0.1, 1}. Při nastání jakékoliv jiné možnosti je vypsána chybová hláška.

```
8 ...
9 pom3=Map[Flatten,vektor,{1}];
10 cisla=RandomReal[UniformDistribution[pomInterval],
   {Length[vektor],pom[[1]]}];
11 pom4=pom3*cisla;
12 If[typ===Diskretni,pom4=Sign[pom4]];
```

Princip samotného zašumění spočívá ve vygenerování vektoru náhodných hodnot o stejné struktuře, jako je struktura originálních nepoškozených dat (proměnná *cisla*). Výstupní vektor zašuměných dat je pak získán jednoduše součinem originálních dat (*pom3*) s daty náhodně vygenerovanými (*cisla*). Rozšiřujícím prvkem kódu je úprava výstupního vektoru tak, aby odpovídal diskrétním hodnotám v případě takto zvoleného typu pomocí parametru funkce. To zajistí funkce signum aplikovaná na zmíněný vektor.

Grafický a textový výstup funkce *PoskodZnaky* je pak definován obdobně jako u funkce *VykresliZnaky*. Jedná se o grafické řady znaků ve formě mřížek, kde každý bod mřížky zastupuje každou číselnou hodnotu z definice znaku. Textový výstup informuje o počtu vykreslených znaků a o typu zašumění znaků (viz obrázek *Obr. 18*).

Ukázka výstupu



Obr. 18: Ukázka obou možných grafických výstupů funkce PoskodZnaky

5.3.3 Funkce *Hopfield*

Hopfield stejně jako u předchozích dvou sítí nabývá dvou významů. Jednak reprezentuje model Hopfieldovy sítě s daným počtem vstupů (neuronů). Jedná se o datový typ, který v sobě uchovává data, což je matice vah všech spojů mezi neurony v síti. Také nese informaci o tom, zda pracuje síť s přenosovými funkcemi se spojitými nebo diskrétními výstupy a tím určuje, pro jaký vstupní signál je síť připravena. Jako funkce se *Hopfield* opět používá ke kontrole správné naučenosti Hopfieldovy sítě. Funkce vyhodnotí předložená vstupní data (určité vzory) a poté vrátí po dosažení stabilního stavu příslušné hodnoty na svých výstupech – diskrétní nebo spojité (podle typu sítě). Ty je pak možno vizualizovat funkcí *VykresliZnaky* (viz kap. 5.3.1) nebo přímo volbou parametrů funkce.

Funkce *Hopfield* používá následující parametry s výchozími hodnotami:

Tab. 5: Parametry funkce Hopfield

Název	Výchozí hodnota	Popis
UkazAsociace	False	Určuje, zda se vykreslí postupné asociace při vybavování vzorů.

Implementace

Důležité úseky zdrojového kódu:

```
1 Options[Hopfield]={UkazAsociace->False};
2 Hopfield[vahy_,sit_][data_List,priznaky_:{UkazAsociace->False}]:=
```

Na počátku se klasicky definují výchozí hodnoty parametrů funkce a také vlastní funkce. Ta je navrhnuta stejným způsobem jako u předchozích dvou typů sítí. Data *vahy* a *sit* v prvních hranatých závorkách jsou interní data proměnné typu *Hopfield*, která nelze při použití funkce měnit. Tato data se nastavují po skončení učení Hopfieldovy sítě prostřednictvím návratové hodnoty funkce *HopfieldTrenuj*. Uživatelsky zadaná data jsou položka *data* v druhých hranatých závorkách spolu s eventuálními parametry funkce zastoupenými položkou *priznaky*. Přehled příznaků funkce *Hopfield* je uveden v tabulce *Tab. 5*.

Kontrola korektnosti dat, která v této fázi nastává pravidelně, nespočívá jenom v testování správného počtu, typu a rozměru číselných hodnot znaků. Musí souhlasit i rozměr dat vzhledem ke struktuře sítě, které je předkládáme. Pokud toto není splněno, běh funkce se přeruší a uživatel je upozorněn příslušnou chybovou hláškou.

Další částí pokračuje kód procesem vybavování.

```
3 ...
4 If[typ===Diskretni,
5   pom3=Sign[pom2.vahy+prah];
6   pom4=ReplaceAll[pom3,0->-1],
7
8   pom4=Tanh[pom2.vahy+prah];
9 ];
```

Protože je proces vybavování narozdíl od jednorázového učení iterativní, probíhá v cyklech. Hlavní cyklus zkoumá stav výstupů neuronů a pokud zůstávají stabilní (tzn. nedochází už k jejich změnám), vybavování je u konce. Výstupy neuronů jsou počítány standardně podle vztahu (2), kde tvar přenosové funkce je dán typem sítě definovaným parametrem funkce. Diskrétní typ představuje funkce signum, přičemž hodnoty 0 jsou nahrazeny hodnotami -1 (tím je získána bipolární skoková funkce namísto unipolárního výstupu funkce signum). Při volbě spojité přenosové funkce je použit hyperbolický tangens se spojitým výstupem (-1, 1).

```
10 ...
11 If[typ===Diskretni,
12    AppendTo[grafy,ArrayPlot[#,Mesh->All,Frame->True,AspectRatio->1,
      ColorRules->{1->Black,-1->White}]&/@ven];
13 ];
14 If[typ===Spojita,
15    AppendTo[grafy,ArrayPlot[#,Mesh->All,Frame->True,AspectRatio->1,
      ColorFunction->"GrayTones",ColorRules->{1.->White}]&/@-ven];
16 ];
```

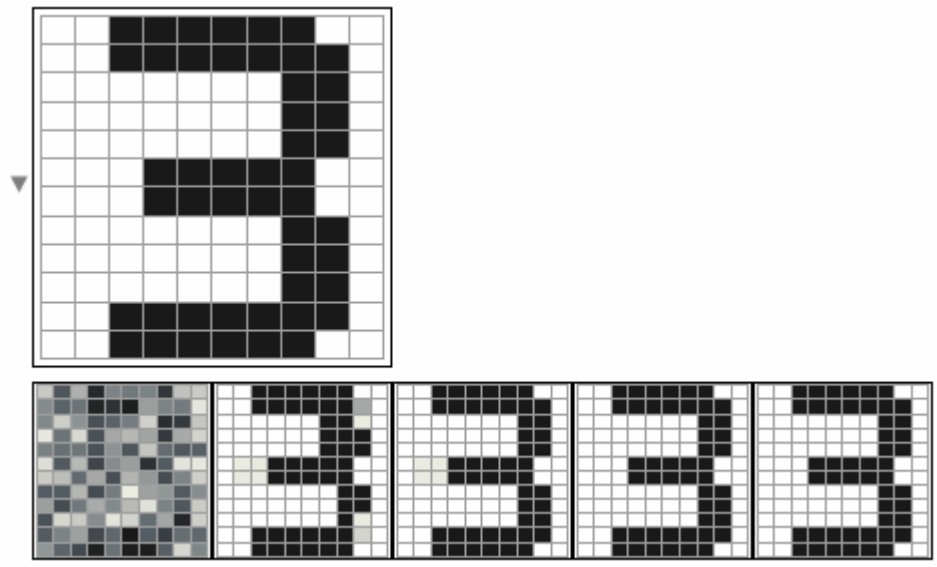
V tomtéž cyklu probíhá připojení aktuálního stavu výstupu sítě pro všechny znaky do množiny, která slouží po ukončení funkce ke grafické vizualizaci průběhu vybavování sítě. Typ sítě opět ovlivňuje pouze barevné vyobrazení jednotlivých bodů znaků.

```
17 ...
18 OpenerView[{Show[grafy[[delka,i]]],Row[Table[Flatten[grafy[[j,i]]],
   {j,prvku[[1]]}]]}];
```

Tímto příkazem se definuje grafický rozbalovací seznam průběhu vybavování pro každý jednotlivý znak.

Nakonec už jen následuje příkaz Return pro určení návratové hodnoty funkce. Jedná se o asociované znaky ve formě stejných číselných matic, jaké jsou například přiváděny i na vstup funkce *Hopfield* či *VykresliZnaky*.

Ukázka výstupu



Obr. 19: Volitelný grafický výstup funkce Hopfield

5.3.4 Funkce *HopfieldTrenuj*

Funkce *HopfieldTrenuj* se používá k učení Hopfieldovy sítě. Pracuje s daty trénovací množiny, která reprezentují vzory určené k zapamatování. Znaky se zadávají jednotlivě, a to po řádcích, kde hodnota -1 odpovídá zcela bílému bodu a +1 zcela černému bodu.

Funkce vrací proměnnou typu *Hopfield*[*w,druh*], kde *w* je matice nalezených vah spojů mezi neurony a *druh* je typ sítě (diskrétní nebo spojitá).

Je nutné také poznamenat, že vstupní data by měla obsahovat méně znaků, než je 15 % neuronů (vstupů) dané Hopfieldovy sítě. Toto je známé omezení Hopfieldovy sítě a jeho nesplnění může vést ke špatnému naučení sítě a tím pádem ke špatné asociaci naučených vzorů. Také minimální Hammingova vzdálenost předložených znaků by měla být co největší, aby nedocházelo k nesprávným asociacím.

Funkce *HopfieldTrenuj* používá následující parametry s výchozími hodnotami:

Tab. 6: Parametry funkce HopfieldTrenuj

Název	Výchozí hodnota	Popis
TypSite	Diskretni	Určuje typ trénované Hopfieldovy sítě (přenosových funkcí). Možné volby jsou Spojita nebo Diskretni.

Implementace

Důležité úseky zdrojového kódu:

```
1 Options[HopfieldTrenuj]={TypSite->Diskretni};
2 HopfieldTrenuj[x_List,priznaky:OptionsPattern[]]:=
```

Definice parametrů funkce s jejich inicializační hodnotou a definice funkce samotné se vyskytují samozřejmě hned na počátku zdrojového kódu. Funkce *HopfieldTrenuj* pracuje mimo její parametry pouze s jednou proměnnou *x* typu vektor. Vektor obsahuje další vektory (resp. matice) celočíselných diskrétních hodnot, které představují znaky určené pro učení sítě.

Po nezbytné kontrole formátu, typu a rozměru vstupních dat se kontroluje také typ sítě, jenž byl určen parametrem *TypSite*. Zde se podle hodnoty tohoto parametru uloží

informace o tom, zda mají být dále v kódu použity přenosové funkce neuronů diskrétního nebo spojitého charakteru.

Následuje stěžejní úsek kódu reprezentující učení Hopfieldovy sítě.

```
3 ...
4 pracovni=Map[Flatten,x,{1}];
5 pom3=Outer[Times,#,#]&/@pracovni;
6 pracovni=N[Plus@@pom3];
7 vahy=ReplacePart[pracovni,{i_,i_}->0];
```

Speciálně u Hopfieldovy sítě se v případě procesu učení nejedná o iterativní děj, jak bylo zvykem doposud, ale jde o jednorázový proces nastavení všech vah v síti. Váhy spojů mezi neurony, které jsou propojeny každý s každým, tvoří diagonálně symetrickou čtvercovou matici. Je to z toho důvodu, že váhy, které jsou na spojích z jednoho neuronu do druhého, jsou v obou směrech stejné. Proces učení (nastavení vah) tedy představuje základní maticové operace, kdy jsou vynásobeny všechny znaky mezi sebou a součtem těchto dílčích maticí se dostane výsledná matice vah. Navíc se na diagonále matice nastaví hodnoty vah na hodnotu 0, protože na vstupy neuronů nejsou přiváděny jejich vlastní výstupy.

```
8 ...
9 vzdalenost=MinHamming[x];
10 If[vzdalenost==0,Message[HopfieldTrenuj::Hamming]];
11 If[pom2>(0.15*pom1),Message[HopfieldTrenuj::MaxZnaku,pom2,Round[
   (0.15*pom1),.1]]];
```

Před vlastním výstupem funkce jsou také kontrolovány podmínky správné naučenosti Hopfieldovy sítě. Volá se proto funkce *MinHamming* a zjišťuje se tak minimální Hammingova vzdálenost všech znaků. Pokud je takto nalezená vzdálenost nulová (například z důvodu dvou zadaných totožných znaků), vypíše se uživateli varovná hláška. Jiná hláška se vypíše v případě, že je počet znaků určených k asociaci menší než 15% neuronů v síti. To je známé omezení Hopfieldovy sítě a v případě nesplnění této podmínky není zaručeno korektní naučení sítě. Proto je uživatel informován i o této události.

```
12 vystupHopfield=Hopfield[vahy,TypSite->typ];
13 Return[vystupHopfield];
```

Posledním krokem je naplnění proměnné *vystupHopfield*, která se stává použitím příkazu Return návratovou hodnotou funkce *HopfieldTrenuj*.

Ukázka výstupu

```
Hopfield s 30 neurony byl uspesne naucen.


Typ Hopfieldovy site: Diskretni
Celkovy pocet vah site: 900
Pocet naucenych vzoru: 3
```

Obr. 20: Textový výstup funkce HopfieldTrenuj

5.4 Dokumentace k toolboxu

Spolu se samotným vytvořením toolboxu byla vypracována také nápověda, která může sloužit i jako dokumentace. Účelem nápovědy je vysvětlit především začínajícím uživatelům funkčnost toolboxu a objasnit jim jeho používání.

Na tvorbu dokumentačních textů ve formě nápovědy lze v Mathematice využít výkonný nástroj Wolfram Workbench. Tento nástroj je však složitější a lehce těžkopádný, nicméně pomocí něj lze vytvořit k rozšiřujícím toolboxům nápovědný systém profesionální úrovně. Taková nápověda se používá především v nejnovější verzi Mathematicy a využívá i podporující prvky, jako jsou například vizuální efekty v jazyce Java a podobně. Dalším možným způsobem je tvorba klasické nápovědy staršího typu, která je kompatibilní s nápovědou Mathematicy verze 5.2. Tento typ lze poměrně jednoduše vypracovávat manuálně v Mathematice, a proto byla nápověda k tomuto toolboxu dělána tímto způsobem.

Základem nápovědy je zdrojový soubor *BrowserCategories.m*, který vlastně obsahuje její stromovou strukturu. Obsahuje popisy jednotlivých sekcí nápovědy s definicí klasických notebook souborů s konkrétním obsahem. Obsah tohoto souboru je taktéž uveden v příloze práce a na přiloženém disku CD.

Struktura nápovědy toolboxu Neuronové sítě je následující:

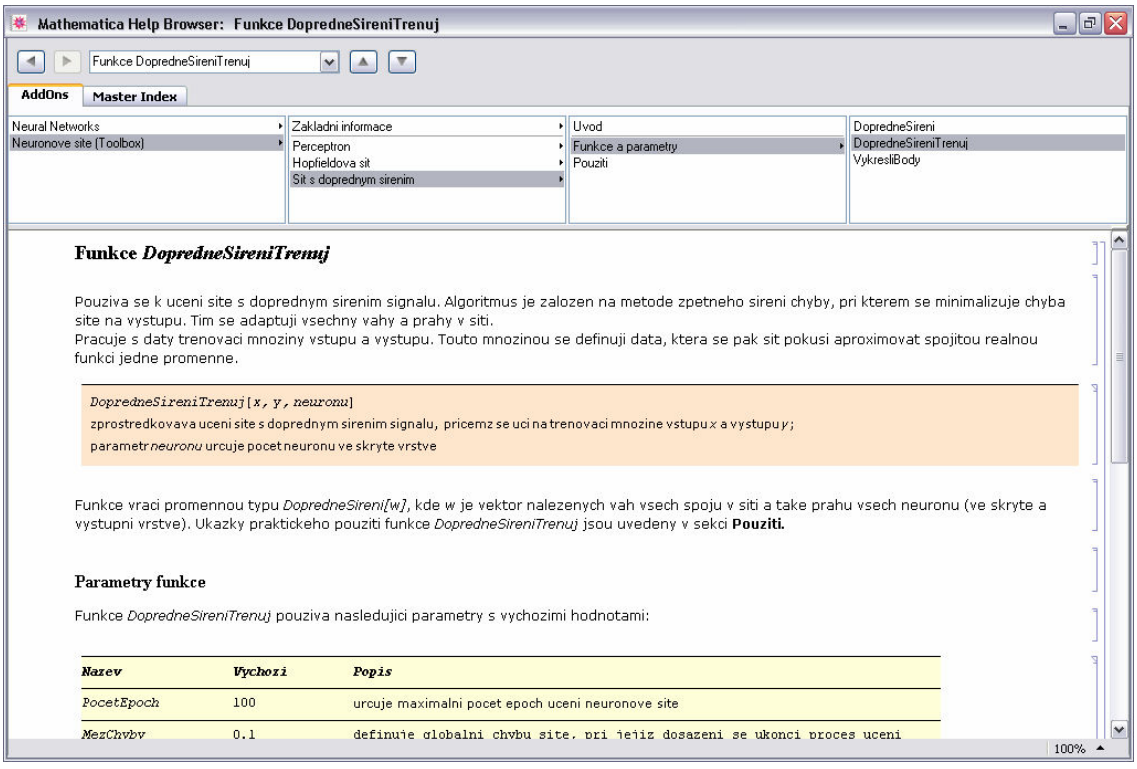
- Základní informace
    - Představení
    - Přehled zpracovaných sítí
    - Podmínky užití
- Perceptron

- Hopfieldova síť
- Síť s dopředným šířením

V sekci *Představení* je vysvětlen účel vzniku toolboxu a základní údaje o jeho tvorbě, jako je verze Mathematicy, ve které byl tvořen nebo zpětná kompatibilita toolboxu. Také je uvedeno základní vysvětlení problematiky neuronových sítí. Názorný výčet všech funkcí je vypsán v oddíle *Přehled zpracovaných sítí*. Sekce *Podmínky užití* slouží k představení podmínek, za kterých se může toolbox využívat a je zde uveden i e-mailový kontakt na autora toolboxu a na vedoucí diplomové práce.

Každá síť pak obsahuje podčásti *Úvod*, *Funkce a parametry* a *Použití*. V sekci *Úvod* jsou uvedené teoretické informace vztahující se k dané síti, jako je struktura sítě, její funkčnost, způsob učení a využití. Oddíl *Funkce a parametry* obsahuje popisy jednotlivých funkcí používané pro práci se sítí a v sekci *Použití* jsou uvedeny krátké ukázky používaní všech těchto funkcí a jejich parametrů.

K nápovědě se přistupuje přes dokumentační centrum Mathematicy vyvolané klávesou F1. Obsah nápovědy je pak snadno přístupný pod jednotlivými nápovědami k rozšiřujícím balíčkům (Add-Ons) Mathematicy.



Obr. 21: Ukázka použití nápovědy k toolboxu

Hlavním motivem dokumentace je představení funkcí používaných k práci s neuronovými sítěmi jak je vidět na obrázku *Obr. 21*. Užitečné jsou i ukázky jejich použití, na kterých si uživatel osvětlí základní postupy při práci s danou sítí.

5.5 Implementace toolboxu do Mathematicy

Toolbox byl tvořen systémem balíčků (Packages) představujících jednotlivé sítě. Název balíčku (toolboxu) jako celku zní „NeuronoveSite“. Jedná se vlastně o adresář a ten se sestává ze zdrojových souborů a dalších adresářů. Struktura souborů toolboxu je následující:

- \NeuronoveSite
  - \Documentation
    - \English
      - *BrowserCategories.m*
      - *NS_DopredneSireni.nb*
      - *NS_Hopfield.nb*
      - *NS_Perceptron.nb*
      - *Predstaveni.nb*
      - *Prehled.nb*
      - *Uziti.nb*
  - \Kernel
    - *init.m*
  - *pracovnidata.dat*
  - *NeuronoveSite.m*
  - *NSDopredneSireni.m*
  - *NSHopfield.m*
  - *NSPerceptron.m*

K řádné implementaci toolboxu do Mathematicy je zapotřebí nahrát složku *NeuronoveSite* s toolboxem do adresářové struktury, kde je Mathematica nainstalována, a to konkrétně do adresáře „\AddOns\Applications“ (platí pro systém Microsoft Windows).

Pro použití toolboxu v Mathematice už jej pak jen stačí po spuštění kernelu Mathematicy nalinkovat jedním z následujících příkazů (viz také *Obr. 9*):

- <<NeuronoveSite`
- Needs["NeuronoveSite`"]

Od této chvíle už by měl uživatel bez problémů moci využívat všechny funkce vypracovaného toolboxu *NeuronoveSite*.

ZÁVĚR

Hlavním cílem při tvorbě této diplomové práce bylo vytvořit vhodnou pomůcku pro práci s neuronovými sítěmi v předmětu Metody umělé inteligence vyučovaném na Fakultě aplikované informatiky Univerzity Tomáše Bati ve Zlíně. Po dohodě s vedoucí práce byly konkrétně stanoveny neuronové sítě jednoduchý Perceptron, vícevrstvá síť s dopředným šířením signálu s učícím algoritmem Back-Propagation a Hopfieldova síť. Pomůcka měla být vyhotovena v prostředí Mathematica jako takzvaný toolbox (balíček).

Celkově se podařilo úspěšně naprogramovat všechny zadané sítě. Jednoduchý Perceptron byl vypracován tak, aby měl využití v problematice lineární klasifikace prvků ve dvou třídách. Síť s dopředným šířením signálu najde uplatnění v obecné aproximaci dat (funkcí). Hopfieldova síť zase demonstruje její autoasociační schopnosti při identifikaci zašuměných znaků. Navržená funkcionalita u všech tří sítí je založena na stejném principu. Každá síť obsahuje vlastní funkce, pomocí kterých se se sítí pracuje. Jedná se v podstatě o funkce (resp. struktury) zastupující jednotlivé sítě, o funkce sloužící k trénování (učení) a vybavování sítě a o funkce starající se o vizuální zobrazení rozličných dat.

Prostředí Mathematica, které bylo k tomuto účelu použito, se ukázalo jako vhodný nástroj pro tvorbu neuronových sítí. Symbolický jazyk a funkce, které Mathematica používá, jsou srozumitelné a lehce zapamatovatelné. Proto byla práce v Mathematice příjemná a také efektivní. Výsledný produkt je distribuován ve formě tzv. toolboxu (balíčku), s jehož pomocí tak lze Mathematicu rozšířit o možnost práce s neuronovými sítěmi. Toto je standardní způsob tvorby nových toolboxů, který Mathematica podporuje. Vytvořený toolbox lze do Mathematicy snadno naimportovat, takže studenti by s jeho používáním neměli mít žádné problémy. Ke snazšímu pochopení slouží také dokumentace k toolboxu, která obsahuje stručný úvod do problematiky neuronových sítí, představení jednotlivých vypracovaných sítí (funkcí) a také praktické ukázky jejich použití.

ZÁVĚR V ANGLIČTINĚ

The main goal of this paper work was to create the utility appropriate for work with artificial neural networks in the issue of Methods of artificial intelligence which is taught at the Faculty of Applied Informatics in the Thomas Bata University in Zlín. With agreement of the supervisor there were specified these neural networks – the simple Perceptron, feed-forward network with Back-Propagation teaching algorithm and Hopfield network. The utility has been specified to be evolved in the environment Mathematica as so called toolbox (package).

All of defined networks were programmed successfully. The simple Perceptron network was created as a linear classifier of elements filed in two classes. Feed-forward network is intended towards function (or data) approximation problems. Finally Hopfield network demonstrates its auto-association ability in the identification of damaged characters. The program functionality of all the networks was designed similarly. Every network contains its own functions, which are appointed to work with the certain net. There are functions (or more precisely structures), which represent networks themselves, other functions are suitable to train or recollect data from neural networks and also graphic visualization functions.

The environment Mathematica, which was used to development of the utility, transpired as an appropriate tool for evolving artificial neural networks. Mathematica uses symbolic program language that is understandable and easy-to-remember so the work in this environment was enjoyable and also effective. The final product is distributed as so called toolbox (or package) and through this toolbox Mathematica can be easily enhanced. This is standard way to develop new toolboxes, which are supported by Mathematica. The mentioned toolbox is easily importable to Mathematica so students should not have problems during work with toolbox. There is also a documentation source that contains short introducing of neural networks, description of functions and some examples of practical use. This helps students to understand this topic.

SEZNAM POUŽITÉ LITERATURY

[1] JIRSÍK, Václav, HRÁČEK, Petr. *Neuronové sítě, expertní systémy a rozpoznávání řeči*. Brno : VUT Brno, [2002]. 106 s. Elektronický text.

[2] ZELINKA, Ivan. *Umělá inteligence I*. Zlín : VUT Brno, 1998. 126 s. Elektronický text.

[3] KUBA, Martin. *Neuronové sítě*. [s.l.] : [s.n.], 1995. Dostupný z WWW: <http://tf.czu.cz/~votruba/WAN/Neuronove%20site.pdf>. s. 28.

[4] ŠÍMA, Jiří, NERUDA, Roman. *Teoretické otázky neuronových sítí*. 1. vyd. Praha : MATFYZPRESS, 1996. 390 s. Dostupný z WWW:

<http://www2.cs.cas.cz/~sima/kniha.pdf>.

[5] BLAŽEK, Petr. *Hraní hry Go počítačem*. Brno, 2008. 17 s. Ústav inteligentních systémů FIT VUT v Brně. Vedoucí semestrální práce Ing. Bohuslav Křena, Ph.D. Dostupný z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=3757>.

[6] VOLNÁ, Eva. *Neuronové sítě 1*. 2. vyd. Ostrava : Ostravská univerzita v Ostravě, 2008. 86 s. Elektronický text. Dostupný z WWW:

<http://albert.osu.cz/oukip/volna/materialy/NEURONOVE_SITE_1/XNES1.pdf>.

[7] ŠNOREK, Miroslav. *Neuronové sítě a neuropočítače*. Praha : Vydavatelství ČVUT, 2004. 156 s. ISBN 80-01-02549-7.

[8] POSPÍŠIL, Jan. *Porovnání modelů pro dolování z dat*. Brno, 2007. 35 s. Ústav informačních systémů FIT VUT v Brně. Vedoucí bakalářské práce Ing. Roman Lukáš, Ph.D. Dostupný z WWW:

<http://www.fit.vutbr.cz/study/DP/rpfile.php?id=5841>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

UI	Umělá inteligence
NS	Neuronová síť
Θ	Práh neuronu
η	Koeficient učení algoritmu Back-Propagation
μ	Koeficient setrvačnosti (Momentum)

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH