Neuronové sítě v prostředí webMathematica

Neural Networks in the environment webMathematica

Bc. Leona Vysloužilová

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Leona VYSLOUŽILOVÁ**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Neuronové sítě v prostředí webMathematica**

Zásady pro vypracování:

1. **Seznamte se s prostředím webMathematica.**
2. **Seznamte se s různými typy neuronových sítí.**
3. **Vytvořte vhodné moduly různých typů neuronových sítí pro prostředí webMathematica.**
4. **Vytvořte vhodné demonstrační úlohy, které budou sloužit pro ukázky v předmětu Metody umělé inteligence.**
5. **Závěr.**

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

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

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

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

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

Ve Zlíně dne 19. února 2010

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Tato diplomová práce je zaměřena na problematiku umělé inteligence, konkrétně na umělé neuronové sítě. Cílem práce je vytvořit moduly neuronových sítí s podporou programu webMathematica od společnosti Wolfram Research pro využití v laboratořích předmětu Metody umělé inteligence vyučovaném na Fakultě aplikované informatiky Univerzity Tomáše Bati ve Zlíně.

V teoretické části práce je popsána historie neuronových sítí, jejich struktura a funkcionalita. Dále je přiblíženo vývojové prostředí od společnosti Wolfram Research Mathematica 7.0 for Students a program webMathematica 3, dále také skriptovací jazyk JavaScript, který je pro tvorbu modulů nezbytný. Praktická část zahrnuje konkrétní vypracované moduly a jejich popis. Konkrétně se jedná o přenosové funkce a neuronové sítě Perceptron, Adaline a vícevrstvou síť Feedforward.

Klíčová slova: neuron, umělá neuronová síť, Perceptron, Adaline, neuronová síť s dopředným šířením signálu, Backpropagation, Mathematica, webMathematica, JavaScript

ABSTRACT

This diploma thesis is focused on artificial intelligence specifically artificial neural networks. The aim of this document is to create modules of neural networks with support of the webMathematica program from the Wolfram Research company for use in the laboratories of Methods for artificial intelligence course taught at Faculty of Applied Informatics at Tomas Bata University in Zlin.

The theoretical part describes the history of neural networks their structure and functionality. Also illustrated is a development environment from Wolfram Research Mathematica 7.0 for Students the program webMathematica 3 and scripting language JavaScript which is essential for creating modules. The practical part includes specific developed modules and their description. Specifically it is about the transfer function and neural networks, Perceptron, Adaline and multilayer net Feedforward.

Keywords: neuron, artificial neural network, Perceptron, Adaline, neural network with forward spreading signal, Backpropagation, Mathematica, webMathematica, JavaScript

Ráda bych tímto chtěla poděkovat vedoucí diplomové práce Ing. Zuzaně Oplatkové, Ph.D., za odborné vedení, konzultace, podnětné rady, cenné připomínky, trpělivost, čas a trvalý zájem, který mi věnovala při tvorbě 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.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD

Inteligence je vlastností některých živých organismů dodávající jim v přírodě mimořádné postavení. Tato vlastnost vznikla a vyvíjela se v průběhu dlouhého vývoje. Některým živým organismům již dnes umožňuje efektivně reagovat na složité projevy prostředí a aktivně je využívat k dosažení vlastních cílů [1].

S rozvojem techniky si lidé souběžně kladou otázku, zda u uměle vytvořených systémů lze dosahovat reakcí a chování, které by se u živých organismů dalo nazvat inteligentní. Takovéto otázky si lidstvo kladlo již dávno a snažilo se jimi zabývat v rámci tehdejší filozofie a později i psychologie. Například nad otázkou, zda stroje mohou myslet, se zamýšleli již v 17. století významní filozofové jako Pascal, Descartes či o století později La Metrie. Jejich závěry samozřejmě neposkytovaly žádný návod, jak u stroje myšlení dosáhnout, měly pouze filozofickou podobu a byly zaměřeny na zodpovězení otázek, zda vůbec stroje myslet mohou [1].

Teprve ve 30. letech minulého století některé významné výsledky v matematické logice a teorii algoritmů, spojené se jménem německého matematika Gödela, a zejména pak prudký rozmach výpočetní techniky po druhé světové válce, se staly skutečnými katalyzátory úsilí napodobit intelektuální schopnosti nejvyspělejšího živého tvora. Postupně byly a jsou navrhovány a experimentálně ověřovány metody, postupy a algoritmy umožňující napodobovat určité hledisko inteligentního chování. Jsou k tomu využívány nejen techniky vycházející z detailní analýzy činnosti živých organismů na úrovni biologické, kde lze zařadit neuronové sítě, genetické algoritmy, atd., tak i techniky vycházející z matematické abstrakce mentálních procesů lidského mozku na úrovni psychologické a kognitivní, např. kvalitativní modelování, metody učení založené na modelech, atd.

Veškeré postupy a algoritmy vedoucí ve svém důsledku k určitému napodobení projevů inteligentního chování člověka, jsou předmětem zkoumání vědní disciplíny umělé inteligence [1].

I. TEORETICKÁ ČÁST

1 NEURONOVÉ SÍTĚ

Umělá neuronová síť (dále již jen neuronová síť) je obecně struktura pro distribuované paralelní zpracování dat složená obvykle z velmi vysokého počtu vzájemně propojených výkonných prvků. Neuronové sítě jsou inspirovány architekturou lidského mozku, proto jsou schopny učit se a analyzovat rozsáhlé a komplexní množiny dat, které lineární algoritmy jen těžko zvládnou [2].

1.1 Historie

Již déle než dvacet let se neuronové sítě těší mimořádné úspěšnosti, avšak historie vzniku neuronových sítí spadá do první poloviny 20. století, kdy byla Američanem W. S. McCullochem publikována první práce o neuronech a jejich modelech. Ve 40. letech minulého století se svým studentem W. Pittsem vypracoval model neuronu, který se v podstatě používá dodnes. V roce 1958 F. Rosenblatt, na základě jejich výsledků, vytvořil první funkční perceptronovou síť. Ta ovšem měla jeden velký nedostatek, nebyla schopna řešit lineárně neseparabilní problémy. Na tuto slabost upozorňovala kniha „Perceptron“ (1969), kterou publikoval M. Minsky a S. Papert. Díky této knize opadl zájem o neuronové sítě na dlouhou dobu.

Až v polovině 80. let došlo k renesanci neuronových sítí. V této době vznikla práce pojednávající o vícevrstvých neuronových sítích, které řeší i problémy, jenž nejsou lineárně separabilní. Postupně pak vznikaly i další typy sítí jako např. Hopfieldova síť, Kohonenova síť, Grossbergova ART síť [3].

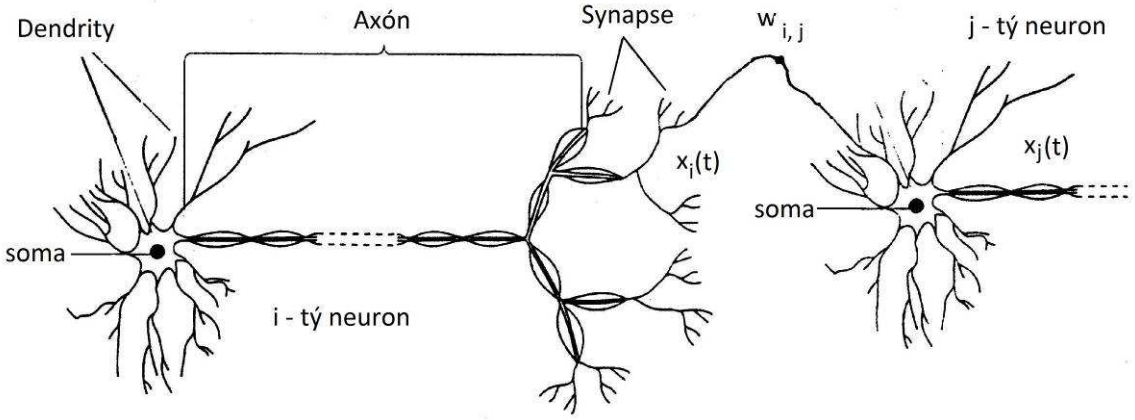
1.2 Základní pojmy

Původním cílem výzkumu neuronových sítí byla snaha pochopit, jakým způsobem myslíme a jak funguje lidský mozek. Získané poznatky umožnily vytvořit zjednodušené matematické modely, které se dají využít pro neurovýpočty při řešení praktických úloh z umělé inteligence. K tomuto zdroji inspirací je užitečné se vracet nejen pro nové inspirace, ale také je jej možné využít při popisu vlastností matematického modelu [4], [5].

1.2.1 Neurofyziologický neuron

Neurony jsou základními stavebními prvky nervové soustavy, převážně mozku. Jsou to živé buňky zaměřené na sbírání, uchovávání, zpracování a přenos informací. Mozková kůra člověka je tvořena přibližně 13 až 15 miliardami neuronů. Každý z nich může být

spojen s 5000 dalšími neurony. Existuje celá řada různých druhů neuronů, avšak ve všech případech se neuron skládá z těla – somy, do kterého přicházejí informace po vstupních větvích – dendritech, kterých je kolem deseti tisíc, a z kterého informace vycházejí po jediném výstupu – axonu, který je však na svém konci bohatě rozvětven. Tyto větve se nazývají terminály a jsou zakončeny blánou, která se stýká s výběžky dendritů jiných neuronů.



Obr. 1.1 Schéma neurofyziologického neuronu [6]

K přenosu informace slouží unikátní mezineuronové rozhraní, tzv. synapse. Nositelem všech význačných informací je tzv. míra synaptické propustnosti.

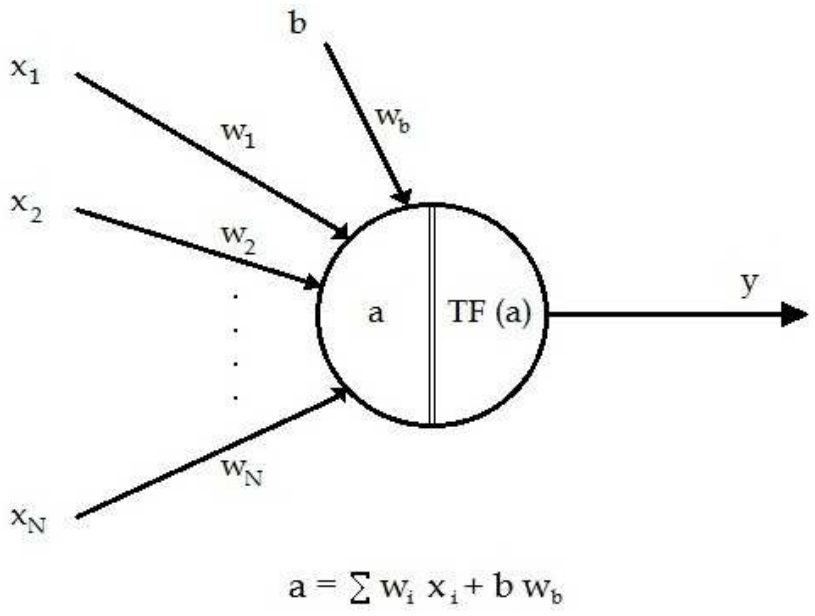
Soma i axon jsou obaleny membránou, která má schopnost za jistých okolností generovat elektrické impulsy, čímž je umožněno šíření informace. Tyto impulsy jsou z axonu přenášeny na dendrity jiných neuronů synaptickými branami, které svojí propustností určují intenzitu podráždění dalších neuronů. Takto podrážděné neurony při dosažení určité hraniční meze, tzv. prahu, samy generují impuls a zajišťují tak šíření příslušné informace. Synaptická propustnost se mění po každém průchodu signálu, což je předpokladem paměťové schopnosti neuronů.

Inteligentní a výkonné chování neuronových sítí je z velké části dáno právě dokonalostí vzájemného propojení. Propojení neuronů prodělává během života organismu svůj vývoj. V průběhu učení se vytváří nové paměťové stopy nebo se při zapomínání synaptické spoje přerušují. Neurony v neuronových sítích jsou uspořádány tak, že výpadek jednoho neuronu nemůže ohrozit funkci celku [4], [5].

1.2.2 Formální neuron

Základem matematického modelu neuronové sítě je formální neuron získaný přeformulováním zjednodušené funkce neurofyziologického neuronu do matematické řeči.

Formální neuron (dále jen neuron) má obecně *n* reálných vstupů $x_1, \ldots, x_n$, které modelují dendrity. Vstupy jsou ohodnoceny obecně reálnými synaptickými váhami $w_1, \ldots, w_n$, které určují jejich propustnost. Synaptické váhy, ve shodě s neurofyziologickou motivací, mohou být záporné. Tím je vyjádřen jejich inhibiční charakter.



Obr. 1.2 Schéma formálního neuronu

Vážená suma vstupních hodnot představuje vnitřní potenciál neuronu:

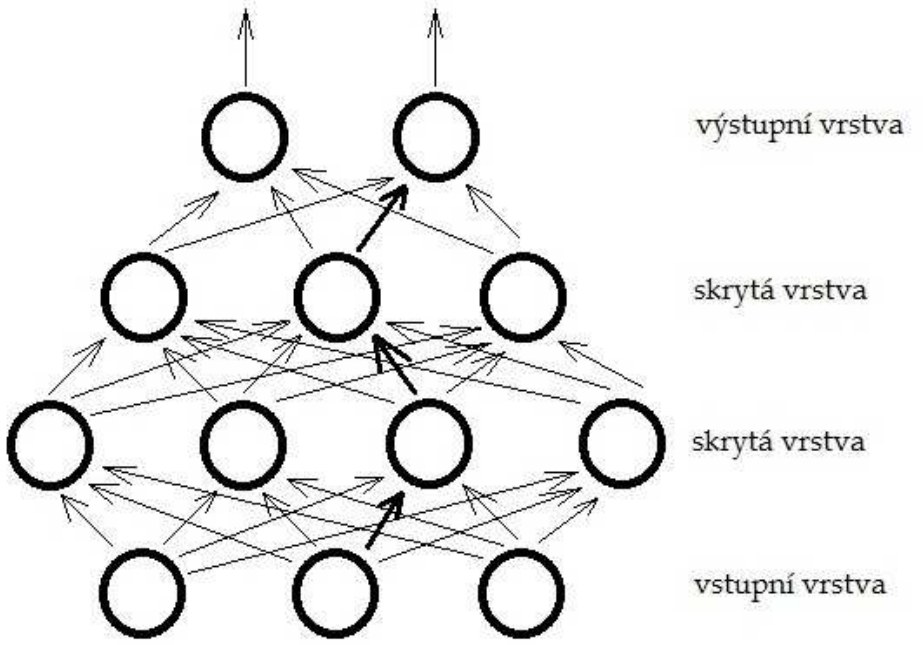
$$a = \sum w_i x_i + b w_b \tag{1}$$

Hodnota vnitřního potenciálu po dosažení tzv. prahové hodnoty *b* indukuje výstup neuronu *y*, který modeluje elektrický impuls axonu. Nelineární nárůst výstupní hodnoty *y* při dosažení prahové hodnoty *b* je dán tzv. aktivační, neboli přenosovou funkcí *TF(a)*. Formální úpravou lze docílit, že funkce *TF(a)* bude mít nulový práh a vlastní práh neuronu se záporným znaménkem budeme chápat jako váhu, tzv. bias, dalšího formálního vstupu s konstantní jednotkovou hodnotou [4], [7].

1.2.3 Neuronová síť a její topologie

Neuronová síť vzniká propojením určitých vstupů a výstupů neuronů tak, že výstup jednoho neuronu je vstupem obecně libovolnému počtu neuronů v další vrstvě. Vrstvy jsou oblasti s podobnou funkcionalitou, přičemž každá vrstva se skládá z „libovolného“ počtu neuronů. Počet neuronů je limitován technickými podmínkami.

Neuronové sítě se v průběhu učení mění, jednotlivé váhy se upravují a na základě těchto úprav se mění stav jednotlivých neuronů [3], [4], [7].



Obr. 1.3 Příklad struktury vícevrstvé neuronové sítě

Topologie sítě je definována typem propojení jednotlivých neuronů, vrstev a také počtem neuronů, vstupů a výstupů. Lze do ní zahrnout i parametry, jako typ přenosové funkce ve vrstvách, parametr učení, atd.

V topologii sítě obvykle platí, že každý neuron je spojen s každým neuronem ve vyšší vrstvě. Existují i jednovrstvé sítě, jakou je např. Hopfieldova síť, ve které je spojen každý neuron se všemi ostatními, či Kohonenova síť. U vícevrstvých sítí je první vrstva vždy větvící, tzn. neurony ve vstupní vrstvě pouze distribují vstupní hodnoty do další vrstvy. Vrstvy následující po vrstvě vstupní jsou nazývány skryté vrstvy. Může jich být hned několik. Výstupní vrstvou bývá označována poslední vrstva sítě, která na svém výstupu v podstatě předkládá výstup celé sítě.

Až ve druhé polovině dvacátého století byl u vícevrstvých sítí „vyřešen“ problém vhodného počtu vrstev. Vlastní funkce neuronové sítě je v podstatě transformační funkce přiřazující jistému vstupnímu obrazu obraz výstupní. Matematický důkaz 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 [3], [4], [6].

1.2.4 Přenosová funkce neuronu

Přenosová funkce neuronu je funkce transformující v intervalech 0 až 1 a -1 až +1 vstupní signál na signál výstupní. Tato funkce musí být monotónní, tzn. přiřazení odezev výstupu na vstup je jednoznačné, a může být skoková či spojitá. Jakou přenosovou funkci zvolíme je důležité pro správný chod neuronu a neuronových sítí.

Přenosová funkce udává odezvu výstupu na vstupní podnět. U různých druhů funkcí obecně platí, že jejich hodnota má být v intervalu -1 až +1 a že mají být spojité, např. sigmoida, hyperbolický tangens, nebo s nespojitostí prvního druhu, např. binární funkce 0-1).

Volba funkce závisí na problému, který chceme řešit. Např. chceme-li klasifikovat, zda je výrobek dobrý či špatný, postačí nám binární funkce. Použijeme-li funkci spojitou, pak musíme rozhodnout, jaká její hodnota (0.7, 0.8,…) znamená dobrý a jaká špatný.

Mezi nejčastěji používané přenosové funkce řadíme perceptron, binární, logistická sinoida a hyperbolický tangens [3].

Funkce perceptron

Je to funkce lineární, která byla použita v Rosenblattově první neuronové síti. Díky své lineárnosti byla schopna řešit pouze lineárně separabilní problémy. Její zápis je

$$f\ (a) = \lambda \times a \ \forall\ a \geq 0\ jinak\ a = 0 \tag{2}$$

Funkce binární

Ze zápisu je vidět, že jde o dvouhodnotovou funkci, která může mít hodnotu 0 nebo 1. Nazývá se též skoková funkce. Její zápis je

$$f(a) = 1\ \forall\ a \geq 0\ jinak\ a = 0 \tag{3}$$

Funkce logistická sigmoida

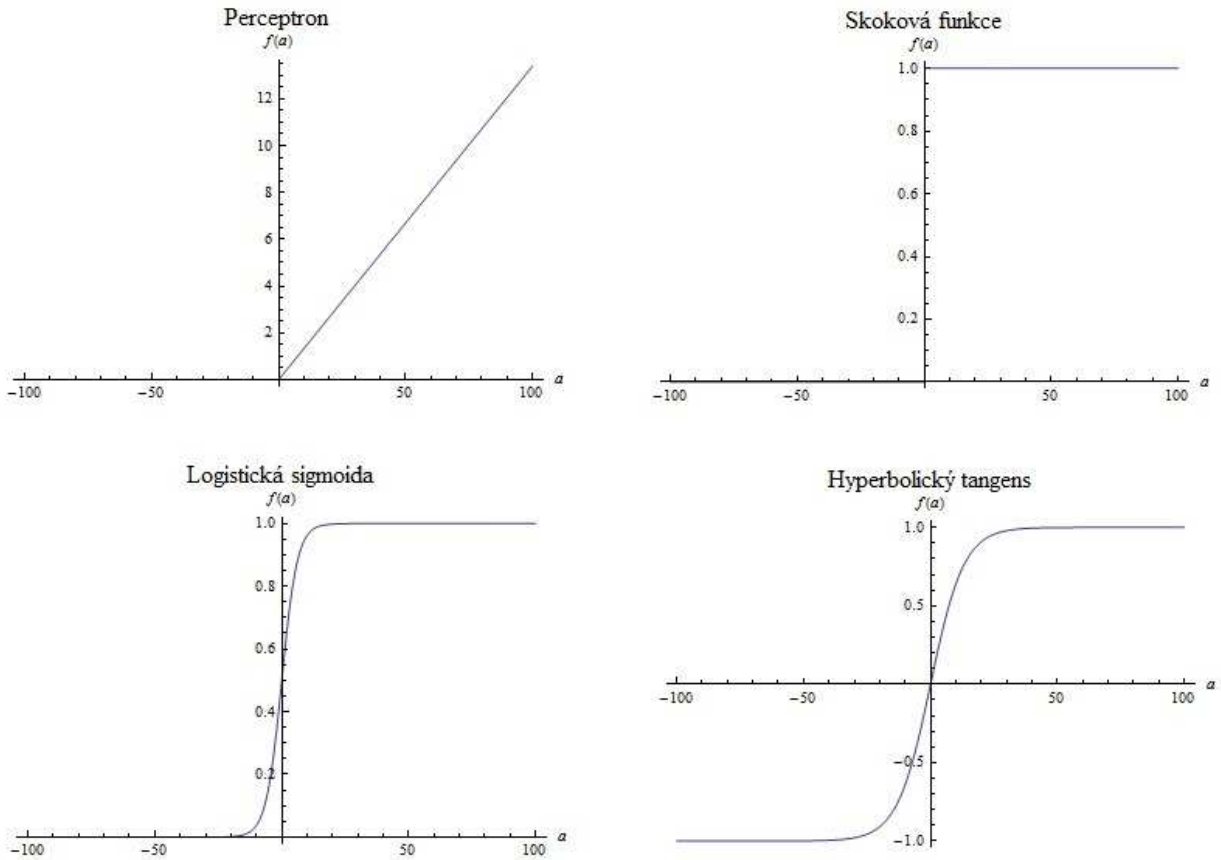
Jedna z používanějších funkcí, která byla odvozena jako aproximace přenosové funkce biologického neuronu. Její zápis je

$$f(a) = \frac{1}{1 + e^{-\lambda \times a}} \tag{4}$$

Funkce hyperbolický tangens

Je obdobou logistické funkce, s rozdílem, že může nabývat hodnot -1 až +1, což znamená, že poskytuje mimo jiné i větší lineární úsek okolo počátku. Její zápis je

$$f(a) = \frac{e^{\lambda \times a} - e^{-\lambda \times a}}{e^{\lambda \times a} + e^{-\lambda \times a}} \tag{5}$$



Obr. 1.4 Přenosové funkce neuronu

1.2.5 Učení sítě

Pro funkčnost sítě je jedním ze základních předpokladů její naučení, tzv. adaptace na řešený problém. Učící proces se skládá ze dvou fází - adaptační a aktivační, během kterých se nastavují váhy sítě. Tyto fáze ke své činnosti potřebují tzv. trénovací množinu, což je skupina vektorů obsahující informace o daném problému potřebné pro učení. Pokud požadované chování sítě modeluje učitel, který pro vzorové vstupy vyhodnotí výstupy, tzv. síť s učitelem, pak jsou to dvojice vektorů vstup-výstup. Jiným typem adaptace je tzv. samoorganizace. V takovémto případě trénovací množina obsahuje pouze vstupní vektory, učitel není k dispozici, a proto se tomuto způsobu adaptace říká učení bez učitele.

Používáme-li jen aktivační fázi, mluvíme o vybavování. Tato fáze se samostatně používá jen tehdy, když je síť naučena. Cyklické střídání obou fází je učení.

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

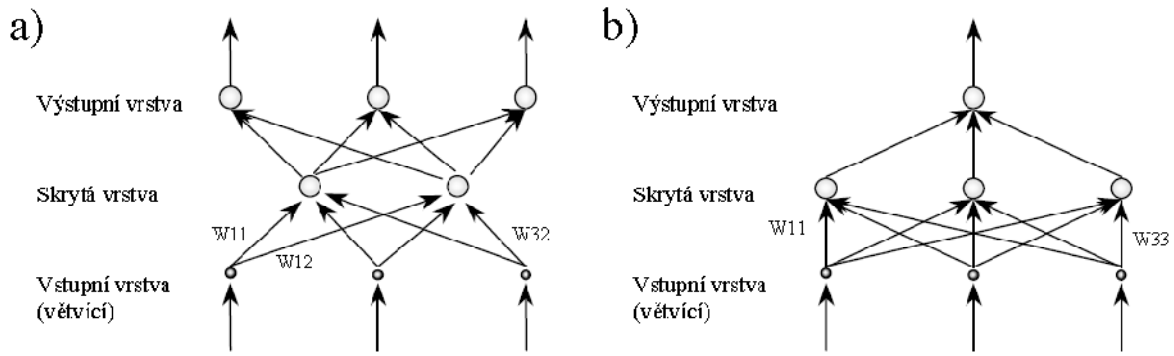
Adaptační fáze je proces, při kterém je minimalizována lokální chyba sítě tak, že se přepočítávají váhy jednotlivých spojů směrem z výstupu na vstup za účelem co největší podobnosti výstupní odezvy s originálním vektorem. Následuje opakování aktivační fáze. Další získaný rozdíl, lokální odchylka, se přičte k předchozímu atd. Pokud se tímto postupem projde celá trénovací množina, je hotová jedna epocha. Sumě odchylek za jednu epochu se říká globální odchylka. Je-li globální odchylka menší než námi požadovaná chyba, pak proces učení končí.

Proces učení není nic jiného, než přelévání informací ze vstupu na výstup a naopak. Zda se síť naučí správným odezvám na dané podněty, závisí např. na množství vektorů a jejich velikosti, topologii sítě, odlišnosti charakteristických vlastností jednotlivých tříd, přípravě trénovací množiny, atd. [3], [7], [8].

1.2.6 Dělení neuronových sítí

Neuronové sítě se dělí podle několika kritérií, jak podle počtu vrstev na jednovrstvé či vícevrstvé, tak podle typu algoritmu učení - s učitelem nebo bez učitele, nebo také podle stylu učení na sítě s učením deterministickým nebo stochastickým.

Dělení podle počtu vrstev znamená, že rozlišujeme, z kolika vrstev je daná síť složena. Pro topologii sítí obvykle platí pravidlo, že každý neuron bývá spojen s každým neuronem ve vrstvě vyšší. Každý spoj je ohodnocen vahami nabývajících různých hodnot. Ty vyjadřují význam daného spoje pro daný neuron.



Obr. 1.5 Různé topologie vícevrstvých sítí [3]

Učení s učitelem znamená, že se síť pokouší přizpůsobit svou odezvu na vstupní informace, tzn. upravit momentální výstup, tak aby se co nejvíce podobal požadovanému originálu. Při učení bez učitele síť vychází z informací obsažených ve vstupních vektorech. Styl učení rozlišuje, jak se přistupuje k zjištění vah sítě. Jedná-li se o zjištění výpočtem, mluvíme o učení deterministickém. Jestliže jsou váhy získávány pomocí generátoru náhodných čísel, jde o stochastický styl učení. Tento způsob získání vah sítě se používá obvykle jen při startu sítě [3].

1.2.7 Využití neuronových sítí

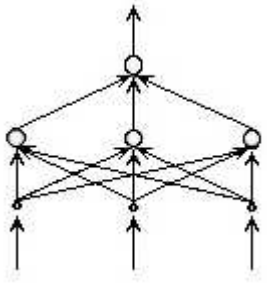
Využití neuronových sítí je opravdu široké a nabývá čím dál více na významu. Můžeme vyjmenovat následující oblasti využití neuronových sítí pro:

- 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

1.3 Neuronová síť Perceptron

Perceptron je síť s jednou pevnou vstupní vrstvou a výstupní vrstvou s měnícími se vahami. Její neurony jsou schopny funkční transformace ze vstupu na výstup díky měnitelnosti prahů a vah ve výstupní vrstvě [3].



Obr. 1.6 Schéma sítě Perceptron [3]

Autorem této nejjednodušší neuronové sítě je Frank Rosenblatt, který poprvé publikoval algoritmus vhodný k nastavení parametrů perceptronu 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.

Typickým perceptronem je neuronová síť s jedním pracovním neuronem spojeným s n vstupy (x_1, x_2,…, x_n). Každému spojení je přiřazena váhová hodnota (w_1, w_2,…, w_n). Váhové hodnoty neuronů jsou adaptovány dle adaptačního pravidla perceptronu, tzn. tak, aby diference mezi skutečným a požadovaným výstupem byla co nejmenší.
Mimo klasických vstupů mají neurony v učící vrstvě zpravidla jeden vstup s konstantně nastavenou hodnotou na 1 navíc. Je zde přidán z důvodů nastavování prahu neuronu. Takovéto rozšíření jednotkového vstupu znamená rozšíření každého vstupního vektoru o jeden vstup s hodnotou 1 a bývá nazýván rozšířený vektor.

Přenášený signál je buď binární, tzn., má hodnotu 0 nebo 1, nebo bipolární, tzn., má hodnotu -1, 0 nebo 1. Perceptron s jedním výkonným prvkem umožní nejvýše klasifikaci do dvou tříd. Pokud bychom zvětšili počet výkonných prvků i počet vrstev v perceptronu, byla by možná klasifikace do více tříd. Tyto třídy již nemusí být lineárně separabilní, ale separabilní být musí [3], [7].

K výpočtu nových vah můžeme použít gradientní metodu či metodu fixních přírůstků, jejíž koeficient může být pevný či modifikován absolutní popřípadě zlomkovou korekcí. V případě použití pravidla fixních přírůstků, jehož koeficient je pevný, se nové váhy vypočítají podle vzorce

$$w_{(n+1)} = w_n + c \times x \tag{6}$$

kde *w* jsou váhy, *c* je učící koeficient, *x* je vstup a *n* je krok. Tento vztah platí ovšem jen když

$$w_n . x \leq 0 \tag{7}$$

kde *x* je vstup, jinak se nové váhy nepřepočítávají a platí vztah

$$w_{(n+1)} = w_n \tag{8}$$

1.4 Neuronová síť Adaline

ADALINE neboli adaptive linear neuron je jednovrstvá neuronová síť. Byla vyvinuta, krátce po objevu perceptronu, profesorem Bernardem Widrowem a jeho postgraduálním studentem Tedem Hoffem na Stanford University v roce 1960. Je založen na McCulloch-Pittsově neuronu.

Tato síť byla vybavena novým výkonným učícím pravidlem, které dodnes nebylo změněno. Widrow se svým studentem jeho funkčnost demonstroval na mnoha jednoduchých typových příkladech.

Adaline pro své vstupy obvykle používá bipolární aktivaci (1 nebo -1), výstupní hodnota je nejčastěji také bipolární. Adaline má bias chovající se jako regulovatelná váha přiřazena spojení, které vychází z neuronu, jehož aktivace je vždy 1.

Při učení Adaline inicializujeme váhy malými náhodnými hodnotami. Koeficient učení α obvykle volíme jako

$$0{,}1 < n \times \alpha < 1 \tag{9}$$

kde n je počet vstupů. Protože pokud bychom dosadili za α příliš velkou hodnotu, adaptační algoritmus by nekonvergoval, pokud bychom dosadili naopak příliš malou hodnotu, proces učení by byl extrémně pomalý [7].

Rozdíl mezi Adaline a standardním perceptronem je v adaptační fázi. Ve standardním perceptronu, je čistě předána aktivační funkce a výstup funkce je použit pro úpravu vah [7].

Hodnota aktivační funkce se při učení sítě vypočítá podle vzorce

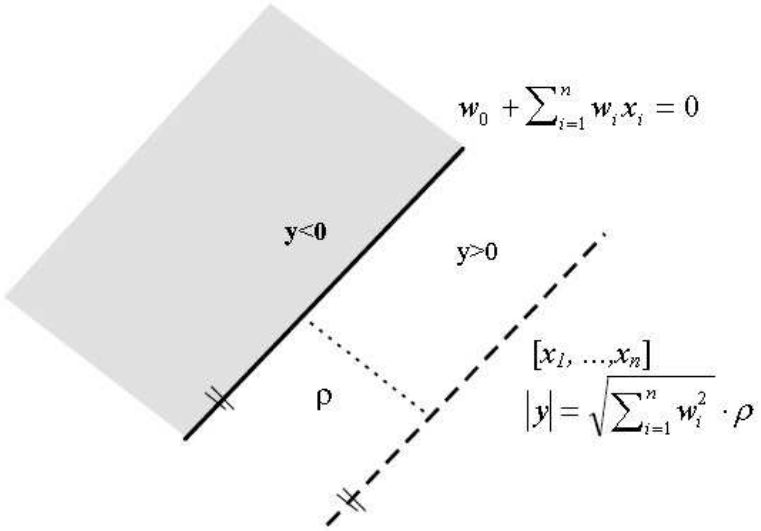
$$a = \sum w_i x_i + b w_b \tag{10}$$

kde a je výstup, x_i je vstup, w_i jsou váhy, b je práh a w_b je váha prahu. Nové váhy se přepočítávají podle vzorce

$$w_{(n+1)} = w_n + c \times (d - a) \times x \tag{11}$$

kde c je učící koeficient a d je předpokládaná třída.

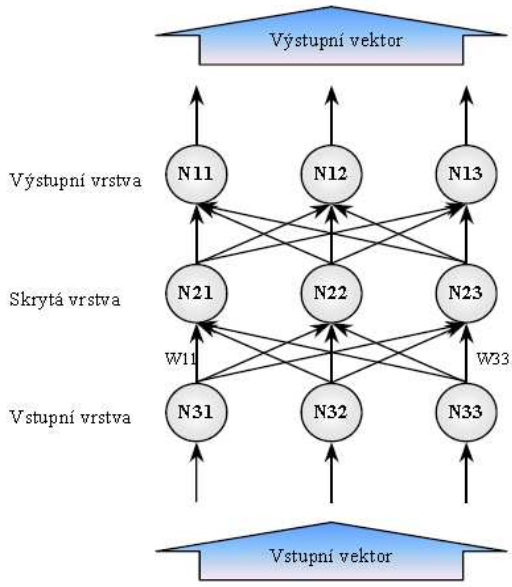
Existuje také rozšířené Adaline, tzv. Madaline [7].



Obr. 1.7 Geometrická interpretace funkce neuronu Adaline [7]

1.5 Vícevrstvá síť a algoritmus Backpropagation

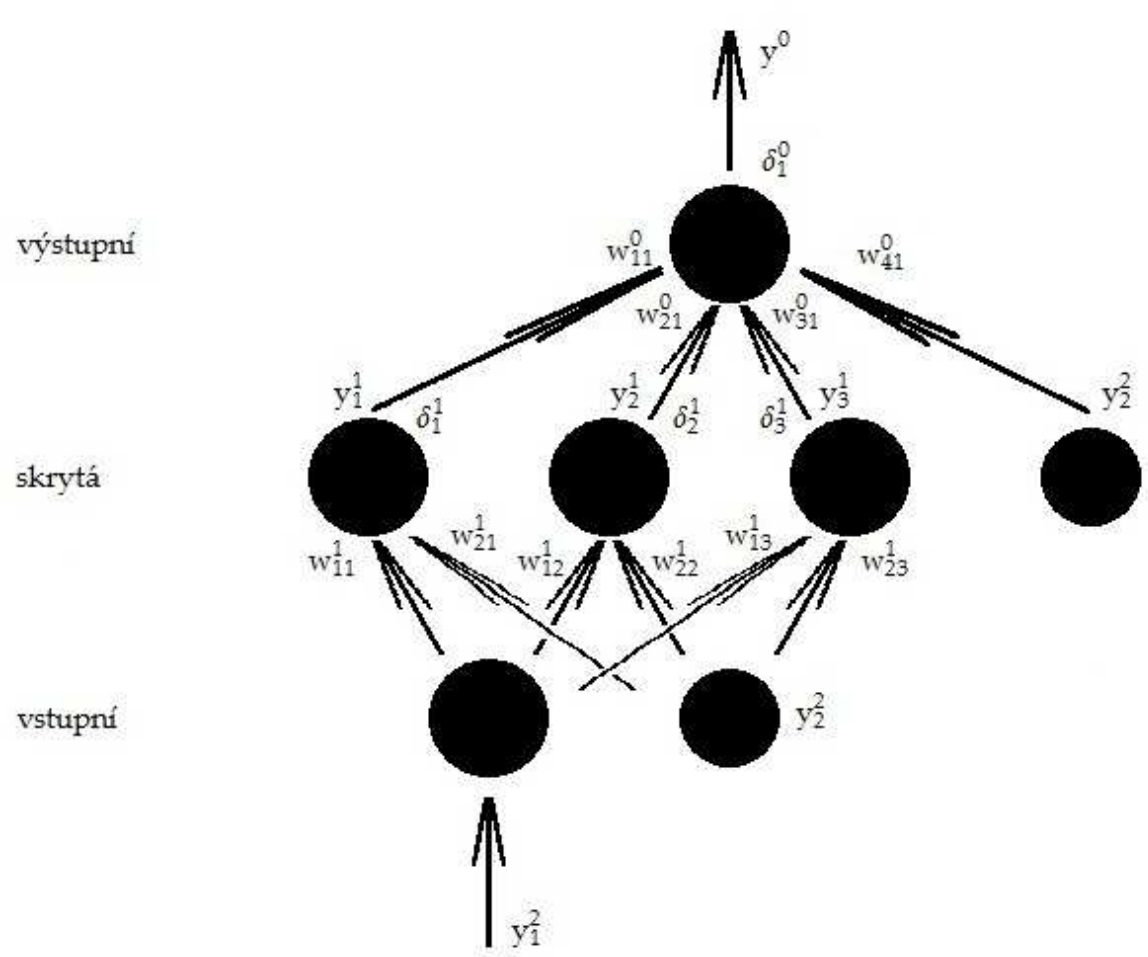
Vícevrstvá síť je síť složená z vrstev propojených mezi sebou směrem ze vstupu na výstup. Obvykle se jako učící algoritmus používá Backpropagation, jež při adaptační fázi přenastavuje váhy tak, aby se odezva sítě co nejvíce přiblížila požadované hodnotě [3].



Obr. 1.8 Ukázka klasické vícevrstvé sítě [3]

Feedforward je vícevrstvá síť s dopředným šířením signálu a zpětným šířením chyby, která je tvořena minimálně třemi vrstvami neuronů: vstupní, výstupní a alespoň jednou skrytou vrstvou. Mezi dvěma sousedními vrstvami se pak nachází úplné propojení neuronů, tedy každý neuron nižší vrstvy je spojen se všemi neurony vrstvy vyšší.

Učení se v neuronové síti realizuje nastavováním vah synapsí mezi neurony. U sítě s algoritmem Backpropagation, který je pro tuto síť vhodný, probíhá učení metodou učení s učitelem, kdy neuronová síť se učí srovnáním aktuální hodnoty výstupu neuronové sítě s žádanou hodnotou. Postupným nastavováním vah synapsí se algoritmus snaží dosáhnout minimálního rozdílu mezi žádanou hodnotou a hodnotou na výstupu neuronové sítě. Míru nepřesnosti mezi predikovanou hodnotou výstupu neuronové sítě a skutečnou hodnotou výstupu objektu vyjadřuje predikční chyba [3], [7].



Obr. 1.9 Topologie vícevrstvé neuronové sítě s dopředným šířením o jednom vstupu a prahu, 3 neuronech ve skryté vrstvě a prahu a výstupním neuronu

Backpropagation je základní algoritmus, pomocí kterého se příslušná vícevrstvá neuronová síť může učit. V literatuře někdy bývá síť využívající tento algoritmus chybně nazývána jako síť Backpropagation, avšak je to pouze algoritmus vytvořený pro učení vícevrstvých neuronových sítí s učitelem nastavující váhy jednotlivých spojů zpětným chodem tak, aby jejich velikosti byly z hlediska řešeného problému pokud možno optimální, tzn., hledá se globální minimum chybové funkce. Šíření vstupní informace probíhá v opačném směru než nastavení vah, tzn. ze vstupu na výstup. U tohoto algoritmu také rozeznáváme dvě fáze - aktivační a adaptační.

Při inicializaci sítě se váhy nastaví na vhodnou hodnotu v rozmezí 0,5 až -0,5, buď pomocí generátoru náhodných čísel, nebo existují i jiné metody pro prvopočáteční nastavení vah, např. simulované žíhání, genetické algoritmy.

Aktivační fáze je používána při učení a vybavování sítě. Je to aktivita, při které se vstupní informace zmodifikovaná momentální množinou vah a přenosovými funkcemi ve vlastních neuronech dostane na výstup. Vstupní vektor je ve vstupní vrstvě rozvětven a každá „větev“ je ohodnocena vahou (hodnota spoje * váha). V každém neuronu, do kterého vstupují takto ohodnocené spoje, se provede jejich součet. Takto získané číslo je použito jako argument přenosové funkce, jejíž výsledná hodnota je výstupem z neuronu, a slouží jako vstupní hodnota do dalších neuronů ve vyšší vrstvě. Pokud se jedná o vrstvu výstupní, pak výstupní hodnota z neuronů je vlastní výstupní vektor.

Při učení sítě o jednom vstupu a prahu, 3 neuronech ve skryté vrstvě a prahu a výstupním neuronu se v aktivační fázi počítá podle vzorců na Obr. 1. 9, kde *y* je výstup, *w* jsou váhy, y_2^2 je práh a w_{41}^0 je váha prahu.

$$y_1^1 = \frac{1}{1 + e^{-(y_1^2 * W_{11}^1 + y_2^2 * w_{21}^1)}}$$

$$y_2^1 = \frac{1}{1 + e^{-(y_1^2 * W_{12}^1 + y_2^2 * w_{22}^1)}}$$

$$y_3^1 = \frac{1}{1 + e^{-(y_1^2 * W_{13}^1 + y_2^2 * w_{23}^1)}}$$

$$y^0 = y_1^1 * w_{11}^0 + y_2^1 * w_{21}^0 + y_3^1 * w_{31}^0 + y_2^2 * w_{41}^0$$

Obr. 1.10 Vzorce pro aktivační fázi sítě o jednom vstupu a prahu, 3 neuronech ve skryté vrstvě a prahu a výstupním neuronu

V adaptační fázi (vlastní Backpropagation) je výstupní vektor, neboli odezva sítě na vstupní vektor, porovnána s požadovaným originálem a rozdíl mezi oběma vektory je použit pro výpočet nových vah tak, že se nejprve opraví váhy u spojů, které vstupují do výstupní (nejvyšší) vrstvy. Pak jsou opraveny váhy u nižší vrstvy, atd. Tato fáze je ukončena až se dosáhne vrstvy vstupní, poté se opakuje fáze aktivační.

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 [3], [7].

$$\delta_1^0 = \left(y^0 - d^0\right)$$

$$\Delta w_{11}^0(t) = \eta * \delta_1^0 * y_1^1 + \mu \Delta w_{11}^0(t-1)$$

$$\Delta w_{21}^0(t) = \eta * \delta_1^0 * y_2^1 + \mu \Delta w_{21}^0(t-1)$$

$$\Delta w_{31}^0(t) = \eta * \delta_1^0 * y_3^1 + \mu \Delta w_{31}^0(t-1)$$

$$\Delta w_{41}^0(t) = \eta * \delta_1^0 * y_4^1 + \mu \Delta w_{41}^0(t-1)$$

$$w_{11}^0(t+1) = w_{11}^0(t) + \Delta w_{11}^0(t)$$

$$w_{21}^0(t+1) = w_{21}^0(t) + \Delta w_{21}^0(t)$$

$$w_{31}^0(t+1) = w_{31}^0(t) + \Delta w_{31}^0(t)$$

$$w_{41}^0(t+1) = w_{41}^0(t) + \Delta w_{41}^0(t)$$

$$\delta_1^1 = y_1^1\left(1 - y_1^1\right) * w_{11}^0 * \delta_1^0$$

$$\delta_2^1 = y_2^1\left(1 - y_2^1\right) * w_{21}^0 * \delta_1^0$$

$$\delta_3^1 = y_3^1\left(1 - y_3^1\right) * w_{31}^0 * \delta_1^0$$

$$\Delta w_{11}^1(t) = \eta * \delta_1^1 * y_1^2 + \mu \Delta w_{11}^1(t-1)$$

$$\Delta w_{12}^1(t) = \eta * \delta_2^1 * y_1^2 + \mu \Delta w_{12}^1(t-1)$$

$$\Delta w_{13}^1(t) = \eta * \delta_3^1 * y_1^2 + \mu \Delta w_{13}^1(t-1)$$

$$\Delta w_{21}^1(t) = \eta * \delta_1^1 * y_2^2 + \mu \Delta w_{21}^1(t-1)$$

$$\Delta w_{22}^1(t) = \eta * \delta_2^1 * y_2^2 + \mu \Delta w_{22}^1(t-1)$$

$$\Delta w_{23}^1(t) = \eta * \delta_3^1 * y_2^2 + \mu \Delta w_{23}^1(t-1)$$

$$w_{11}^1(t+1) = w_{11}^1(t) + \Delta w_{11}^1(t)$$

$$w_{12}^1(t+1) = w_{12}^1(t) + \Delta w_{12}^1(t)$$

$$w_{13}^1(t+1) = w_{13}^1(t) + \Delta w_{13}^1(t)$$

$$w_{21}^1(t+1) = w_{21}^1(t) + \Delta w_{21}^1(t)$$

$$w_{22}^1(t+1) = w_{22}^1(t) + \Delta w_{22}^1(t)$$

$$w_{23}^1(t+1) = w_{23}^1(t) + \Delta w_{23}^1(t)$$

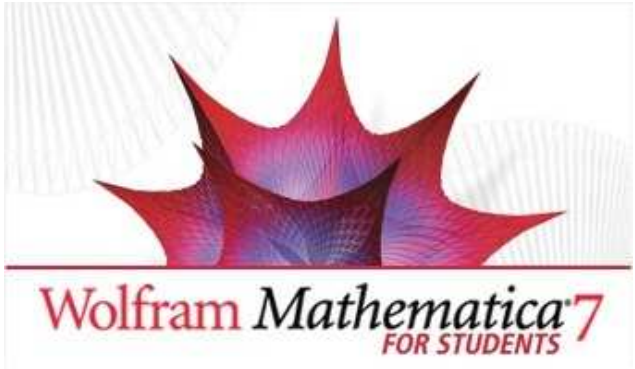
Obr. 1.11 Vzorce pro adaptační fázi sítě o jednom vstupu a prahu, 3 neuronech ve skryté vrstvě a prahu a výstupním neuronu

Pro síť o jednom vstupu a prahu, 3 neuronech ve skryté vrstvě a prahu a výstupním neuronu se v adaptační fázi používají vzorce na Obr. 1. 10, kde y je výstup, w jsou váhy, d je předpokládaná hodnota výstup, y_2^2 je práh a w_{41}^0 je váha prahu, η je učící koeficient, δ je pomocná proměnná pro přepočet vah, μ je momentum ovlivňující rychlost učení, w_{21}^1 , w_{22}^1, w_{23}^1 jsou váhy prahů vstupující do skryté vrstvy a Δ je přírůstek vah.

2 MATHEMATICA

Mathematica je nejmocnější výpočetní systém na světě. Poprvé byla vydána v roce 1988, což mělo hluboký vliv na způsob, jakým jsou počítače používány v technických a dalších oborech.

Software Mathematica paří do rodiny aplikací určených nejen pro matematické výpočty, jako i např. Matlab. Z důvodu uživatelsky přívětivého prostředí a symbolického, logicky snadno zapamatovatelného jazyka je Mathematica velice oblíbeným vývojovým prostředím. Díky rostoucí podpoře produktu a mnoha rozšířením je Mathematica také vhodná i pro práci s neuronovými sítěmi [9].



Obr. 2.1 Logo Mathematica 7.0 for Students

2.1 Popis

Software Mathematica je vyvíjen americkou společností Wolfram Research, kterou založil v roce 1987 anglický matematik Stephen Wolfram. Z počátku byla Mathematica jediným produktem, avšak se zvyšujícím se zájmem o tento software společnost začala vyvíjet rozšiřující, sofistikovanější nástroje. Dnes společnost nabízí kromě Mathematicy 7.0 programy pro webová rozhraní (webMathematica), rychlejší verze pro paralelní nasazení (gridMathematica) nebo nástroj pro spouštění aplikací vytvořených v Mathematice aniž by bylo nutné její pořízení (Mathematica Player).

Uživatel má při práci k dispozici obsáhlou a kvalitní podporu ve formě aplikace Documentation Center dostupnou i na webovém serveru společnosti Wolfram, velké množství návodů a ukázkových projektů.

Mathematica pracuje na modulárním principu, tzn., mimo práce s vlastními systémovými knihovnami je možné zakomponovat do programu rozšiřující balíčky, tzv. Add-Ons či toolboxy, které podporují různé specifické problematiky. Tyto balíčky vydává buď sama společnost jako doplňky nebo také běžní uživatelé pracující s Mathematicou.

Nejnovější verze Mathematicy samozřejmě podporuje nejrozšířenější operační systémy, jako např. Microsoft Windows, Linux, Apple's Mac OS X i Sun's Solaris i s podporou 64bitových systémů.

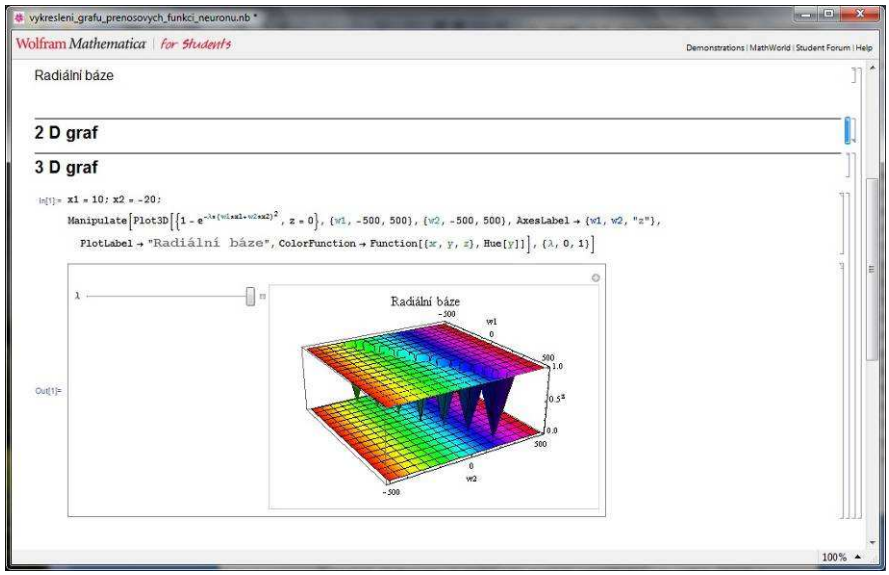
Mathematicu je možné využít v širokém spektru oborů, od problematik matematických či vědeckých až po využití v technice nebo v aplikacích v chemickém modelování.

Může vystupovat jako programovací jazyk vyšší úrovně, jazyk pro popis grafických objektů či jako systém pro psaní technických dokumentů, atd. [10].

2.2 Prostředí

Práce v prostředí Mathematica je intuitivní a jednoduchá. Pro styk s uživatelem prostředí používá tzv. front-end rozhraní, samotné výpočty zajišťuje na pozadí jádro, tzv. kernel. Běžnou práci v prostředí tvoříme v interaktivních dokumentech v tzv. noteboocích s příponou *.nb. Ty umožňují kombinovat vstupy a výstupy Mathematicy s textem, grafickými objekty a jinými formáty dat.

Při práci stačí do notebooku napsat požadovaný výpočet, tzn. vstup a pro jeho vyhodnocení stisknout kombinaci kláves Shift+Enter nebo numerický Enter. Kernel požadavek vyhodnotí a vrátí výsledek, tzv. výstup ve front-endu. Výstupy mohou být v různých podobách nejen v číselné, např. texty, 2D grafy, 3D modely s možností rotace, obrázky, animace atd.



Obr. 2.2 Ukázka práce v prostředí Mathematica 7.0 for Students

Jednotlivé položky notebooku jsou považovány za samostatné a nazývají se buňky. Zpřehledňují práci v programu, mohou mít hierarchickou strukturu a také se mohou

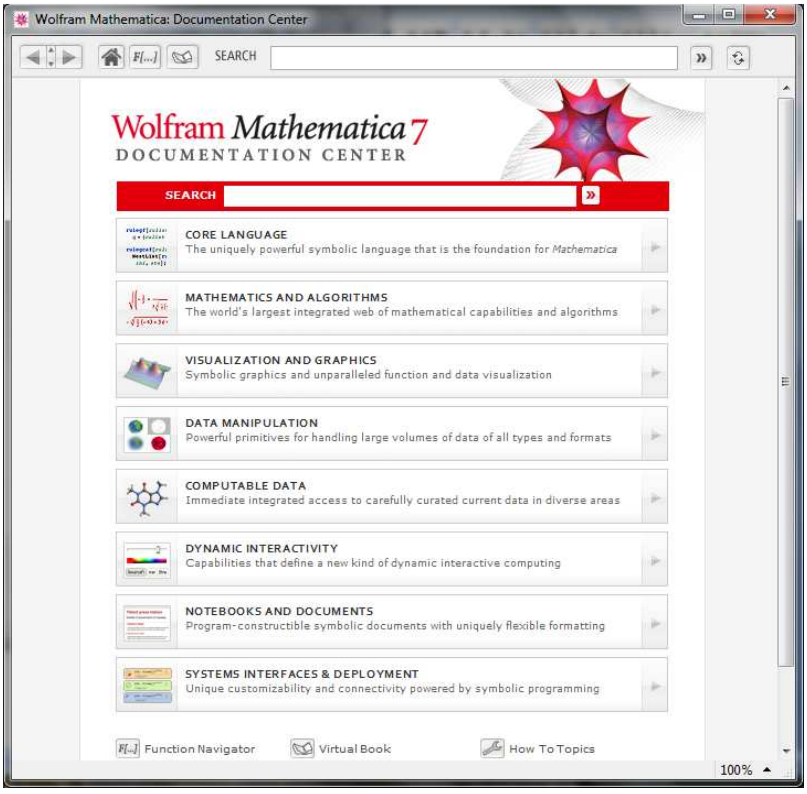
shromažďovat či sbalovat, tzn. skrýt požadovaný obsah pro jeho lepší přehlednost. Je zde i možnost stylování a formátování obsahu notebooku buď dle uživatele, nebo prostřednictvím předdefinovaných šablon.

Jazyk Mathematicy je symbolický, avšak všechny příkazy a symboly lze nahradit jejich textovou formou. Mathematica obsahuje přes 2500 vestavěných funkcí a příkazů, jejichž názvy jsou většinou voleny podle jejich účelu. Program je citlivý na velikost písmen v textu a názvosloví. Při výpočtech Mathematica pracuje na volitelnou přesnost výsledků.

Argumenty funkce se zadávají do hranatých závorek. Pomocí parametrů funkce lze upravit její chování dle vlastních potřeb. Je umožněno také vnořování funkcí.

Při výběru funkce z vestavěných funkcí Mathematicy je k dispozici rozsáhlá interaktivní nápověda s definicí, popisem a ukázkami použití dané funkce.

Stiskem klávesy F1 se vyvolává nápověda, která obsahuje mimo jiné řadu podrobných tutoriálů a návodů, jak s programem Mathematica začít pracovat.



Obr. 2.3 Ukázka nápovědy v prostředí Mathematica 7.0 for Students

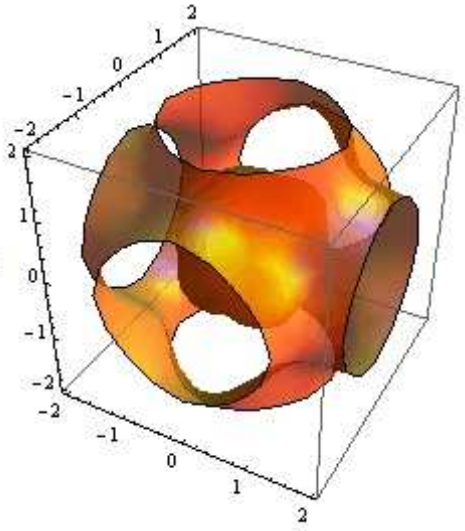
V prostředí Mathematica je možné také pracovat s proměnnými, ve vyšších verzích i s dynamickými, a podporuje také příkazy standardního sekvenčního programování. Díky tomu lze definovat a vytvářet vlastní funkce či programovat vlastní algoritmy.

Mathematica v nejnovějších verzích nabízí i dynamickou interaktivitu, tzn., je možné vytvářet například modifikovatelné panely okamžitě reagující na změnu parametrů.

U tohoto programu je důležitou předností rozsáhlá podpora importu a exportu nejznámějších formátů dat a také, že prostřednictvím protokolu MathLink komunikuje kernel přímo i s jinými rozhraními jako např. Java, C++, MySQL, webové servery [10].

```
In[1]:= ContourPlot3D[x^4 + y^4 + z^4 - (x^2 + y^2 + z^2)^2 + 3 (x^2 + y^2 + z^2) == 3,
         {x, -2, 2}, {y, -2, 2}, {z, -2, 2}, Mesh → None,
         ContourStyle → Directive[Orange, Opacity[0.8], Specularity[White, 30]]]
```

Out[1]=



Obr. 2.4 Ukázka funkce ContourPlot 3D v prostředí Mathematica 7.0 for Students

2.3 Toolboxy

Funkce Mathematicy lze obohatit díky rozšiřujícím balíčkům, tzv. toolboxům, které je možné do programu importovat. Tyto balíčky vytvořené v Mathematice se jednoduše nakopírují do adresářové struktury programu a před jejich použitím se připojí pomocí jediného příkazu. Poté jsou k dispozici všechny funkce a objekty z tohoto balíčku.

Definice balíčků jsou obsaženy v souborech s příponou *.m, kvůli odlišení od klasických notebooků. Balíček může mít nadefinovanou vlastní nápovědu či dokumentaci, která se po importu automaticky zobrazí v nápovědě Mathematicy.

Balíčky je možné vytvořit buď manuálně, tzn., tvůrce balíčku sepíše obsah, popř. nápovědu ručně a poté si sám otestuje a tím zajistí jeho funkčnost, nebo je možné použít vývojové prostředí Wolfram Workbench, které společnost Wolfram také nabízí a které je k tomuto účelu přímo určeno [10].

3 WEBMATHEMATICA

WebMathematica přidává interaktivní výpočty a vizualizace na webové stránky integrací Mathematicy pomocí nejnovějších technologií webového serveru. Bezproblémově pracuje s klientskými technologiemi jako je JavaScript [11].

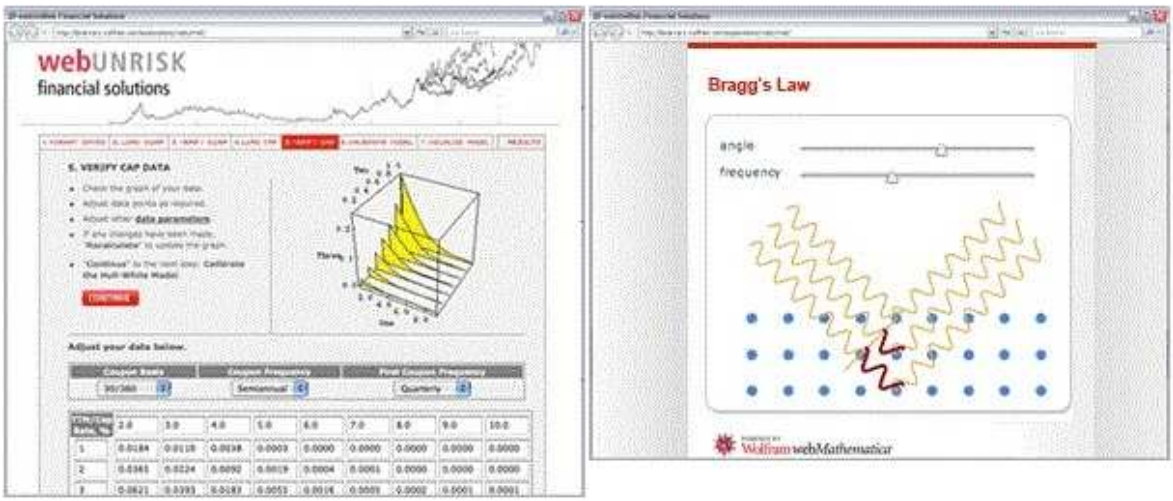
Obr. 3.1 Logo Wolfram webMathematica 3

3.1 Popis

WebMathematica dynamicky zobrazuje a generuje grafiku a vizualizace. Lze v ní interaktivně upravovat výpočty. Umožňuje používat prvky známé pro webové rozhraní, jako jsou tlačítka, drop-down listy či textová pole pro kontrolu výpočtů. Klientské štítky ve webMathematice integrují standardní webové Java technologie - Java Servlet a Java Server Pages. Pomocí posuvníků a dalších dynamických možností pro nastavení parametrů lze vytvářet nové interaktivní výsledky.

Výstupy můžeme zobrazovat na webové stránce nebo vytvářet přehledy ke stažení v různých formátech, např. PDF, Mathematica notebook (*.nb), atd.

Použitím šablon při tvorbě můžeme dosáhnout konzistentní a profesionální vzhled [11].



Obr. 3.2 Ukázka vlastností

3.2 Rozdíl mezi Mathematicou a webMathematicou

Mathematica a webMathematica mají stejný základní „motor“, ale poskytují podstatně lišící se uživatelské rozhraní a jsou zaměřeny na různé typy uživatelů.

webMathematica nabízí přístup ke konkrétním Mathematica aplikací prostřednictvím webového prohlížeče nebo jiných webových klientů. Efektivní využití standardního rozhraní předpokládá jen malé znalosti. Uživatelé ve většině případů ani nemusí být obeznámeni s Mathematicou, ani potřebují vědět, že Mathematicu používají.

U webMathematicy je tomu podobně. Vývojáři potřebují pouze základní znalosti HTML a Mathematicy a jsou schopni vytvořit kompletní, full-featured webové stránky. Jiné technické programy vyžadují Java dovednosti a umožňují pouze vytváření malých appletů. Také proto, že webMathematica umožňuje přistupovat v plném rozsahu k výpočetním schopnostem zabudovaných do Mathematicy, vývojáři nemusí pracovat s extra knihovnami.

V jistém smyslu, Mathematica je vývojové prostředí pro webMathematica stránky, tzn., můžete vypracovat kód v Mathematice, např. modely neuronových sítí, kód, který pak může být umístěn do webMathematica stránky umožní ostatním spuštění modelu a využití jeho výsledků pro jejich pravidelnou práci [11].

3.3 Inovace ve webMathematice 3

webMathematica 3 implementuje revoluční funkce, včetně rychlého interaktivního pro server základního umístění na webu. Zahrnuta je i řada vylepšení pro optimalizaci vysokého provozu stránek.

V této verzi webMathematicy je k dispozici automatizovaný okamžitý interaktivní nástroj, tzv. Manipulate. S jeho pomocí lze tvořit webové stránky, které obsahují různé grafické prvky, jako jsou posuvníky, zaškrtávací políčka a rozbalovací menu, atd.

Dalším vylepšením je nový systém hromadné obsluhy pro dlouhoběžící a asynchronní výpočty.

Významnou novinkou je také podpora Wolfram Workbench, který nabízí velké množství funkcí urychlujících vývoj obsahu webMathematicy, tzn. kód Mathematicy lze ladit, když běží na serveru.

Dále také umožňuje používat webové služby REST a SOAP, které využívá i Mathematica. Výrazného zlepšení se dočkalo i monitorování jádra. Nový kód pro sledování využití paměti, času běhu, souběžných požadavků a Java objektů pomáhá zlepšit spolehlivost serveru, což umožňuje spouštění a ukončování jednotlivých jader a zrušení jednotlivých výpočtů.

webMathematica 3 lépe interaguje s jádrem Mathematicy. Jádra se spouští jakmile se spouští server, tzn. všechna jádra se spouští současně, což pomáhá vylepšit start serveru.

Má také řadu nových konfiguračních nástrojů, které mohou omezit využívání času a paměti jádra a zvýšit tak spolehlivost serveru. Jádra jsou automaticky na pozadí obnovována, takže server může pracovat bez přerušení [12].

3.4 Klíčové výhody webMathematicy 3

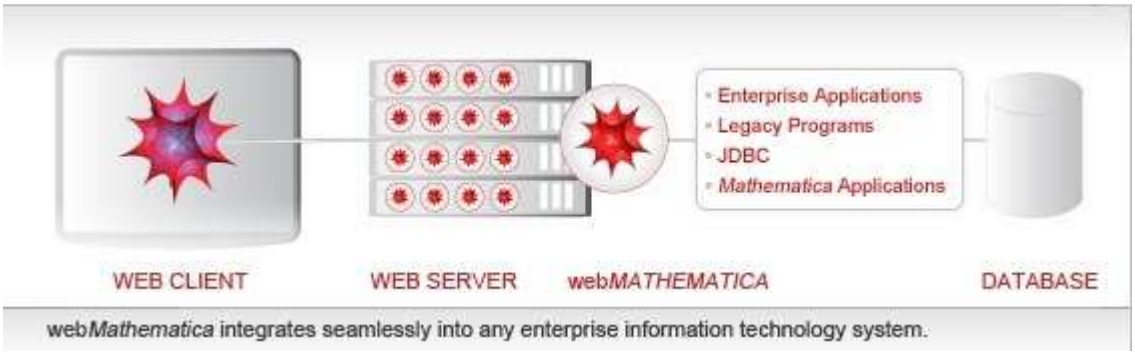
Dynamické webové stránky, které počítají vlastní výsledky a komunikují s uživateli, obvykle vyžadují složité programování a nespolehlivé funkce. Ale webMathematica 3 poskytuje řešení od startu k cíli, které kombinuje výpočetní výkon a znalosti Mathematicy s mocným jazykem pro snadný vývoj a škálovatelné rozmístění.

Při použití webMathematicy v organizaci je výhodné, že z obsahu stránek může čerpat více návštěvníků nebo lze vytvořit celou výpočetní infrastrukturu služeb, což snižuje počáteční investice. Uživatelům umožňuje on-line zpracování dat. webMathematica dokonce poskytuje aplikace do mobilních zařízení, takže personál má vždy přístup k nejnovějším nástrojům.

Obr. 3.3 Klíčové výhody pro organizace

Vzhledem k tomu, že ve webMathematice jsou k dispozici všechny funkce Mathematicy, je vývoj internetových stránek řešením v řádu minut, nikoli měsíců.

Tedy bez ohledu na velikost vytvářené aplikace, webMathematica sníží čas na vývoj aplikace, která bude v konečném výsledku stabilnější a její použití a údržba snadnější [13].



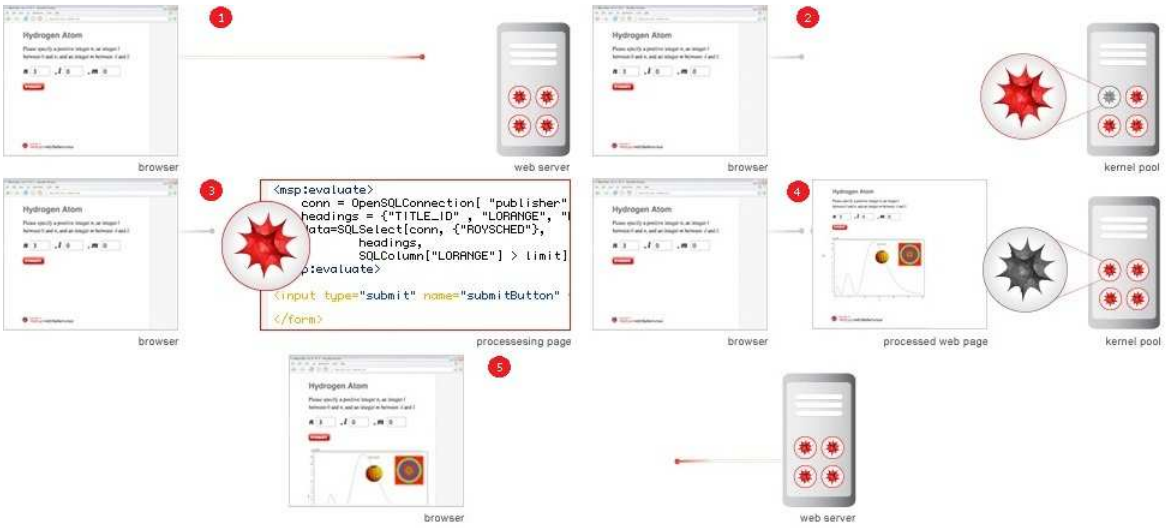
Obr. 3.4 Nákres propojení

3.5 Technologie

webMathematica snadno začleňuje standardní webové technologie. Vyberete, nainstalujete a konfigurujete webový server, Javu a servlet kontejnery dle svého výběru na podporované platformě dle svého výběru.

3.5.1 Jak webMathematica zpracovává požadavek?

Nejdříve prohlížeč odesílá HTTP požadavek na webový server. Požadavek odkazuje na konkrétní webMathematica stránku a obsahuje proměnné a jejich hodnoty. Webový server pak vykoná jeho předzpracování, tzn. ověřování, a předá jej webMathematice. Ta získá jádro Mathematicy ze skupiny předspuštěných jader. Do tohoto jádra jsou odesílány jakékoli proměnné a hodnoty. Jádro Mathematicy načte stránku a zpracuje všechny webMathematica značky. Jádro sestaví a poté vrátí výsledek. webMathematica přijme odezvu a dodá všechny potřebné HTTP hlavičky pro návrat do prohlížeče. Poté odstraní dočasné nastavení jádra Mathematicy a uvolní jej do portfolia dostupných jader. Webový server následně zpracuje veškeré kroky a vrátí HTML odpověď, kterou mohou používat aplety, plug-iny, nebo jiné dynamické HTML prvky v prohlížeči. Odpověď může být i v jiném formátu, např. MathML, TeX, nebo Mathematica notebook [14].



Obr. 3.5 Nákres postupu zpracování požadavku webMathematicou

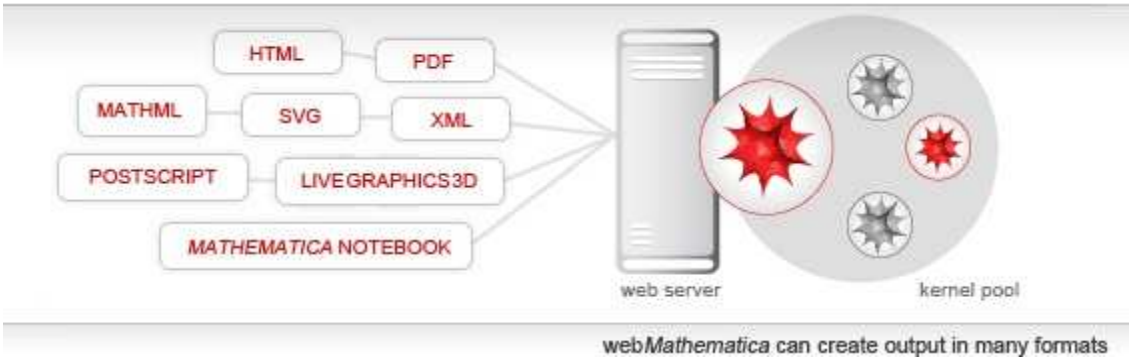
3.5.2 webMathematica stránky

webMathematica je založena na Mathematice a dvou standardních Java technologiích - Java Servlets a JavaServer Pages (JSP). Servlety jsou speciální programy v jazyce Java, který běží na servech, kde je Java odblokována a který je obvykle nazýván "servlet kontejner" (nebo někdy "servlet engine"). Existuje mnoho různých druhů servlet kontejnerů, které mohou běžet na mnoho různých operačních systémech a architekturách. Servlet kontejnery mohou být integrovány do jiných webových serverů, např. webový server Apache.

Pomocí webMathematicy se zobrazují HTML stránky rozšířené o příkazy Mathematicy. Pokud je požadavek podán u jedné z těchto stránek, Mathematica příkaz vyhodnotí a vypočítaný výsledek je vložen do stránky.

Technologie webMathematicy využívá požadavky / odpovědi standardně vykonávané webovými servery. Vstup může pocházet z HTML formuláře, appletů, JavaScript či web aplikací. Je možné také zasílat datové soubory na server webMathematicy ke zpracování. Jako výstup lze použít mnoho různých formátů, např. HTML, obrázky, Mathematica notebooky, MathML, SVG, XML, PostScript i PDF.

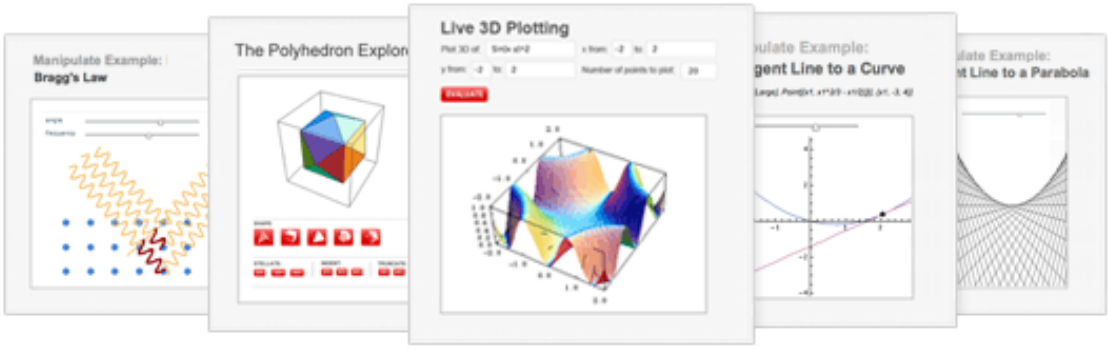
Zabezpečení webMathematicy je zajištěno tím, že je plně kompatibilní se standardním web-serverovým zabezpečením, jako jsou firewally, důvěryhodní hostitelé a HTTP-specifické rysy. Mathematica má specifické zabezpečení zabraňující spuštění systémových příkazů [14].



Obr. 3.6 Možné výstupní formáty webMathematicy

3.6 Interaktivní příklady

Na stránky webMathematicy můžeme umístit různé druhy interaktivních výpočtů z různých oblastí, např. vědy, financí, grafiky atd. [15].



Obr. 3.7 Interaktivní příklady

4 JAVASCRIPT

JavaScript je multiplatformní, objektově orientovaný a nejrozšířenější skriptovací jazyk na internetu. Byl navržen tak, aby přidával interaktivitu do webové stránky. JavaScript se zpravidla používá jako interpretovaný programovací jazyk často vkládaný přímo do HTML kódu stránky. Obvykle jsou jím ovládány různé interaktivní prvky GUI jako např. tlačítka či textová políčka nebo tvořeny animace a efekty obrázků [16], [17].

Obr. 4.1 Logo JavaScript

4.1 Přiblížení

Z hlediska syntaxe patří do rodiny jazyků C/C++/Java. Slovo Java je však součástí jeho názvu pouze z marketingových důvodů a s programovacím jazykem Java jej vedle názvu spojuje jen podobná syntaxe.

Původně byl JavaScript obchodní název implementace společnosti Netscape. Zde byl vyvíjen nejprve pod názvem Mocha, později LiveScript. V prosinci 1995 společně se společností Sun Microsystems byl ohlášen jako doplněk k jazykům HTML a Java. Pro verzi firmy Microsoft se používá název JScript. JScript je podporován platformou .NET.

Standardizován byl v červenci 1997 asociací ECMA a v srpnu 1998 ISO. Standardizovaná verze JavaScriptu je pojmenována jako ECMAScript, z které byly odvozeny i další implementace, např. ActionScript [16], [17].

Obr. 4.2 Logo ECMA INTERNATIONAL

4.2 Vlastnosti

Charakteristickou vlastností JavaScriptu je, že pracuje na straně uživatele a ne na serveru, tzn. spouští se až po stažení WWW stránky z Internetu, na rozdíl od např. PHP a ASP, které se spouštějí na straně serveru ještě před stažením z Internetu. Z toho vyplývá jisté bezpečností omezení, např. že JavaScript nemůže pracovat se soubory, čímž neohrozí soukromí uživatele.

Na straně serveru lze JavaScript použít také. Roku 1996 byla vypuštěna firmou Netscape první implementace JavaScriptu na straně serveru, tzv. LiveWire. Dnes existuje několik možností.

Základy JavaScriptu se snadno učí a snadno používají. Ke spuštění JavaScriptu uživatel potřebuje pouze webový prohlížeč s povoleným JavaScriptem (tato volba může být vypnuta) a případně také HTML editor pro psaní scriptů.

JavaScript má velmi široké spektrum použití. Od psaní textu na stránce v závislosti na podmínkách (např. pozdrav v závislosti na denní době), přes výzvy, varování z kontroly obsahu, až po uživatelem definované funkce, jako např. roztažení obrazu apod. [16], [17].

II. PRAKTICKÁ ČÁST

5 MODULY NEURONOVÝCH SÍTÍ PRO PROSTŘEDÍ WEBMATHEMATICA

Demonstrační úlohy jsem vytvářela v programech Wolfram Mathematica 7 for Students verze 7.0.0, Wolfram Workbench verze 2.0.

Obr. 5.1 Logo Wolfram Workbench 2.0

5.1 Přenosové funkce

Před programováním stránky byly nejdříve v Mathematice připraveny základní funkce, a to nejen z důvodů ověření jejich funkčnosti, ale také možnosti rychlé nápravy nedostatků.

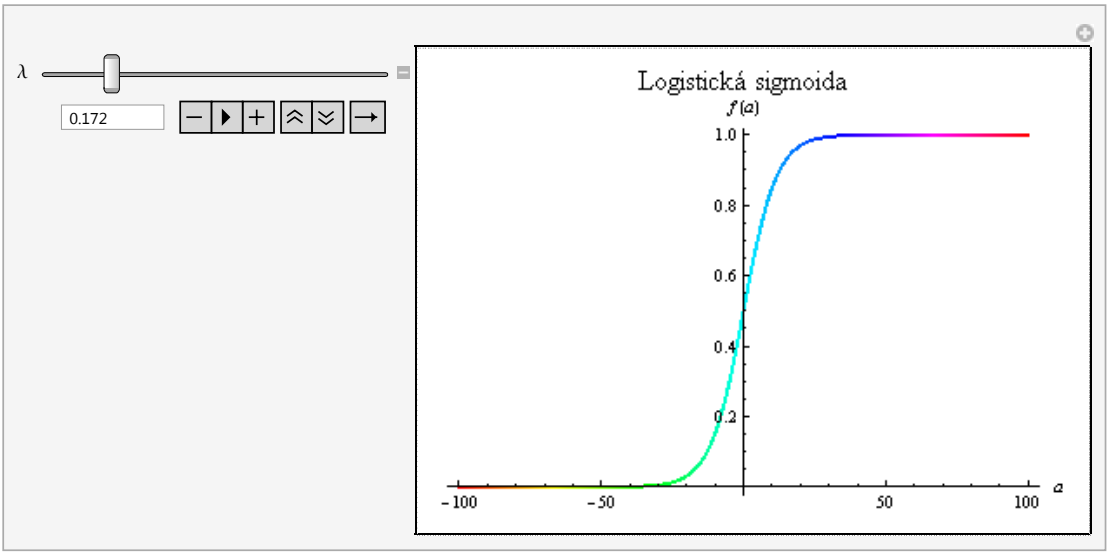
Následně tyto funkce byly přepsány v programu Wolfram Workbench 2 do tzv. *Form page*, který musí obsahovat takové elementy dokumentu, aby si s jeho překladem webMathematica poradila.

V přípravném notebooku a na webové stránce budou vykresleny grafy ve 2D i 3D všech přenosových funkcí, které nám byly představeny na přednáškách předmětu Metody umělé inteligence, tzn. logistická sigmoida, hyperbolický tangens, skoková funkce, perceptron, lineární funkce, omezená funkce a Gaussova funkce.

5.1.1 Tvorba zdrojového kódu v Mathematice

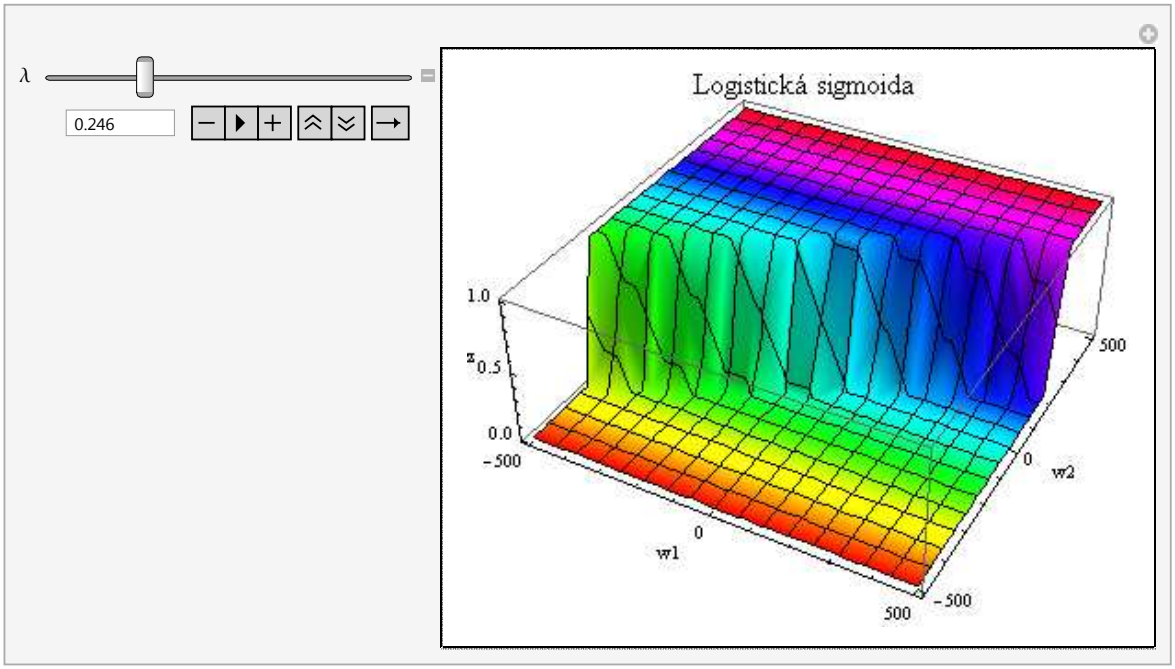
Spuštěním programu Wolfram Mathematica se otevře i prázdný notebook, který je nutné při práci průběžně ukládat pro případ selhání programu.

Funkce logistická sigmoida má funkční předpis daný vzorcem (4) uvedeným v kapitole 1. 2. 4 Přenosové funkce neuronu. Pomocí funkce *Plot* obsažené standardně v knihovně funkcí Mathematica byl vykreslen průběh této funkce ve 2D. Použitím funkce *Manipulate* a nadefinováním proměnné λ v intervalu od 0 do 1, která určuje strmost průběhu funkce, lze pozorovat jak se průběh funkce při různém nastavení λ mění. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 2.



Obr. 5.2 Výsledný graf funkce logistická sigmoida ve 2D

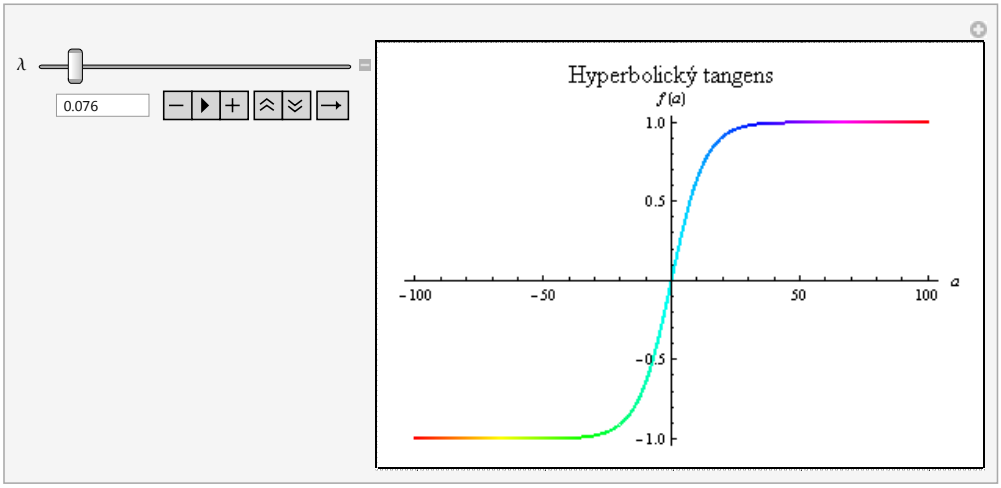
Pro 3D je předpis i kód funkce obdobný, ovšem s tím rozdílem, že byl přidán do předpisu třetí rozměr. A pro vykreslení grafu byla použita funkce *Plot* pro 3D, tzn. *Plot3D*, a pomocné proměnné *x1*, *x2*. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 3.



Obr. 5.3 Výsledný graf funkce logistická sigmoida ve 3D

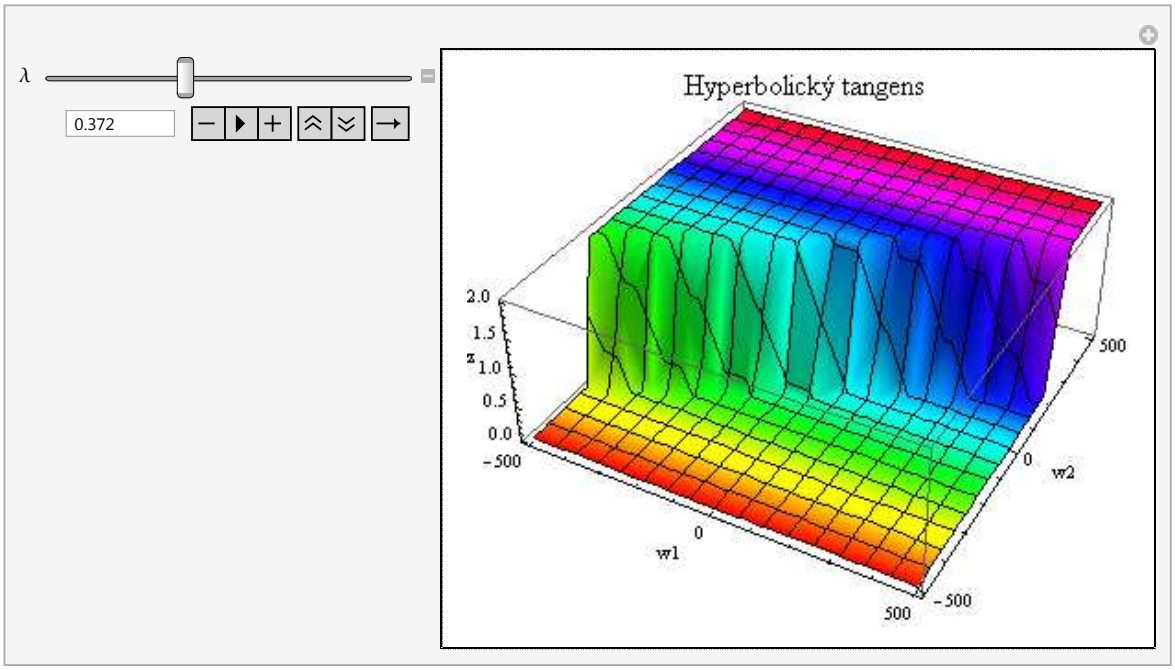
Hyperbolický tangens má funkční předpis daný vzorcem (5) uvedeným v kapitole 1. 2. 4 Přenosové funkce neuronu. Také i v tomto případě byl vykreslen průběh této funkce ve 2D za pomoci funkce *Plot* a použitím funkce *Manipulate* a nadefinováním proměnné λ v intervalu od 0 do 1 bude moci být ovlivněna strmost průběhu funkce při různém

nastavení hodnoty λ. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 4.



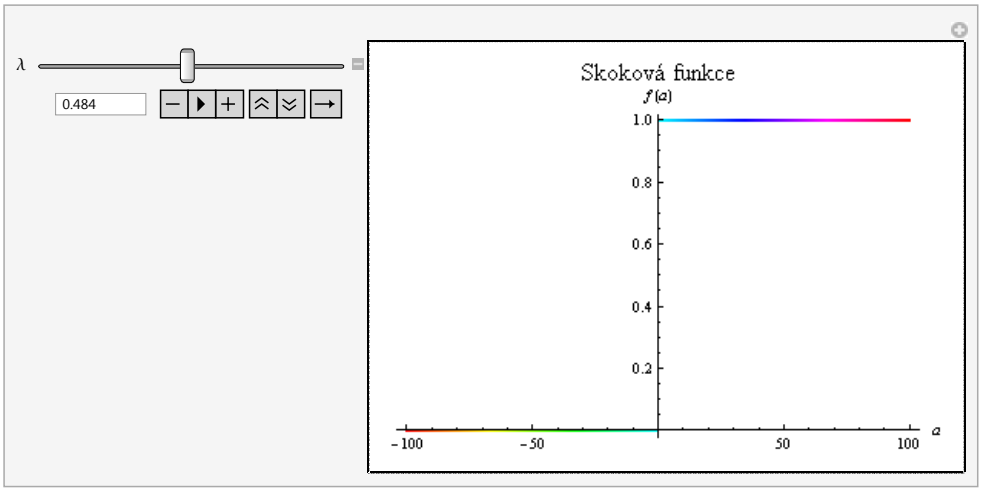
Obr. 5.4 Výsledný graf funkce hyperbolický tangens ve 2D

Podobně jako u funkce logistická sigmoida je i v tomto případě postup vedoucí k vykreslení 3D grafu stejný, tzn., do předpisu funkce byl přidán třetí rozměr a byla použita funkce *Plot3D* a pomocné proměnné *x1, x2*. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 5.



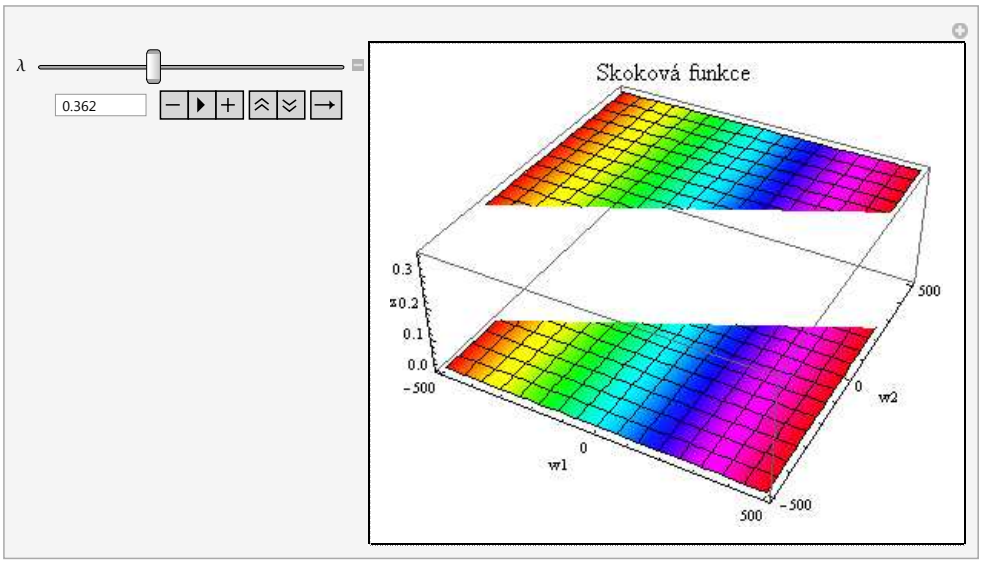
Obr. 5.5 Výsledný graf funkce hyperbolický tangens ve 3D

Kód pro vykreslení skokové funkce, která má funkční předpis daný vzorcem (3) uvedeným v kapitole 1. 2. 4 Přenosové funkce neuronu, se mírně od předchozích přenosových funkcí odlišuje. K vykreslení bylo nutné použít funkci *UnitStep*, která je také standardně obsažená v knihovně funkcí Mathematica. Další postup, který vede k vykreslení 2D grafu funkce je stejný, tzn., byla použita funkce *Plot* a *Manipulate*. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 6.



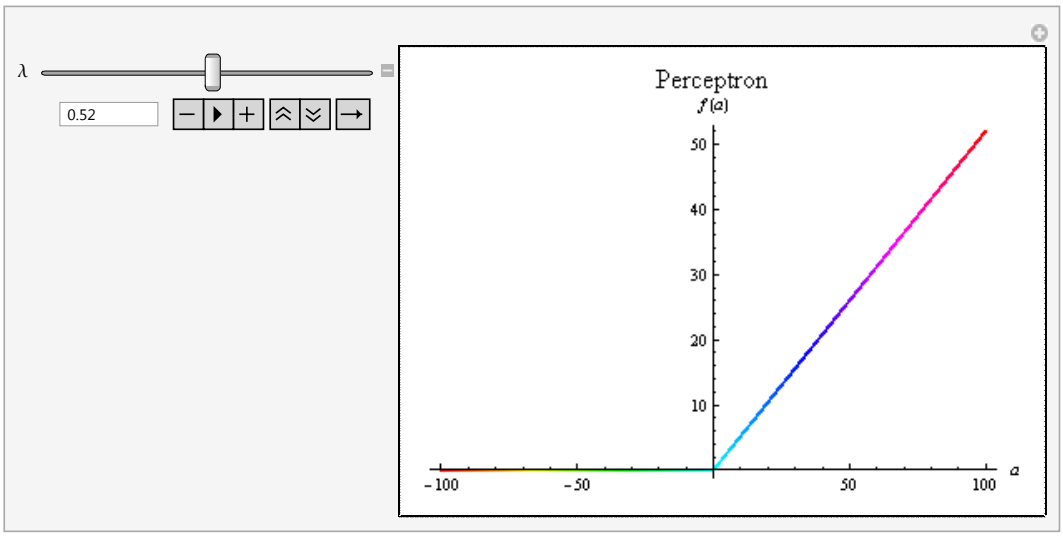
Obr. 5.6 Výsledný graf skokové funkce ve 2D

Pro vykreslení skokové funkce ve 3D je kód podobný jako pro tuto funkci ve 2D. Také je nezbytné použití funkce *UnitStep* jako u 2D grafu. Ovšem aby tuto funkci bylo možné vykreslit i ve 3D bylo opět nutné použít funkce *Plot3D a Manipulate* a pomocné proměnné *x1*, *x2*. Kód pro vykreslení je v příloze P I a výsledek po jeho spuštění je zobrazen na Obr. 5. 7.



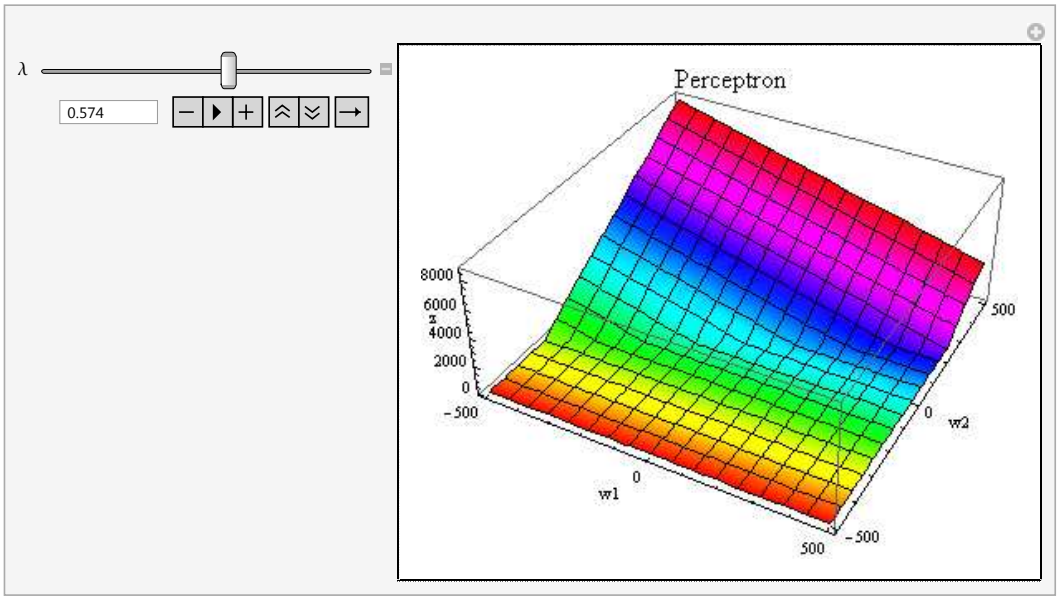
Obr. 5.7 Výsledný graf skokové funkce ve 3D

Přenosová funkce Perceptron má funkční předpis daný vztahem (2) uvedeným v kapitole 1. 2. 4 Přenosové funkce neuronu. K tomu, abych byl dodržen funkční předpis této funkce, je nutné použít funkci *If*, která vyjadřuje podmínku a je také standardně obsažená v knihovně funkcí Mathematica. Využití funkcí *Plot* a *Manipulate* je již samozřejmostí. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 8.



Obr. 5.8 Výsledný graf funkce Perceptron ve 2D

Aby bylo možné vykreslit graf přenosové funkce Perceptron i ve 3D bylo opět nutné využít funkce *If*, *Plot3D* a *Manipulate* a pomocných proměnných *x1*, *x2*. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 9.

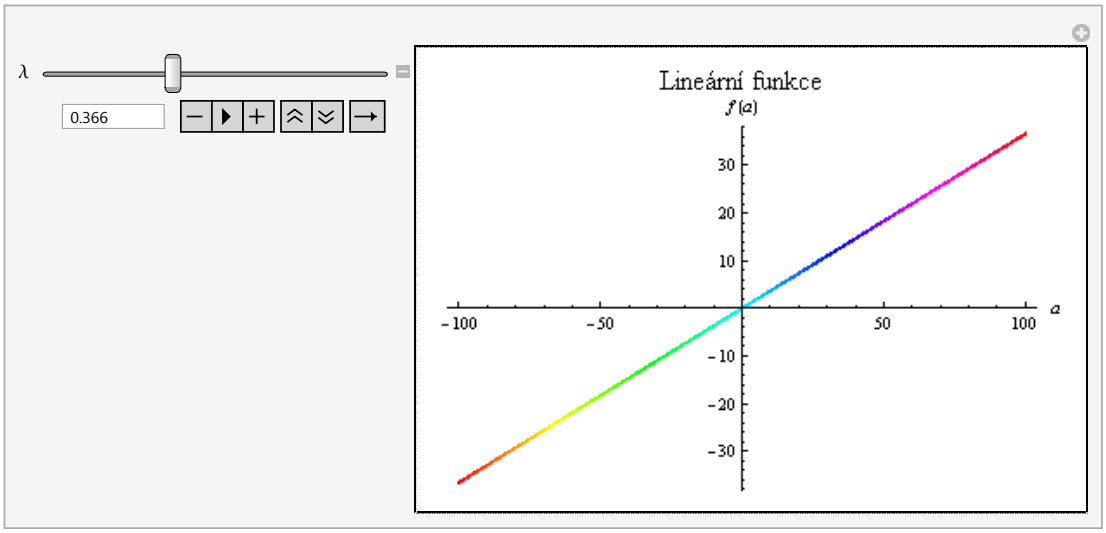


Obr. 5.9 Výsledný graf funkce Perceptron ve 3D

Přenosová funkce lineární má funkční předpis daný vztahem

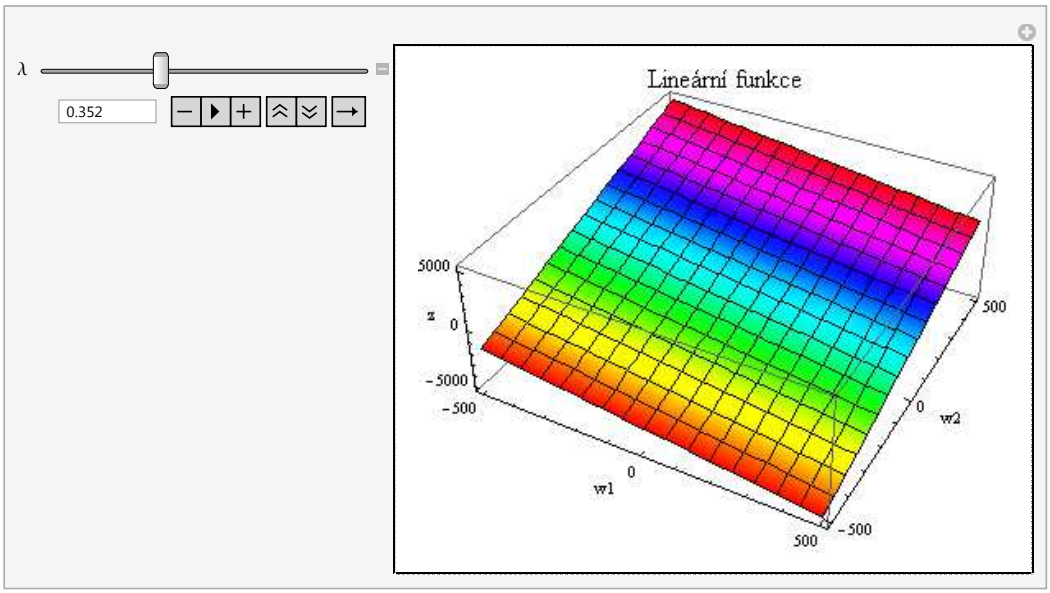
$$f(a) = \lambda \times a \; \forall \; a \tag{12}$$

I v tomto případě byl vykreslen průběh této funkce ve 2D za pomoci funkce *Plot* a použitím funkce *Manipulate.* Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 10.



Obr. 5.10 Výsledný graf lineární funkce ve 2D

K získání grafu této funkce i v 3D bylo nezbytné rozšířit předpis o třetí rozměr a použít opět funkcí *Plot3D* a *Manipulate* a samozřejmě také pomocné proměnné *x1*, *x2*. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 11.

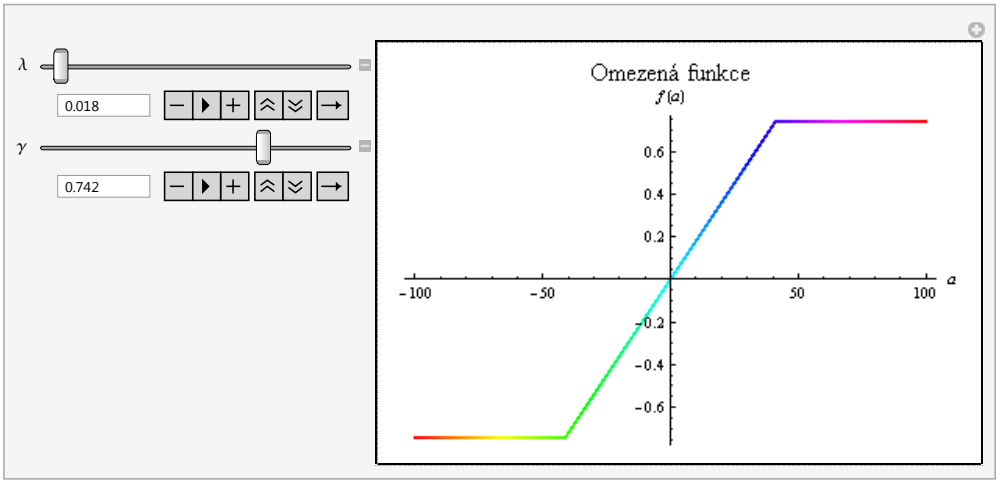


Obr. 5.11 Výsledný graf lineární funkce ve 3D

Funkční předpis omezené funkce je dán vztahem

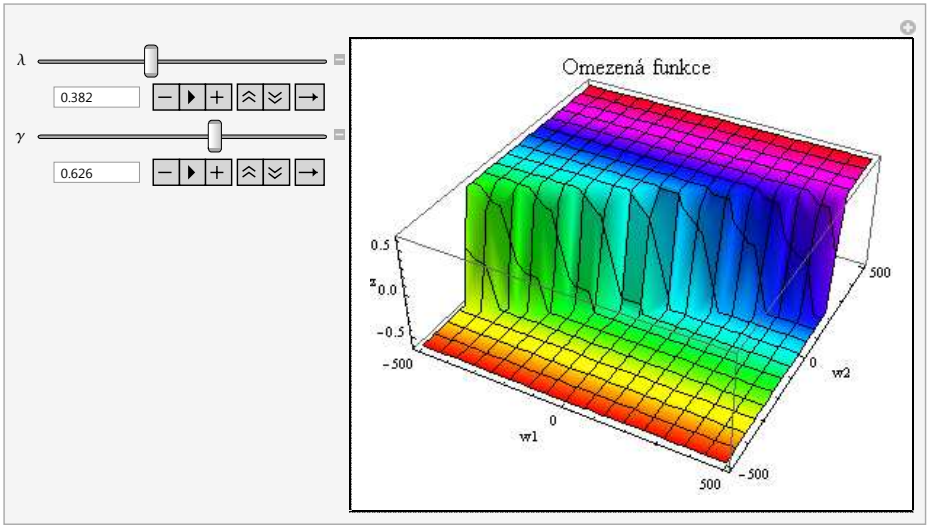
$$f(a) = \gamma \forall\, a > h_1 \vee \lambda \times a \forall\, a \in \langle -h_1; h_1 \rangle \vee -\gamma \forall\, a < -h_1 \tag{13}$$

K vykreslení grafu této funkce bylo potřebné využít funkci *Clip*, která je také standardně obsažená v knihovně funkcí Mathematica. Další postup přispívající k vykreslení 2D grafu funkce byl opět stejný, tzn., byla použita funkce *Plot* a *Manipulate.* Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 12.



Obr. 5.12 Výsledný graf omezené funkce ve 2D

K zobrazení omezené funkce i ve 3D bylo opět žádoucí použití funkce *Clip* jako u 2D grafu. Přirozeně, že k vykreslení bylo nutné použít funkce *Plot3D a Manipulate* a pomocné proměnné *x1*, *x2*. Kód pro vykreslení je uveden v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 13.

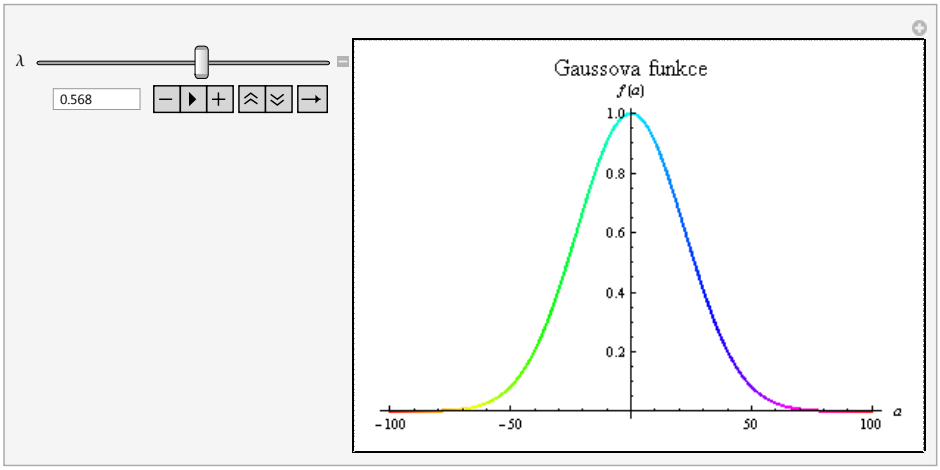


Obr. 5.13 Výsledný graf omezené funkce ve 3D

Funkční předpis Gaussovy funkce je dán vztahem

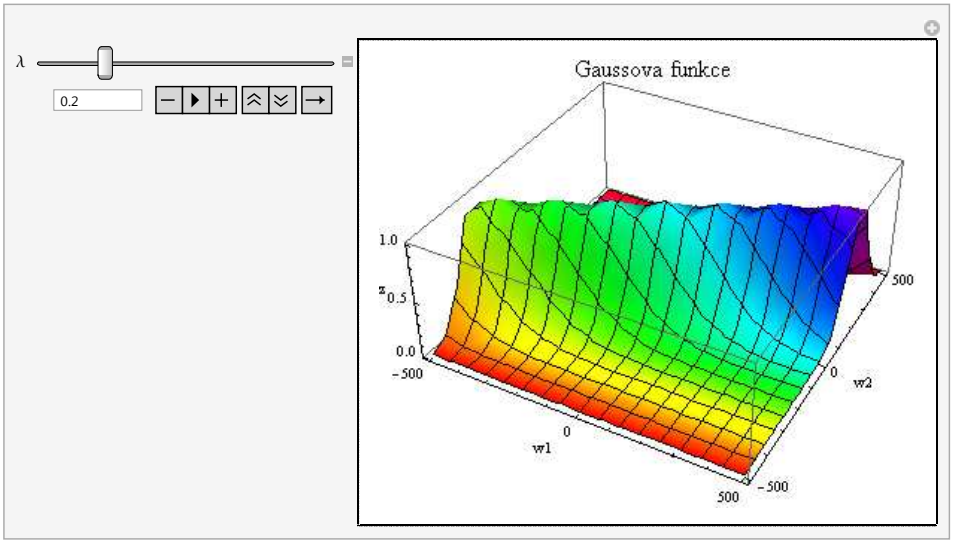
$$f(a) = e^{\frac{(\lambda \times a - m_j)^2}{\sigma_j^2}} \tag{14}$$

K tomu, aby byl vykreslen průběh této funkce, bylo nezbytné nadefinovat si proměnné *m* a *σ*. Proměnná *m* znamená střední hodnotu vstupů a proměnná *σ* představuje rozptyl vstupů. Využití funkcí *Plot* a *Manipulate* je samozřejmé. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 14.



Obr. 5.14 Výsledný graf Gaussovy funkce ve 2D

Aby bylo možné vykreslit graf Gaussovy funkce i ve 3D bylo opět nutné nadefinovat si proměnné *m* a *σ*, ovšem tentokrát s vyššími hodnotami. Žádoucí bylo opětovné využití funkcí *Plot3D* a *Manipulate* a pomocných proměnných *x1*, *x2*. Kód pro vykreslení je v příloze P I a výsledek po spuštění kódu zobrazen na Obr. 5. 15.



Obr. 5.15 Výsledný graf Gaussovy funkce ve 3D

Výše popsané je obsahem souboru **prenosfunkce.nb** na přiloženém CD.

5.1.2 Tvorba zdrojového kódu ve Wolfram Workbench 2.0

Při spouštění programu se nejdříve zobrazí dotaz, kde se má vytvořit tzv. workspace. Po odsouhlasení, popř. změně umístění se otevře okno programu s několika podokny. V největším z nich, kde se jako první otevře uvítací stránka, na které je možné zvolit online dokumentaci, webovou stránkou či dalšími pomocné průvodce k programu, bude tvořen kód stránky.

Jako první, bylo nezbytné vytvořit nový projekt přes menu *File – New – Project*, poté byla v nově otevřeném okně průvodce rozbalena nabídka *Mathematica*, ve které byla podsložka *webMathematica*, kde byla zvolena možnost *Form Project*. Poté bylo zadáno jméno projektu a stisknuto *Finish*. Tím se zobrazilo původní okno s tím rozdílem, že v podokně *Package Explorer* byla zobrazena složka nově vytvořeného projektu. Po jejím rozkliknutí podsložka *WebPages* již obsahovala soubor jsp, který se dvojklikem otevře v největším okně. Obsahem tohoto souboru byly již nejen základní elementy pro zobrazení stránky na webu, ale také ukázkový příklad, který začátečníkovi napoví, jak se taková stránka má tvořit.

Pokud je žádoucí do projektu přidávat další jsp soubory stačí pravým tlačítkem myši kliknout na podsložku *WebPages*, zvolit nabídku *New – Other*, otevře se okno průvodce, kde stejně jako při tvorbě nového projektu se rozbalí nabídka *Mathematica*, ve které je podsložka *webMathematica*, ale tentokrát je nutné zvolit možnost *Form Page*. Poté už jen změnit dle potřeby jméno jsp souboru a stisknout *Finish*. V podokně *Package Explorer* se již v podsložce *WebPages* zobrazí nový jsp soubor, který také obsahuje základní elementy pro zobrazení stránky na webu a ukázkový příklad.

Před tvorbou stránky o přenosových funkcích bylo nutné nejdříve vymazat původní ukázkový příklad. Při tvorbě bylo nezbytné mít na paměti, že kód, který má být vyhodnocen programem webMathematica, musí být „ohraničen“ tagy <msp:evaluate> a </msp:evaluate>.

Jako první, byl připsán následující kód, který zajistí správný chod používaných funkcí, tzn., inicializuje příslušné knihovny funkcí. Kód je uveden v příloze P I.

Aby bylo na dané stránce možné použít funkci *Manipulate* pro webMathematicu, tzn., *MSPManipulate*, musí být tato funkce „předdefinována“. Kód je uveden v příloze P I.

Další důležitá část je kód tzv. formuláře v HTML jazyce, jehož kód je uveden v příloze P I.

Atribut *action* má význam - skript, který bude zpracovávat data, a atribut *method* znamená způsob předávání dat. *Method = "post"* zabalí odesílaná data a odesílá je nezávisle, takže nejsou vidět. *Post* je dobré nastavit u delších formulářů.

Při psaní kódu pro jednotlivé přenosové funkce byl použit kód předpřipravený v programu Mathematica. Odlišné bylo, že místo funkce *Manipulate* pro program Mathematica musela být použita funkce *MSPManipulate* pro program webMathematica a tento kód „uzavřen“ mezi tagy , . Dále znak e, který v programu Mathematica označuje exponenciální funkci a lze ji zde místo funkce *Exp* použít, musel být ve vzorcích, které tento znak obsahovali, nahrazen funkcí *Exp*. A také proměnné označené řeckými písmeny musely být vypsány jejich názvy slovně, aby program webMathematica je nezaměnil za funkce standardně obsažené v knihovně funkcí.

Do kódu 3D grafů všech přenosových funkcí byla přidána proměnná *i*, která v těchto grafech zajistí možnost různého úhlu pohledu.

Tyto úpravy kódu byly provedeny u všech přenosových funkcí, čímž je základní funkcionalita splněná. Ukázka výsledného kódu je uvedena v příloze P I.

V dalším kroku byly „schovány“ jednotlivé grafy tak, aby se ukázaly stiskem příslušného tlačítka. Stiskem stejného tlačítka znovu se graf nazpět skryje. K této funkci byl využit jednoduchý script. Kód je uveden v příloze P I.

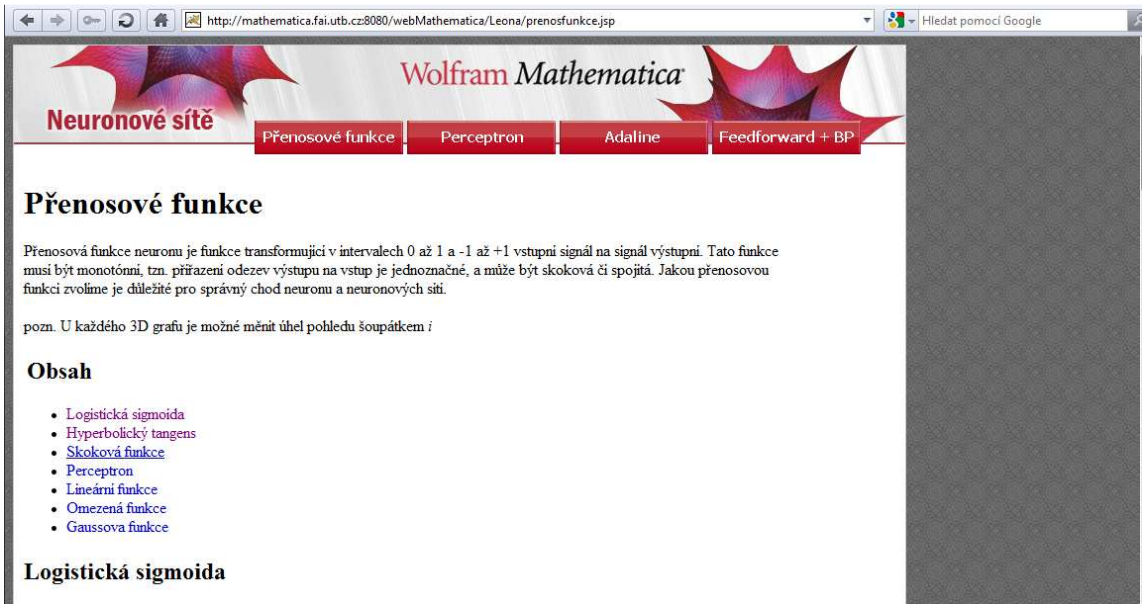
Poté před kód každého vykreslení grafu bylo nutné vložit buttonky, které budou grafy ukazovat/schovávat. A také kód každého grafu „obalit“ tagy , . Ukázka kódu pro buttonek je uvedena v příloze P I.

V další fázi bylo přidáno několik vylepšení pro lepší orientaci. Na začátek stránky byl připsán seznam přenosových funkcí, u kterého každá položka přesune odkazovanou část stránky na začátek okna prohlížeče. K nadpisu byl připojen tag odkazu na záložku a za každé vykreslení grafů dané přenosové funkce tag odkazu na onu záložku. Tím byl zajištěn z kterékoli části stránky rychlý přesun na začátek stránky. Kódy jsou uvedeny v příloze P I.

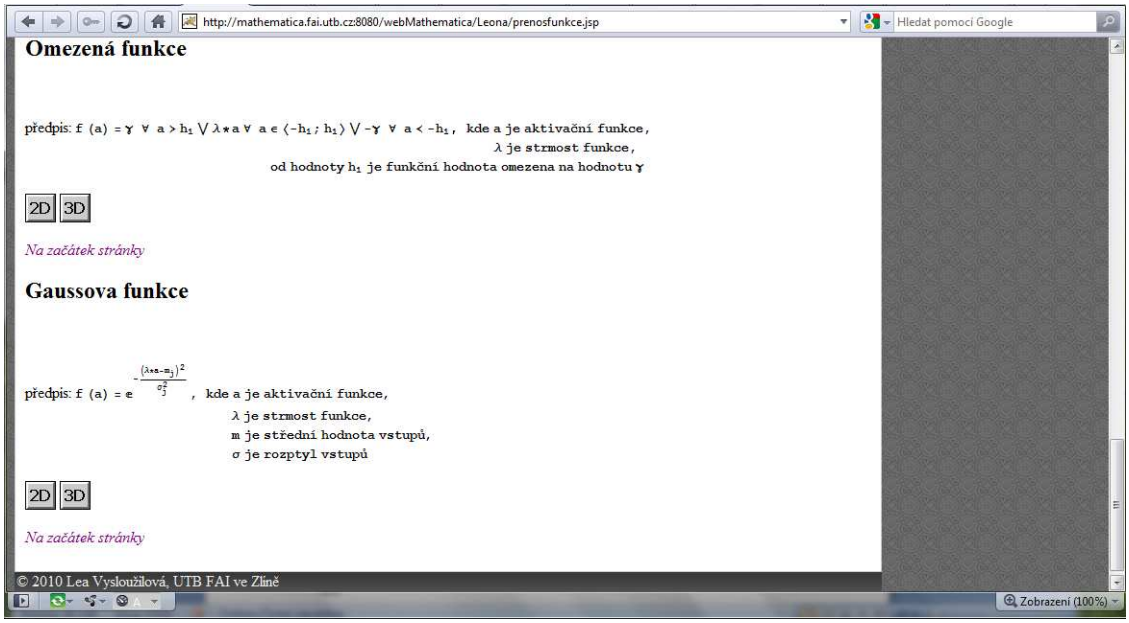
K tomu, aby stránka mohla sloužit jako výuková pomůcka, bylo nutné připsat informace o přenosových funkcích.

Poslední fází bylo dodat stránce formu. K úpravě vzhledu stránky, byl vytvořen vzhled v grafickém editoru Gimp a webové stránky byly napsány použitím HTML jazyka a kaskádových stylů. Tento styl byl použit i u dalších stránek, které jsou součástí diplomové práce.

Výše popsané je obsahem souboru **prenosfunkce.jsp** na přiloženém CD.



Obr. 5.16 Konečný vzhled stránky Přenosové funkce v prohlížeči Opera – začátek stránky



Obr. 5.17 Konečný vzhled stránky Přenosové funkce v prohlížeči Opera – konec stránky

5.2 Perceptron

V příkladě, který je zde popsán, byla použita metoda fixních přírůstků, jejíž koeficient je pevný, a nové váhy se v takovémto příkladě vypočítají podle vztahu (6). Ovšem tento vztah platí jen tehdy, platí-li vztah (7), jinak se nové váhy nepřepočítávají a platí vztah (8). Vztahy jsou uvedeny v kapitole 1. 3 Neuronová síť Perceptron.

Obdobně jako u programování stránky Přenosové funkce byly nejdříve v Mathematice připraveny základní funkce.

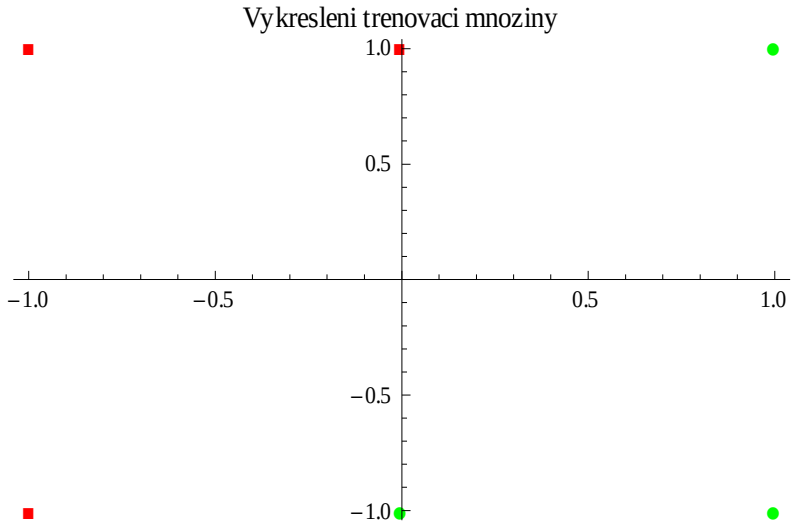
Tyto funkce byly následně přepsány v programu Wolfram Workbench 2 do tzv. *Form page*.

5.2.1 Tvorba zdrojového kódu v Mathematice

Do prázdného notebooku, který se otevře po spuštění programu Mathematica byly v prvním kroku, kdy bude síť Perceptron učena, nadefinovány všechny vstupní proměnné, jako jsou počáteční váhy, maximální počet epoch a trénovací množina. Další proměnné byly dle potřeby definovány v průběhu. Ukázka kódu nadefinování vstupních proměnných pro trénování sítě je uvedena v příloze P II.

Aby funkce pro trénování sítě probíhala do maximálního počtu epoch, nebo do doby kdy globální chyba bude nulová, bylo zajištěno použitím funkce *Table* a vnořením podmínkové funkce *If*. Použitím další funkce *If* bylo zajištěno, aby počet epoch nepřesáhl jejich maximální počet. Další vnořenou funkcí *Table* bylo ošetřeno, aby byly použity při přepočítávání všechny body trénovací množiny a přepočet vah. Aplikací další vnořené podmínky *If* byla zaručena změna vah podle výše uvedených vztahů. Následující vnořenou podmínkou *If* byla na konci epochy vypočítána globální chyba. Ukázka výsledného kódu zajištujícího učení sítě je uvedena v příloze P II.

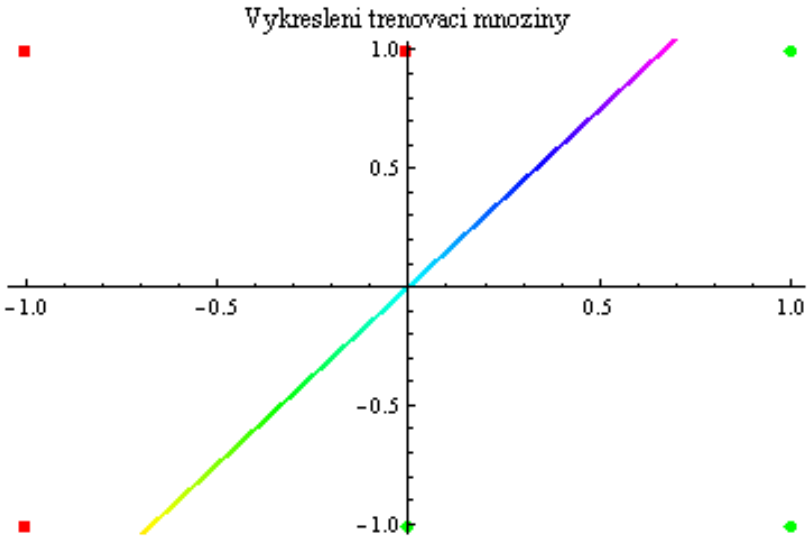
Kód, kterým byla vykreslena trénovací množina rozdělená do plusové a minusové třídy je uveden v příloze P II.



Obr. 5.18 Graf trénovací množiny

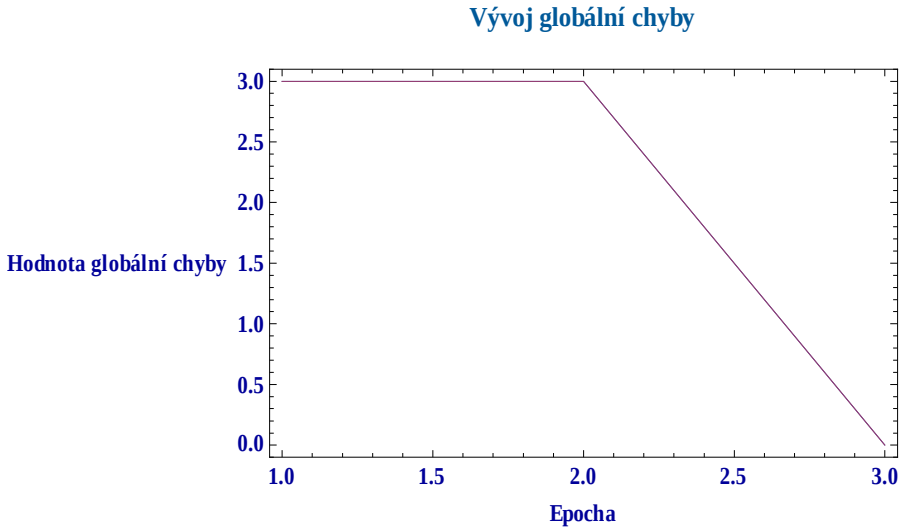
Několika podmínkami *If* byly ošetřeny různé případy, které mohou nastat při vykreslování hranice. Jejich kód je uveden v příloze P II.

Vykreslením hranice do grafu trénovací množiny byla zajištěna názorná ukázka, zda se jedná o případ lineární, popř. nelineární separability.



Obr. 5.19 Graf trénovací množiny s hranicí

Dalším grafem, který je nezbytné vykreslit je graf vývoje globální chyby. Kód pro jeho vykreslení je uveden v příloze P II.



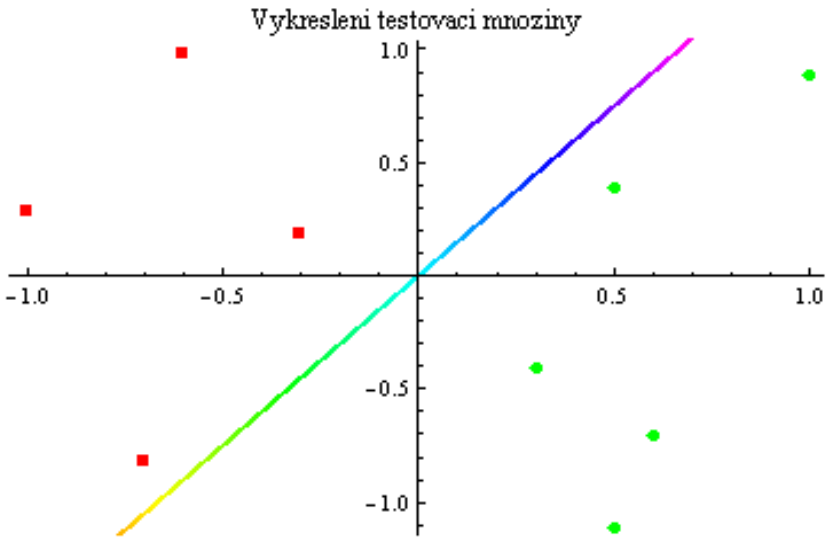
Obr. 5.20 Graf vývoje globální chyby

Ve fázi testování sítě bylo nutné si opět nejdříve nadefinovat vstupní proměnné, jako je testovací množina a předpokládané umístění bodů. Další proměnné byly postupně dodefinovány dle potřeby. Ukázka kódu nadefinování vstupních proměnných pro testování sítě je uvedena v příloze P II.

Aby funkce pro testování prošla při testování celou testovací množinu, byla použita funkce *Table*. První vnořenou podmínkovou funkcí *If* byly rozřazeny testované body do dvou tříd na kladné a záporné. Druhou vnořenou podmínkovou funkcí *If* byly porovnány, zda se rozřazení shoduje s předpokládaným umístěním bodů. Ukázka výsledného kódu testování sítě je uvedena v příloze P II.

Po testování sítě zbývá jen grafické znázornění výsledků, což zajištěno kódem uvedeným v příloze P II.

Výsledný graf:



Obr. 5.21 Graf testovací množiny s hranicí

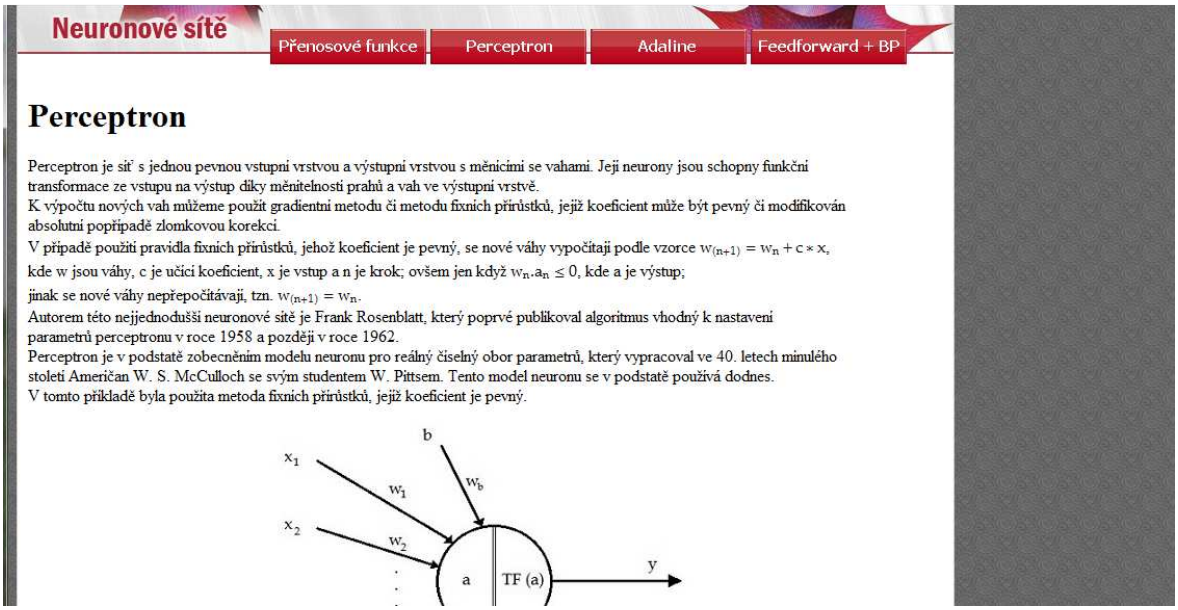
Výše popsané je obsahem souboru **sit_perceptron.nb** na přiloženém CD.

5.2.2 Tvorba zdrojového kódu ve Wolfram Workbench

Struktura této stránky byla při její tvorbě rozvržena na hlavní stránku a dvě podstránky, které budou sloužit jako ukázky lineárně separabilního a nelineárně separabilního problému. Každá stránka byla naprogramována vždy do nového jsp souboru.

Na hlavní stránce jsou uvedeny základní informace týkající se sítě Perceptron a obrázek modelu umělého neuronu. Na konci stránky jsou uvedeny odkazy, které přesměrují na výše zmíněné příklady. Kód je uveden v příloze P II.

K úpravě vzhledu stránky byl opět použit vytvořený styl, jako u předchozí stránky.



Obr. 5.22 Konečný vzhled stránky Perceptron v prohlížeči Opera

Při tvorbě stránky, která bude zobrazovat lineárně separabilní problém jako první, byl připsán kód, který zajistí správný chod používaných funkcí, tzn., inicializuje příslušné knihovny funkcí.
Na této stránce byl opět použit skript, který požadovanou část stránky po kliknutí na tlačítko skryje nebo ukáže.

Další důležitá část je kód tzv. formuláře v HTML jazyce, jehož atributy byly vysvětleny při tvorbě předchozí stránky.

Při načtení stránky je vidět pouze část stránky se zadáváním hodnot, zbylá část stránky je skryta. Políčka v zadávací části stránky při jejím načtení již hodnoty obsahují, ale lze vložit i vlastní hodnoty. Původní hodnoty jsou nastaveny tak, že po jejich přepočítání a zobrazení skryté části stránky se ukáže, že daný problém je lineárně separabilní. Ukázka části kódu pro zadávání hodnot je uvedena v příloze P II.

Funkce *DoPerceptron* je funkce, která obsahuje kód pro učení a testování sítě.

Následující část kódu je přiřazením hodnot z políček do proměnných, se kterými pracuje funkce *DoPerceptron*. Ukázka kódu je uvedena v příloze P II.

Pod tím to kódem následují dvě tlačítka. Tlačítko *Proveď*, pro přepočítání se zadanými hodnotami a tlačítko *Zobraz* pro zobrazení výsledků. Kód je uveden v příloze P II.

Kódem, který následuje, bylo zajištěno provedení funkce *DoPerceptron*. Kód je uveden v příloze P II.

Další částí kódu byl vykreslen graf trénovací množiny s hranicí. Kód pro její vykreslení je uveden v příloze P II.

Poté byla do tabulky vypsána historie vah a konečné váhy. Zbývá už jen vykreslit graf testovací množiny, výsledky testovaných bodů a graf vývoje globální chyby. Kód je uveden v příloze P II.

Na konec stránky zbylo doplnit odkazy na začátek stránky, na hlavní stránku Perceptron i na podstránku s nelineárně separabilním problémem. Kód je uveden v příloze P II.

K formátování, tzn., úpravě vzhledu stránky, byl použit styl, který byl použit i u předchozí stránky.

Obr. 5.23 Konečný vzhled stránky Perceptron v prohlížeči Opera – ukázka lineárně separabilní

Stránka nelineárně separabilní ukázky je obdobou předchozí stránky s tím rozdílem, že původní hodnoty jsou nastaveny tak, že po jejich přepočítání a zobrazení skryté části stránky se ukáže, že daný problém je nelineárně separabilní.

Výše popsané je obsahem souborů **perceptron.jsp, perceptronlinsep.jsp, perceptronnelinsep.jsp** na přiloženém CD.

Obr. 5.24 Konečný vzhled stránky Perceptron v prohlížeči Opera – ukázka nelineárně separabilní

5.3 Adaline

V příkladě, který je popsán níže se hodnota aktivační funkce při učení sítě vypočítá podle vzorce (10). Nové váhy se přepočítávají podle vzorce (11). Vzorce jsou uvedeny v kapitole 1. 4 Neuronová síť Adaline.

Obdobně jako u programování předchozích stránek byly nejdříve v programu Mathematica připraveny základní funkce.

Tyto funkce byly následně přepsány v programu Wolfram Workbench 2 do nového *Form page*.

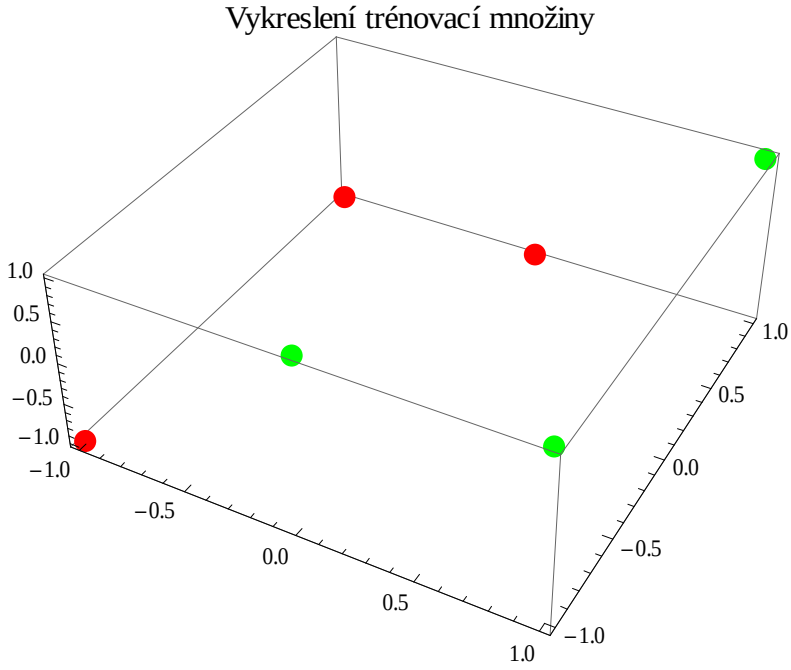
5.3.1 Tvorba zdrojového kódu v Mathematice

Do prázdného notebooku, který se otevře po spuštění programu Mathematica byly v prvním kroku, kdy bude síť Adaline učena, nadefinovány všechny vstupní proměnné, jako jsou počáteční váhy, maximální počet epoch, trénovací množinu, třídu předpokládaného umístění, práh a učící koeficient. Další proměnné byly dle potřeby definovány v průběhu. Ukázka kódu nadefinování vstupních proměnných pro trénování sítě je uvedena v příloze P III.

Aby funkce pro trénování sítě probíhala do maximálního počtu epoch, nebo do doby kdy globální chyba bude nulová, bylo zajištěno použitím funkce *Table* a vnořením podmínkové funkce *If*. Použitím další funkce *If* bylo zajištěno, aby počet epoch nepřesáhl jejich

maximální počet. Další vnořenou funkcí *Table* bylo ošetřeno, aby byly použity při přepočítávání všechny body trénovací množiny a přepočet vah. Následující vnořenou podmínkou *If* byla na konci epochy vypočtena globální chyba. Ukázka výsledného kódu zajištujícího učení sítě je uvedena v příloze P III.

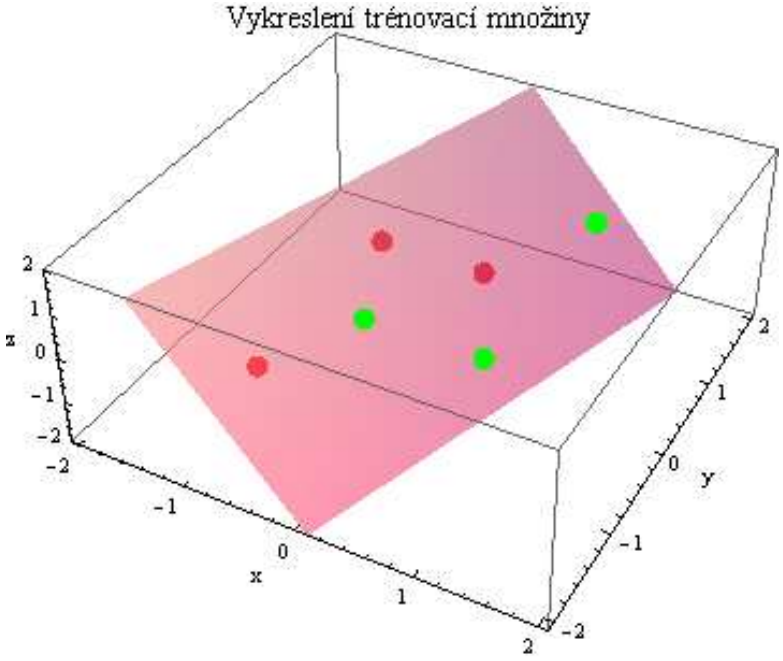
Následujícím kódem byla vykreslena trénovací množina rozdělená do plusové a minusové třídy. Kód je uveden v příloze P III.



Obr. 5.25 Graf trénovací množiny

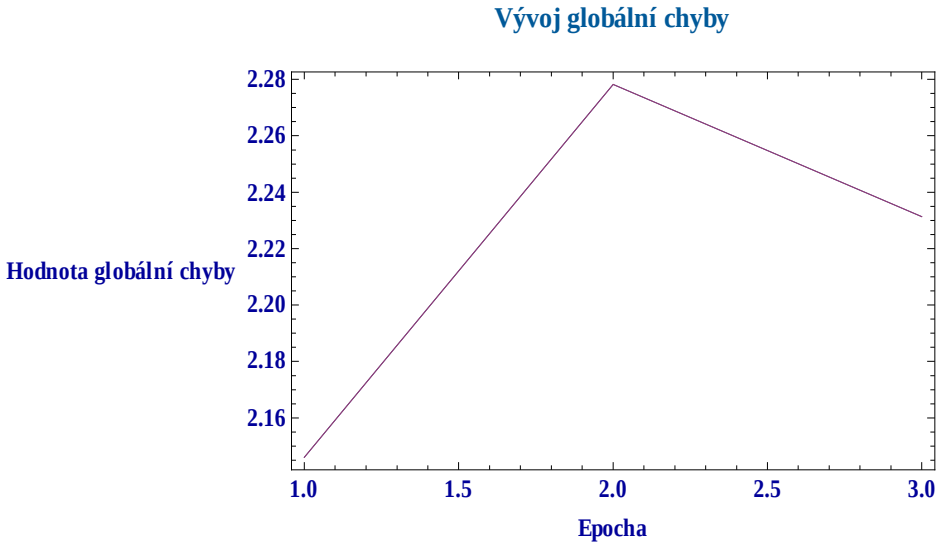
Následným kódem byly ošetřeny různé případy, které mohou nastat při vykreslování hranice a vykreslení trénovací množiny s hranicí. Kód je uveden v příloze P III.

Vykreslením hranice do grafu trénovací množiny byla zajištěna názorná ukázka, zda se jedná o případ lineární, popř. nelineární separability. V následujícím grafu je vidět, že se jedná o lineárně separabilní problém, protože prvky byly zařazeny správně, tzn., všechny červené body leží pod hranicí a všechny zelené leží nad ní.



Obr. 5.26 Graf trénovací množiny s hranicí

Dalším grafem, který bylo nezbytné vykreslit je graf vývoje globální chyby. Kód pro jeho vykreslení je uveden v příloze P III.



Obr. 5.27 Graf vývoje globální chyby

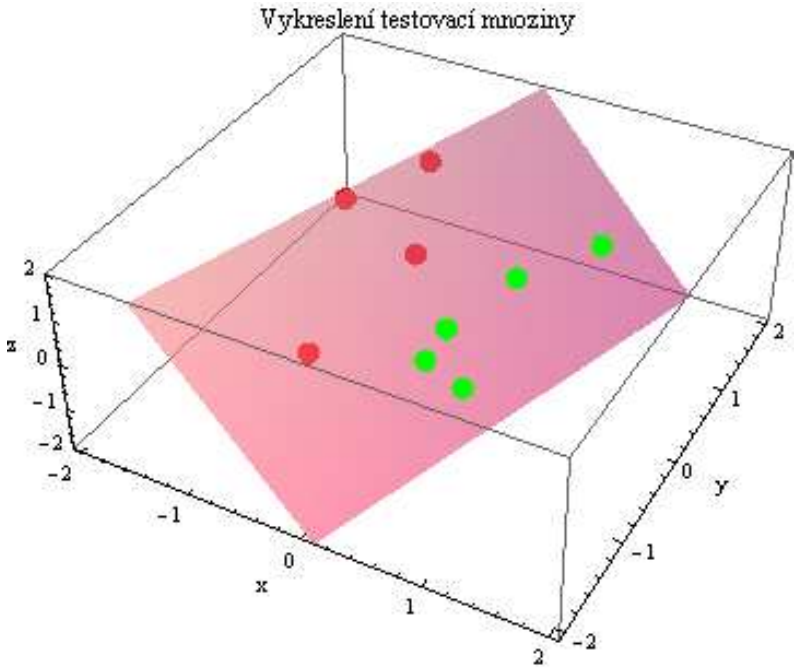
Ve fázi testování sítě byly opět nejdříve nadefinovány vstupní proměnné, jako je testovací množina a předpokládané umístění bodů. Další proměnné byly postupně dodefinovány dle

potřeby. Ukázka kódu nadefinování vstupních proměnných pro testování sítě je uvedena v příloze P III.

Aby funkce pro testování prošla při testování celou testovací množinu, byla použita funkce *Table*. První vnořenou podmínkovou funkcí *If* byly testované body rozřazeny do dvou tříd na kladné a záporné. Druhou vnořenou podmínkovou funkcí *If* bylo porovnáno, zda se rozřazení shoduje s předpokládaným umístěním bodů. Ukázka výsledného kódu testování sítě je uvedena v příloze P III.

Po testování sítě zbývá jen grafické znázornění výsledků, což bylo zajištěno kódem uvedeným v příloze P III.

Výsledný graf:



Obr. 5.28 Graf testovací množiny s hranicí

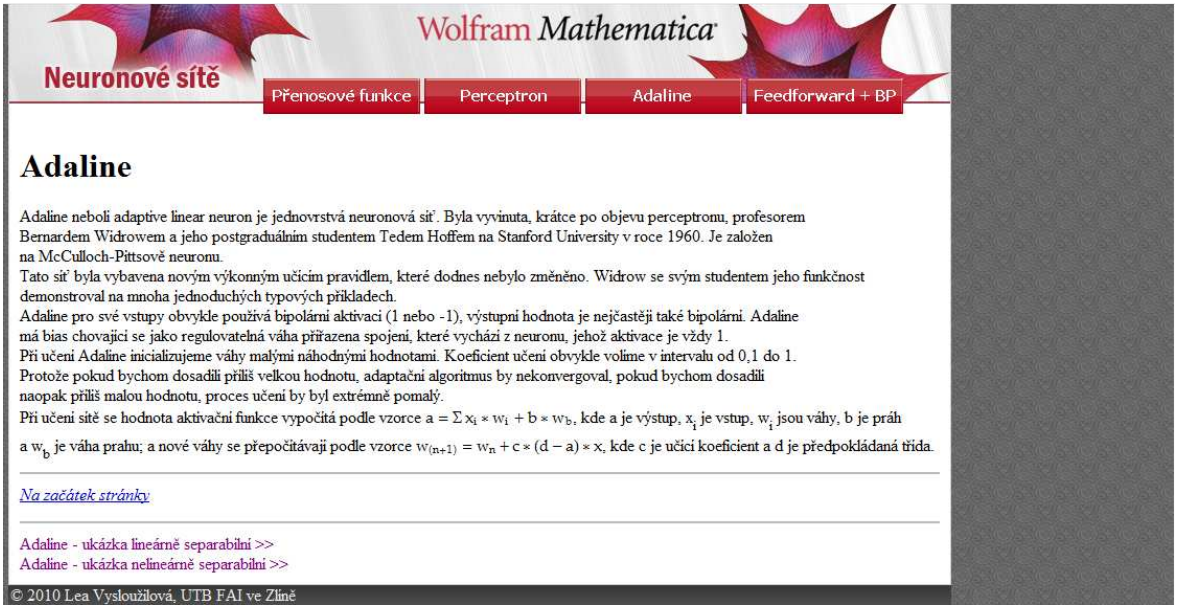
Výše popsané je obsahem souboru **sit_adaline.nb** na přiloženém CD.

5.3.2 Tvorba zdrojového kódu ve Wolfram Workbench

Struktura této stránky byla při její tvorbě rozvržena na hlavní stránku a dvě podstránky, které budou sloužit jako ukázky lineárně separabilního a nelineárně separabilního problému. Každá stránka byla naprogramována vždy do nového jsp souboru.

Na hlavní stránce jsou uvedeny základní informace týkající se sítě Adaline. Na konci stránky jsou uvedeny odkazy, které přesměrují na výše zmíněné příklady. Kód je uveden v příloze P III.

K úpravě vzhledu stránky byl opět použit vytvořený styl, jako u předchozích stránek.



Obr. 5.29 Konečný vzhled stránky Adaline v prohlížeči Opera

Při tvorbě stránky, která bude zobrazovat lineárně separabilní problém jako první, byl připsán kód, který zajistí správný chod používaných funkcí, tzn., inicializuje příslušné knihovny funkcí.

Na této stránce byl opět použit skript, který požadovanou část stránky po kliknutí na tlačítko skryje nebo ukáže.

Další důležitá část je kód tzv. formuláře v HTML jazyce, jehož atributy byly vysvětleny při tvorbě stránky o přenosových funkcích neuronu.

Při načtení stránky je vidět pouze část stránky, kde lze zadávat hodnoty, zbylá část stránky je skryta. Políčka v zadávací části stránky při jejím načtení již hodnoty obsahují, ale lze vložit i vlastní hodnoty. Původní hodnoty jsou nastaveny tak, že po jejich přepočítání a zobrazení skryté části stránky se ukáže, že daný problém je lineárně separabilní.

Ukázka části kódu pro zadávání hodnot je uvedena v příloze P III.

Funkce *DoAdaline* je funkce, která obsahuje kód pro učení a testování sítě. Část kódu, která následuje, je přiřazením hodnot z políček do proměnných, se kterými pracuje funkce *DoAdaline*. Kód je také uveden v příloze P III.

Následuje kód pro dvě tlačítka. Tlačítko *Proveď*, pro přepočítání se zadanými hodnotami a tlačítko *Zobraz* pro zobrazení výsledků. Následujícím kódem bylo zajištěno provedení funkce *DoAdaline.* Další částí kódu byl vykreslen graf trénovací množiny s hranicí. Kódem, který následuje, byla vypsána historie vah a konečné váhy do tabulky. Tyto kódy jsou také uvedeny v příloze P III.

Zbývá vykreslit graf testovací množiny, výsledky testovaných bodů a graf vývoje globální chyby. Kód je také uveden v příloze P III.

Na konec stránky byly už jen doplněny odkazy na začátek stránky, na hlavní stránku Adaline i na podstránku s nelineárně separabilním problémem. Kód je uveden v příloze P III.

K formátování, tzn., úpravě vzhledu stránky, byl použit styl, který byl použit i u předchozích stránek.

Obr. 5.30 Konečný vzhled stránky Adaline v prohlížeči Opera – ukázka lineárně separabilní

Stránka nelineárně separabilní ukázky je obdobou předchozí stránky s tím rozdílem, že původní hodnoty jsou nastaveny tak, že po jejich přepočítání a zobrazení skryté části stránky se ukáže, že daný problém je nelineárně separabilní.

Obr. 5.31 Konečný vzhled stránky Adaline v prohlížeči Opera – ukázka nelineárně separabilní

Výše popsané je obsahem souborů **adaline.jsp, adalinelinsep.jsp, adalinenelinsep.jsp** na přiloženém CD.

5.4 Feedforward

Obdobně jako u předchozích stránek byly nejdříve v Mathematice připraveny základní funkce a tím ověřena jejich funkčnost.

Tyto funkce byly následně přepsány v programu Wolfram Workbench 2 do nového *Form page*.

Příklad, který je popsán níže se jedná o vícevrstvou neuronovou síť s dopředným šířením signálu a zpětným šířením chyby o jednom vstupu a prahu, 3 neuronech ve skryté vrstvě a prahu a výstupním neuronu. Vzorce pro aktivační a adaptační fázi jsou uvedeny na Obr. 1. 9 a 1. 10 v kapitole 1. 5 Vícevrstvá síť a učící algoritmus Backpropagation.

5.4.1 Tvorba zdrojového textu v Mathematice

Do prázdného notebooku, který se otevře po spuštění programu Mathematica byly nejdříve naprogramovány potřebné funkce.

Funkce *epocha* vytvoří vektor vah pro každou epochu. Její kód je uveden v příloze P IV.

Funkce *Uceni3* nejdříve projde aktivační fázi pro daný prvek z trénovací množiny a následně vypočítá veškeré přírůstky vah, pomocné proměnné a nové váhy.

Funkcí *DejVystup* byl získán výstup ze sítě, nebo také y-ovou hodnotu funkce pro vstup x.

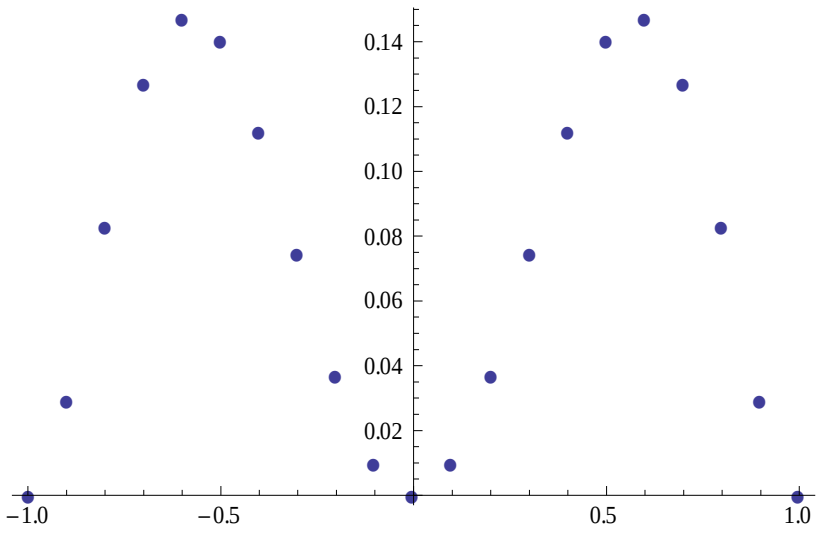
Funkce *vyplodvahy* na začátku inicializuje váhy. Kódy těchto funkcí jsou uvedeny v příloze P IV.

Následující kód, který je také uveden v příloze P IV, je nastavením funkce ve skryté vrstvě a na výstupu neuronu.

Kódem, který dále následuje, byly připraveny souřadnice pro požadovaný průběh funkcí sinus, cosinus, sextic a quintic [18]. Kód je také uveden v příloze P IV.

Dále byly nadefinovány všechny vstupní proměnné, jako je učící koeficient, momentum ovlivňující rychlost učení, počet neuronů ve skryté vrstvě, pomocné proměnné pro přepočet vah, maximální počet epoch, funkci skryté a výstupní vrstvy a počáteční váhy. Kód je také uveden v příloze P IV.

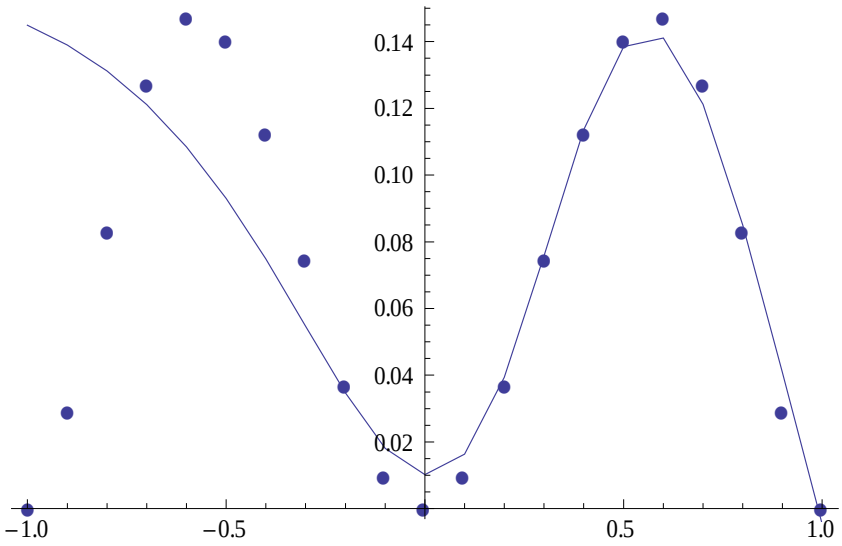
Vykreslení grafu požadovaného průběhu funkce, bylo docíleno kódem také uvedeným v příloze P IV.



Obr. 5.32 Graf požadovaného průběhu funkce sextic

Kódem, který následuje a je uveden v příloze P IV, byly získány hodnoty pro vykreslení vypočteného průběhu funkce.

Zbývá vykreslení grafu funkce aproximované neuronovou sítí vůči požadovaným hodnotám. Požadovanou funkcí v tomto případě je funkce sextic. Kód pro její vykreslení je uveden v příloze P IV.



Obr. 5.33 Graf funkce aproximované neuronovou sítí vůči požadovaným hodnotám

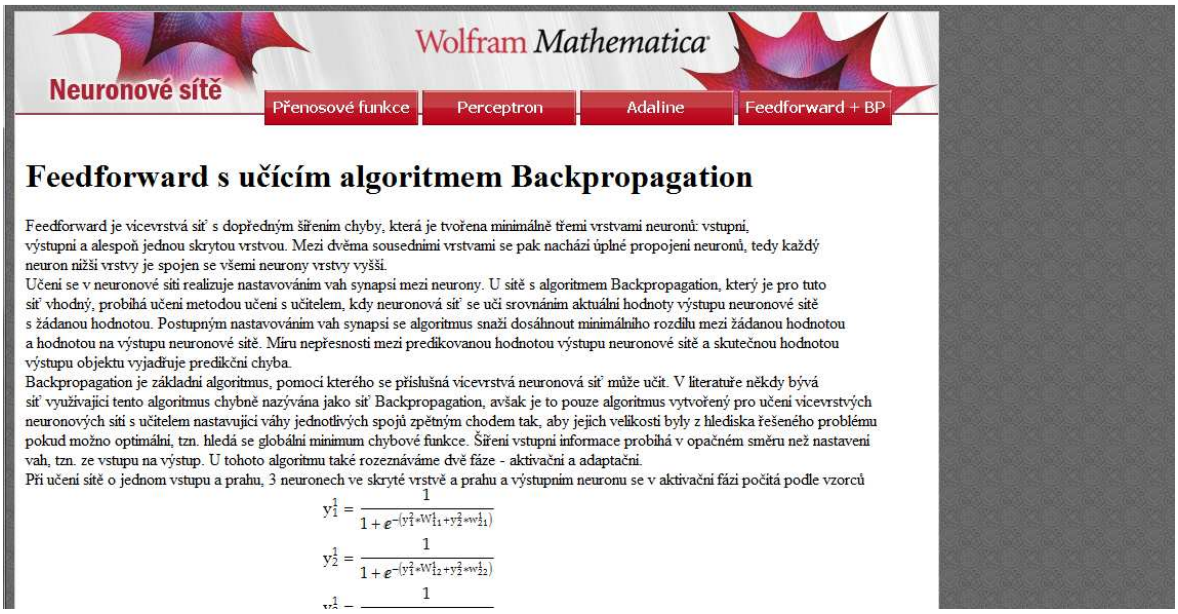
Výše popsané je obsahem souboru **sit_feedfrwBP.nb** na přiloženém CD.

5.4.2 Tvorba zdrojového textu ve Wolfram Workbench

Struktura této stránky byla také při její tvorbě rozvržena na hlavní stránku a jednu podstránku, která bude sloužit jako ukázka. Každá stránka byla naprogramována vždy do nového jsp souboru.

Na hlavní stránce jsou uvedeny základní informace týkající se neuronové sítě Feedforward a učícího algoritmu Backpropagation. Také jsou zde uvedeny vzorce používané v aktivační a adaptační fázi tohoto příkladu, i obrázek topologie sítě, která je v tomto příkladu použita. Na konci stránky je uveden odkaz, které přesměruje na výše zmíněný příklad. Kód je uveden v příloze P IV.

K úpravě vzhledu stránky byl opět použit vytvořený styl, jako u předchozích stránek.



Obr. 5.34 Konečný vzhled stránky Feedforward s učícím algoritmem Backpropagation v prohlížeči Opera

Při tvorbě stránky ukázky jako první, byl připsán kód zajišťující správný chod používaných funkcí, tzn., že inicializuje příslušné knihovny funkcí.
Na této stránce byl opět použit skript, který požadovanou část stránky po kliknutí na tlačítko skryje nebo ukáže.

Další důležitá část je kód tzv. formuláře v HTML jazyce, jehož atributy byly vysvětleny při tvorbě stránky o přenosových funkcích neuronu.

Při načtení stránky je opět vidět pouze část stránky, kde lze zadávat hodnoty, zbylá část stránky je skryta. Políčka v zadávací části stránky při jejím načtení již hodnoty obsahují, ale lze vložit i vlastní hodnoty. V zadávací části je možné vybrat i úlohu, kterou síť bude řešit. Na výběr je funkce sinus, cosinus, sextic a quintic. Další část kódu ohraničená tagy <msp:evaluate>, </msp:evaluate> obsahuje a vyhodnocuje funkci *switchfunction*, která zajistí, že následující funkce budou pracovat s úlohou vybranou v zadávací části. Dále obsahuje funkce např. *epocha*, *Uceni3*, *vystupZeSite*, *vystupZeSkryte*, *DejVystup*, *vyplodvahy*, atd. Tyto funkce byly popsány v předchozí kapitole.

Dalšími částmi kódu byly připraveny zbylé potřebné hodnoty. Kód je uveden v příloze P IV.
Následuje část kódu, která přiřazuje hodnoty z políček do proměnných, se kterými pracují vytvořené funkce. Kód je uveden v příloze P IV.

Kódem, který je uveden v příloze P IV, byly vypočteny počáteční váhy a proveden konečný výpočet.

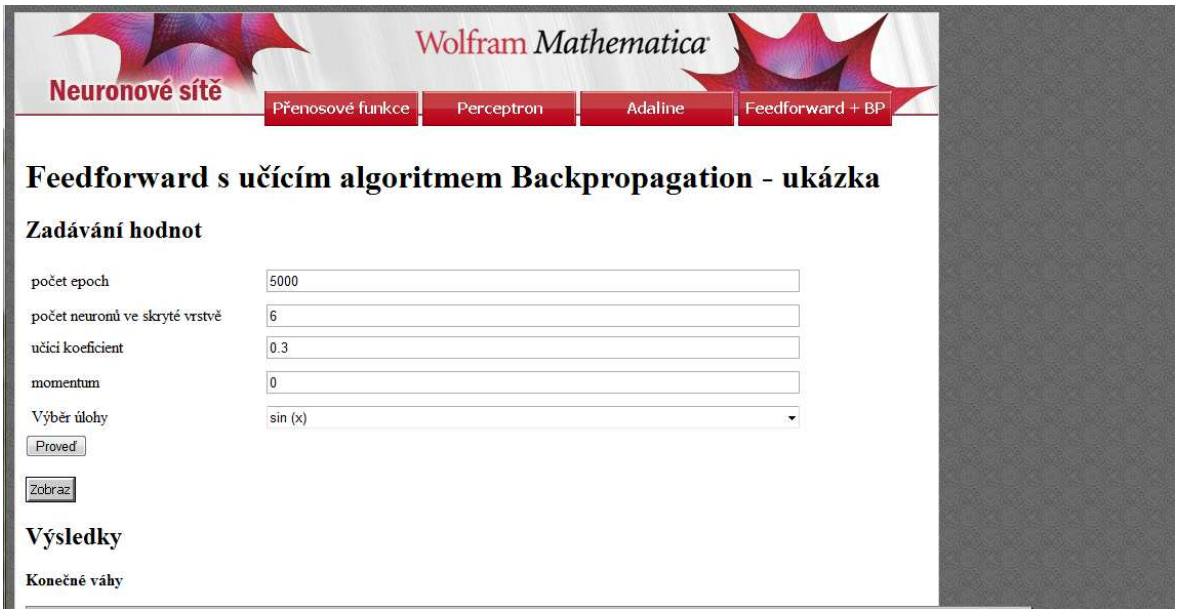
Pod tímto kódem následují dvě tlačítka. Tlačítko *Proveď*, pro přepočítání se zadanými hodnotami a tlačítko *Zobraz* pro zobrazení výsledků.

Následujícím kódem byly vypsány do tabulky konečné váhy. Další částí kódu byl vykreslen konečný graf funkce aproximované neuronovou sítí vůči požadovaným hodnotám.

Zbývající částí kódu byl vykreslen vývoj grafů. Jelikož počet probíhajících epoch se může pohybovat v řádech tisíců a všechny grafy vykreslit by bylo náročné pro prohlížeč, byl zvolen příklad z historie grafů mezi prvním a posledním. Byl vybrán stý graf od konce. Kód je uveden v příloze P IV.

Na konec stránky zbývá už jen doplnit odkazy na začátek stránky a na hlavní stránku.

K formátování, tzn., úpravě vzhledu stránky, byl použit styl, který byl použit i u předchozích stránek.



Obr. 5.35 Konečný vzhled stránky Feedforward s učícím algoritmem Backpropagation - ukázka v prohlížeči Opera

Výše popsané je obsahem souborů **backpropagation.jsp, backpropagationuk.jsp** na přiloženém CD.

ZÁVĚR

Hlavním cílem této diplomové práce bylo vytvořit moduly neuronových sítí s podporou programu webMathematica od společnosti Wolfram Research vhodné jako „výuková“ webová stránka neuronových sítí, která bude moci být využita v laboratořích předmětu Metody umělé inteligence vyučovaném na Fakultě aplikované informatiky Univerzity Tomáše Bati ve Zlíně. Po dohodě s vedoucí práce byly jmenovitě stanoveny neuronové sítě jednoduchý Perceptron, Adaline a Feedforward s učícím algoritmem Backpropagation a dále také přenosové funkce neuronu.

V teoretické části práce je nejdříve stručně popsána historie neuronových sítí a shrnutí základních pojmů, které se jich týkají, jako je neurofyziologický a formální neuron, neuronová síť a její topologie, dále také přenosové funkce neuronu, učení a dělení sítí. Následně byly přiblíženy výše zmíněné neuronové sítě. V další kapitole je popsáno vývojové prostředí Mathematica 7.0 for Students od společnosti Wolfram Research, které bylo použito pro přípravu a odladění funkcí. Následující kapitola seznámí čtenáře s programem webMathematica 3, který přidává interaktivní výpočty a vizualizace na webové stránky integrací programu Mathematica pomocí nejnovějších technologií webového serveru. Jsou zde uvedeny také rozdíly mezi programy Mathematica a webMathematica, inovace a klíčové výhody ve webMathematice 3 a také technologie, které tento program používá při zpracování požadavku a zobrazení stránky. Poslední kapitola teoretické části přibližuje multiplatformní, objektově orientovaný a nejpoužívanější skriptovací jazyk na internetu – JavaScript, a také jeho vlastnosti.

V praktické části je popsána tvorba jednotlivých webových stránek. Přenosové funkce jsou zpracovány tak, aby sloužili jako názorná ukázka. Jednoduchý Perceptron je vypracován tak, že ho je možné využít k řešení problematiky lineární klasifikace prvků ve dvou třídách. Obdobným způsobem je vypracována i síť Adaline, kterou lze také k této problematice použít. Síť Feedforward, neboli síť s dopředným šířením signálu a zpětným šířením chyby, se uplatní v obecné aproximaci funkcí.

Funkcionalita u všech tří sítí je založena na stejném principu. Na každé stránce má uživatel možnost vyzkoušet vlastní data. Jednotlivé sítě obsahují vlastní funkce sloužící k trénování, vybavování sítě a grafickému zobrazení dat.

K samotné tvorbě výsledných jsp souborů bylo použito vývojové prostředí Wolfram Workbench 2.0, nabízené společností Wolfram. Je k tomuto účelu přímo určené a také se ukázalo jako vhodný nástroj pro tvorbu těchto souborů.

Přiložené CD obsahuje text diplomové práce ve formátu PDF, notebooky vytvořené v programu Mathematica a vytvořené soubory jsp v programu Wolfram Workbench 2.0.

CONCLUSION

The main objective of this thesis was to develop neural network modules with support from the webMathematica from the Wolfram Research company, appropriate as an ”educational“ website of neural networks, which will be used in the laboratories of Methods for artificial intelligence course, taught at Faculty of Applied Informatics at Tomas Bata University in Zlin. After an agreement with the leader of this document, the neural networks Perceptron, Adaline and Feedforward with learnt Backpropagation algorithm, and also the transfer functions of neurons, have been specifically established.

The theoretical part of this thesis first briefly describes the history of neural networks, and a summarizes the basic concepts, which concern them, such as neurophysiological and formal neuron, neural network and its topology, also the neuron transfer function, learning and networking division. Subsequently the neural network mentioned above, was sketched. The next chapter describes the development environment of Mathematica 7.0 for Students by the Wolfram Research company, which was used for the preparation and debugging functions. The following chapter will introduce the reader to the webMathematica 3 program, which adds interactive calculations and visualization on the website by integration of the program Mathematica using the latest web server technology. Also presented in this thesis are the differences between Mathematica and webMathematica program, innovation and key advantages in webMathematica 3, and also the technology that the program uses to process the request and page views. The last chapter of the theoretical part, shows more about cross-platform, object-oriented and most widely used scripting language on the internet - JavaScript, and its properties.

The practical part describes the formation of individual websites. The transfer functions are processed to serve as a demonstration. Simple Perceptron is drafted so that it can be used to tackle linear classification of elements in two classes. In a similar way ,Adaline network is also prepared, which can also be applied to this issue. Feedforward network or network of forward spreading of the signal and reverse spreading of the error applies to the general function approximation.

The functionality of all three networks is based on the same principle. On each page the user has the opportunity to try its own data. Individual networks include custom functions used for training, equipping networks and graphic display data. For the very creation of the resulting jsp files a development environment Wolfram Workbench 2.0, offered by

Wolfram company,was used. It is specifically for this purpose, and also it turned out as a good tool for creating these files.

The enclosed CD contains the text of the diploma thesis in PDF format, notebooks created in Mathematica program and jsp files created in the Wolfram Workbench 2.0 program.

SEZNAM POUŽITÉ LITERATURY

[1] MAŘÍK, Vladimír; ŠTĚPÁNKOVÁ, Olga; LAŽANSKÝ, Jiří. Umělá inteligence. Praha : Academia, 2000. 264 s. ISBN 80-200-0496-3.

[2] Science WORLD [online]. 2001 [cit. 2010-02-15]. Dostupný z WWW: <http://scienceworld.cz/technologie/co-jsou-to-umele-neuronove-site-4077>.

[3] Zelinka Ivan. Umělá inteligence I. Volume 1. Zlín: Vutium, Brno, 1998. 126 p. ISBN 80-214-1163-5.

[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] ŠNOREK, Miroslav, JIŘINA, Marcel. Neuronové sítě a neuropočítače. [s.l.] : [s.n.], 1996. 124 s. ISBN 80-01-01455.

[6] BÍLA, Jiří. Umělá inteligence a neuronové sítě v aplikacích. [s.l.] : [s.n.], 1995. 115 s. ISBN 80-01-01275-1.

[7] 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>.

[8] 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>.

[9] Wolfram Mathematica: History & Background [online]. 2009 [cit. 2010-02-26]. History & Background. Dostupné z WWW: <http://www.wolfram.com/products/mathematica/history.html>.

[10] Wolfram Mathematica: Home Page [online]. 2009 [cit. 2010-03-04]. Wolfram Mathematica. Dostupné z WWW: <http://www.wolfram.com/products/mathematica/index.html>.

[11] Wolfram webMathematica 3:What is webMathematica? [online]. 2009 [cit. 2010-03-04]. What is webMathematica?. Dostupné z WWW: <http://www.wolfram.com/products/webmathematica/whatis.html>.

[12] Wolfram webMathematica 3: New Features in webMathematica [online]. 2009 [cit. 2010-03-07]. New Features in webMathematica. Dostupné z WWW: <http://www.wolfram.com/products/webmathematica/newfeatures.html>.

[13] Wolfram webMathematica: Key Advantages [online]. 2009 [cit. 2010-03-07]. Key Advantages. Dostupné z WWW: <http://www.wolfram.com/products/webmathematica/advantages/>.

[14] Wolfram webMathematica:Technology [online]. 2009 [cit. 2010-03-07]. Technology. Dostupné z WWW: <http://www.wolfram.com/products/webmathematica/technology/>.

[15] Wolfram webMathematica: Featured Sites and Interactive Examples [online]. 2009 [cit. 2010-03-08]. Featured Sites and Interactive Examples. Dostupné z WWW: <http://www.wolfram.com/products/webmathematica/examples/>.

[16] ZAKAS, Nicholas Z. . JavaScript pro webové vývojáře. [s.l.] : Computer Press, listopad 2009. 832 s. ISBN 978-80-251-2509-0.

[17] JavaScript and HTML DOM Reference [online]. 2006 [cit. 2010-03-01]. JavaScript and HTML DOM. Dostupné z WWW: <http://www.w3schools.com/jsref/default.asp>.

[18] Oplatkova, Z.: Metaevolution: Synthesis of Optimization Algorithms by means of Symbolic Regression and Evolutionary Algorithms, Lambert Academic Publishing, 154 p., 2009, ISBN: 978-3-8383-1808-0.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

.NET	dotnet
ASP	Active Server Pages
ECMA	Europen Computer Manufacturers Association
GUI	Graphical User Interface
HTML	HyperText Markup Language
ISO	International Organization for Standardization
PHP	Hypertext Preprocessor, původně Personal Home Page
SQL	Structured Query Language
WWW	World Wide Web

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: PŘENOSOVÉ FUNKCE - UKÁZKY KÓDU

Ukázky zdrojového kódu z programu Mathematica:

Vykreslení grafu přenosové funkce logistická sigmoida ve 2D:

```
Manipulate[Plot[1/(1+E^(-\[Lambda]*a)),{a,-100,100},AxesLabel-
>{a,f[a]},PlotLabel->"Logistická sigmoida",PlotStyle-
>Thick,ColorFunction->Function[{x,y},Hue[x]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové funkce logistická sigmoida ve 3D:

```
x1=10;x2=-20;Manipulate[Plot3D[{1-(1/(1+E^(-
\[Lambda]*(w1*x1+w2*x2))))},{w1,-500,500},{w2,-500,500},AxesLabel-
>{w1,w2,"z"},PlotLabel->"Logistická sigmoida",ColorFunction-
>Function[{x,y,z},Hue[y]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové funkce hyperbolický tangens ve 2D:

```
Manipulate[Plot[(E^(\[Lambda]*a)-E^(-\[Lambda]*a))/(E^(\[Lambda]*a)+E^(-
\[Lambda]*a)),{a,-100,100},AxesLabel->{a,f[a]},PlotLabel->"Hyperbolický
tangens",PlotStyle->Thick,ColorFunction-
>Function[{x,y},Hue[x]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové funkce hyperbolický tangens ve 3D:

```
x1=10;x2=-20;Manipulate[Plot3D[{1-(E^(\[Lambda]*(w1*x1+w2*x2))-E^(-
\[Lambda]*(w1*x1+w2*x2)))/(E^(\[Lambda]*(w1*x1+w2*x2))+E^(-
\[Lambda]*(w1*x1+w2*x2)))},{w1,-500,500},{w2,-500,500},AxesLabel-
>{w1,w2,"z"},PlotLabel->"Hyperbolický tangens",ColorFunction-
>Function[{x,y,z},Hue[y]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové skokové funkce ve 2D:

```
Manipulate[Plot[UnitStep[\[Lambda]*a],{a,-100,100},AxesLabel-
>{a,f[a]},PlotLabel->"Skoková funkce",PlotStyle->Thick,ColorFunction-
>Function[{x,y},Hue[x]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové skokové funkce ve 3D:

```
x1=10;x2=-20;Manipulate[Plot3D[{\[Lambda]*(1-
UnitStep[(w1*x1+w2*x2)])},{w1,-500,500},{w2,-500,500},AxesLabel-
>{w1,w2,"z"},PlotLabel->"Skoková funkce",ColorFunction-
>Function[{x,y,z},Hue[x]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové funkce Perceptron ve 2D:

```
vykresleni[a_]:=If[a>=0,a,0];Manipulate[Plot[\[Lambda]*(vykresleni[a]),{a
,-100,100},AxesLabel->{a,f[a]},PlotLabel->"Perceptron",PlotStyle-
>Thick,ColorFunction->Function[{x,y},Hue[x]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové funkce Perceptron ve 3D:

```
vykresleni[a_]:=If[a>=0,a,0];x1=10;x2=-
20;Manipulate[Plot3D[{(\[Lambda]*(vykresleni[1-(w1*x1+w2*x2)]))},{w1,-
500,500},{w2,-500,500},AxesLabel->{w1,w2,"z"},PlotLabel-
>"Perceptron",ColorFunction->Function[{x,y,z},Hue[y]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové lineární funkce ve 2D:

```
Manipulate[Plot[\[Lambda]*a,{a,-100,100},AxesLabel->{a,f[a]},PlotLabel-
>"Lineární funkce",PlotStyle->Thick,ColorFunction-
>Function[{x,y},Hue[x]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové lineární funkce ve 3D:

```
x1=10;x2=-20;Manipulate[Plot3D[{(\[Lambda]*(1-(w1*x1+w2*x2)))},{w1,-
500,500},{w2,-500,500},AxesLabel->{w1,w2,"z"},PlotLabel->"Lineární
funkce",ColorFunction->Function[{x,y,z},Hue[y]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové omezené funkce ve 2D:

```
Manipulate[Plot[Clip[\[Lambda]*a,{-\[Gamma],\[Gamma]}],{a,-
100,100},AxesLabel->{a,f[a]},PlotLabel->"Omezená funkce",PlotStyle-
>Thick,ColorFunction-
>Function[{x,y},Hue[x]]],{\[Lambda],0,1},{\[Gamma],0,1}]
```

Vykreslení grafu přenosové omezené funkce ve 3D:

```
x1=10;x2=-20;Manipulate[Plot3D[{Clip[(\[Lambda]*(1-(w1*x1+w2*x2))),{-
\[Gamma],\[Gamma]}]},{w1,-500,500},{w2,-500,500},AxesLabel-
>{w1,w2,"z"},PlotLabel->"Omezená funkce",ColorFunction-
>Function[{x,y,z},Hue[y]]],{\[Lambda],0,1},{\[Gamma],0,1}]
```

Vykreslení grafu přenosové Gaussovy funkce ve 2D:

```
m=Mean[{-3,3}];\[Sigma]=Variance[{-3,3}];Manipulate[Plot[E^(-
((\[Lambda]*a-m)^2/\[Sigma]^2)),{a,-100,100},AxesLabel-
>{a,f[a]},PlotLabel->"Gaussova funkce",PlotStyle->Thick,ColorFunction-
>Function[{x,y},Hue[x]]],{\[Lambda],0,1}]
```

Vykreslení grafu přenosové Gaussovy funkce ve 3D:

```
x1=10;x2=-20;m=Mean[{10,-20}];\[Sigma]=Variance[{10,-
20}];Manipulate[Plot3D[{E^(-((\[Lambda]*(w1*x1+w2*x2)-
m)^2/\[Sigma]^2))},{w1,-500,500},{w2,-500,500},AxesLabel-
>{w1,w2,"z"},PlotLabel->"Gaussova funkce",ColorFunction-
>Function[{x,y,z},Hue[y]]],{\[Lambda],0,1}]
```

Ukázky zdrojového kódu z programu webMathematica:

Inicializování příslušných knihoven funkcí.

```
<msp:evaluate>
      Needs["MSPManipulate`"]
      Needs["MSP`"]
      SetSecurity[];
</msp:evaluate>
```

Předdefinování *MSPManipulate*:

```
<msp:evaluate>
      MSPManipulateHeader[$$updateArgs, $$manipulateNumber]
</msp:evaluate>
```

Formulář v HTML jazyce:

```
<form action="prenosfunkce.jsp" method="post">
```

Ukázka výsledného kódu první fázi pro přenosovou funkci logistická sigmoida ve 2D:

```
<div class="manipulate">
      <center><msp:evaluate>
            MSPManipulate[ Plot[1/(1 + Exp[(-lambda*a)]), {a, -100, 100},
AxesLabel -> {a, f[a]}, PlotLabel-> "Logistická sigmoida", PlotStyle ->
Thick, ColorFunction -> Function[{x, y}, Hue[x]]], {lambda, 0, 1},
OutputSize->{600, 370}]
      </msp:evaluate></center>
</div>
```

Script pro skrývání:

```
<script>
      function zobrazSkryj(idecko){
      el=document.getElementById(idecko).style;
      el.display=(el.display == 'block')?'none':'block';
      }
</script>
```

```
<style>
      h3 {cursor: pointer; cursor: hand;}
      .skryvany {display: none}</style>
<style>
            a:link, a:visited    {text-decoration: none}
</style>
```

Ukázka kódu pro buttonek:

```
<br /><img src="2D_button.gif" alt="2D" onclick="zobrazSkryj('logsim1')"
/>
```

Ukázka kódu pro položku seznamu - pro přenosovou funkci logistická sigmoida:

```
<li class="toclevel-1 tocsection-1"><a href="#log_sim"><span
class="tocnumber"></span> <span class="toctext">Logistická
sigmoida</span></a></li>
```

Ukázka tagu odkazu na záložku:

```
<h2 id="log_sim">Logistická sigmoida</h2>
```

Ukázka tagu odkazu na záložku u nadpisu stránky:

```
<h1 id="zacatek">Přenosové funkce</h1>
```

Tag odkazu na onu záložku:

```
<em><a href="#zacatek">Na začátek stránky</a></em>
```

PŘÍLOHA P II: PERCEPTRON - UKÁZKY KÓDU

Ukázky zdrojového kódu z programu Mathematica:

Ukázka kódu nadefinování vstupních proměnných pro trénování sítě:

```
Tplus={{1,1},{1,-1},{0,-1}};
Tminus={{-1,-1},{-1,1},{0,1}};
maxEpoch=5;
wahy={1,0,0};
```

Ukázka výsledného kódu zajištujícího učení sítě:

```
historievah={};vaha2={};zmenilysevahy={};globchyb=1;epocha=1;i=0;souradni
cey={};vektor={};matice={};konec=100;souradnicex={};sou={};
Table[If[globchyb==0,konec=epocha,If[epocha>maxEpoch,konec=maxEpoch,Print
[epocha,". epocha"];Print[i," i1"];
    Table[Print[i,"
i2"];ukazatelzmenyvah=w.TPZ[[i]];Print[ukazatelzmenyvah," ukazatel"];
If[ukazatelzmenyvah==0,w=w+TPZ[[i]];zmenilysevahy=Join[zmenilysevahy,{1}]
,w=w;zmenilysevahy=Join[zmenilysevahy,{0}]];
     Print[zmenilysevahy," zmenavah1"];
vaha2=Join[vaha2,{w[[2]]}];historievah=Join[historievah,{w}];vektor=Join[
vektor,{w}];Print[vektor," vektor1"];Print[historievah," historie"];
     If[i== pocetbodu,globchyb=Total[zmenilysevahy];Print[globchyb,"
globchyb"];souradnicey=Join[souradnicey,{globchyb}];Print[zmenilysevahy,"
zmenavah2"];zmenilysevahy={};Print[zmenilysevahy,"
zmenavah3"],],{i,pocetbodu}];
    Print[vektor," vektor2"];matice=Join[matice,{vektor}];vektor={};
Print[vektor," vektor3"];Print[matice," matice"];
    Print[globchyb," globchyb"];Print[epocha,". epocha"];souradnicex=
Join[souradnicex,{epocha}];epocha++]],{konec}];
```

Ukázka kódu vykreslení trénovací množinu rozdělenou do plusové a minusové třídy:

```
graf=ListPlot[{Tplus,Tminus},PlotLabel->"Vykresleni trenovaci
mnoziny",PlotMarkers->Automatic,PlotStyle->{Green,Red}]
```

Ošetření různých případů, které mohou nastat při vykreslování hranice:

```
hranice=If[w[[2]]!=0,If[w[[1]]==0,Plot[-w[[3]]/w[[2]],{x1,-
1,1},PlotStyle->Thick,ColorFunction->Function[{x,y},Hue[x]]],Plot[(((-
x1*w[[1]])-(x3*w[[3]]))/(w[[2]])),{x1,-1,1},PlotStyle-
>Thick,ColorFunction-
>Function[{x,y},Hue[y]]]],If[w[[1]]==0,ListPlot[{0,0}],ListPlot[{{-
```

```
w[[3]]/w[[1]],-maxxim},{-w[[3]]/w[[1]],maxxim}},Joined->True,PlotStyle-
>Thick,ColorFunction->Function[{x,y},Hue[y]]]];
```

Ukázka kódu vykreslení grafu vývoje globální chyby:

```
vyvoj=ListPlot[ Table[Transpose[sou],{i,
Dimensions[sou][[1]]}],PlotRange->All,PlotJoined->True,Axes-
>False,PlotLabel->StyleForm["Vývoj globální chyby",FontSize-
>14,FontColor->RGBColor[0,0.35,0.6]],TextStyle->{FontWeight-
>"Bold",FontColor->RGBColor[0,0,0.6],FontSize->12},Frame-
>True,RotateLabel->False,FrameLabel->{"Epocha","Hodnota globální
chyby",""},ImageSize->500]
```

Ukázka kódu nadefinování vstupních proměnných pro testování sítě:

```
testmnozina = {{0.5, 0.4}, {-0.6, 1}, {-0.7, -0.8}, {0.3, -0.4}, {1,
0.9}, {-1, 0.3}, {0.6, -0.7}, {0.5, -1.1}, {-0.3, 0.2}};

predpoklad={1,-1,-1,1,1,-1,1,1,-1};
```

Ukázka výsledného kódu testování sítě:

```
kporovnani={};testovanekladne={};testovanezaporne={};vyhodnoceni={};
testmnozinaP=Table[Join[testmnozina[[i]],{1}],{i,Length[testmnozina]}];
pocettestbodu=Length[testmnozinaP];

Table[vysledky=w.testmnozinaP[[i]];If[vysledky>0,testovanekladne=Join[tes
tovanekladne,{testmnozina[[i]]}];Print[testovanekladne,"
testovanekladne"];kporovnani=Join[kporovnani,{1}];Print[kporovnani,"
kporovnani"],

testovanezaporne=Join[testovanezaporne,{testmnozina[[i]]}];Print[testovan
ezaporne,"testovanezaporne"];kporovnani=Join[kporovnani,{-
1}];Print[kporovnani," kporovnani"]];

If[predpoklad[[i]]==kporovnani[[i]],vyhodnoceni=Join[vyhodnoceni,{splnil}
],vyhodnoceni=Join[vyhodnoceni,{nesplnil}]];Print[vyhodnoceni,"vyhodnocen
i"]

,{i,pocettestbodu}];
```

Ukázka kódu pro grafické znázornění výsledků:

```
testgraf=ListPlot[{testovanekladne,testovanezaporne},PlotLabel-
>"Vykresleni testovaci mnoziny",PlotMarkers->Automatic,PlotStyle-
>{Green,Red}]

testvystup=Show[testgraf,hranice]
```

Ukázky zdrojového kódu z programu webMathematica:

Ukázka kódu na odkazy:

```
<a href="perceptronlinsep.jsp">Perceptron - ukázka lineárně separabilní >></a><br />
<a href="perceptronlinnesep.jsp">Perceptron - ukázka lineárně neseparabilní >></a><br />
```

Ukázka části kódu pro zadávání hodnot:

```
<h2>Zadávání hodnot</h2>

<table align="left" border="0" cellpadding="3" cellspacing="3" >
	<tr>
		<td style="width:300px;"><b>Trénovací množina</b></td>
		<td style="width:700px;"><b></b></td>
	</tr>
	<tr>
	</tr>
	<tr>
		<td style="width:300px;">Tplus</td>
		<td style="width:700px;"><input type="text" name="varTplus" value="<msp:evaluate>MSPValue[ $$varTplus, "{{1, 1}, {1, -1}, {0, -1}}"]</msp:evaluate>" style="width: 520px;" /></td>
	</tr>
	<tr>
		<td style="width:300px;">Tminus</td>
		<td style="width:700px;"><input type="text" name="varTminus" value="<msp:evaluate>MSPValue[ $$varTminus, "{{-1, -1}, {-1, 1}, {0, 1}}"]</msp:evaluate>" style="width: 520px;" /></td>
	</tr>
```

Ukázka kódu přiřazení hodnot z políček do proměnných, se kterými pracuje funkce *DoPerceptron*.

```
<msp:evaluate>
	Tplus=If[ MSPValueQ[$$varTplus],MSPToExpression[$$varTplus]];
	Tminus=If[ MSPValueQ[$$varTminus],MSPToExpression[$$varTminus]];
	wahy=If[ MSPValueQ[$$varwahy],MSPToExpression[$$varwahy]];
	const=If[ MSPValueQ[$$varConst],MSPToExpression[$$varConst]];
	TestMn=If[ MSPValueQ[$$varTestmnozina],MSPToExpression[$$varTestmnozina]];
	predp=If[ MSPValueQ[$$varPredpoklad],MSPToExpression[$$varPredpoklad]];
```

```
      maxE=If[ MSPValueQ[$$varMaxEpoch],MSPToExpression[$$varMaxEpoch]];
</msp:evaluate>
```

Kód pro tlačítka:

```
<br /><input type="submit" name="submitButton" value="Proveď" />
<br />
<br /><img src="zobraz.gif" alt="Zobraz" onclick="zobrazSkryj('grafy')"/>
```

Provedení funkce *DoPerceptron*:

```
<msp:evaluate>
      Perceptron = DoPerceptron[Tplus, Tminus, const, wahy, TestMn, maxE,
predp];
</msp:evaluate>
```

vykreslení grafu trénovací množiny s hranicí:

```
<msp:evaluate>
      graf=ListPlot[{Tplus,Tminus},PlotLabel -> "Vykresleni trenovaci
mnoziny",PlotMarkers->Automatic, PlotStyle -> {Green, Red}];
</msp:evaluate>
<br />
<msp:evaluate>
      zkouska=Show[graf,hranice];
      HTMLTableForm[MSPShow[zkouska],TableAttributes->{"cellpadding" ->
"0", "cellspacing" -> "0","align"->"center"}]
</msp:evaluate>
```

Kód pro vykreslení grafu testovací množiny, výsledků testovaných bodů a graf vývoje globální chyby:

```
<msp:evaluate>
      testgraf =  ListPlot[{testovanekladne, testovanezaporne}, PlotLabel
-> "Vykresleni testovaci mnoziny", PlotMarkers -> Automatic,
   PlotStyle -> {Green, Red}];
</msp:evaluate>
<br />
<msp:evaluate>
      testvystup = Show[testgraf, hranice];
      HTMLTableForm[MSPShow[testvystup],TableAttributes->{"cellpadding" -
> "0", "cellspacing" -> "0","align"->"center"}]
</msp:evaluate>

<msp:evaluate>
```

```
      HTMLTableForm[radky,TableHeadings->{{},{"bod", "predpoklad",
      "skutecne", "ne/splneno"}},TableAttributes->{"border" ->
      "1","cellspacing"->"3","cellpadding"->"3","bgcolor"-
      >"#C9C9C9","align"->"center"}]
</msp:evaluate>

<msp:evaluate>
      vyvoj = ListPlot[ Table[Transpose[sou], {i, Dimensions[sou][[1]]}],
PlotRange -> All, PlotJoined -> True, Axes -> False,
         PlotStyle -> Thick, PlotLabel -> StyleForm["Vývoj globální
chyby", FontSize -> 14, FontColor -> RGBColor[0, 0.25, 0.6]],
        TextStyle -> {FontWeight -> "Bold", FontColor -> RGBColor[0, 0,
0.6], FontSize -> 12}, Frame -> True,
        RotateLabel -> False, FrameLabel -> {"Epocha", "Hodnota globální
chyby", ""}, ImageSize -> 500];
  </msp:evaluate>
<br />
<msp:evaluate>
      HTMLTableForm[MSPShow[vyvoj],TableAttributes->{"cellpadding" ->
"0", "cellspacing" -> "0","align"->"center"}]
</msp:evaluate>
```

Odkazy na začátek stránky, na hlavní stránku Perceptron i na podstránku s nelineárně separabilním problémem:

```
<em><a href="#zacatek">Na začátek stránky</a></em>
<a href="perceptron.jsp">Perceptron >></a><br />
<a href="perceptronlinnesep.jsp">Perceptron - ukázka nelineárně
separabilní >></a><br />
```

PŘÍLOHA P III: ADALINE - UKÁZKY KÓDU

Ukázky zdrojového kódu z programu Mathematica:

Ukázka kódu nadefinování vstupních proměnných pro trénování sítě:

```
TMN={{1,1,1},{1,-1,1},{0,-1,1},{-1,-1,-1},{-1,1,-1},{0,1,-1}};

maxEpoch=3;
wahy={1,0,0,0};w=wahy;
b=1; prah=b;
ucicikoef=0.1;
predpokladtrida={1,1,1,-1,-1,-1};
```

Ukázka výsledného kódu zajišťujícího učení sítě:

```
historievah={};vaha2={};zmenilysevahy={};globchyb=1;epocha=1;i=0;souradni
cey={};vektor={};matice={};konec=100;souradnicex={};sou={};lokalchyba=0;a
ktivacnifunkce=0;lokal=0;aktvfun={};pocetepoch={};
Tplus={};Tminus={};Table[If[predpokladtrida[[i]]==1,Tplus=Join[Tplus,{TMN
[[i]]}],Tminus=Join[Tminus,{TMN[[i]]}]],{i,Length[TMN]}];Print[Tplus,"tpl
us"];Print[Tminus,"tminus"];
vypis={{"prvek","predpokladana trida","aktivacni funkce","vahy"}};
Table[

If[globchyb==0,konec=epocha,If[epocha>maxEpoch,konec=maxEpoch,Print[epoch
a,". epocha"];Print[i," i1"];
    Table[Print[i,"
i2"];aktivacnifunkce=TRM[[i]].w;Print[aktivacnifunkce,"
aktivacnifunkce"];
     aktvfun=Join[aktvfun,{aktivacnifunkce}];Print[aktvfun,"aktvfun"];
     w=w+ucicikoef*(predpokladtrida[[i]]-aktivacnifunkce)*TRM[[i]];

vypis=Join[vypis,{{TMN[[i]],predpokladtrida[[i]],aktivacnifunkce,w}}];
     lokal=Abs[predpokladtrida[[i]]-aktivacnifunkce];Print[lokal,"
lokal"];lokalchyba=Join[lokalchyba,lokal];Print[lokalchyba,"
lokalchyba"];

vaha2=Join[vaha2,{w[[2]]}];historievah=Join[historievah,{w}];vektor=Join[
vektor,{w}];Print[vektor," vektor1"];Print[historievah," historie"];
     If[i==pocetbodu,globchyb=Total[lokalchyba];Print[globchyb,"
globchyb"];souradnicey=Join[souradnicey,{globchyb}];Print[lokalchyba,"
```

```
lokalchyba2"];lokalchyba=0;Print[lokalchyba,"
lokalchyba3"]],{i,pocetbodu}];
    Print[vektor,"
vektor2"];matice=Join[matice,{vektor}];vektor={};Print[vektor,"
vektor3"];Print[matice," matice"];
    Print[globchyb," globchyb2"];Print[epocha,".
epocha"];souradnicex=Join[souradnicex,{epocha}];
    Print[Grid[vypis,Frame→All]];
    vypis={{"prvek","predpokladana trida","aktivacni
funkce","vahy"}};pocetepoch=Join[pocetepoch,{epocha}];Print[pocetepoch,"p
ocetepoch"];epocha++
    ]
   ],{konec}];
```

Ukázka kódu vykreslení trénovací množinu rozdělenou do plusové a minusové třídy:

```
Print[graf=ListPointPlot3D[{Tplus,Tminus},PlotLabel->"Vykreslení
trénovací množiny",PlotStyle-
>{{Green,PointSize[0.03]},{Red,PointSize[0.03]}}]];
```

Ošetření různých případů, které mohou nastat při vykreslování hranice:

```
vx={0,1,2};

vy={4,-1,3};

vz=Table[-(w[[1]]*vx[[i]]+w[[2]]*vy[[i]]+w[[4]])/w[[3]],{i,3}];

v1={vx[[2]]-vx[[1]],vy[[2]]-vy[[1]],vz[[2]]-vz[[1]]};

v2={vx[[3]]-vx[[1]],vy[[3]]-vy[[1]],vz[[3]]-vz[[1]]};

Show[ListPointPlot3D[{Tplus,Tminus},PlotLabel->"Vykreslení trénovací
množiny",PlotStyle->{{Green,PointSize[0.03]},{Red,PointSize[0.03]}}],

ParametricPlot3D[{vx[[1]]+v1[[1]]*t+v2[[1]]*s,

vy[[1]]+v1[[2]]*t+v2[[2]]*s,vz[[1]]+v1[[3]]*t+v2[[3]]*s},{s,-5,5},{t,-
5,5},PlotStyle-
>Directive[Purple,Opacity[0.6],Specularity[White,25]],Mesh-
>None],AxesLabel->{"x","y","z"},PlotRange->{{-2,2},{-2,2},{-2,2}}]
```

Ukázka kódu vykreslení grafu vývoje globální chyby:

```
vyvoj=ListPlot[ Table[Transpose[sou],{i,
Dimensions[sou][[1]]}],PlotRange->All,PlotJoined->True,Axes-
>False,PlotLabel->StyleForm["Vývoj globální chyby",FontSize-
>14,FontColor->RGBColor[0,0.35,0.6]],TextStyle->{FontWeight-
>"Bold",FontColor->RGBColor[0,0,0.6],FontSize->12},Frame-
```

```
>True,RotateLabel->False,FrameLabel->{"Epocha","Hodnota globální
chyby",""},ImageSize->500]
```

Ukázka kódu nadefinování vstupních proměnných pro testování sítě:

```
testmnozina={{0.5,0.4,0.3},{-0.6,1,1.3},{-0.7,-0.8,-0.6},{0.3,-
0.4,0.2},{1,0.9,0.7},{-1,0.3,1.1},{0.6,-0.7,-0.4},{0.5,-1.1,0.8},{-
0.3,0.2,0.5}};
predpoklad={1,1,-1,1,1,1,-1,1,1};
```

Ukázka výsledného kódu testování sítě:

```
kporovnani={};testovanekladne={};testovanezaporne={};vyhodnoceni={};prah=
1;
testmnozinaP=Table[Join[testmnozina[[i]],{prah}],{i,Length[testmnozina]}]
;
pocettestbodu=Length[testmnozinaP];
Table[vysledky=w.testmnozinaP[[i]];

If[vysledky>=0,testovanekladne=Join[testovanekladne,{testmnozina[[i]]}];P
rint[testovanekladne,"
testovanekladne"];kporovnani=Join[kporovnani,{1}];Print[kporovnani,"
kporovnani"],

testovanezaporne=Join[testovanezaporne,{testmnozina[[i]]}];Print[testovan
ezaporne," testovanezaporne"];kporovnani=Join[kporovnani,{-
1}];Print[kporovnani," kporovnani"]];

If[predpoklad[[i]]==kporovnani[[i]],vyhodnoceni=Join[vyhodnoceni,{splnil}
],vyhodnoceni=Join[vyhodnoceni,{nesplnil}]];Print[vyhodnoceni,"vyhodnocen
i"]
  ,{i,pocettestbodu}];
```

Vykreslení grafu testovací množiny:

```
vx={0,1,2};

vy={4,-1,3};

vz=Table[-(w[[1]]*vx[[i]]+w[[2]]*vy[[i]]+w[[4]])/w[[3]],{i,3}];

v1={vx[[2]]-vx[[1]],vy[[2]]-vy[[1]],vz[[2]]-vz[[1]]};

v2={vx[[3]]-vx[[1]],vy[[3]]-vy[[1]],vz[[3]]-
vz[[1]]};Print[Show[ListPointPlot3D[{testovanekladne,testovanezaporne},Pl
otLabel->"Vykreslení testovací mnoziny",PlotStyle-
>{{Green,PointSize[0.03]},{Red,PointSize[0.03]}}],ParametricPlot3D[{vx[[1
```

```
]]+v1[[1]]*t+v2[[1]]*s,vy[[1]]+v1[[2]]*t+v2[[2]]*s,vz[[1]]+v1[[3]]*t+v2[[
3]]*s},{s,-5,5},{t,-5,5},PlotStyle-
>Directive[Purple,Opacity[0.6],Specularity[White,25]],Mesh-
>None],AxesLabel->{"x","y","z"},PlotRange->{{-2,2},{-2,2},{-2,2}}]]
testvystup=Show[testgraf,hranice]
```

Ukázky zdrojového kódu z programu webMathematica:

Kód na odkazy:

```
<a href="adalinelinsep.jsp">Adaline - ukázka lineárně separabilní
>></a><br />
<a  href="adalinenelinsep.jsp">Adaline  -  ukázka  nelineárně  separabilní
>></a><br />
```

Ukázka části kódu pro zadávání hodnot:

```
<h2>Zadávání hodnot</h2>

<table align="left" border="0" cellpadding="3" cellspacing="3" >
      <tr>
            <td style="width:300px;"><b>Trénovací množina</b></td>
            <td style="width:700px;"><b></b></td>
      </tr>
      <tr>
      </tr>
      <tr>
            <td style="width:300px;">trénovací body</td>
            <td style="width:700px;"><input type="text"
name="varTrenmnozina" value="<msp:evaluate>MSPValue[ $$varTrenmnozina,
"{{1, 1, 1}, {1, -1, 1}, {0, -1, 1},{-1, -1, -1}, {-1, 1, -1}, {0, 1, -
1}}"]</msp:evaluate>" style="width: 520px;" /></td>
      </tr>
      <tr>
            <td style="width:300px;">předpoklad umístění</td>
            <td style="width:700px;"><input type="text"
name="varTrenPredp" value="<msp:evaluate>MSPValue[ $$varTrenPredp, "{1,
1, 1,-1, -1, -1}"]</msp:evaluate>" style="width: 520px;" /></td>
      </tr>
```

Ukázka kódu přiřazení hodnot z políček do proměnných, se kterými pracuje funkce *DoAdaline*:

```
<msp:evaluate>
```

```
      TrenMn=If[
MSPValueQ[$$varTrenmnozina],MSPToExpression[$$varTrenmnozina]];
      trenPredp=If[
MSPValueQ[$$varTrenPredp],MSPToExpression[$$varTrenPredp]];
      wahy=If[ MSPValueQ[$$varwahy],MSPToExpression[$$varwahy]];
      const=If[ MSPValueQ[$$varConst],MSPToExpression[$$varConst]];
      TestMn=If[
MSPValueQ[$$varTestmnozina],MSPToExpression[$$varTestmnozina]];
      predp=If[
MSPValueQ[$$varPredpoklad],MSPToExpression[$$varPredpoklad]];
      maxE=If[ MSPValueQ[$$varMaxEpoch],MSPToExpression[$$varMaxEpoch]];
      uckoef=If[ MSPValueQ[$$varUcKoef],MSPToExpression[$$varUcKoef]];
</msp:evaluate>
```

Kód pro tlačítka:

```
<br /><input type="submit" name="submitButton" value="Proveď" / >
<br />
<br /><img src="zobraz.gif" alt="Zobraz"
onclick="zobrazSkryj('grafiky')"/>
```

Provedení funkce *DoAdaline*:

```
<msp:evaluate>
      Adaline = DoAdaline[TrenMn, trenPredp, const, wahy, TestMn, maxE,
predp, uckoef];
</msp:evaluate>
```

Vykreslení grafu trénovací množiny s hranicí:

```
<msp:evaluate>
      graf = Show[ListPointPlot3D[{Tplus, Tminus}, PlotLabel ->
"Vykresleni trenovaci mnoziny", PlotStyle -> {{Green, PointSize[0.03]},
{Red, PointSize[0.03]}}],
       ParametricPlot3D[{vx[[1]] + v1[[1]]*t + v2[[1]]*s, vy[[1]] +
v1[[2]]*t + v2[[2]]*s, vz[[1]] + v1[[3]]*t + v2[[3]]*s}, {s, -5, 5}, {t,
-5, 5},
        PlotStyle -> Directive[Purple, Opacity[0.6], Specularity[White,
25]], Mesh -> None], AxesLabel -> {"x", "y", "z"}, PlotRange -> {{-2, 2},
{-2, 2}, {-2, 2}}];
 </msp:evaluate>
<br />
<msp:evaluate>
```

```
      HTMLTableForm[MSPShow[graf],TableAttributes->{"cellpadding" -> "0",
"cellspacing" -> "0","align"->"center"}]
</msp:evaluate>
```

Výpis historie vah a konečných vah:

```
<h4>Historie vah</h4>
<msp:evaluate>
HTMLTableForm[radkytestmn,TableHeadings->{{},{"epocha", "bod",
"predpokladana trida", "aktivacni funkce", "vahy"}},TableAttributes-
>{"border" -> "1","cellspacing"->"3","cellpadding"->"3","bgcolor"-
>"#C9C9C9","align"->"center"}]
</msp:evaluate>
<h4>Konečné váhy</h4>
<msp:evaluate>
      HTMLTableForm[{konecvahy},TableAttributes->{"border" ->
"1","cellspacing"->"3","cellpadding"->"3","bgcolor"->"#C9C9C9","align"-
>"center"}]
</msp:evaluate>
```

vykreslení grafu testovací množiny, výsledků testovaných bodů a graf vývoje globální chyby:

```
<msp:evaluate>
      testgraf = Show[ListPointPlot3D[{testovanekladne,
testovanezaporne}, PlotLabel -> "Vykreslení testovací mnoziny", PlotStyle
-> {{Green, PointSize[0.03]}, {Red, PointSize[0.03]}}],
                        ParametricPlot3D[{vx[[1]] + v1[[1]]*t +
v2[[1]]*s, vy[[1]] + v1[[2]]*t + v2[[2]]*s, vz[[1]] + v1[[3]]*t +
v2[[3]]*s}, {s, -5, 5}, {t, -5, 5},  PlotStyle ->
                        Directive[Purple, Opacity[0.6],
Specularity[White, 25]], Mesh -> None], AxesLabel -> {"x", "y", "z"},
PlotRange -> {{-2, 2}, {-2, 2}, {-2, 2}}];
</msp:evaluate>
<br />

<msp:evaluate>
      HTMLTableForm[MSPShow[testgraf],TableAttributes->{"cellpadding" ->
"0", "cellspacing" -> "0","align"->"center"}]
</msp:evaluate>

<msp:evaluate>
```

```
HTMLTableForm[radky,TableHeadings->{{},{"bod", "predpoklad", "skutecne",
"ne/splneno"}},TableAttributes->{"border" -> "1","cellspacing"-
>"3","cellpadding"->"3","bgcolor"->"#C9C9C9","align"->"center"}]
</msp:evaluate>
<msp:evaluate>
      vyvoj = ListPlot[ Table[Transpose[sou], {i, Dimensions[sou][[1]]}],
PlotRange -> All, PlotJoined -> True, Axes -> False,
         PlotStyle -> Thick, PlotLabel -> StyleForm["Vývoj globální
chyby", FontSize -> 14, FontColor -> RGBColor[0, 0.25, 0.6]],
        TextStyle -> {FontWeight -> "Bold", FontColor -> RGBColor[0, 0,
0.6], FontSize -> 12}, Frame -> True,
        RotateLabel -> False, FrameLabel -> {"Epocha", "Hodnota globální
chyby", ""}, ImageSize -> 500];
  </msp:evaluate>
<br />

<msp:evaluate>
      HTMLTableForm[MSPShow[vyvoj],TableAttributes->{"cellpadding" ->
"0", "cellspacing" -> "0","align"->"center"}]
</msp:evaluate>
```

Odkazy na začátek stránky, na hlavní stránku Adaline i na podstránku s nelineárně separabilním problémem:

```
<em><a href="#zacatek">Na začátek stránky</a></em>
<a href="adaline.jsp">Adaline >></a><br />
<a  href="adalinenelinsep.jsp">Adaline  -  ukázka  lineárně  neseparabilní
>></a><br />
```

PŘÍLOHA P IV: FEEDFORWARD - UKÁZKY KÓDU

Ukázky zdrojového kódu z programu Mathematica:

Funkce *epocha*:

```
epocha[vahy_]:=Module[{},iterator=1;vysledek=Nest[Uceni3,{vstupniVektor,vahy,neuronfce,blbost},Length[vstupniVektor]]; Return[vysledek[[2]]]]
```

Funkce *Uceni3:*

```
Uceni3[{vstupni_,w2_,neuronfce_,blbost_}]:=Module[{},

{vstuprozsireny,vystupsit}=vstupni[[iterator]];


vystupZeSite=neuronfce[[2]][sumace[Join[neuronfce[[1]][sumace[vstuprozsireny,#]&/@w2[[1]]],{1.}],#]&/@w2[[2]]][[1]];

vystupZeSkryte=Join[neuronfce[[1]][sumace[vstuprozsireny,#]&/@w2[[1]]],{1.}];


deltaNula=(vystupsit[[1]]-vystupZeSite);

deltaWahNula=eta*deltaNula*vystupZeSkryte+mi*deltaWahNulaMinule;

deltaWahNulaMinule=deltaWahNula;

vahyNulaNew=w2[[2,1]]+deltaWahNula;

pomocne=Join[{#}&/@Most[vystupZeSkryte],{#}&/@Most[w2[[2,1]]],2];

deltaJedna=#[[1]]*(1-#[[1]])*#[[2]]*deltaNula&/@pomocne;


deltaWahJedna=((eta*#*vstuprozsireny)&/@deltaJedna)+mi*deltaWahJednaMinule;

deltaWahJednaMinule=deltaWahJedna;

vahyJednaNew=w2[[1]]+deltaWahJedna;

wNew={vahyJednaNew,{vahyNulaNew}};

iterator++;

Return[{vstupni,wNew,neuronfce,vstuprozsireny}]]
```

Funkce *DejVystup*:

```
DejVystup[vstuprozsireny_,w2_,neuronfce_]:=Module[{},vystupZeSite=neuronf
ce[[2]][sumace[Join[neuronfce[[1]][sumace[vstuprozsireny,#]&/@w2[[1]]],{1
}],#]&/@w2[[2]]];

Return[vystupZeSite]]
```

Funkce *vyplodvahy:*

```
vyplodvahy[pocetvstupu_,skryte_,pocetvystupu_]:=Module[{},pocty=Flatten[{
pocetvstupu,skryte,pocetvystupu}];vahy=Table[Table[Table[RandomReal[{-
1,1}],{i,pocty[[k]]+1}],

{j,pocty[[k+1]]}],{k,Length[pocty]-1}]; Return[vahy]]
```

Nastavení funkce ve skryté vrstvě a na výstupu neuronu:

```
 sigmoida[suma_]:=1/(1+Exp[-suma])

linear[suma_]:=suma
```

Připravení souřadnic pro požadovaný průběh funkcí:

```
 vstup=Table[{i},{i,-1,1,0.1}]

vystup=Sin[vstup]

vystup=Cos[vstup]

fce[x_]:=x^6-2 x^4+x^2

x^5-2 x^3+x;

vystup=fce[vstup]

vystup=x^6-2 x^4+x^2/.x-> vstup
```

Definování:

```
Iterator = 1;

eta = 0.3;

mi = 0;

skryte = {3};

deltaWahNulaMinule = Table[0,{skryte[[1]]+1}];

deltaWahJednaMinule = Table[{0,0},{skryte[[1]]}];

pocetepoch = 8000;

neuronfce = {sigmoida,linear}

vahy2 = vyplodvahy[1,skryte,1]
```

Vykreslení grafu požadovaného průběhu funkce:

```
puvodni=ListPlot[Join[vstup,vystup,2],PlotMarkers->Automatic]
```

Získání hodnot pro vykreslení vypočteného průběhu funkce:

```
vysledkygraf=DejVystup[#,Last[final],neuronfce]&/@vstuproz
```

Vykreslení grafu funkce aproximované neuronovou sítí vůči požadovaným hodnotám:

```
vysledny=Show[puvodni,ListPlot[Join[vstup,vysledkygraf,2],Joined->True]]
```

Ukázky zdrojového kódu z programu webMathematica:

Kód na odkaz:

```
<a href="backpropagationuk.jsp">Feedforward s učícím algoritmem Backpropagation - ukázka >></a><br />
```

Kód přípravy proměnných:

```
<msp:evaluate>
vstup = Table[{i}, {i, 0, 10, 0.5}];
vystup = switchfunction[ToExpression[$$varVystup]];
vstuproz = Table[Join[vstup[[i]], {1}], {i, Length[vstup]}] // N;
vstupniVektor = {Take[#, 2], {Last[#]}} & /@ Join[vstuproz, vystup, 2];
iterator = 1;
neuronfce = {sigmoida, linear};
</msp:evaluate>
```

Kód pro přiřazení hodnoty z políček do proměnných, se kterými pracují vytvořené funkce:

```
<msp:evaluate>
      skryte1={If[
MSPValueQ[$$varSkryteNeurony],MSPToExpression[$$varSkryteNeurony]]};
      pocetepoch=If[ MSPValueQ[$$varPocet],MSPToExpression[$$varPocet]];
      eta=If[ MSPValueQ[$$varEta],MSPToExpression[$$varEta]];
      mi=If[ MSPValueQ[$$varMi],MSPToExpression[$$varMi]];
</msp:evaluate>
```

Kód pro výpočet počátečních vah a provedení konečného výpočtu:

```
<msp:evaluate>
      vahy2=vyplodvahy[1,skryte1,1];
      final = NestList[epocha, vahy2, pocetepoch];
</msp:evaluate>
```

Kód pro vykreslení vývoje grafů:

```
<h4>Vykreslení vývoje grafů</h4>

<h5>První</h5>

<msp:evaluate>
      puvodni = ListPlot[Join[vstup, vystup, 2], PlotMarkers ->
Automatic];
      vysledkygraf = DejVystup[#, First[final], neuronfce] & /@ vstuproz;
      vysledny = ListPlot[Join[vstup, vysledkygraf, 2], Joined -> True];
      dohromady=Show[puvodni,vysledny];
</msp:evaluate>
<msp:evaluate>
      HTMLTableForm[MSPShow[dohromady],TableAttributes->{"cellpadding" ->
"0", "cellspacing" -> "0","align"->"center"}]
</msp:evaluate>
<br />
<h5>Stý od konce</h5>
<br />
<msp:evaluate>
      puvodni = ListPlot[Join[vstup, vystup, 2], PlotMarkers ->
Automatic];
      vysledkygraf = DejVystup[#, final[[-100]], neuronfce] & /@
vstuproz;
      vysledny = ListPlot[Join[vstup, vysledkygraf, 2], Joined -> True];
      dohromady=Show[puvodni,vysledny];
</msp:evaluate>
<msp:evaluate>
      HTMLTableForm[MSPShow[dohromady],TableAttributes->{"cellpadding" ->
"0", "cellspacing" -> "0","align"->"center"}]
</msp:evaluate>
<br />
<br />pozn. pro příklad z historie grafů mezi prvním a posledním byl
zvolen stý od konce
<br />
<h5>Poslední</h5>
<br />
<msp:evaluate>
      puvodni = ListPlot[Join[vstup, vystup, 2], PlotMarkers ->
Automatic];
```

```
      vysledkygraf = DejVystup[#, Last[final], neuronfce] & /@ vstuproz;
      vysledny = ListPlot[Join[vstup, vysledkygraf, 2], Joined -> True];
      dohromady=Show[puvodni,vysledny];
</msp:evaluate>
<msp:evaluate>
      HTMLTableForm[MSPShow[dohromady],TableAttributes->{"cellpadding" ->
"0", "cellspacing" -> "0","align"->"center"}]
</msp:evaluate>
```