

Srovnání dostupných implementací umělých neuronových sítí

Bc. Petr Juráček

Diplomová práce
2016



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Petr Juráček**
Osobní číslo: **A14842**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Informační technologie**
Forma studia: **kombinovaná**

Téma práce: **Srovnání dostupných implementací umělých neuronových sítí v různých programovacích jazycích**

Téma anglicky: **A Comparison of the Implementation of Artificial Neural Networks in Different Programming Languages**

Zásady pro vypracování:

1. Seznamte se s umělými neuronovými sítěmi.
2. Nalezněte dostupné implementace.
3. Zvolte vhodné datasety k porovnání jednotlivých dostupných implementací.
4. Srovnajte výsledky jednotlivých simulačních úloh pro různé implementace neuronových sítí.
5. Zhodnoťte výhody a nevýhody nalezených řešení.
6. Vytvořte vhodné tutoriály pro použití u nejvhodnějších implementací.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. ZELINKA, Ivan. Umělá inteligence I: Neuronové sítě a genetické algoritmy. 1. vyd. Brno: VUT v Brně, 1998, 126 s. ISBN 80-214-1163-5.
2. ŠÍMA J., NERUDA R.: Teoretické otázky neuronových sítí. Vyd. 1. Praha, 1996: Matfyzpress, 390 s., ISBN 80-85863-18-9
3. KŘIVAN, Miloš. Úvod do umělých neuronových sítí. Vyd. 3., přeprac. Praha: Oeconomica, 2014, 44 s. ISBN 978-80-245-2024-7.
4. PRATA, Stephen. Mistrovství v C++. 4., aktualiz. vyd. Brno: Computer Press, 2013, 1176 s. Bestseller (Computer Press). ISBN 978-80-251-3828-1.
5. SCHILDT, Herbert. Mistrovství - Java. 1. vyd. Brno: Computer Press, 2014, 1224 s. Mistrovství. ISBN 978-80-251-4145-8.
6. VRÁNA, Jakub: 1001 tipů a triků pro PHP. Vyd. 1. Brno: Computer Press, 2010, 456 s. ISBN 9788025129401.
7. GUTMANS, Andi, Stig S?ther BAKKEN a Derick RETHANS. Mistrovství v PHP 5. Vyd. 1. Brno: CP Books, 2005, 655 s. ISBN 80-251-0799-x.

Vedoucí diplomové práce:

doc. Ing. Zuzana Komínková Oplatková, Ph.D.
Ústav informatiky a umělé inteligence

Konzultant:

Ing. Erik Král, Ph.D.
Ústav počítačových a komunikačních systémů

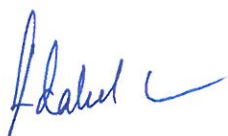
Datum zadání diplomové práce:

5. února 2016

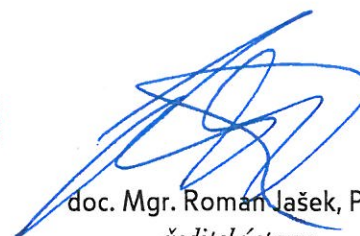
Termín odevzdání diplomové práce:

20. května 2016

Ve Zlíně dne 5. února 2016



doc. Mgr. Milan Adámek, Ph.D.
děkan



doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

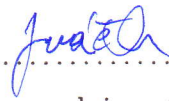
Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové 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 diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové 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 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 práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové 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 práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové 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, dne 13.5.2016

.....

.....
podpis autora

ABSTRAKT

Tato diplomová práce představuje dostupné knihovny pro implementaci umělých neuronových sítí v různých programovacích jazycích a dostupný software pro vytváření modelů a simulace funkce neuronových sítí.

Klíčová slova: Umělé neuronové sítě, implementace neuronových sítí, knihovny pro neuronové sítě

ABSTRACT

This master's thesis presents libraries available for the implementation of artificial neural networks in various programming languages and software available for modeling and simulation function of artificial neural networks.

Keywords: Artificial neural networks, implementation of neural networks, library for neural networks

Tímto bych chtěl poděkovat své vedoucí práce doc. Ing. Zuzaně Komínkové Oplatkové, Ph.D. za odborné vedení při práci a současně konzultantu mé diplomové práce Ing. et Ing. Eriku Králi, Ph.D. za cenné rady. Dále bych chtěl poděkovat své přítelkyni a rodině za podporu při studiu.

OBSAH

ÚVOD	10
I TEORETICKÁ ČÁST	11
1 NEURONOVÉ SÍTĚ	12
1.1 NEURON	12
1.1.1 Biologický neuron	12
1.1.2 Synapse	13
1.1.3 Umělý neuron	14
1.1.4 Přenosová funkce	15
1.2 PRINCIP FUNKCE NEURONOVÉ SÍTĚ	18
1.2.1 Pracovní režimy neuronové sítě	19
1.2.2 Učící algoritmy	19
1.3 DĚLENÍ NEURONOVÝCH SÍTÍ	19
2 KNIHOVNY PRO IMPLEMENTACI NEURONOVÝCH SÍTÍ	21
2.1 NEURONOVÉ SÍTĚ V C/C++	21
2.1.1 Knihovna FANN	21
2.1.2 Knihovna Open NN (C)	23
2.1.3 Knihovna OpenNN (C++)	23
2.2 NEURONOVÉ SÍTĚ V C#	24
2.2.1 Encog Machine Learning Framework	24
2.2.2 Knihovna FannCSharp	24
2.3 NEURONOVÉ SÍTĚ V JAVA	24
2.3.1 Encog Machine Learning Framework	24
2.3.2 Knihovna Fannj	24
2.3.3 Framework Neuroph	24
2.4 NEURONOVÉ SÍTĚ V PHP	26
2.4.1 Knihovna PHP FANN	26
2.4.2 Knihovna ANN for PHP 5	27
3 SOFTWARE PRO MODELOVÁNÍ NEURONOVÝCH SÍTÍ	28
3.1 MATHWORKS – MATLAB	28
3.1.1 Neural Network Toolbox™	28
3.2 WOLFRAM – MATHEMATICA	29
3.2.1 Nástroj Neural Networks	30
3.3 GOOGLE – DEEP DREAM	31
II PRAKTICKÁ ČÁST	32

4	PŘÍKLADY POUŽITÍ KNIHOVEN	34
4.1	DATASETY PRO TESTOVÁNÍ KNIHOVEN	34
4.1.1	Dataset: Logická funkce XOR	34
4.1.2	Dataset: Logická funkce s více vstupy a výstupy.....	34
4.1.3	Dataset: Goniometrická funkce sinus	35
4.1.4	Dataset: Fisherův Iris dataset	35
4.2	FANN PRO JAZYK C/C++.....	36
4.2.1	Instalace.....	36
4.2.2	Inicializace sítě.....	37
4.2.3	Nastavitelné parametry neuronové sítě.....	39
4.2.4	Přenosové funkce	40
4.2.5	Učící algoritmy.....	42
4.2.6	Proces učení neuronové sítě	42
4.2.7	Grafické rozhraní FANNTool.....	43
4.2.8	Spuštění a testování naučené neuronové sítě	45
4.3	FANN PRO JAZYK C#	45
4.3.1	Instalace.....	45
4.3.2	Inicializace.....	46
4.3.3	Nastavitelné parametry, přenosové funkce, učící algoritmy	46
4.3.4	Proces učení neuronové sítě	47
4.3.5	Spuštění a testování naučené neuronové sítě	47
4.4	ANN FOR PHP5.....	48
4.4.1	Instalace.....	48
4.4.2	Inicializace.....	48
4.4.3	Rozsah vstupních a výstupních dat.....	49
4.4.4	Proces učení neuronové sítě	50
4.4.5	Logování změn vah při učení.....	51
4.4.6	Spuštění a testování naučené neuronové sítě	53
4.5	NEURONOVÉ SÍTĚ V MATLABU	54
4.5.1	Průvodce vytvořením neuronové sítě.....	54
4.5.2	Postup vytvoření neuronové sítě.....	55
4.6	NEURONOVÉ SÍTĚ V MATHEMATICÉ.....	68
4.6.1	Instalace AddOnu.....	68
4.6.2	Postup vytvoření neuronové sítě.....	70
5	SROVNÁNÍ KNIHOVEN A SOFTWARE	75
5.1	SROVNÁNÍ KNIHOVEN.....	75
5.1.1	FANN C/C++	75

5.1.2	FANN C#	76
5.1.3	ANN for PHP5.....	76
5.1.4	Rychlostní testy	77
5.1.5	Srovnání.....	78
5.2	SROVNÁNÍ SOFTWARE	79
5.2.1	MATLAB	79
5.2.2	Mathematica.....	79
5.2.3	FANNTool.....	80
5.2.4	Srovnání.....	80
ZÁVĚR.....		82
SEZNAM POUŽITÉ LITERATURY		84
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK		87
SEZNAM OBRÁZKŮ		88
SEZNAM TABULEK		89

ÚVOD

Již od samotných počátků vzniku výpočetní techniky se pokoušíme pomocí počítačů napodobit činnost lidského mozku. Spolu s těmito pokusy vznikl také pojem umělá inteligence (UI). UI představuje počítač nebo stroj, který vykazuje známky inteligentního chování. Jednou z hlavních příčin vzniku umělých neuronových sítí byly pokusy simulovat funkci nervových tkání (např. sítnice oka) a tím získat podrobnější a hlubší znalosti o tom, jak naše nervová soustava funguje.

"Mezi hlavní problémy v rámci výzkumu umělé inteligence patří uvažování, znalosti, plánování, učení, zpracovávání přirozeného jazyka (komunikace), vnímání a schopnost se pohybovat či manipulovat s předměty." [1]

Umělé neuronové sítě jsou jakýmsi zjednodušeným matematickým modelem biologických neuronů a jejich vzájemného propojení. Představují jeden z výpočetních modelů používaných v UI, který se stává čím dál více rozšířeným v oblasti informačních technologií.

Neuronové sítě jsou v současné době využívány pro řešení různých typů úloh. Mezi základní typy úloh, které jsou schopny neuronové sítě řešit patří klasifikační úlohy, kdy síť předkládáme objekty a ta je musí zařadit do správné klasifikační třídy, nebo predikční úlohy, kdy má síť za úkol na základě předešlých dat predikovat (předpovědět) budoucí stav určité veličiny. Neuronová síť nebo také samostatné neurony mohou fungovat také jako logické členy – tedy reagovat na různé vstupní signály binárním výstupem 0 a 1. Mezi náročnější úlohy, které jsou neuronové sítě schopny řešit patří rozpoznávání obrazů a zvuků, komprese, rozpoznání spamu a jeho filtrace, šifrování a dešifrování zpráv a mnoho dalších.

Velké společnosti působící v IT oblasti vyvíjí vlastní neuronové sítě pro různé účely. Příkladem je společnost Google, Inc., která vyvinula vlastní neuronovou síť schopnou rozpoznávat objekty na fotografiích. Při procházení obrázků síť přiřazuje fotografiím klíčová slova podle obsahu a současně se dále učí rozpoznávat nové objekty nebo obměny stávajících objektů. Síť je tedy schopna bez ohledu na název fotografie zařadit fotografii do příslušných kategorií. Nedávno společnost svůj algoritmus zpřístupnila volně ke stažení.

V současné době máme také k dispozici open-source řešení nebo i placené nástroje pro vytvoření vlastních neuronových sítí pro různé účely. Tyto řešení jsou distribuovány ve formě knihoven pro různé programovací jazyky, nebo již jako hotový software pro vytvoření neuronové sítě podle požadovaného účelu. Mým cílem je tyto knihovny a softwarové nástroje zdokumentovat a vytvořit ucelený přehled dostupných implementací a jejich porovnání z různých hledisek.

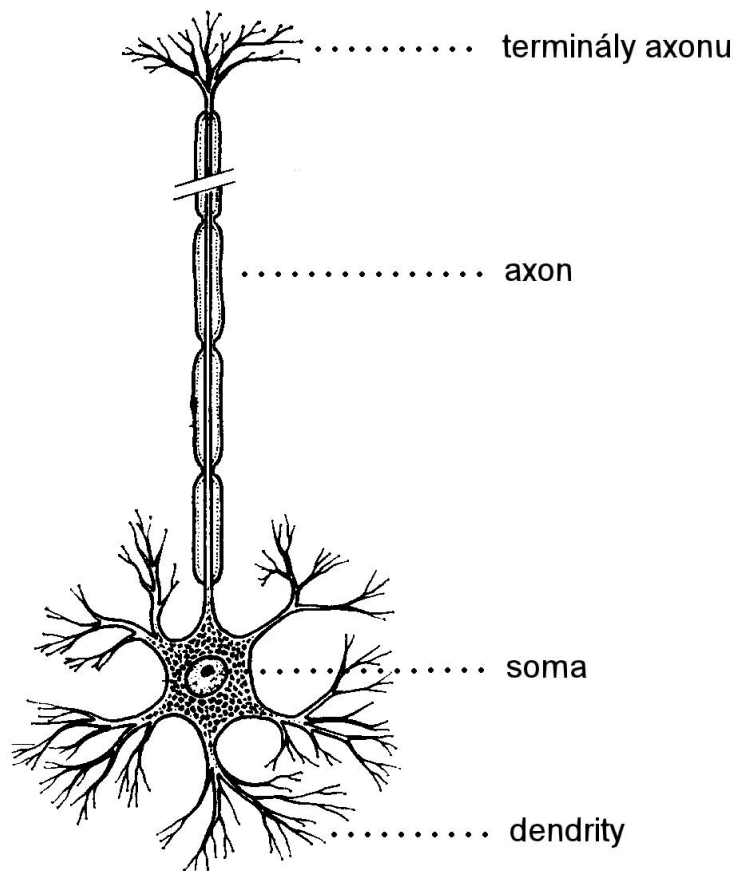
I. TEORETICKÁ ČÁST

1 NEURONOVÉ SÍTĚ

1.1 Neuron

1.1.1 Biologický neuron

Neuron neboli nervová buňka je základní funkční prvek nervové soustavy. Funkcí neuronu je přenos, zpracování a uchování informace. U biologického neuronu tyto funkce v nervové soustavě probíhají pomocí chemických reakcí, které vyvolávají impulsy. Mozková kůra člověka je tvořena 13 až 15 miliardami neuronů. Každý neuron je schopen se propojit až s 5 tisíci jinými neurony. [2]



Obr. 1.1 Biologický neuron [2]

Soma

Soma je tělo neuronu, které zpracovává vstupní signály a vysílá je axonem dále na své terminály.

Dendrity

Dendrity jsou vstupy neuronu, přes které neuron přijímá vstupní signály. Na dendrity neuronu se napojují terminály axonu jiných neuronů a vznikají tak propojení mezi neurony, které se nazývají synapse.

Axon

Axon je výstupním kanálem neuronu. Vede signál z neuronu dále do svých terminálů a následně dále do neuronové sítě.

Terminály axonu

Terminály axon rozvětvují a slouží k vytvoření spojení, tedy synapsí, s dalšími neurony.

1.1.2 Synapse

Synapse (Obr. 1.2) se nazývá místo propojení dvou neuronů - čili místo, kde se setkává terminál axonu prvního neuronu s dendritem neuronu druhého. Toto propojení slouží k předávání vzruchů mezi neurony.

V synapsích nedochází k přímému dotyku mezi terminálem axonu a dendritem navazujícího neuronu, ale je mezi nimi tzv. synaptická štěrbina široká asi 20 nm. [3]

Synapse dělíme na základní dva typy [3]:

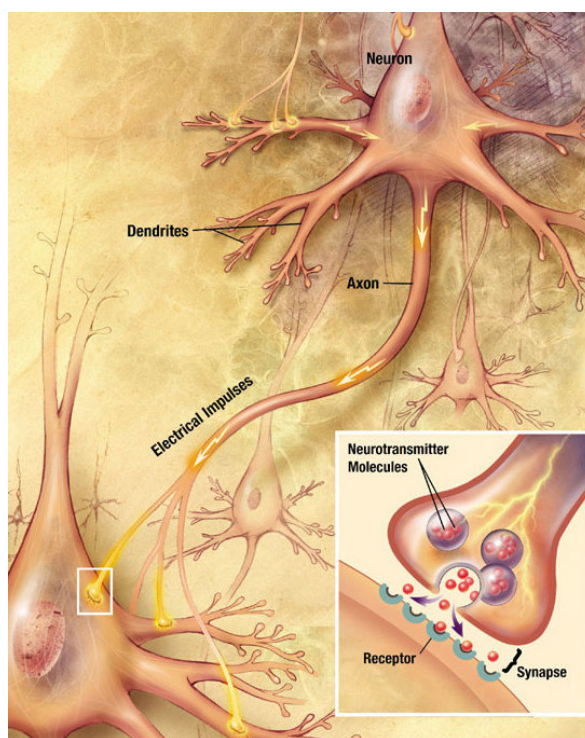
Excitační synapse

Jedná-li se o tento typ synapse, tak je postsynaptický neuron excitován (drážděn) převodem signálu.

Inhibiční synapse

Inhibiční synapse způsobuje inhibici (útlum) postsynaptického neuronu. Signál přenesen tedy není a tím je neuron tlumen.

Přesto, že je signál axonem veden elektrickým signálem, není signál v synapsích přenášen elektricky. Elektrický signál, který axonem dorazí až k synapsi, tedy na konec terminálu axonu, vyvolá uvolnění molekul neurotransmiteru do synaptické štěrby. Neurotransmitter je chemická látka, která se následně na druhé straně synaptické štěrby, tedy na dendritu navazujícího neuronu, váže na specializované molekuly zvané receptory, které díky této vazbě vyvolají vzruch v postsynaptickém neuronu. [3].



Obr. 1.2 Synapse mezi neurony [3]

1.1.3 Umělý neuron

Umělý neboli formální neuron je matematický model biologického neuronu. Tento model se snaží co nejvíce přiblížit funkci reálné nervové buňky tak, aby bylo možné co nejpodobněji simulovat funkci biologických neuronů nebo celých neuronových sítí. [2]

Model neuronu obsahuje podobné části, jako biologický neuron [2]:

Vstupy neuronu

Vstupy neuronu $x_1 \dots x_n$ jsou vstupní hodnoty signálu do neuronu, které neuron následně zpracovává pomocí své přenosové funkce. Odpovídají funkci dendritů u biologického neuronu. Vstupem neuronu je reálné číslo.

Synaptické váhy

Synaptické váhy $w_1 \dots w_n$ slouží k ohodnocení jednotlivých vstupů. Tyto váhy mohou nabývat i záporných hodnot, což vyjadřuje jejich inhibiční (utlumující) charakter. Synaptické váhy mají reálnou hodnotu a existuje nekonečně mnoho kombinací vah jednotlivých neuronů.

Vnitřní potenciál

Zvážená suma vstupních hodnot signálu se nazývá vnitřní potenciál nebo také aktivační funkce.

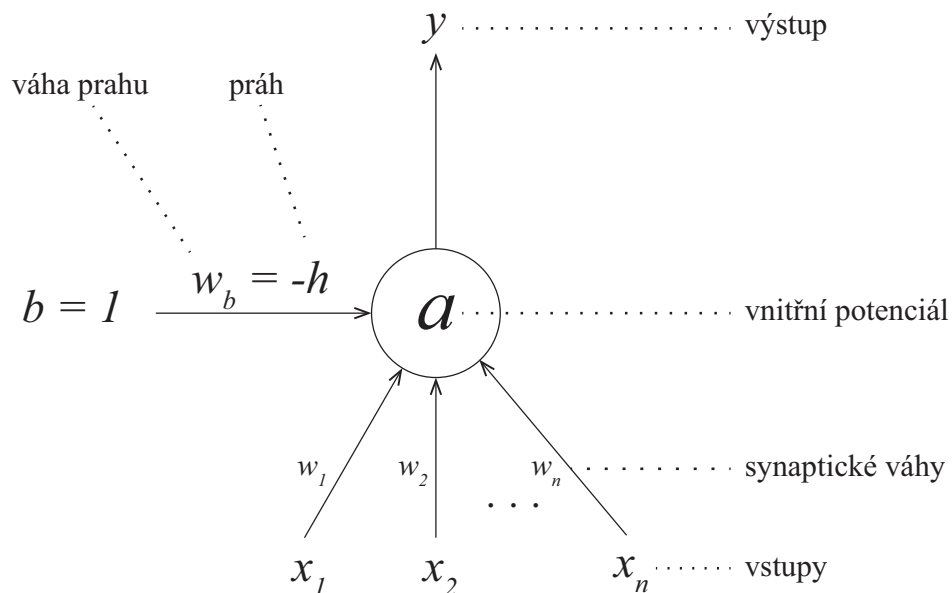
$$a = \sum_{i=1}^n w_i x_i + b w_b \quad (1.1)$$

Práh

Překročí-li vnitřní potenciál práh h , neuron se aktivuje a generuje výstup y . Pokud je vnitřní potenciál menší než práh h , zůstává neuron pasivní.

Výstup

Výstupem rozumíme signál, který je vyslán směrem ven z neuronu – nahrazuje funkci axonu biologického neuronu.



Obr. 1.3 Umělý neuron [2]

1.1.4 Přenosová funkce

"Přenosová funkce neuronu je funkce, která transformuje vstupní signál na signál výstupní v intervalech 0 až 1 a -1 až +1. Tato funkce může být skoková i spojitá a musí být monotónní - tzn. že přiřazení odezev výstupu na vstup je jednoznačné." [4]

Přenosovou funkci získáme z vnitřního potenciálu neuronu vztahem [5]

$$f(a) = TF \left(\sum_{i=1}^n w_i x_i + b w_b \right) \quad (1.2)$$

kde symbol TF představuje přenosovou funkci (z angličtiny Transfer Function).

Přenosovou funkci neuronu volíme podle toho, k jakému účelu má neuronová síť sloužit. Tato funkce určuje, jaká bude odezva na vstupní signál přicházející do neuronu. Existují různé druhy přenosových funkcí. Obecně však platí, že výstupní hodnota u všech druhů přenosových funkcí se nachází v intervalu -1 až $+1$. [4]

Chceme-li například neuronovou síť roztřídit množinu kruhů na modré a červené, postačí nám funkce binární, kde hodnota 0 bude reprezentovat modrou barvu a hodnota 1 červenou barvu (nebo opačně). Tím nám síť vrátí jednoznačný výsledek 0 nebo 1 . Pokud by jsme zvolili spojitý typ přenosové funkce, museli bychom řešit problém s reálnými hodnotami mezi 0 a 1 . [4]

Rozlišujeme šest základních typů přenosových funkcí [4][5]

- Perceptron

$$f(a) = a \quad \forall a > 0 \quad \text{jinak } a = 0 \quad (1.3)$$

- Binární funkce

$$f(a) = 1 \quad \forall a > 0 \quad \text{jinak } a = 0 \quad (1.4)$$

- Lineární přenos

$$f(a) = ka \quad (1.5)$$

- Logistická sigmoida

$$f(a) = \frac{1}{1 + e^{-ka}} \quad (1.6)$$

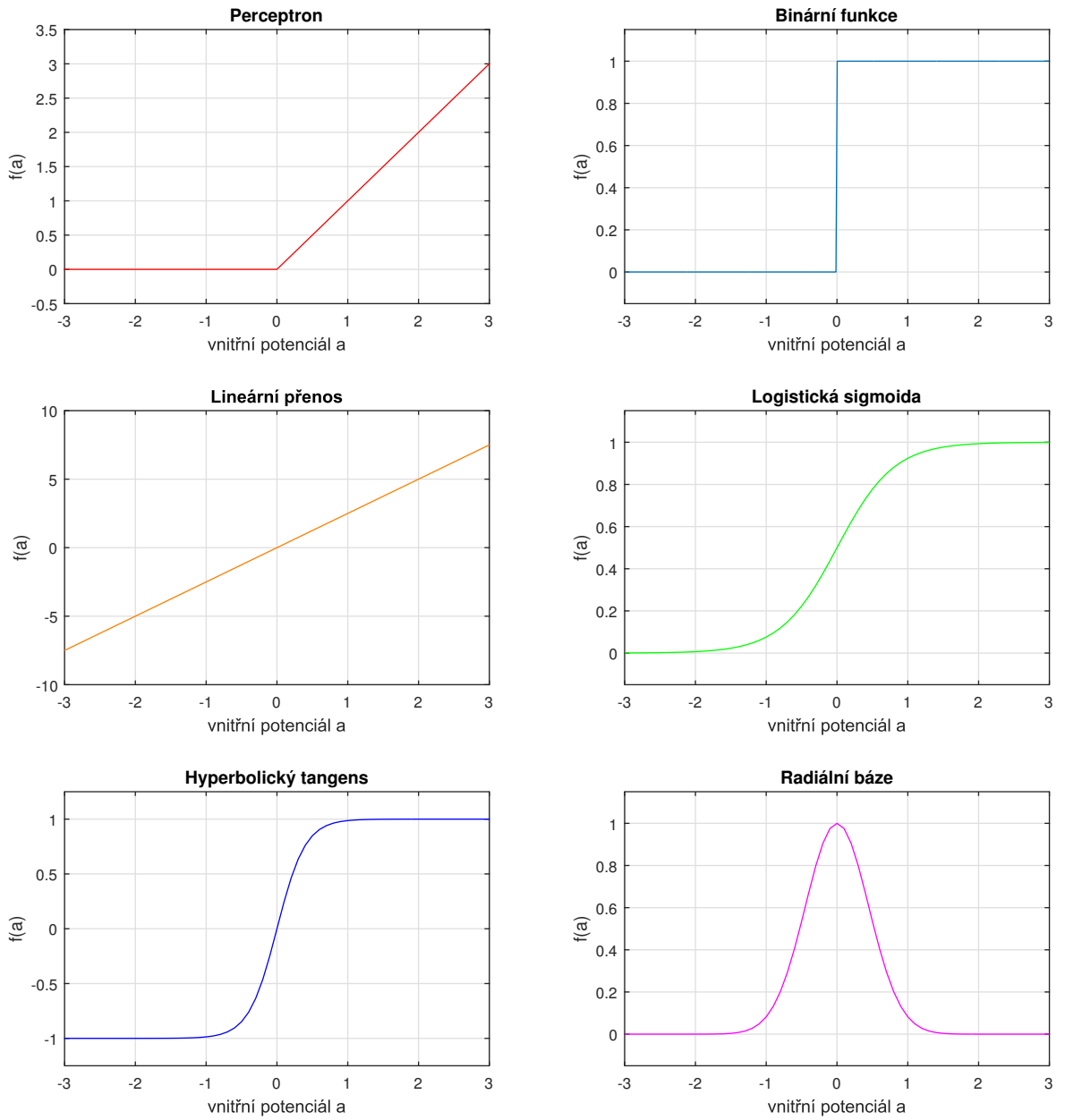
- Hyperbolický tangens

$$f(a) = \frac{e^{ka} - e^{-ka}}{e^{ka} + e^{-ka}} \quad (1.7)$$

- Radiální báze

$$f(a) = e^{-ka^2} \quad (1.8)$$

Parametr k je směrnice, která ovlivňuje strmost průběhu funkce.



Obr. 1.4 Typy přenosových funkcí

1.2 Princip funkce neuronové sítě

Každá nově vytvořená síť není schopná řešit žádný úkol. V tomto stavu se jedná o nenaučenou neuronovou síť, tedy jakéhosi technického "novorozence", který musí být nejprve naučen. Proces učení se podobá klasickému učení dítěte. Chceme-li dítě naučit rozeznávat barvy, musíme mu barvu nejprve ukázat, a poté mu říci, jak se barva nazývá a dítě se po nás snaží opakovat název a přiřadit jej dané barvě. [4]

Algoritmus učení je obvykle rozdělen na dvě fáze [4]:

1. Aktivační (vybavovací)
2. Adaptační (učící)

Obě fáze ke své činnosti potřebují učící množinu. Učící množina je množina vektorů definující daný problém. Jedná-li se o učící algoritmus s učitelem, obsahují vektory kombinaci vstupních a výstupních hodnot sítě. U učícího algoritmu bez učitele obsahuje tato množina pouze vstupní vektory. Učením se nazývá cyklické opakování obou fází.

Při aktivační fázi dochází k přepočtu vstupního vektoru neuronovou sítí za použití přenosových funkcí jednotlivých neuronů a synaptických vah. Výsledkem je výstupní vektor. Pokud aktivační fáze probíhá v procesu učení, porovnáváme aktuální výstupní vektor neuronové sítě s výstupním vektorem učící množiny, tedy požadovaným výstupním vektorem. Rozdíl mezi těmito vektory se nazývá lokální odchylka a je ukládán do paměťové proměnné.

Při adaptační fázi se snažíme o minimalizaci lokální chyby tím, že přepočítáváme, tedy adaptujeme, synaptické váhy směrem od výstupu ke vstupu neuronové sítě. Cílem je co největší podobnost výstupního vektoru sítě se vzorovým výstupním vektorem, který obsahuje učící množina. [4]

Tab. 1.1 Zpracování úlohy pomocí NS a klasického počítače [4]

Neuronová síť	Počítač
Je určena nastavováním vah, prahů a struktury	Je programován instrukcemi (if, else, then, switch, go to...)
Paměťové a výkonné prvky jsou uspořádány spolu	Proces a paměť pro něj jsou separovány
Paralelismus	Sekvenčnost
Tolerují odchylky od originálních informací	Netolerují odchylky
Samoorganizace během učení	Neměnnost programu

1.2.1 Pracovní režimy neuronové sítě

Umělá neuronová síť se může nacházet ve třech různých pracovních režimech. U biologických neuronových sítí tyto pracovní režimy nerozlišujeme, protože probíhají všechny současně. [2]

Pracovní režimy [2]:

1. Organizační – v této fázi dochází ke změně organizace sítě, tedy změna topologie a synapsí.
2. Adaptivní – učící fáze, kdy dochází ke změně (adaptaci) vah neuronů
3. Aktivní – neuron je aktivní, přijímá, zpracovává a dále do sítě vysílá signál

1.2.2 Učící algoritmy

Učící algoritmy dělíme na učení s učitelem a učení bez učitele.

U algoritmů s učitelem není myšleno užití učitele jako objektu, ale činnost, kdy předkládáme vzory a vyžadujeme jejich správné zopakování. Má-li neuronová síť za úkol rozpoznávat jednotlivá písmena na vstupních znacích, musíme síti nejprve "ukázat", tedy předložit znak a definovat o jaký znak se jedná, tedy určit, jaký má být výstupní vektor neuronové sítě k danému vstupnímu vektoru, pokud rozpozná tento znak.

Neuronové sítě využívající algoritmy učení bez učitele vycházejí pouze z informací obsažených ve vstupních vektorech. Může se jednat například o úlohu, kdy musí neuronová síť roztřídit množinu těles na předem nedefinovaný počet skupin na základě vzájemné podobnosti těchto těles. Neuronová síť tedy třídí jedno těleso po druhém, přičemž vytváří nové skupiny, pokud nalezne více podobných těles nespadaajících do některé z existujících skupin. Nalezne-li neuronová síť více těles, které nelze přiřadit do existujících skupin, ale nejsou si mezi sebou podobné, vytvoří novou skupinu, kterou lze charakterizovat jako "Nerozeznatelné". [4]

1.3 Dělení neuronových sítí

Neuronové sítě dělíme podle několika kritérií [4].

Podle počtu vrstev

- Jednovrstvé
- Vícevrstvé

Podle algoritmu učení

- S učitelem
- Bez učitele

Podle stylu učení

- Deterministické – zjištění vah výpočtem
- Stochastické – váhy generovány náhodně

2 KNIHOVNY PRO IMPLEMENTACI NEURONOVÝCH SÍTÍ

Knihovna (anglicky library) je v informatice označení pro soubor funkcí a procedur (v objektovém programování též objektů, datových typů a zdrojů), který může být sdílen více počítačovými programy. Knihovna usnadňuje programátorovi tvorbu zdrojového kódu tím, že umožňuje použít již vytvořený kód i v jiných programech. Knihovna navenek poskytuje své služby pomocí API (Application Programming Interface), což jsou názvy funkcí (včetně popisu jejich činnosti), předávané parametry a návratové hodnoty. [6]

2.1 Neuronové sítě v C/C++

2.1.1 Knihovna FANN

FANN (Fast Artificial Neural Network Library) je open-source knihovna implementující vícevrstvé neuronové sítě v jazyce C. Knihovna podporuje plně propojené i řídky propojené neuronové sítě. K dispozici je také několik grafických rozhraní pro tuto knihovnu. FANN obsahuje také framework pro snadnou manipulaci s učícími daty. [7]

Klíčové vlastnosti a funkce [7]

- Učící algoritmus backpropagation (RPROP, Quickprop, Batch, Incremental)
- Vyvíjející se topologie při učení, která dynamicky tvoří a trénuje neuronovou síť (Cascade2)
- Snadné použití (vytváření, učení a běh neuronové sítě pomocí volání pouze tří funkcí)
- Rychlost (až 150 krát rychlejší, než jiné knihovny)
- Všestranný (lze nastavit mnoho parametrů a funkcí za běhu)
- Dobrá dokumentace (podrobný manuál a rozsáhlá komunita)
- Multiplatformní (konfigurační skript pro Linux a UNIX, DLL knihovny pro Windows, projektové soubory pro MSVC++ a Borland kompilátory)
- Implementováno několik různých přenosových funkcí (včetně stupňovité lineární funkce pro navýšení rychlosti)
- Jednoduché ukládání a načítání uložených neuronových sítí
- Několik jednoduchých příkladů použití

- Možnost použití plovoucí desetinné čárky i celých čísel (ve skutečnosti jsou obě float, double a int jsou k dispozici)
- Využívání cache pro zrychlení
- Open-source řešení, ale může být použit i v komerčních aplikacích (licence LGPL)
- Framework pro jednoduchou manipulaci s učitelskými daty
- Grafické rozhraní
- Propojení s mnoha různými programovacími jazyky (více než 20 programovacích jazyků)
- Široce využíván (přibližně 100 stažení denně)

Grafické rozhraní [7]

FANN má více možností, jak se sítí pracovat. Je zde implementováno několik grafických rozhraní.

FANNTool

FANNTool je snadno použitelné GUI pro knihovnu FANN. FANNTool obsahuje také několik dalších pokročilých funkcí pro automatické nastavení parametrů FANN.

Agil Neural Network

Agil Neural Network je jednoduché rozhraní, které je soustředěno na učící data a poskytuje možnosti pro vizualizaci rozdílu mezi požadovaným a skutečným výstupem neuronové sítě.

NeuralView

NeuralView je multiplatformní simulátor neuronových sítí s grafickým rozhraním.

FannExplorer

FannExplorer je přenosné grafické prostředí pro vývoj, učení a testování neuronových sítí, které podporuje animaci učícího procesu. FannExplorer poskytuje snadno použitelný prohlížeč naprogramované neuronové sítě. Toto grafické rozhraní vyžaduje webový prohlížeč s aktivním JavaScriptem, Flash verze 7 a vyšší a fannKernel.

sfann

sfann je nástroj příkazové řádky, který umožňuje snadné učení, testování a celkovou manipulaci s neuronovou sítí.

2.1.2 Knihovna Open NN (C)

Open NN je ucelená open-source knihovna pro implementaci neuronové sítě typu vícevrstvý perceptron. Obsahuje několik funkcí a učících algoritmů, které umožňují řešit celou řadu různých problémů.

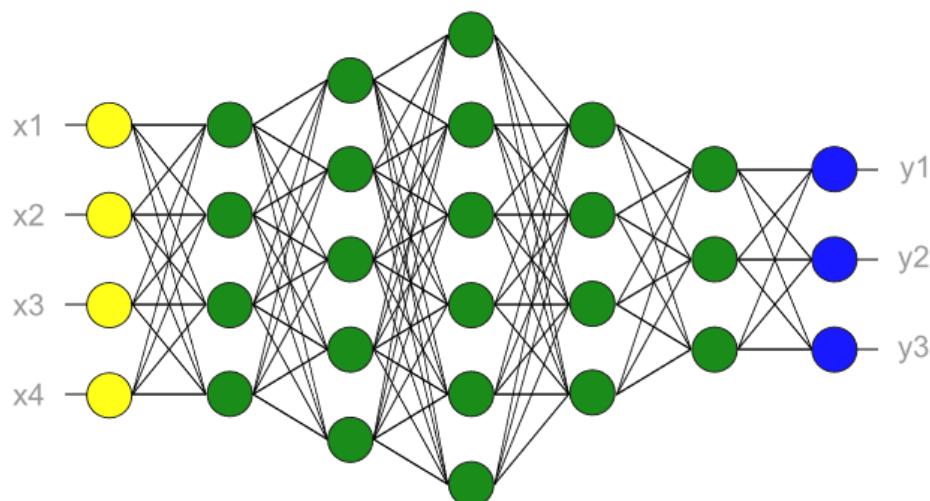
Tato knihovna také obsahuje efektivní framework pro výzkum a vývoj algoritmů neuronových sítí a jejich použití v aplikacích. [8]

Klíčové vlastnosti a funkce [8]

- Implementaci vícevrstvé perceptronové sítě
- Vzorové příklady užití knihovny
- Jednotkové testy

2.1.3 Knihovna OpenNN (C++)

OpenNN je open-source knihovna napsaná v programovacím jazyce C++. Tato knihovna implementuje neuronové sítě. Je určena pro pokročilé uživatele, s vysokou znalostí jazyka C++. Knihovna je schopna implementovat libovolný počet vrstev neuronů. Učení sítí probíhá pomocí učitele. Rozsáhlá architektura umožňuje návrh neuronových sítí s univerzálními vlastnostmi. Na obrázku 2.1 je ukázková topologie neuronové sítě vytvořené pomocí této knihovny.



Obr. 2.1 Příklad topologie neuronové sítě [9]

Hlavní výhodou OpenNN je vysoký výkon. Tato knihovna vyniká rychlostí a optimálním alokováním paměti. Proces je neustále optimalizován a paralelizován s cílem co nejvíce maximalizovat efektivitu. [9]

2.2 Neuronové sítě v C#

2.2.1 Encog Machine Learning Framework

Encog je moderní framework pro strojové učení, který podporuje celou řadu pokročilých algoritmů a podpůrných tříd. Obsahuje algoritmy strojového učení, jako je Support Vector Machines, umělé neuronové sítě, genetické programování, Bayesovské sítě. Většina učicích algoritmů tohoto frameworku jsou vícevláknové a dobře pracují na vícejádrových procesorech. Je zde možno také využívat GPU k dalšímu navýšení rychlosti zpracování. Pro rychlejší a snadnější učení sítě lze využít grafického rozhraní Encog Workbench. Encog je v aktivním vývoji od roku 2008. [10]

2.2.2 Knihovna FannCSharp

Jedná se o knihovnu FANN napsanou v jazyce C/C++, použitelnou pro aplikace v C#. Z webových stránek projektu jsou ke stažení předkompilované DLL knihovny spolu s takzvaným wrapperem pro jazyk C#. Wrapper představuje rozhraní pro manipulaci s předkompilovanými DLL knihovnami. Funkce a vlastnosti knihovny zůstávají zachovány z původní knihovny pro jazyk C/C++. [7]

2.3 Neuronové sítě v Java

2.3.1 Encog Machine Learning Framework

Jedná se o stejný framework jako Encog Machine Learning Framework pro jazyk C#. Encog pracuje stejně pod jazykem Java.

2.3.2 Knihovna Fannj

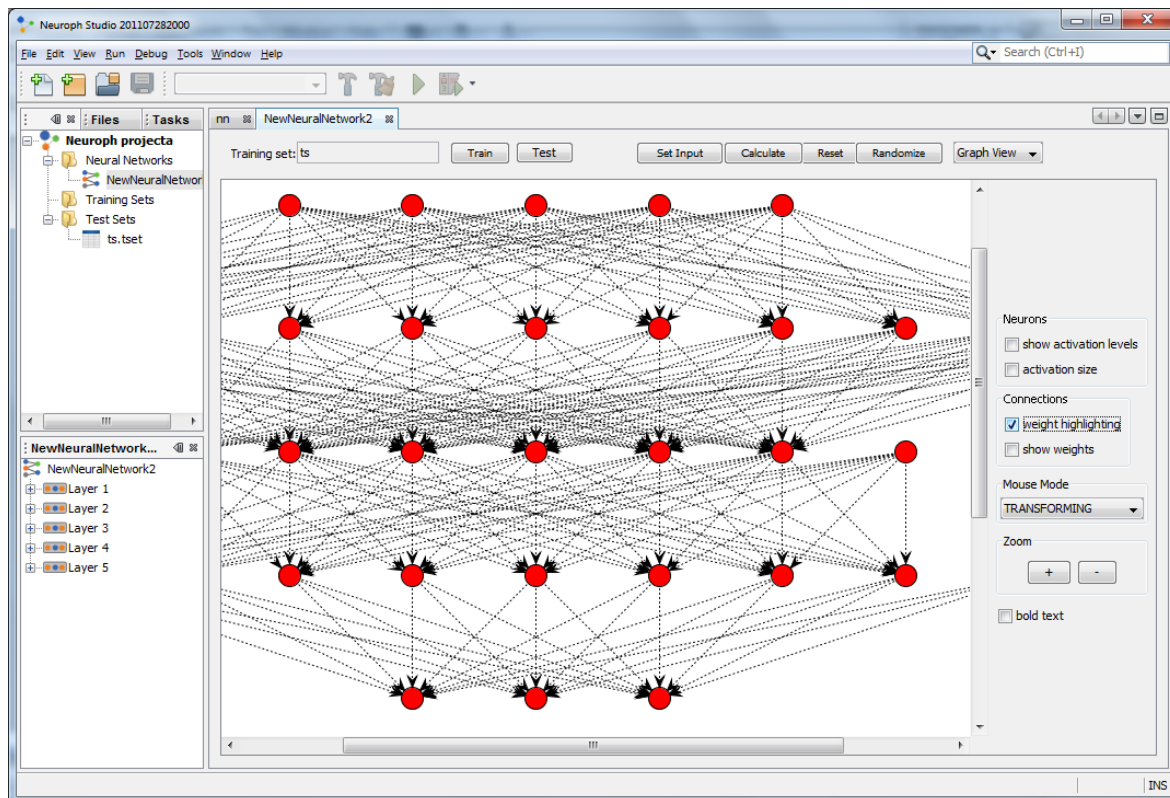
Jedná se o knihovnu FANN napsanou v jazyce C/C++ pro vývoj aplikací v jazyce Java. Ke stažení jsou knihovny pro jazyk Java, tedy soubory s příponou *.jar. [7]

2.3.3 Framework Neuroph

Neuroph je jednoduchý framework pro neuronové sítě v jazyce Java. Obsahuje dobře navrženou open-source knihovnu s malým počtem základních tříd, které odpovídají základním pojmům neuronových sítí. Obsahuje také hezké a přehledné GUI obsahující editor pro rychlou editaci a tvorbu komponent neuronové sítě. Neuroph je open-source pod licencí Apache 2.0.

Neuroph zjednodušuje vývoj neuronových sítí tím, že poskytuje Java knihovnu neuronové sítě a GUI nástroj, který podporuje tvorbu, učení a ukládání neuronové sítě. Díky jednoduchému grafickému rozhraní je vhodný i pro začátečníky, kteří nechtějí za-

cházet do hlubší teorie, a chtějí pouze vyzkoušet, jak neuronové sítě fungují Framework je malý, dobře zdokumentovaný, snadno ovladatelný, a velmi flexibilní. [11]



Obr. 2.2 Neuroph Studio [11]

Klíčové vlastnosti a funkce [11]

- Malý počet základních tříd v jádru knihovny (pouze 10), který je znovu použitelný a snadno rozšiřitelný
- Podpora učení s učitelem i bez učitele
- Snadné sledování struktury a logiky
- Java & Neural Network IDE, Neuroph Studio, založené na platformě NetBeans
- Podpora rozpoznávání obrazu
- Podpora OCR
- Predikce akciového trhu
- Vizualizace učících vzorů

- Normalizace dat

Podporované architektury neuronových sítí [11]

- Adaline
- Perceptron
- Vícevrstvý perceptron s učícím algoritmem Backpropagation, Momentum on Resilient Propagation
- Hopfieldova síť
- Bidirectional Associative Memory
- Kohonenova síť
- Hebbianova síť
- Maxnet
- Kompetitivní síť
- Instar
- Outstar
- RBF síť
- Neuro Fuzzy Reasoner

2.4 Neuronové sítě v PHP

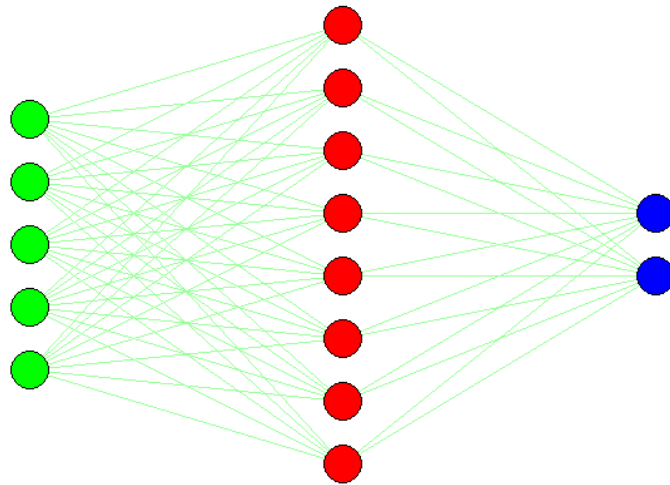
2.4.1 Knihovna PHP FANN

Jedná se o výše popsanou knihovnu FANN, původně napsanou v jazyce C/C++, ve verzi pro jazyk PHP. Je tedy možno spouštět PHP skripty na serveru, pokud je knihovna na tomto serveru nainstalována. [7]

2.4.2 Knihovna ANN for PHP 5

Knihovna ANN for PHP 5 implementuje pouze neuronové sítě s topologií vícevrstvý perceptron. Velkou výhodou knihovny je, že má implementovány metody pro vykreslení neuronové sítě, což nám umožňuje zobrazit si a následně uložit danou neuronovou síť jako PNG obrázek (viz Obr. 2.3). [12]

Podrobný rozbor a ukázka praktického využití této knihovny byly řešeny v rámci bakalářské práce s názvem Umělé neuronové sítě v PHP [13]. Součástí práce byla také jednoduchá webová aplikace založená na této knihovně.



Obr. 2.3 Ukázka zobrazení topologie sítě pomocí ANN for PHP

Klíčové vlastnosti a funkce [12]

- Řešení detekčních problémů (lineární nebo binární)
- Řešení klasifikačních problémů
- Logování vah a chyb sítě
- Model klient-server
- Grafické znázornění topologie sítě v PNG formátu
- Zobrazení detailů sítě
- Asociace řetězců

3 SOFTWARE PRO MODELOVÁNÍ NEURONOVÝCH SÍTÍ

3.1 MathWorks – Matlab

MATLAB (MATrix LABoratory) je interaktivní programové prostředí a skriptovací programovací jazyk čtvrté generace. Program MATLAB je vyvíjen společností MathWorks. MATLAB umožňuje počítání s maticemi, vykreslování 2D i 3D grafů funkcí, implementaci algoritmů, počítačovou simulaci, analýzu a prezentaci dat i vytváření aplikací včetně uživatelského rozhraní.



Obr. 3.1 Logo Matlab

Původně byl jazyk určen pro matematické účely, ale časem byl upraven, byly přidány nové funkce a rozšíření, rozrostl se různými směry a dnes je využitelný v široké paletě aplikací. V roce 2004 měl MATLAB přes jeden milion uživatelů a to především z řad vědeckotechnických pracovníků, studentů a zaměstnanců vysokých škol. MATLAB je využíván pro vědecké a výzkumné účely, a to jak v soukromém sektoru, tak i v akademických řadách. Hlavní oblastí využití jsou technické obory a ekonomie.

Název MATLAB vznikl zkrácením slov MATrix LABoratory (volně přeloženo „maticová laboratoř“), což odpovídá skutečnosti, že klíčovou datovou strukturou při výpočtech v MATLABu jsou matice. Vlastní programovací jazyk vychází z jazyka Fortran. [16]

3.1.1 Neural Network Toolbox™

Pro modelování neuronových sítí v Matlabu slouží Neural Network Toolbox™. Tento nástroj poskytuje funkce a aplikace pro modelování složitých nelineárních systémů, které nelze snadno modelovat pomocí uzavřených soustav rovnic.

Neural Network Toolbox™ podporuje učení s učitelem s dopředným šířením, radiální bází a dynamickými sítěmi. Podporuje také učení bez učitele se samoorganizujícími mapami. Pomocí panelu nástrojů je možné navrhnout, učít, vizualizovat a simulovat neuronové sítě. Neural Network Toolbox™ je možno použít pro aplikace, jako jsou rozpoznávání vzorů, shlukování, predikce časových řad a modelování dynamických systémů a jejich řízení.

Pro rychlejší učení neuronové sítě a zpracování rozsáhlých datových sad jsou výpočty zpracovávány pomocí Parallel Computing Toolbox™, který výpočty distribuuje na

vícejádrových procesorech. [17]

Klíčové vlastnosti a funkce [17]

- Učení s učitelem, vícevrstvé neuronové sítě, radiální báze, učící vektorová kvantizace (LVQ) – schopnost klasifikace, časově spožděné neuronové sítě (sítě s omezenou dobou paměti), nelineární autoregresivní exogení sítě (NARX) a rekurentní neuronové sítě
- Učení bez učitele, včetně samo-organizujících se map a kompetitivních vrstev
- Data-fitting, rozpoznávání vzorů a shlukování
- Paralelní výpočty a podpora GPU pro zrychlení učení sítě (pomocí Parallel Computing Toolbox)
- Předzpracování a postprocessing pro zlepšení učení sítě a vyhodnocení výkonu sítě
- Modulární reprezentace sítě pro manipulaci se sítí a vizualizaci sítí libovolné velikosti
- Simulink® bloky pro návrh a řízení sítě a vyhodnocování výstupů

3.2 Wolfram – Mathematica

Mathematica je počítačový software široce využívaný ve vědeckých, technických a matematických oblastech. Software byl původně vytvořen Stephenem Wolframem a následně vyvíjen týmem matematiků a programátorů. Je prodáván firmou Wolfram Research se sídlem v Champaign, Illinois. V programu Mathematica je použit programovací jazyk Wolfram. [14]



Obr. 3.2 Logo Mathematica 10

3.2.1 Nástroj Neural Networks

Neural Networks poskytuje odborníkům a studentům nástroje pro učení, vizualizaci a validaci modelů neuronových sítí. Zahrnuje komplexní sadu struktur neuronových sítí – včetně funkce radiální báze, dopředné sítě, Hopfieldovi sítě, perceptronu, vektorové kvantizace (LVQ), učení bez učitele a Kohonenovy sítě. Neural Networks obsahuje také speciální funkce k řešení typických problémů při analýze dat, jako je například aproximace funkcí, klasifikace a detekce, shlukování, nelineární časové řady a nelineární problémy identifikačního systému.

Neural Networks je stejně vhodný pro pokročilé i nezkušené uživatele. Vestavěné palety usnadňují zadání jakéhokoli parametru pro účely analýzy, hodnocení a učení pomocí učící množiny. Online dokumentace obsahuje řadu podrobných příkladů, které demonstrují různé modely neuronových sítí. Neural Networks také nabízí řadu možností k úpravě učících algoritmů. Výchozí hodnoty jsou nastaveny tak, aby dávaly dobré výsledky pro širokou škálu problémů. To nám umožňuje rychle začít používat jen několik příkazů. Po získání více zkušeností budeme moci přizpůsobit algoritmy s cílem zlepšit výkon, zvýšit rychlost a přesnost našich modelů neuronových sítí.

S Neural Networks a Mathematica, máme přístup k robustnímu modelovacímu prostředí, které umožňuje testovat a analyzovat modely neuronových sítí rychle a snadno. Nevýhodou je vyšší pořizovací cena jak samotného softwaru Mathematica, tak i samotného nástroje Neural Networks, který stojí £585, což je přes 20 000 Kč. [15]

Snadné používání a učení sítě [15]

- Malé množství funkcí konstruováno tak, že uživatel nemusí specifikovat mnoho parametrů
- Dobrá organizace palet s šablonami příkazů, možnosti a odkazy na online dokumentaci
- Inteligentní inicializační algoritmy pro počáteční učení s vysokým výkonem a rychlostí
- Rozsáhlá dokumentace včetně úvodu do teorie neuronových sítí a velmi názorných příkladů aplikací

Podporované architektury neuronových sítí [15]

- Podpora většiny běžně používaných struktur neuronových sítí včetně radiální báze, dopředného šíření, Hopfieldových sítí, perceptronu, vektorové kvantizace (LVQ), sítí s učením bez učitele a Kohonenových sítí

- Podpora pokročilých učících algoritmů včetně Levenberg-Marquardtova, Gauss-Newtonova a nejstrmějšího sestupu, stejně jako tradičních algoritmů, jako například backpropagation
- Podpora pro typické aplikace neuronových sítí včetně aproximace, klasifikace, dynamické modelování systémů, časové řady, auto-asociativní paměti, seskupování, a samoorganizujících se map

Výkonné modelovací prostředí [15]

- Vizualizační nástroje pro sledování modelů sítí, učícího procesu a výkonu sítě
- Speciální objekt neuronové sítě, který je schopen identifikovat typ sítě a zobrazit seznam jejích parametrů a vlastností
- Speciální záznam učení, který následně poskytuje informace o učícím procesu
- Funkce vybavené mnoha pokročilými možnostmi pro úpravu a řízení učících algoritmů
- Podpora vícevrstevných neuronových sítí s libovolným počtem skrytých vrstev a libovolným počtem skrytých neuronů v každé vrstvě

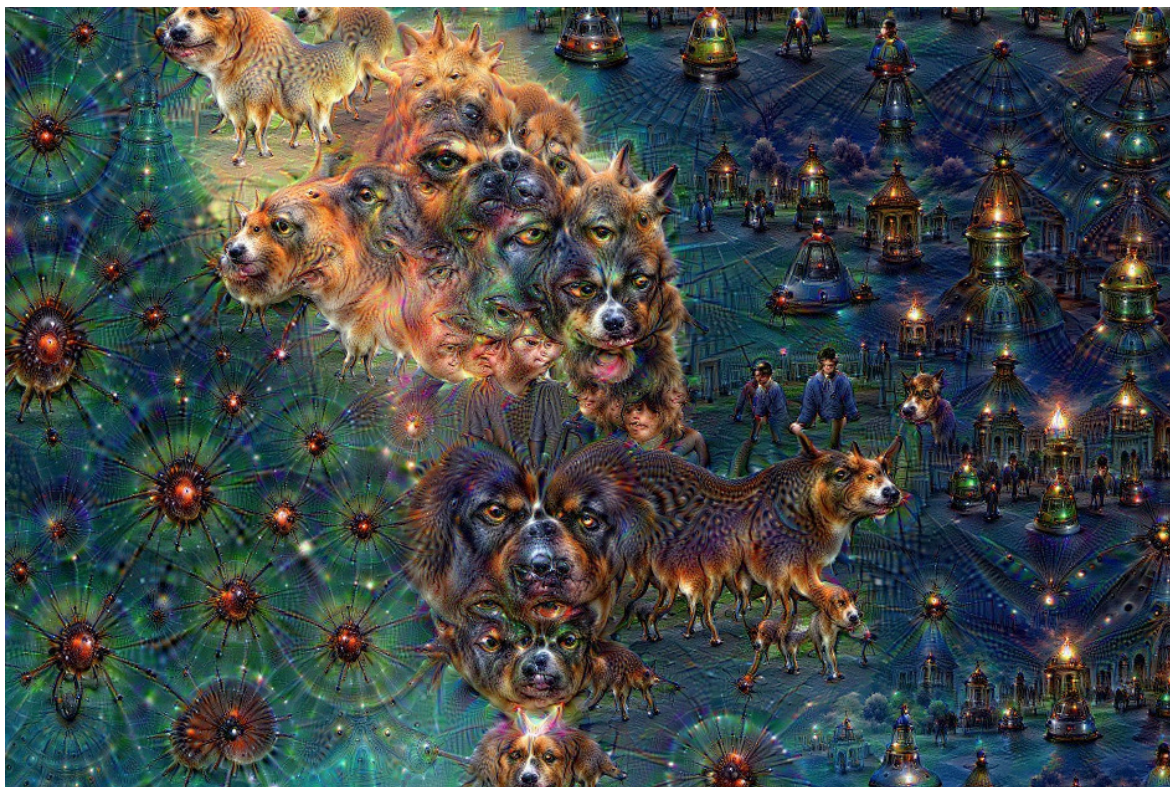
3.3 Google – Deep Dream

Společnost Google, Inc. vyvíjí vlastní systém UI pro indexování fotografií na základě jejich obsahu. Jedná se o robustní umělou neuronovou síť, která prochází databází fotografií a rozpoznává objekty vyskytující se na těchto fotografiích. Neuronová síť "zná", tedy umí rozpoznat, představitele typické pro určitou skupinu a podle toho, co na fotografii "vidí", ji zaindexuje. Neuronová síť se neustálým procházením fotografií a rozpoznáváním objektů stále učí. S každým objektem, který zařadí do určité třídy, roste počet vzorů, které síť bezpečně zná a umí zařadit.

Aby neuronová síť mohla tímto způsobem fungovat, musel Google vytvořit referenční databázi vzorových fotografií. Tato databáze se nazývá ImageNet a je k dispozici na webové adrese `www.image-net.org`. V současné době obsahuje více než 14 milionů obrazových vzorů, které slouží jako anglicko-obrázkový slovník pro neuronovou síť od Google. Fotografie jsou velmi dobře lingvisticky popsány, a proto je neuronová síť schopna se z této databáze učit.

Deep Dream je uzpůsobena pro použití běžnými uživateli. Je ukázkou toho co neuronové sítě dokáží. Tomuto softwaru můžeme předkládat jednotlivé fotografie a neuronová

síť fotografie postupně zpracovává, rozpoznává objekty na fotografii a tyto objekty do fotografie zvýrazňuje. Při dalším průchodu takto zvýrazněné objekty společně dohromady můžou síti připomínat jiný objekt, který také následně vykreslí do této fotografie. Proto se po větším počtu iterací může stát, že se na fotografii zobrazí mutace různých objektů jako například pes s rybí hlavou - proto název Deep Dream. [18]



Obr. 3.3 Ukázka obrazu vygenerovaného pomocí Deep Dream [18]

Proces je časově velmi náročný a je potřeba, aby software měl přístup k databázi obrázků. Nedávno společnost Google poskytla svoji neuronovou síť Deep Dream i se zdrojovými kódy volně ke stažení. [18]

II. PRAKTICKÁ ČÁST

4 PŘÍKLADY POUŽITÍ KNIHOVEN

V následujícím výčtu jednotlivých knihoven a softwaru si ukážeme, jak vybrané knihovny a software nainstalovat a používat. Tyto knihovny jsou následně testovány na zvolených datasetech.

4.1 Datasetsy pro testování knihoven

Pro účely testování výkonu jednotlivých knihoven je potřeba určit několik datasetů, na kterých budou dané knihovny testovány. Datasetsy tedy musí být vhodně zvoleny tak, aby testování generovalo vhodné výsledky, na základě kterých bude možno jednotlivé neuronové sítě porovnat z různých hledisek.

Datasetem rozumíme soubor vstupních a výstupních dat, pomocí kterých bude neuronová síť učena. Dataset tedy představuje učící množinu obsahující vstupní vektory a k nim přidružené výstupní vektory.

4.1.1 Dataset: Logická funkce XOR

Jako první, nejjednodušší dataset byla zvolena logická funkce XOR. Vstupní vektor tvoří vstupy A a B představující dvě binární hodnoty. Výstupní vektor obsahuje pouze jednu binární hodnotu. Plně naučená síť pomocí tohoto datasetu tedy bude pracovat stejně, jako logická součástka XOR.

Tab. 4.1 Dataset: Logická funkce XOR

Vstupy		Výstup
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

4.1.2 Dataset: Logická funkce s více vstupy a výstupy

Další dataset představuje složitější logickou funkci. Vstupní vektor má pět vstupních binárních hodnot, a odezvou je výstupní vektor se třemi binárními hodnotami. Dataset obsahuje 32 vzorů (každé binární kombinaci je přiřazena její odezva v podobě výstupního vektoru). Tabulka 4.2 však obsahuje pouze ukázkou deseti učících vzorů.

Tab. 4.2 Dataset: Složitější logická funkce (5 vstupů, 3 výstupy)

Vstupy					Výstupy		
A	B	C	D	E	X	Y	Z
0	0	0	0	0	1	1	0
0	0	0	0	1	0	1	0
0	0	0	1	0	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	1	0	0	0
0	0	1	1	0	1	0	1
0	0	1	1	1	1	1	1
0	1	0	0	0	0	1	1
0	1	0	0	1	1	0	1

4.1.3 Dataset: Goniometrická funkce sinus

Tento dataset má pouze jeden vstup a jeden výstup. Vstupní hodnotou je hodnota x , která představuje úhel ve stupních. Odezvou sítě je hodnota $\sin(x)$. Neuronová síť naučená tímto datasetem by měla být schopna určit relativně přesný výsledek funkce sinus na zadaný vstupní úhel.

Tab. 4.3 Dataset: Goniometrická funkce sinus

Vstup	Výstup
x [°]	$\sin(x)$
0	0,00000
30	0,50000
45	0,70711
60	0,86603
90	1,00000
120	0,86603
135	0,70711
150	0,50000
180	0,00000

4.1.4 Dataset: Fisherův Iris dataset

Tento dataset představil již v roce 1936 britský statistik a biolog Ronald Fisher. Jedná se o soubor dat obsahující naměřená data kališních a okvětních lístků kvetoucí rostliny Kosatec (lat. Iris). Ronald Fisher objevil závislost rozměrů kališních a okvětních lístků na odrůdě rostliny. Dataset obsahuje data naměřená ze tří druhů této rostliny: Iris

setosa, Iris versicolor a Iris virginica. Tento dataset tedy použijí pro určení druhového jména rostliny (setosa, versicolor, virginica) na základě vstupních naměřených hodnot rozměrů kališních a okvětních lístků. Soubor obsahuje 150 učicích vzorů. Každý učicí vzor obsahuje ve vstupním vektoru 4 reálné hodnoty vyjadřující rozměry určitých lístků v centimetrech. Výstupem je určení jedné ze tří tříd. Jedná se tedy o klasifikační úlohu. Tabulka 4.4 obsahuje pouze ukázkou deseti učicích vzorů z celkových 150. Celkově tedy máme k dispozici 50 zástupců každé ze tří tříd. [19]

Tab. 4.4 Dataset: Fisherův Iris dataset [20]

Vstupy				Výstup
Kališní lístek		Okvětní lístek		Třída
Délka [cm]	Šířka [cm]	Délka [cm]	Šířka [cm]	
5,1	3,5	1,4	0,2	setosa
4,9	3,0	1,4	0,2	setosa
4,7	3,2	1,3	0,2	setosa
7,0	3,2	4,7	1,4	versicolor
6,4	3,2	4,5	1,5	versicolor
6,9	3,1	4,9	1,5	versicolor
6,3	3,3	6,0	2,5	virginica
5,8	2,7	5,1	1,9	virginica
7,1	3,0	5,9	2,1	virginica
6,3	2,9	5,6	1,8	virginica

4.2 FANN pro jazyk C/C++

Knihovna FANN je velmi rozsáhlá a snadno použitelná pro mnoho různých typů neuro-nových sítí. Její obsáhlá a podrobná dokumentace napomáhá velmi rychlému pochopení principů této knihovny.

4.2.1 Instalace

Knihovnu FANN je možno stáhnout v podobě ZIP balíčku pomocí odkazu na webových stránkách projektu [7]. Tento balíček obsahuje předkompilované DLL a LIB soubory, které pouze zahrneme do svého C/C++ projektu spolu s hlavičkovými soubory a následně můžeme knihovnu používat.

Pro zpřístupnění knihoven v našem projektu zahrneme potřebné hlavičkové soubory (Kód 1).

Kód 1 Hlavičkové soubory FANN

```
1 #include <fann.h>
2 #include <fann_cpp.h>
3 #include <floatfann.h>
```

Balíček také obsahuje již hotové funkční příklady, které si můžeme vyzkoušet. Spolu s těmito funkčními příklady obsahuje také datasety ve formě textových souborů pro učení neuronové sítě.

4.2.2 Inicializace sítě

Po načtení hlavičkových souborů můžeme inicializovat svou neuronovou síť, tím že vytvoříme nový objekt neuronové sítě. Následně můžeme tomuto objektu nastavovat různé parametry.

Kód 2 Inicializace neuronové sítě pomocí knihovny FANN v jazyce C/C++

```
1 // Vytvoreni objektu neural_net
2 FANN::neural_net nn;
3
4 // Vytvoreni site zadanim pozadovane topologie
5 nn.create_standard(num_layers, num_input, num_hidden, num_output);
6
7 // Nastaveni uciciho koeficientu
8 nn.set_learning_rate(learning_rate);
9
10 // Nastaveni prenosove funkce skrytych neuronu
11 nn.set_activation_function_hidden(FANN::
12     SIGMOID_SYMMETRIC_STEPWISE);
13
14 // Nastaveni prenosove funkce vystupnich neuronu
15 nn.set_activation_function_output(FANN::
16     SIGMOID_SYMMETRIC_STEPWISE);
```

Vytvořit síť s určitou topologií můžeme několika způsoby. Knihovna obsahuje celkově 6 funkcí pro vytvoření požadované neuronové sítě, které se liší ve způsobu zadávání topologie sítě nebo v míře a způsobu propojení jednotlivých neuronů.

Metoda: `create_standard`

Vytvoří standartní plně propojenou backpropagation neuronovou síť. Parametry definujeme jednotlivě.

Metoda: `create_standard_array`

Vytvoří standartní plně propojenou backpropagation neuronovou síť. Parametry definujeme pomocí pole, které obsahuje informace o topologii jednotlivých vrstev.

Metoda: `create_sparse`

Vytvoří standartní backpropagation neuronovou síť, která není plně propojena - obsahuje parametr míry propojení. Parametry definujeme jednotlivě.

Metoda: `create_sparse_array`

Vytvoří standartní backpropagation neuronovou síť, která není plně propojena - obsahuje parametr míry propojení. Parametry definujeme pomocí pole, které obsahuje informace o topologii jednotlivých vrstev.

Metoda: `create_shortcut`

Vytvoří standartní plně propojenou backpropagation neuronovou síť, která má navíc také zkrácené propojení, což znamená že neurony nejsou propojeny pouze s neurony následující vrstvy, ale také s neurony za následující vrstvou. Parametry definujeme jednotlivě.

Metoda: `create_shortcut_array`

Vytvoří standartní plně propojenou backpropagation neuronovou síť, která má navíc také zkrácené propojení, což znamená že neurony nejsou propojeny pouze s neurony následující vrstvy, ale také s neurony za následující vrstvou. Parametry definujeme pomocí pole, které obsahuje informace o topologii jednotlivých vrstev.

Kód 3 Příklady metod pro vytvoření neuronové sítě

```
1 // Pocty neuronu jednotlivych vrstev ulozeny v poli
2 unsigned int layers[4] = {2, 7, 9, 1};
3
4 // Vytvoreni site o 4 vrstvach: 2 vstupni neurony, 2 skryte
   vrstvy se 7 a 9 neurony, 1 vystupni neuron
5 nn.create_standard(4, 2, 7, 9, 1);
6
7 // Vytvoreni site o 4 vrstvach: 2 vstupni neurony, 2 skryte
   vrstvy se 4 neurony, 1 vystupni neuron
```

```
8 nn.create_standard_array(4, layers);
9
10 // Stejne jako create_standard() s rozdilem propojeni pouze na
    60%
11 nn.create_sparse(0.6, 4, 2, 7, 9, 1);
12
13 // Stejne jako create_standard_array() s rozdilem propojeni
    pouze na 60%
14 nn.create_sparse_array(0.6, 4, layers);
15
16 // Stejne jako create_standard()
17 nn.create_shortcut(4, 2, 7, 9, 1);
18
19 // Stejne jako create_standard_array()
20 nn.create_shortcut_array(4, layers);
```

4.2.3 Nastavitelné parametry neuronové sítě

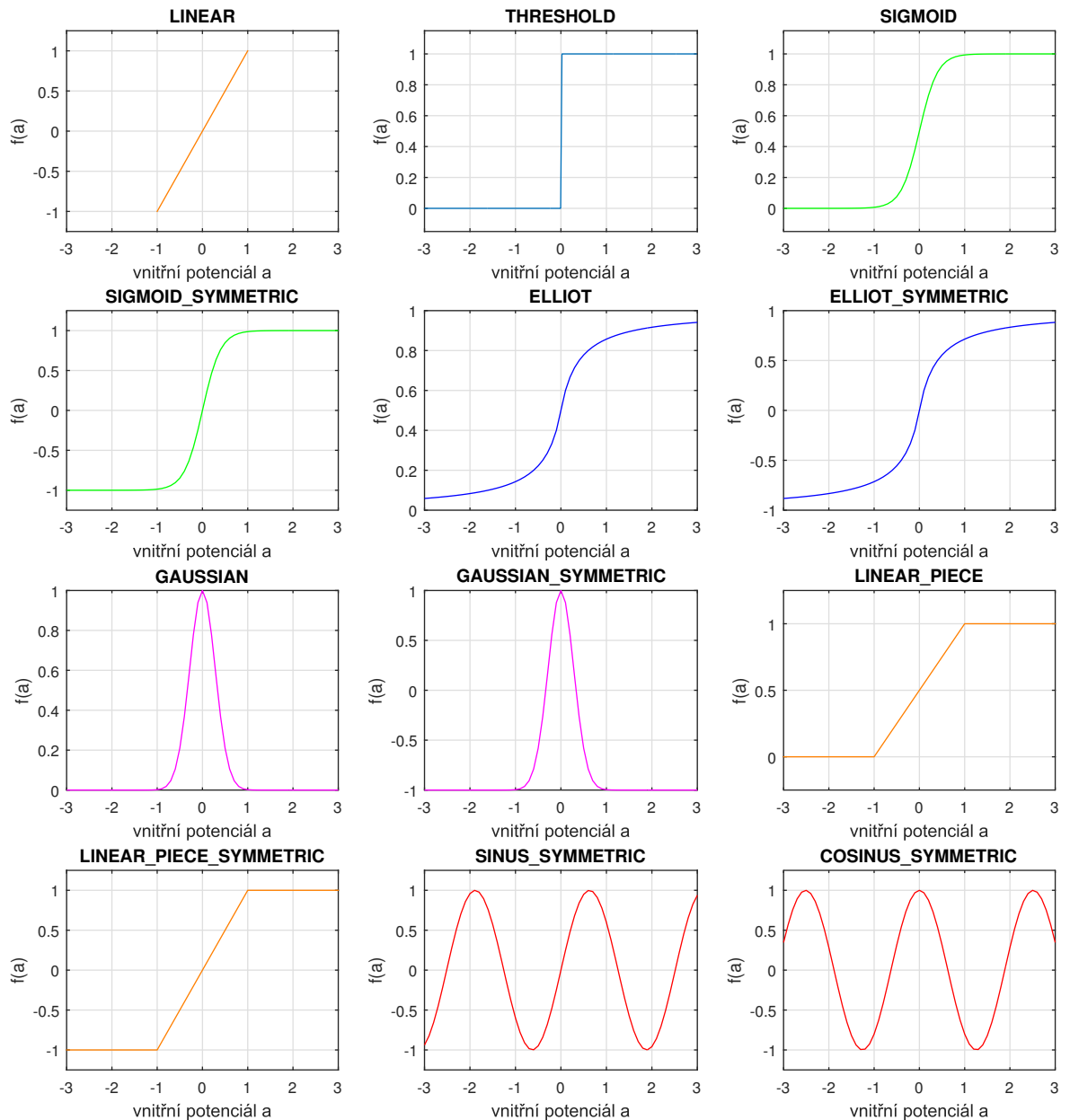
U nově vytvořené neuronové sítě máme možnost nastavit velké množství parametrů a tím ovlivnit rychlost procesu učení a vlastnosti samotné neuronové sítě. V následujícím bloku kódu jsou postupně nastavovány základní parametry neuronové sítě.

Kód 4 Nastavení základních parametrů neuronové sítě

```
1 // Deklarace promenných definující parametry
2 FANN::activation_function_enum actHidden = FANN::
    SIGMOID_SYMMETRIC_STEPWISE;
3 FANN::activation_function_enum actOutput = FANN::
    SIGMOID_SYMMETRIC_STEPWISE;
4 FANN::training_algorithm_enum trainAlgorithm = FANN::
    TRAIN_INCREMENTAL;
5 const float learning_rate = 0.7f;
6
7 // Nastavení ucíciho koeficientu
8 nn.set_learning_rate(learning_rate);
9
10 // Nastavení typu přenosové funkce skrytých neuronu
11 nn.set_activation_function_hidden(actHidden);
12
13 // Nastavení typu přenosové funkce výstupních neuronu
14 nn.set_activation_function_output(actOutput);
15
16 // Nastavení ucíciho algoritmu
17 nn.set_training_algorithm(trainAlgorithm);
```

4.2.4 Přenosové funkce

Knihovna FANN má k dispozici mnoho různých přenosových funkcí (Obr. 4.1). Všechny vycházejí ze základních typů přenosových funkcí uvedených na obrázku 1.4.



Obr. 4.1 Přenosové funkce knihovny FANN

THRESHOLD (SYMMETRIC)

Přenosová funkce *threshold* představuje binární funkci nabývající hodnot 0 a 1. Její verze *symmetric* nabývá hodnot -1 a 1.

SIGMOID (STEPWISE, SYMMETRIC)

Přenosová funkce *sigmoidea* v základní verzi nabývá reálných hodnot v uzavřeném intervalu $\langle 0;1 \rangle$. Je možné použít také verzi *symmetric*, *stepwise* nebo kombinaci *stepwise* a *symmetric*. Symetrická verze představuje sigmoidu nabývající hodnot $\langle -1;1 \rangle$. Typ *stepwise* znamená, že přenosová funkce je aproximována na sigmoidu pomocí menších lineárních úseků, což sníží nároky na výpočetní čas jak při učení neuronové sítě, tak při samotném běhu již naučené neuronové sítě, ale sníží přesnost této přechodové funkce.

GAUSSIAN (STEPWISE, SYMMETRIC)

Tato přenosová funkce představuje funkci zvanou radiální báze. U této funkce, stejně jako u sigmoidy, máme možnost si zvolit, jestli se bude jednat o klasickou nebo symetrickou přenosovou funkci. Také můžeme zvolit verzi *stepwise*, která nám urychlí proces učení i běh naučené neuronové sítě, ale sníží přesnost přenosové funkce.

ELLIOT (SYMMETRIC)

Elliotova funkce je takzvaná rychlá přenosová funkce podobná sigmoidě. Autorem této funkce je David Elliott. Tato funkce není tak hojně používaná, proto nebyla zmíněna v teoretické části diplomové práce. Z toho důvodu je potřeba uvést její matematický zápis.

$$f(a) = \frac{xk}{1 + |xk|} \quad (4.1)$$

LINEAR, LINEAR_PIECE (SYMMETRIC)

Knihovna obsahuje dva typy lineární funkce. Prvním typem je klasická lineární funkce a druhým typem je omezená (ohraničená) lineární funkce. Klasická lineární funkce nemá omezení hodnoty přenosové funkce. Omezená lineární funkce nabývá reálných hodnot ležících na intervalu $\langle 0;1 \rangle$, nebo na intervalu $\langle -1;1 \rangle$, jedná-li se o *symmetric* verzi.

SIN_SYMMETRIC, COS_SYMMETRIC

Jedná se o periodické goniometrické funkce sinus a cosinus. Jejich hodnoty nabývají reálných hodnot v uzavřeném intervalu $\langle -1;1 \rangle$, a proto jsou označeny jako *symmetric*.

4.2.5 Učící algoritmy

Knihovna FANN používá několik typů učícího algoritmu backpropagation.

Incremental

Standardní backpropagation algoritmus, kde jsou váhy aktualizovány po každém učícím vzoru. To znamená, že váhy jsou aktualizovány během jedné epochy mnohokrát. Jednoduché sítě tedy budou učeny velmi rychle, zatímco u pokročilejších sítí nebude učící proces tak efektivní.

Batch

Standardní backpropagation algoritmus, kde jsou váhy aktualizovány po výpočtu střední kvadratické chyby na celé učící množině. To znamená, že váhy jsou aktualizovány pouze jednou během epochy. Některé typy sítí tedy budou učeny pomaleji, ale protože střední kvadratická chyba je vypočtena přesněji než při kumulativním učení, některé typy sítí pomocí tohoto algoritmu dosáhnou lepšího řešení.

RPROP, QUICKPROP, SARPROP

Algoritmus RPPROP (a jeho další varianty) je pokročilým typem učícího algoritmu backpropagation. Tento algoritmus je adaptivní, a proto nevyužívá učící koeficient pro změnu synaptických vah.

4.2.6 Proces učení neuronové sítě

Nejprve je nutné vytvořit učící množinu v externím textovém souboru s příponou `*.train`. Struktura souboru je velmi jednoduchá. Jednotlivé hodnoty oddělujeme pouze mezerou. Na prvním řádku definujeme strukturu učící množiny, tedy počet učících vzorů, počet vstupů a počet výstupů. Následují vstupní a výstupní vektory jednotlivých vzorů. Každý vzor je tedy zapsán na dvou řádcích. První z nich obsahuje vstupní vektor a druhý obsahuje výstupní vektor.

Kód 5 Učící množina pro funkci XOR

```
1  4 2 1 // 4 vzory, kazdy ma 2 vstupy a 1 vystup
2  -1 -1 // vstupni vektor
3  -1 // vystupni vektor
4  -1 1
5  1
6  1 -1
7  1
8  1 1
9  -1
```

Máme-li vytvořený soubor s učící množinou, můžeme vytvořit objekt učící množiny, který následně předáme algoritmu. Do objektu musíme načíst data z našeho souboru.

Na počátku učícího procesu je nutné inicializovat počáteční váhy celé neuronové sítě, což můžeme udělat dvěma způsoby. Knihovna obsahuje pro tento účel dvě odlišné funkce. První možností je použití funkce `randomize_weights(min_weight, max_weight)` se dvěma parametry určující interval volby náhodných synaptických vah. Druhou možností je použití funkce `init_weights(tData)`, kde do parametru funkce vložíme objekt s naší učící množinou. Zde se váhy nevolí náhodně, ale pomocí algoritmu vytvořeného Derrickem Nguyenem a Bernardem Widrowem. Tento algoritmus potřebuje znát rozsah vstupních dat, proto předáváme jako argument objekt učící množiny. Pomocí získaného rozsahu vstupních hodnot, rozloží algoritmus váhy tak, aby následné učení bylo rychlejší.

Naučenou neuronovou síť můžeme následně uložit do souboru pro pozdější použití pomocí funkce `save(file)`.

Kód 6 Proces učení

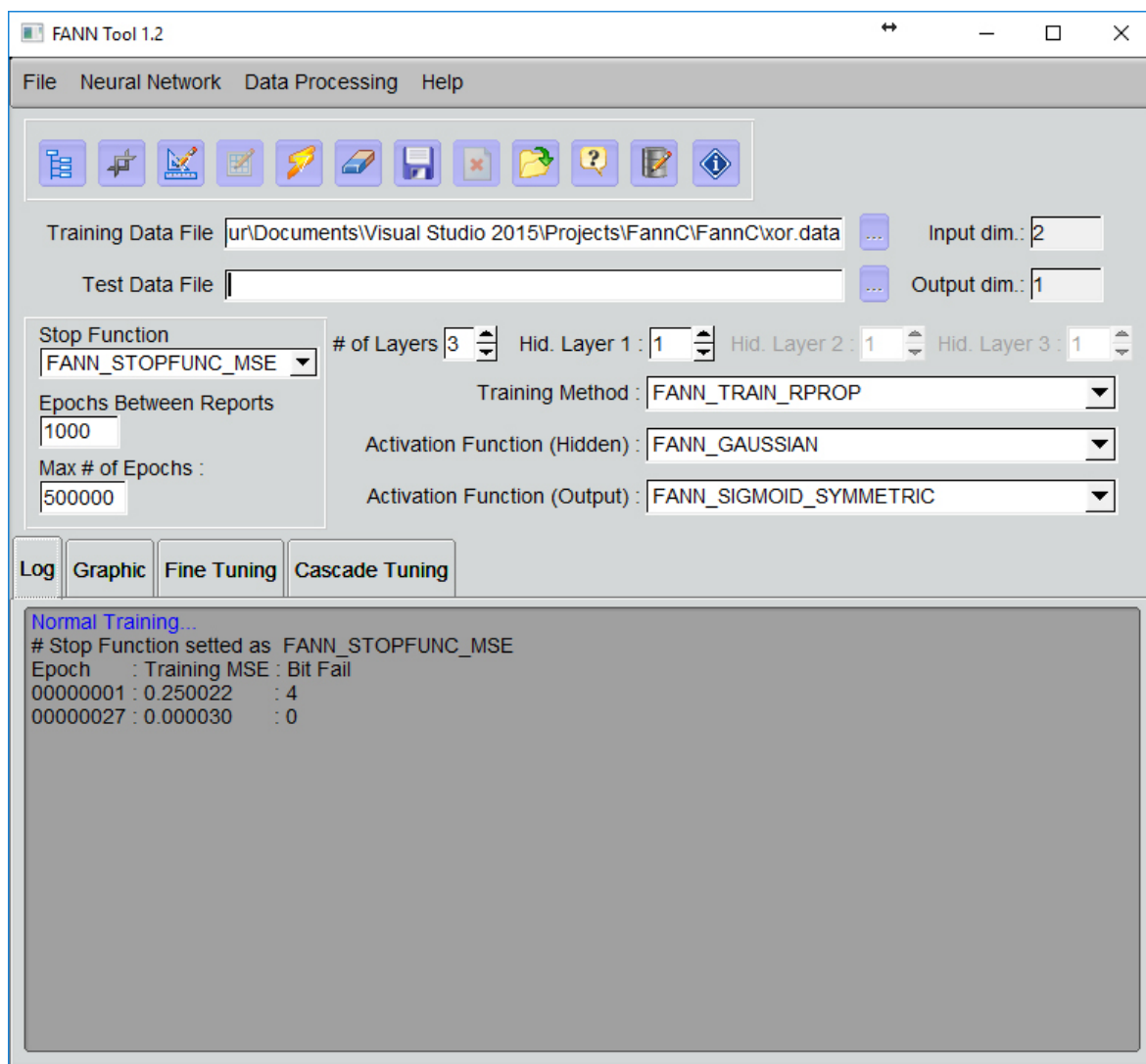
```
1 // Deklarace potrebných promenných
2 const float desired_error = 0.001f;
3 const unsigned int max_iterations = 300000;
4 const unsigned int iterations_between_reports = 1000;
5 FANN::training_data tData;
6
7 if (tData.read_train_from_file("xor.train"))
8 {
9     // Inicializace vah na pocatku uciciho procesu
10    nn.init_weights(tData);
11
12    // Spusteni uciciho procesu
13    nn.train_on_data(tData, max_iterations,
14                    iterations_between_reports, desired_error);
15
16    // Ulozeni site do textoveho souboru
17    nn.save("net_xor.net");
18 }
```

4.2.7 Grafické rozhraní FANNTool

FANNTool je multiplatformní vizuální nástroj založený na knihovně FANN. V této aplikaci je možné vyzkoušet si možnosti této knihovny a přichystat si data ve formátu, který síť podporuje.

Ovládání aplikace je velice intuitivní a jednoduché. Můžeme nastavovat všechny dostupné parametry sítě, topologii sítě, učit síť, testovat síť a sledovat průběh učícího

procesu. Vytvořenou síť si můžeme uložit do souboru pro pozdější použití. Tento soubor s příponou *.net obsahuje kompletní data a parametry neuronové sítě, aby bylo možné ji z tohoto souboru následně sestavit. Učící množinu si definujeme do textového souboru s příponou *.train a testovací data do souboru s příponou *.test.



Obr. 4.2 Nástroj FANNTool

V aplikaci je možné nastavovat všechny dostupné parametry neuronové sítě a po ukončení učícího algoritmu si prohlédnout grafický vývoj chyby. Aplikace také obsahuje funkce pro nalezení optimální přenosové funkce neuronů pro rychlejší průběh učícího procesu.

4.2.8 Spuštění a testování naučené neuronové sítě

Naučenou neuronovou síť načteme ze souboru. V uloženém souboru je plná konfigurace naučené neuronové sítě, a proto ušetříme čas, který by byl nutný pro opětovné naučení neuronové sítě. Vytvoříme vstupní vektor a předáme tento vektor metodě `run(input)` (viz Kód 7).

Kód 7 Inicializace neuronové sítě ze souboru a její spuštění

```
1 // Vytvoreni objektu site a nacteni ze souboru
2 FANN::neural_net nn;
3 nn.create_from_file("net_xor.net");
4
5 // Vstupni vektor
6 fann_type input[] = { 0 , 1 };
7
8 // Spusteni site a ulozeni vystupu do promenne
9 fann_type *out = nn.run(input);
```

4.3 FANN pro jazyk C#

Tato knihovna je totožná s knihovnou FANN pro jazyk C/C++. Jedná se o před-kompilované DLL knihovny, ke kterým je k dispozici takzvaný wrapper. V překladu wrapper znamená obal nebo obálka, což vyjadřuje jeho funkci. Wrapper tvoří pomyslný obal před-kompilovaných DLL souborů a vytváří rozhraní pro přístup ke všem funkcím a metodám pomocí jazyka C#.

4.3.1 Instalace

Pro instalaci FANN knihovny pro jazyk C# je potřeba stáhnout z webových stránek projektu FANN [7] před-kompilované DLL soubory. V jednom ZIP souboru stáhneme knihovnu současně s wrapperem. Po stažení ZIP soubor rozbalíme a zahrneme DLL knihovny do našeho projektu.

Kód 8 Použití knihovny FANN v C# projektu

```
1 using FANNCSsharp;
2 using FANNCSsharp.Float;
```

4.3.2 Inicializace

Inicializace objektu neuronové sítě je velmi podobná jako v jazyce C/C++. Pro vytvoření objektu neuronové sítě máme, stejně jako u FANN knihovny v jazyce C/C++, řadu možností. Možnosti se liší pouze ve formátu vstupních hodnot předaných konstruktoru třídy `NeuralNet`.

Kód 9 Inicializace neuronové sítě v jazyce C#

```
1  uint numLayer = 3;
2  uint[] layer = { 2, 4, 1 };
3
4  // Vytvoreni objektu neural_net
5  NeuralNet nn = new NeuralNet(numLayer, layer);
6
7  // Nastaveni uciciho koeficientu
8  nn.LearningRate = 0.7f;
9
10 // Nastaveni prenosove funkce skrytych a vystupnich neuronu
11 nn.ActivationFunctionHidden = ActivationFunction.
    SIGMOID_SYMMETRIC_STEPWISE;
12 nn.ActivationFunctionOutput = ActivationFunction.
    SIGMOID_SYMMETRIC_STEPWISE;
13
14 // Nastaveni uciciho algoritmu
15 nn.TrainingAlgorithm = TrainingAlgorithm.TRAIN_RPROP;
```

4.3.3 Nastavitelné parametry, přenosové funkce, učící algoritmy

U vytvořené neuronové sítě máme možnost nastavit velké množství parametrů. Princip je stejný jako u knihovny FANN v jazyce C/C++, což se týká také typů přenosových funkcí a učících algoritmů. Rozdíl je pouze v syntaxi, která je jednodušší než v C/C++.

Kód 10 Nastavení parametrů neuronové sítě

```
1  // Nastaveni uciciho koeficientu
2  nn.LearningRate = 0.7f;
3
4  // Nastaveni prenosove funkce skrytych a vystupnich neuronu
5  nn.ActivationFunctionHidden = ActivationFunction.
    SIGMOID_SYMMETRIC_STEPWISE;
6  nn.ActivationFunctionOutput = ActivationFunction.
    SIGMOID_SYMMETRIC_STEPWISE;
7  // Nastaveni uciciho algoritmu
8  nn.TrainingAlgorithm = TrainingAlgorithm.TRAIN_RPROP;
```

4.3.4 Proces učení neuronové sítě

Učící množinu, tedy data si můžeme připravit do textového souboru, nebo je možné přímo ve zdrojovém kódu programu vytvořit datovou strukturu obsahující učící množinu.

Naučenou síť je možné si uložit do souboru s příponou *.net pro pozdější načtení bez nutnosti opětovného učení.

Kód 11 Naučení neuronové sítě

```
1 // Vytvoreni objektu ucici mnoziny a nasledne nacteni dat ze
   // souboru
2 TrainingData data = new TrainingData();
3 data.ReadTrainFromFile("train_xor.train");
4
5 // Inicializace pocatecnich synaptickych vah a spusteni uceni
6 nn.InitWeights(data);
7 nn.TrainOnData(data, maxIterations, iterationsBetweenReports,
   // desiredError);
```

4.3.5 Spuštění a testování naučené neuronové sítě

Neuronovou síť můžeme spustit a otestovat buď ihned po naučení, nebo ji můžeme po naučení uložit do textového souboru a znovu ji inicializovat a spustit až později. Pro spuštění inicializované naučené neuronové sítě potřebujeme připravit vstupní vektory. Nejjednodušší způsob, jak si vytvořit testovací data pro naši neuronovou síť, je vytvořit si seznam vstupních vektorů (viz Kód 12).

Kód 12 Inicializace neuronové sítě ze souboru a její spuštění

```
1 // Inicializace naucene site ze souboru
2 NeuralNet nn = new NeuralNet("net_xor.net");
3
4 // Vytvoreni seznamu vstupnich vektoru
5 List<float[]> input = new List<float[]>();
6 input.Add(new float[] { 0, 0 });
7 input.Add(new float[] { 0, 1 });
8 input.Add(new float[] { 1, 0 });
9 input.Add(new float[] { 1, 1 });
10
11 // Postupne predani vstupnich vektoru siti ke zpracovani
12 foreach (float[] inVector in input)
13 {
14     // Spusteni site pro kazdy vstupni vektor
```

```
15         float[] output = nn.Run(inVector);
16
17         // Zde muzeme vystupni hodnoty vypsat
18     }
```

4.4 ANN for PHP5

Knihovna ANN for PHP byla využita v bakalářské práci [13] pro vytvoření jednoduché aplikace typu klient-server. Aplikace obsahovala 4 typy řešených úloh (klasifikace, detekce, predikce a logická funkce). Ve své bakalářské práci autor popisoval tuto aplikaci z pohledu uživatele. Nyní si zde rozebereme samotnou knihovnu z pohledu programátora.

4.4.1 Instalace

Nejprve je nutné si nainstalovat WAMP server (na Windows) nebo MAMP server (na OS X). Nainstalováním tohoto nástroje získáme Apache2 server, PHP a MySQL databáze přímo na svém počítači. Po instalaci máme vše připraveno pro vývoj webových aplikací v jazyce PHP. Většina webových serverů má v základu nastaven maximální čas vykonávání jednoho skriptu na 30 sekund. Náročnější neuronové sítě se mohou učit mnohem déle, a proto musíme nastavit tuto hodnotu alespoň na 600 sekund. Tato hodnota se nazývá `max_execution_time` a její nastavení provedeme v souboru `php.ini`, který se nachází ve složce, kde máme nainstalován WAMP.

Instalace samotné knihovny je velmi jednoduchá. Stáhneme si ZIP soubor s aktuální stabilní verzí knihovny a vytvoříme si v kořenové složce našeho projektu složku ANN, do které tento ZIP rozbalíme. Následně knihovnu zahrneme do našeho projektu (viz Kód 13), například v hlavní souboru `index.php`.

Kód 13 Načtení knihovny ANN for PHP5 do projektu

```
1 <?php
2     // Nacteni knihovny
3     require_once 'ANN/Loader.php';
4     use ANN\Network;
5 ?>
```

4.4.2 Inicializace

Máme-li knihovnu nainstalovanu a zahrnutu v našem projektu, můžeme inicializovat první objekt neuronové sítě. Konstruktor přijímá tři celočíselné parametry (viz Kód 14). Prvním parametrem je počet skrytých vrstev neuronové sítě, druhým parametrem

definujeme, kolik neuronů se bude nacházet v každé skryté vrstvě a třetí parametr určuje, kolik bude mít neuronová síť výstupů.

Kód 14 Inicializace neuronové sítě pomocí knihovny ANN for PHP5

```
1 <?php
2     // Definovani topologie site
3     $hiddenLayers = 2;
4     $neuronsPerLayer = 5;
5     $outputs = 1;
6
7     // Vytvoreni objektu neuronove site
8     $nn = new Network($hiddenLayers, $neuronsPerLayer, $outputs);
9 ?>
```

4.4.3 Rozsah vstupních a výstupních dat

Knihovna má implementovány třídy pro úpravu vstupních dat pro neuronovou síť. Tyto třídy nám usnadňují definování učící množiny tím, že nám dokáží například datum nebo vložený vstupní řetězec převést na reálná čísla v rozsahu 0 až 1, tedy přizpůsobená pro vstup do neuronové sítě. Pomocí těchto tříd můžeme také definovat požadovaný rozsah vstupních hodnot. Knihovna má také zvláštní třídu pro klasifikační neuronové sítě. Díky této třídě si můžeme výstup nadefinovat například jako název třídy a knihovna si výstupy přizpůsobí tak, abychom dostali na výstup již název třídy.

Kód 15 Definice rozsahu vstupních hodnot neuronové sítě

```
1 <?php
2     // Zahrnuti jmennych prostoru
3     use ANN\InputValue;
4     use ANN\OutputValue;
5     use ANN>DateInputs;
6     use ANN\StringValue;
7     use ANN\Classification;
8
9     // Vstup vyjadrujici teplotu v rozsahu -30 az 50 stupnu
10    // celsia
11    $temperatureInput = new InputValue(-30, 50);
12
13    // Rozsah vstupu vyjadrujiciho procenta
14    $percentInput = new InputValue(0, 100);
15
16    // Vstup ve formatu data
17    $dateInput = new DateInputs('2016-01-03');
```

```
18 // Vstup ve formátu retezce s maximalni delkou 20 znaku
19 $stringInput = new StringValue(20);
20
21 // Rozsah vystupu vyjadrujiciho promile
22 $promileOutput = new InputValue(0, 1000);
23
24 // Vystup pro klasifikacni sit - 3 tridy
25 $classes = new Classification(3);
26 $classes->addClassifier('setosa');
27 $classes->addClassifier('versicolor');
28 $classes->addClassifier('virginica');
29 ?>
```

Pokud máme nadefinovány rozsahy vstupů a výstupů (Kód 15), můžeme vytvořit učící množinu. Využijeme již vytvořených objektů vstupních hodnot s definovanými rozsahy. Tyto objekty následně převedou vstupy na reálné hodnoty, které je neuronová síť schopna přijmout.

Kód 16 Vytvoření učící množiny pomocí knihovny ANN for PHP5

```
1 <?php
2 // Vytvoreni objektu ucici mnoziny
3 $trainData = new Values;
4
5 // Definice prvnio ucicioho vzoru
6 $trainData->train()
7     ->input (
8         $temperatureInput->getInputValue(-4),
9         $percentInput->getInputValue(65),
10        $dateInput->getFirstDayOfWeek(),
11        $stringInput->getInputValue("Hello world!")
12    )
13    ->output (
14        $promileOutput->getOutputValue(832)
15    );
16 ?>
```

4.4.4 Proces učení neuronové sítě

Pro naučení sítě si musíme nejprve přichystat učící množinu. Máme-li učící množinu malých rozměrů, tedy malý počet vstupních a výstupních vektorů, můžeme učící vzory definovat přímo ve zdrojovém kódu. Knihovna má pro tyto účely implementovanou třídu nazvanou `Values`, která slouží pro vytváření učící množiny, a manipulaci s tímto souborem dat.

Kód 17 Vytvoření učící množiny pomocí knihovny ANN for PHP5

```
1 <?php
2     // Pouziti jmenneho prostoru Values
3     use ANN\Values;
4
5     // Vytvoreni objektu ucicich dat
6     $values = new Values;
7     $values->train()
8         ->input(0, 0)->output(0)
9         ->input(0, 1)->output(1)
10        ->input(1, 0)->output(1)
11        ->input(1, 1)->output(0);
12 ?>
```

Pokud bychom však měli rozsáhlejší učící množinu, je lepší ji definovat do textového souboru s příponou *.dat a následně ji načíst z tohoto souboru přímo do objektu typu Values. Připravenou učící množinu předáme neuronové síti k naučení a spustíme učící proces.

Kód 18 Učení neuronové sítě pomocí knihovny ANN for PHP5

```
1 <?php
2     // Predani ucici mnoziny objektu neuronove site
3     $nn->setValues($values);
4
5     // Spusteni uceni a ulozeni vysledku true/false
6     $trained = $nn->train();
7
8     if($trained)
9         echo "OK";
10    else
11        echo "Uceni nedokonceno";
12 ?>
```

Pokud není dosaženo požadované chyby, uloží se do proměnné \$trained hodnota FALSE, protože síť není plně naučena podle učících vzorů. Stejně tak získáme výsledek FALSE, pokud máme nastaven na serveru nízký maximální čas vykonávání skriptu, protože po tomto čase se skript s učícím procesem ukončí a maximální požadované chyby ještě nebude dosaženo.

4.4.5 Logování změn vah při učení

Knihovna má implementovanou metodu pro logování, tedy zapisování změn vah po každé epoše do CSV souboru spolu s aktuální chybou neuronové sítě. Pro měření času po-

třebného pro naučení neuronové sítě musí být tato funkce vypnutá, protože výrazně navýší výpočetní čas.

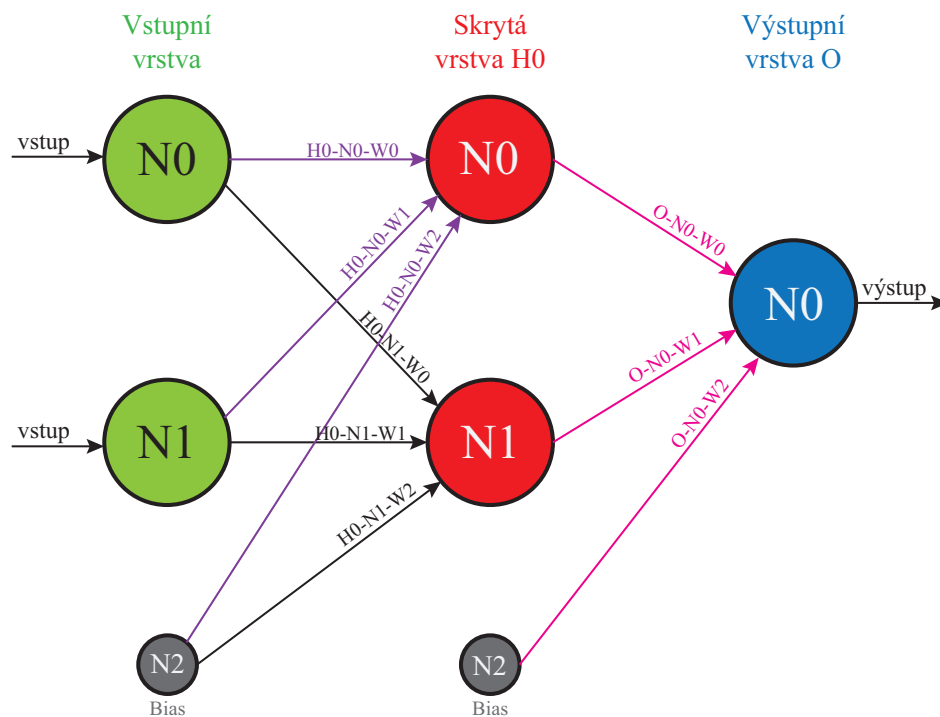
	E	H0 N0 W0	H0 N0 W1	H0 N0 W2	H0 N1 W0	H0 N1 W1	H0 N1 W2	O N0 W0	O N0 W1	O N0 W2
1	1.5	0.64108	0.28679	1.44802	-0.35104	2.01357	-1.45222	-1.6673	0.46671	1.35888
2	1.5	0.6369	0.29804	1.44384	-0.34068	2.03931	-1.44186	-1.66491	0.43052	1.36391
3	1.5	0.63258	0.3092	1.43952	-0.33044	2.06555	-1.43162	-1.66246	0.39371	1.36911
4	0.5	0.63005	0.32073	1.45937	-0.31745	2.09204	-1.40077	-1.74458	0.33415	1.27117
5	1	0.62539	0.33171	1.45472	-0.30695	2.11918	-1.39027	-1.74107	0.29634	1.27767
6	1.5	0.6098	0.3138	1.44361	-0.32356	2.09264	-1.41559	-1.654	0.35271	1.37047
7	1.5	0.6067	0.30706	1.44547	-0.3109	2.09344	-1.41179	-1.6602	0.35382	1.3605
8	1.5	0.59103	0.30706	1.45198	-0.32747	2.09344	-1.41056	-1.64873	0.35072	1.3653
9	1.5	0.57544	0.30706	1.45843	-0.3437	2.09344	-1.40898	-1.63824	0.34725	1.36916
10	0.5	0.57108	0.31837	1.47602	-0.33356	2.12047	-1.38042	-1.71106	0.29074	1.28177

Obr. 4.3 Ukázka logu vah zobrazeného pomocí PHP

Na obrázku 4.3 vidíme ukázkou prvních deseti řádků z CSV souboru logu vah. Tabulka nám podrobně ukazuje vývoj chyby neuronové sítě a jednotlivých vah. Při řešení složitějších problémů, tedy i složitější topologii, nabývá tento soubor obrovských rozměrů, a proto se u rozsáhlejších sítí raději logování nepoužívá.

Nyní si popíšeme, co který sloupec tabulky vyjadřuje, a jak správně hodnoty v tabulce číst a interpretovat. K tomuto účelu nám pomůže také schéma na obrázku 4.4.

- **E** - sloupec zobrazující vývoj chyby neuronové sítě
- **H0 až Hi** - vyjadřuje pořadí skryté vrstvy
- **N0 až Nj** - vyjadřuje pořadí neuronu v dané skryté vrstvě
- **W0 až Wk** - vyjadřuje pořadí synaptické váhy daného neuronu v dané skryté vrstvě, poslední váha vždy vyjadřuje hodnotu biasu



Obr. 4.4 Schéma neuronové sítě znázorňující strukturu logu vah

4.4.6 Spuštění a testování naučené neuronové sítě

V prvním kroku si přichystáme testovací data, což uděláme stejným způsobem jako, když jsme připravovali učící množinu pro učení sítě, pouze vynecháme výstupní hodnoty (viz Kód 19). Poté objekt s testovacími daty předáme neuronové síti a spustíme ji.

Kód 19 Spuštění a testování neuronové sítě v ANN for PHP

```

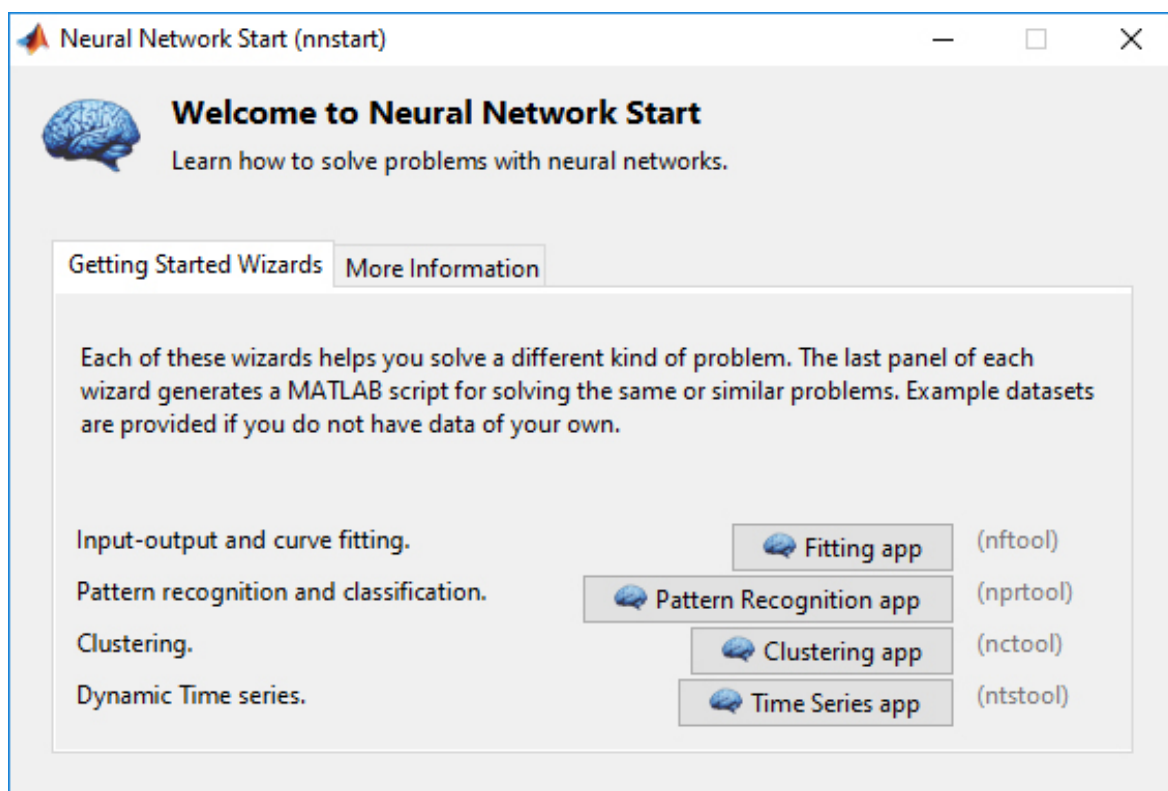
1  <?php
2      // Pouziti jmenneho prostoru
3      use ANN\Values;
4
5      // Vytvoreni objektu testovacich dat a naplneni vstupnimi
        vektory
6      $testData = new Values();
7      $testData
8          ->input(1, 1)
9          ->input(0, 0)
10         ->input(1, 0)
11         ->input(0, 0)
12         ->input(1, 1)
13         ->input(0, 1);
14
15     // Vytvoreni objektu neuronove site
16     $nn->setValues($testData);
17

```

```
18 // Susteni a ulozeni vysledku do pole $output
19 $output = $nn->getOutputs ();
20 ?>
```

4.5 Neuronové sítě v MATLABu

Pro vytváření neuronových sítí v softwaru MATLAB slouží nástroj Neural Network Toolbox. Je možné využívat příkazů tohoto nástroje, tedy konfigurovat neuronovou síť ručně, nebo použít příkaz `nnstart`, který spustí průvodce vytvořením neuronové sítě (viz Obr. 4.5). V tomto průvodci je velmi podrobně a přehledně zpracován každý krok potřebný pro konfiguraci neuronové sítě.



Obr. 4.5 Výchozí okno nástroje Neural Network Toolbox

4.5.1 Průvodce vytvořením neuronové sítě

Průvodce nabízí 4 možnosti představující skupiny úloh, které si můžeme zvolit. Pro každou skupinu úloh máme v úvodním okně tlačítko (Obr. 4.5).

- Fitting app - vstup-výstup a prokládání křivky

- Pattern recognition app - úlohy řešící rozeznávání vzorů a klasifikaci
- Clustering app - objevování přírodních distribucí, kategorií a vztahů mezi kategoriemi (zahrnuje samo-organizující se mapy a kompetitivní vrstvy)
- Time series app - modelování nelineárních dynamických systémů, predikce pomocí sekvenčních údajů

4.5.2 Postup vytvoření neuronové sítě

Nyní si ukážeme postup vytvoření neuronové sítě pro klasifikaci. Použijeme Fisherův Iris dataset. Nejprve si výstupy musíme přizpůsobit tak, aby klasifikace byla jednoznačná a jednoduchá. Proto zvolíme binární výstupy ve stejném počtu jako máme počet výstupních tříd. Výstupy znázorňuje tabulka 4.5.

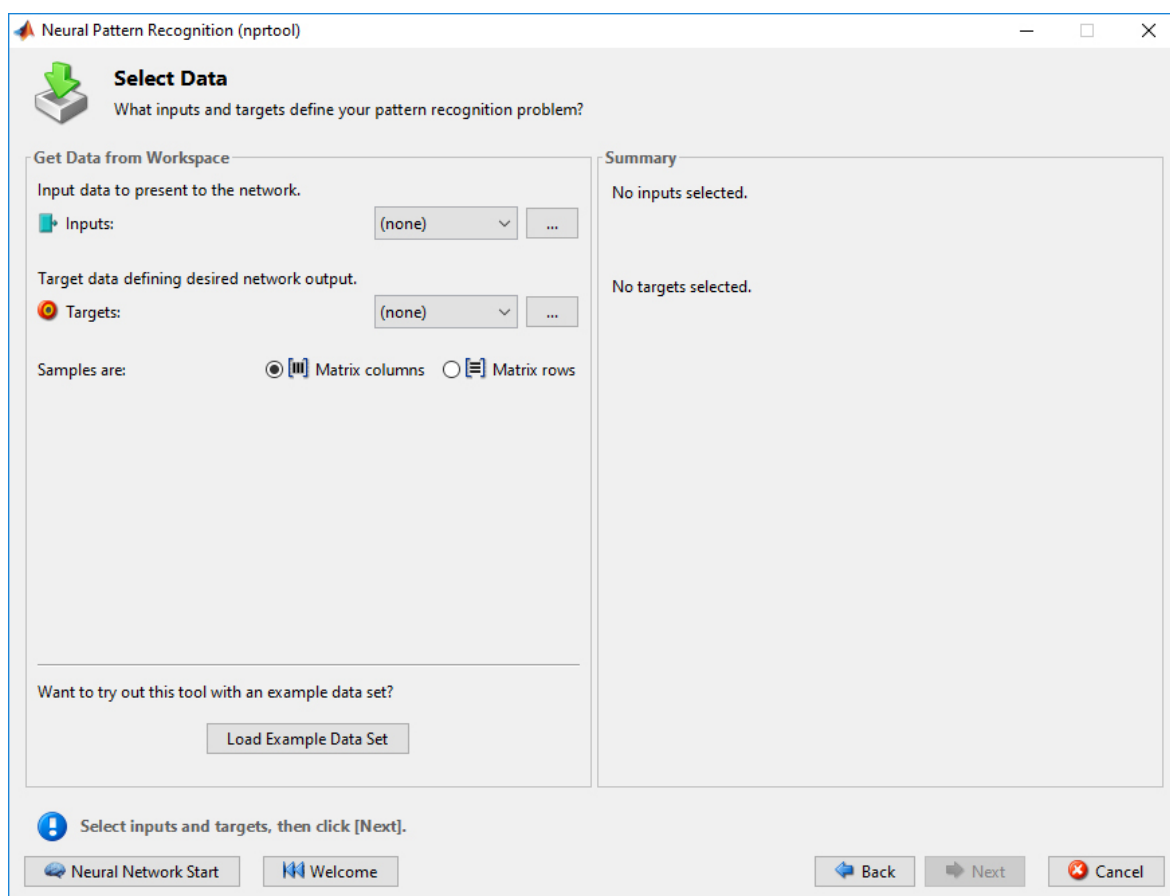
Tab. 4.5 Výstupy neuronové sítě pro klasifikaci

Třída	Výstup 1	Výstup 2	Výstup 3
setosa	0	0	1
versicolor	0	1	0
virginica	1	0	0

Z tabulky 4.5 je vidět, že každé třídě přísluší binární jedna pouze na jednom výstupu. Lze tedy jednoznačně určit třídu, do které neuronová síť vstupní vektor přiřadila. Pokud by se na vstupu objevila kombinace, která by obsahovala jinou kombinaci, než které jsou uvedeny v tabulce, znamenalo by to, že neuronová síť není dostatečně naučena a je potřeba ji doučit.

Po přizpůsobení datasetu, zvolíme z úvodního okna průvodce (Orb. 4.5) tlačítko Pattern recognition app, což znamená, že naše síť bude řešit klasifikační problém. Zobrazí se nám první okno průvodcem, na kterém je vysvětleno jaké problémy tato síť řeší. Je zde také důležitá informace o tom, že tyto sítě jsou pouze dvou-vrstvé dopředné. Tří a vícevrstvé neuronové sítě zde nelze vytvořit. Okno je tedy pouze informační, a proto klikneme na tlačítko Next.

Dalším oknem v průvodci je okno, ve kterém definujeme vstupní a výstupní vektory (Obr. 4.6). Máme možnost hodnoty načíst z XLS souboru, nebo je přímo vložit v příkazovém okně MATLABu do proměnné. Tyto proměnné se následně objeví v rozevřacích seznamech u položek Inputs a Targets (viz Obr. 4.6). Pokud vkládáme data z XLS souboru, uloží se nám také do proměnné. Pod volbou vstupní a výstupní matice musíme ještě zvolit, jestli jsou jednotlivé vzory řádkové nebo sloupcové. Poté můžeme pokračovat tlačítkem Next.

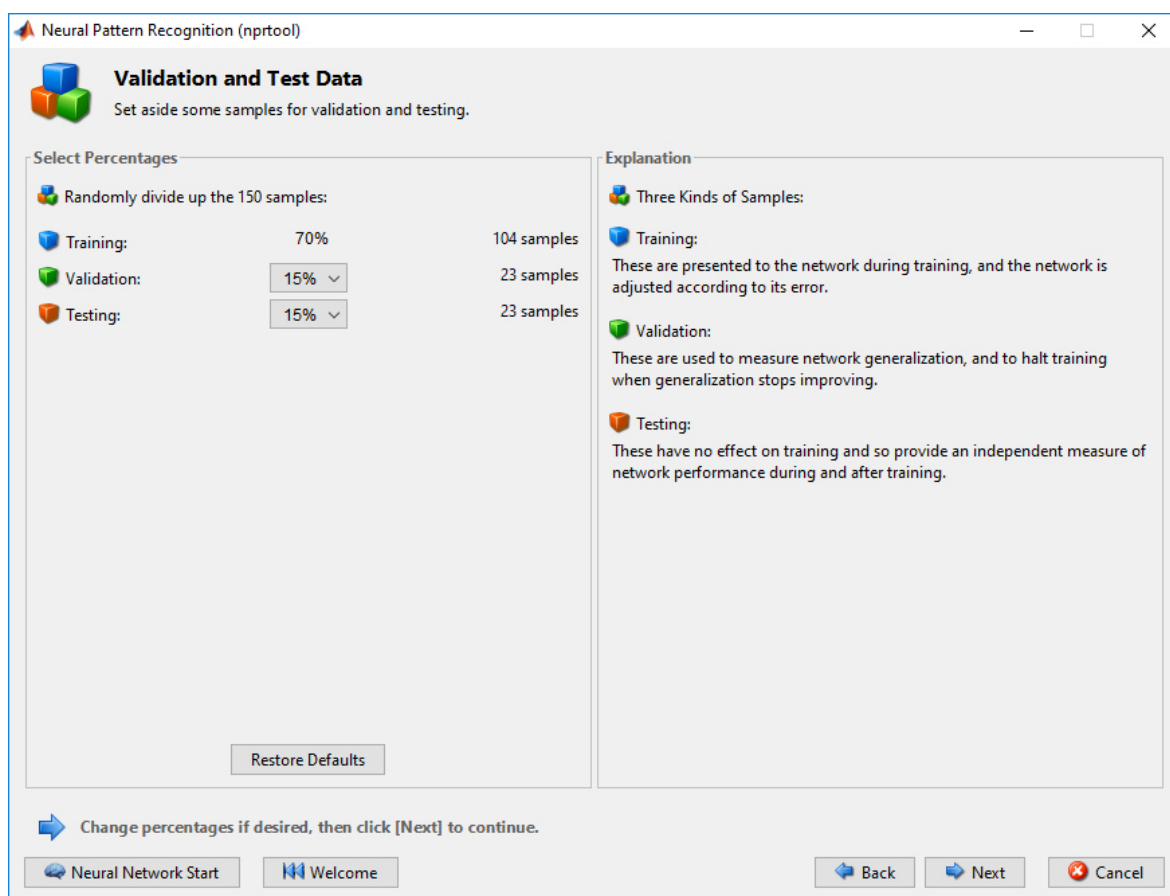


Obr. 4.6 Neural Network Toolbox - definování vstupních a výstupních vektorů

Následující okno slouží pro určení rozložení validačních a testovacích dat (Obr. 4.7). Rozložení bude náhodné v množině učicích dat a učící vzory budou rozloženy v poměru definovaném v procentech.

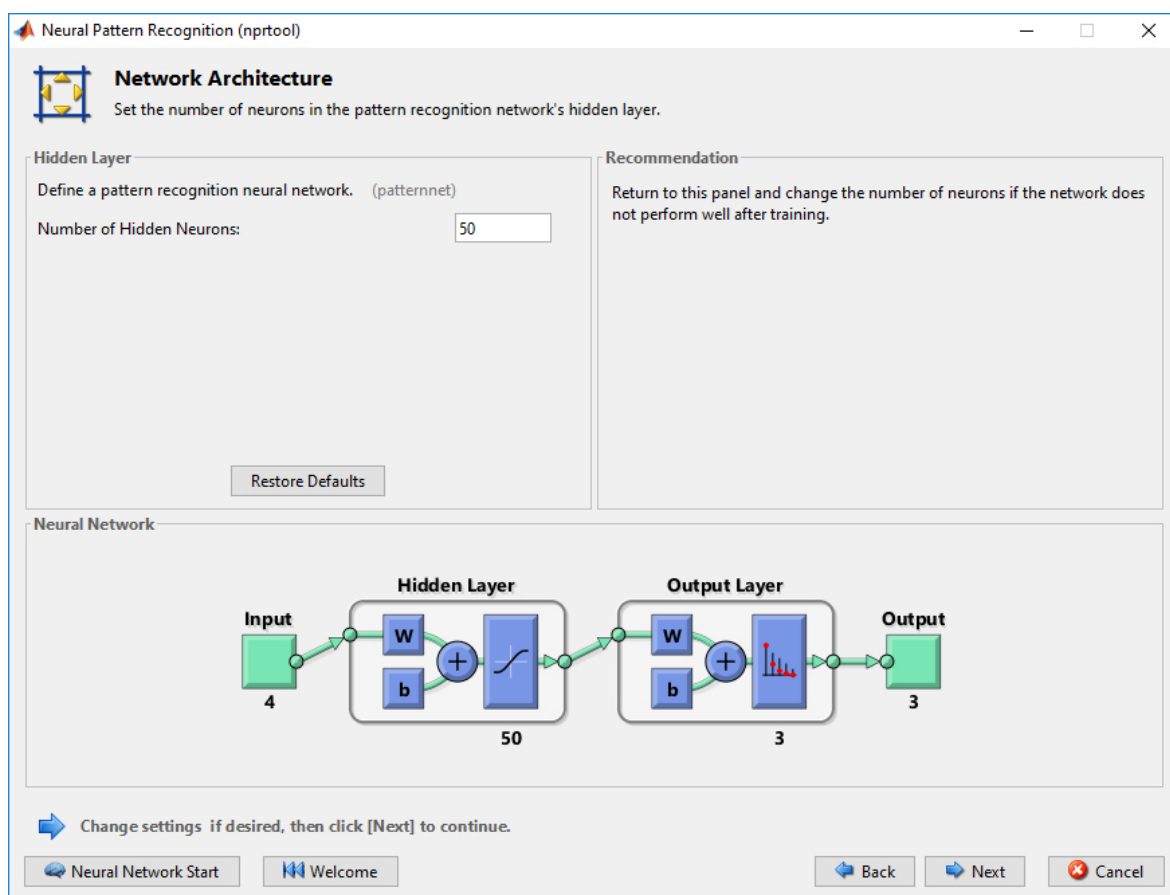
- Training - učící data použítá jako vzory pro učení neuronové sítě
- Validation - ověřovací data slouží pro měření zobecnění sítě a také pro zastavení učení v případě že se výsledek nezlepšuje
- Testing - testovací data, která nemají vliv na samotné učení, pouze se používají pro měření chyby v průběhu učení a po ukončení učení

Při určování procentuálních hodnot dochází k přepočítávání, kolika vzorků se daná hodnota ve skutečnosti bude týkat z celé učící množiny. Pokračujeme opět tlačítkem Next.



Obr. 4.7 Neural Network Toolbox - validační a testovací data

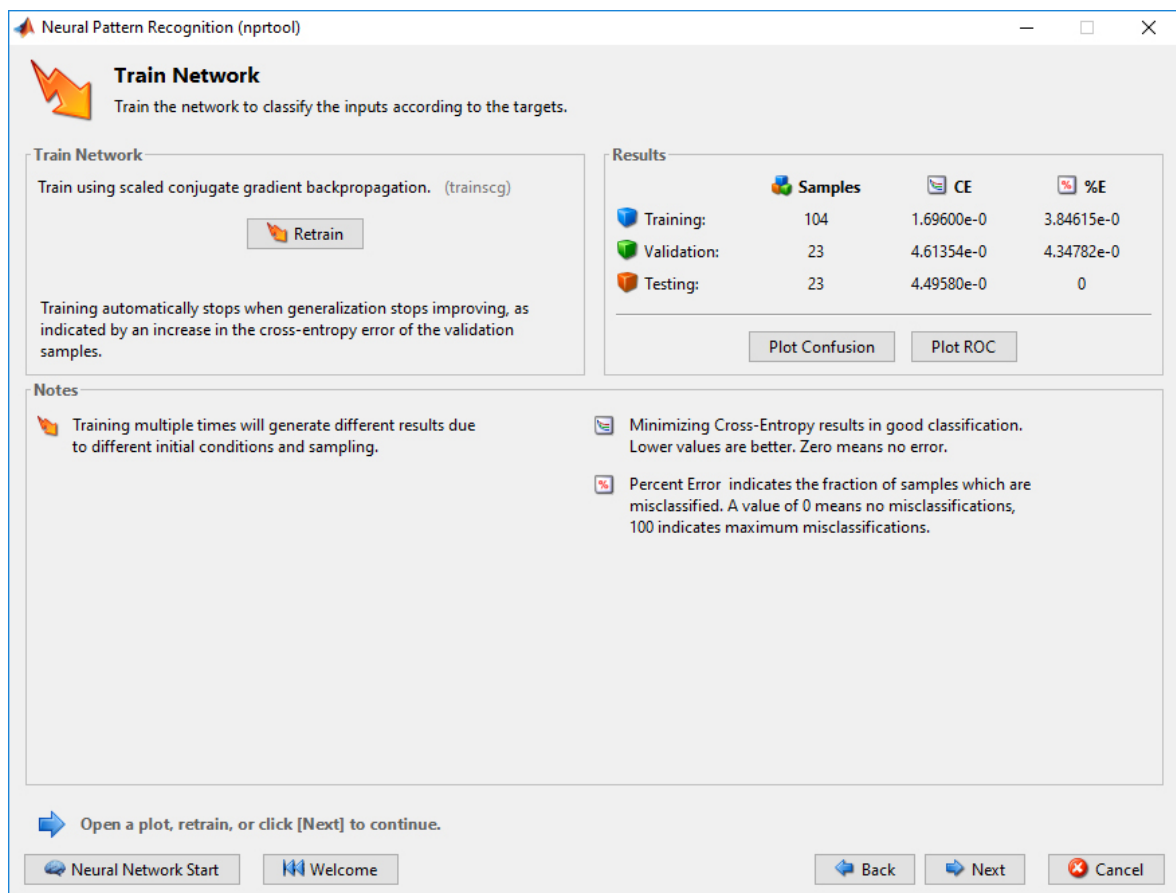
Další okno slouží k nastavení architektury neuronové sítě (Obr. 4.8). Můžeme zde definovat počet neuronů ve skryté vrstvě. Pokud by učení neuronové sítě trvalo příliš dlouhou dobu a síť by nebyla schopna se dostatečně dobře naučit, je možné se v průvodci vrátit k tomuto kroku a změnit počet skrytých neuronů pro zlepšení výsledků učícího procesu. Architekturu neuronové sítě máme vyobrazenu v dolní části okna. Vidíme zde aktuální počet vstupů, výstupů a nastavený počet skrytých neuronů. Počet vstupů a výstupů vychází ze vstupních a výstupních dat, které jsme definovali v okně na obrázku 4.6. Pokračujeme dále tlačítkem Next.



Obr. 4.8 Neural Network Toolbox - architektura sítě

Následuje okno pro učení neuronové sítě (Obr. 4.9). Tlačítkem Train v levé části okna spustíme učicí proces. Výsledky učení vidíme v pravé části okna. Sloupce CE a %E vyjadřují výsledky učení.

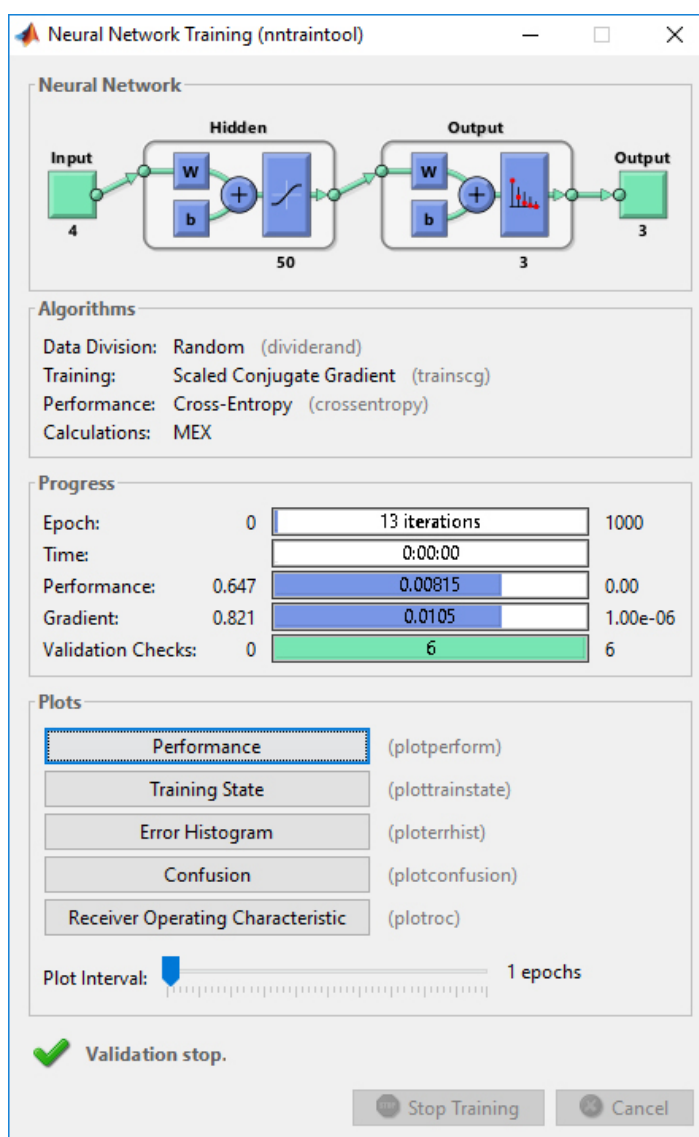
- CE vychází z anglického Cross-Entropy, což vyjadřuje entropii, tedy neurčitost systému, v našem případě neuronové sítě. Účelem učicího procesu je tuto entropii minimalizovat
- %E - procentuální chyba vyjadřuje, kolik procent vzorků bylo rozpoznáno chybně; opět je potřeba tuto hodnotu minimalizovat, tedy dosáhnout v ideálním případě hodnoty 0%.



Obr. 4.9 Neural Network Toolbox - učení sítě

Souběžně s učícím procesem se nám otevře další okno (Obr. 4.10), které nám zobrazuje aktuální architekturu, průběh učení, čas učení, epochy a další údaje. Po ukončení učení je možné si tlačítka ve spodní části okna zobrazit různé grafy.

Okno s přehledem (Obr. 4.10) se někdy otevře na pozadí, takže není na první pohled vidět, ale je možné toto okno zobrazit ze spodní lišty Windows.

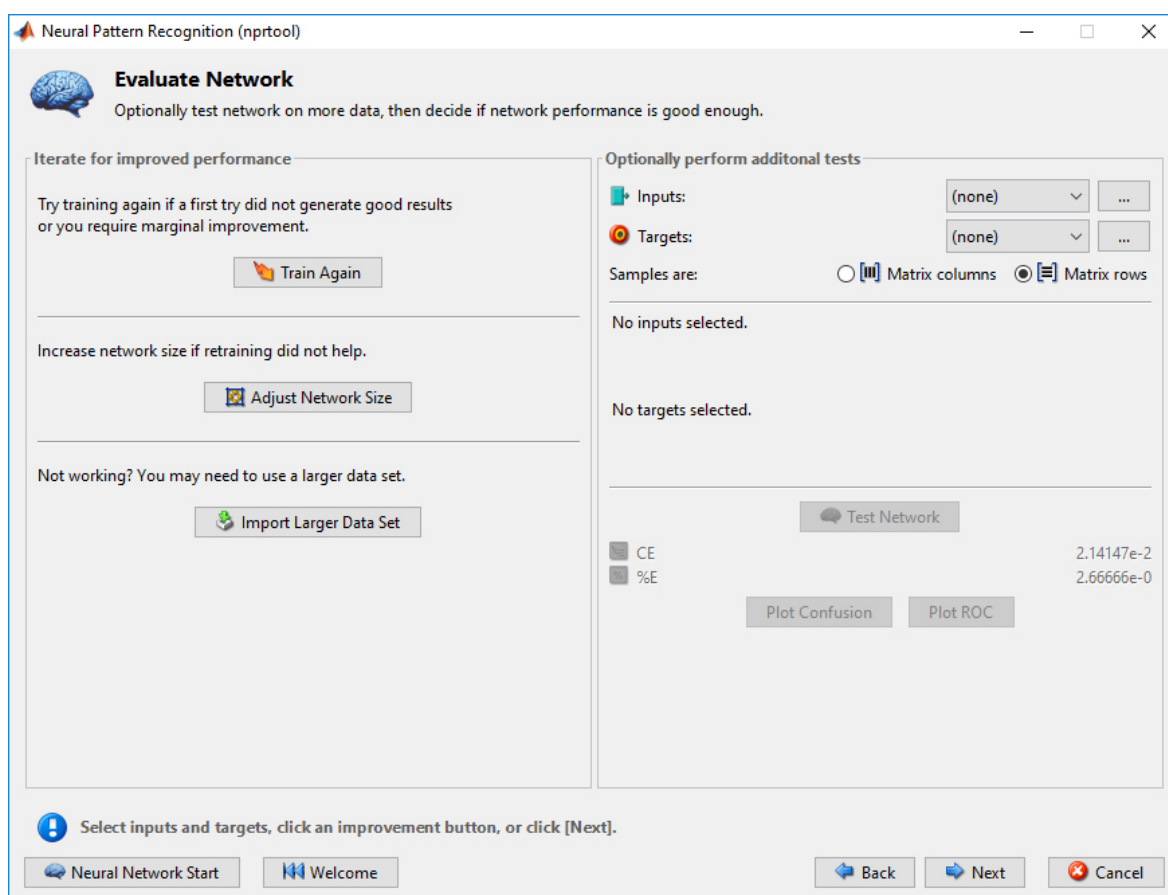


Obr. 4.10 Neural Network Toolbox - přehled průběhu učení

Po naučení neuronové sítě se nám zpřístupní tlačítko Next na okně pro učení sítě (Obr. 4.9) - můžeme tedy pokračovat dále.

Dále pokračujeme na okno pro zhodnocení vytvořené a naučení neuronové sítě (Obr. 4.11). Pokud by nám výsledná neuronová síť nevyhovovala, máme zde nabídku tří možných řešení (levá strana okna).

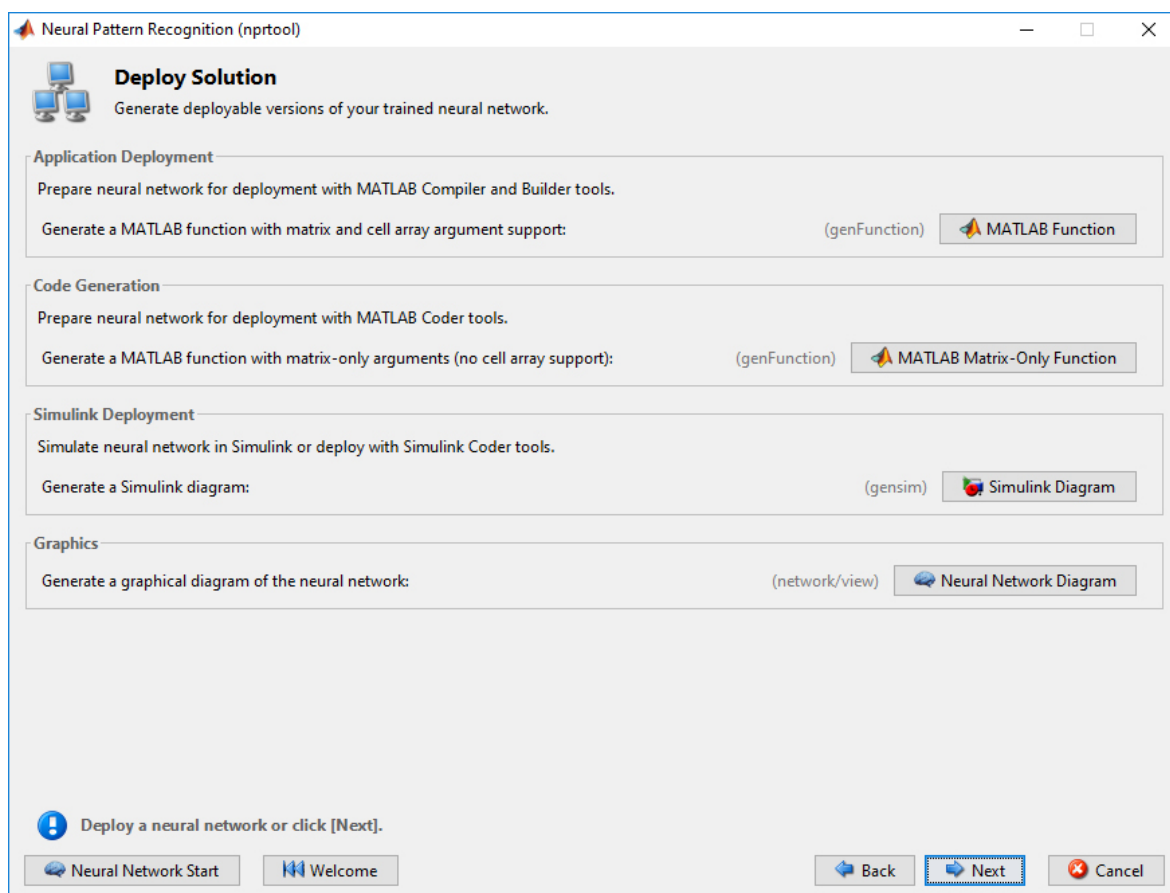
- Učit znovu - pouhé zopakování učení neuronové sítě
- Přizpůsobit velikost sítě - návrat na nastavení architektury (Obr. 4.8)
- Import většího datasetu - návrat na okno, kde jsme definovali vstupní a výstupní vektory, tedy učící vzory (Obr. 4.6)



Obr. 4.11 Neural Network Toolbox - zhodnocení neuronové sítě

Na pravé straně okna máme k dispozici ovládací prvky pro vložení testovacích dat. Je to pouze volitelný krok, ve kterém si svou síť můžeme otestovat na konkrétních připravených datech. Princip je stejný, jako když jsme definovali učící množinu. Pokud jsme s neuronovou sítí spokojeni můžeme pokračovat dále tlačítkem Next.

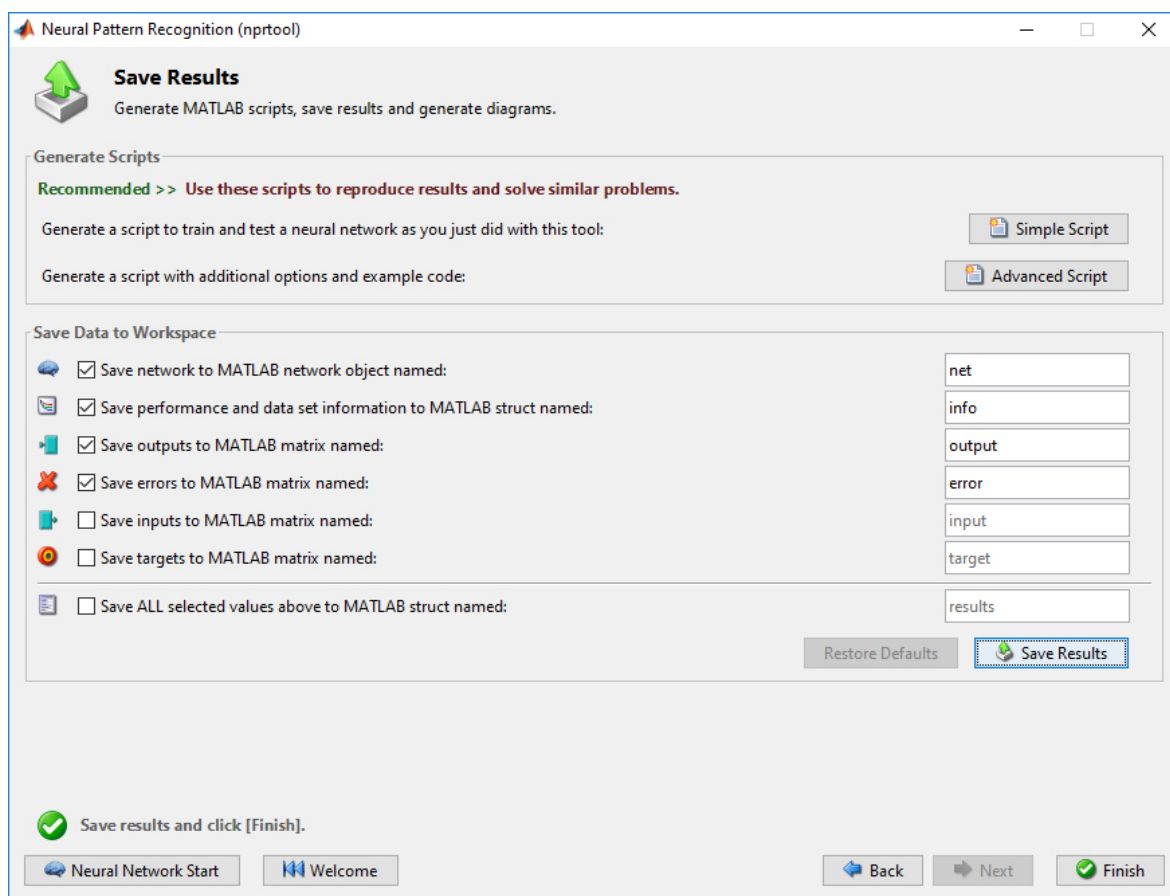
Okno pro nasazení neuronové sítě (Obr. 4.12), kterou jsme vytvořili pomocí průvodce Neural Network Toolboxu, slouží pro přípravu sítě pro použití buď v aplikaci vytvořené v MATLABu nebo pro MATLAB Coder nástroj, což je nástroj, který je schopen převést kód MATLABu do C/C++ kódu. Jsme tedy schopni zde připravit neuronovou síť pro aplikace, které vyvíjíme v C/C++. Dále můžeme v tomto okně vygenerovat diagram v prostředí Simulink nebo pouze zobrazit diagram ve formě obrázku.



Obr. 4.12 Neural Network Toolbox - nasazení vytvořené neuronové sítě

Závěrečné okno celého procesu obsahuje ovládací prvky pro jednoduché uložení výsledků do skriptů nebo pouze do proměnných v MATLABU (do workspace) pro další použití. Máme možnost vybrat co se uloží a také si pojmenovat proměnnou, do které se vybraný objekt uloží.

Použitím tlačítek Simple Script a Advanced Script si můžeme naši neuronovou síť uložit do souboru s příponou *.m, tedy skript pro software MATLAB pro pozdější reprodukci stejné neuronové sítě.



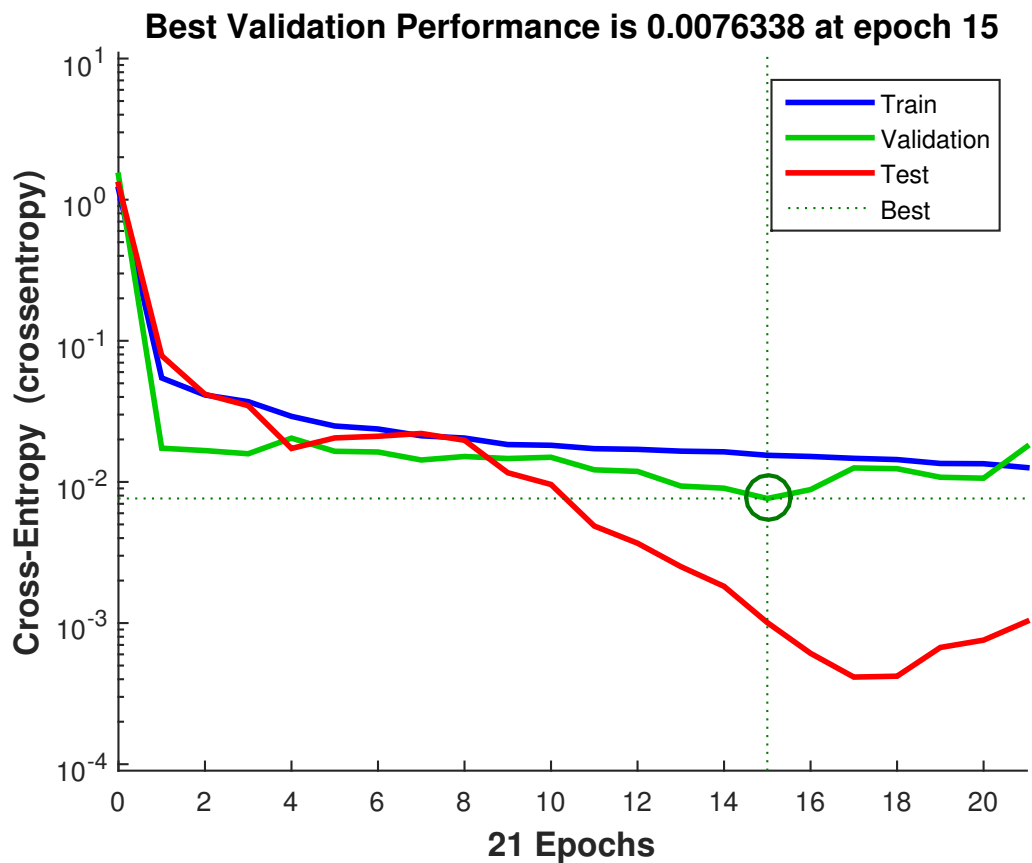
Obr. 4.13 Neural Network Toolbox - uložení výsledků

Necháme si tedy vygenerovat Simple Script. Na konci skriptu jsou zakomentovány příkazy pro vykreslení grafů. Odkomentujeme tyto vykreslení a spustíme skript. Po spuštění skriptu se nám objeví několik oken s grafy, které si můžeme proklikat a prostudovat.

Kód 20 Odkomentování bloku s vykreslením grafů v MATLABu

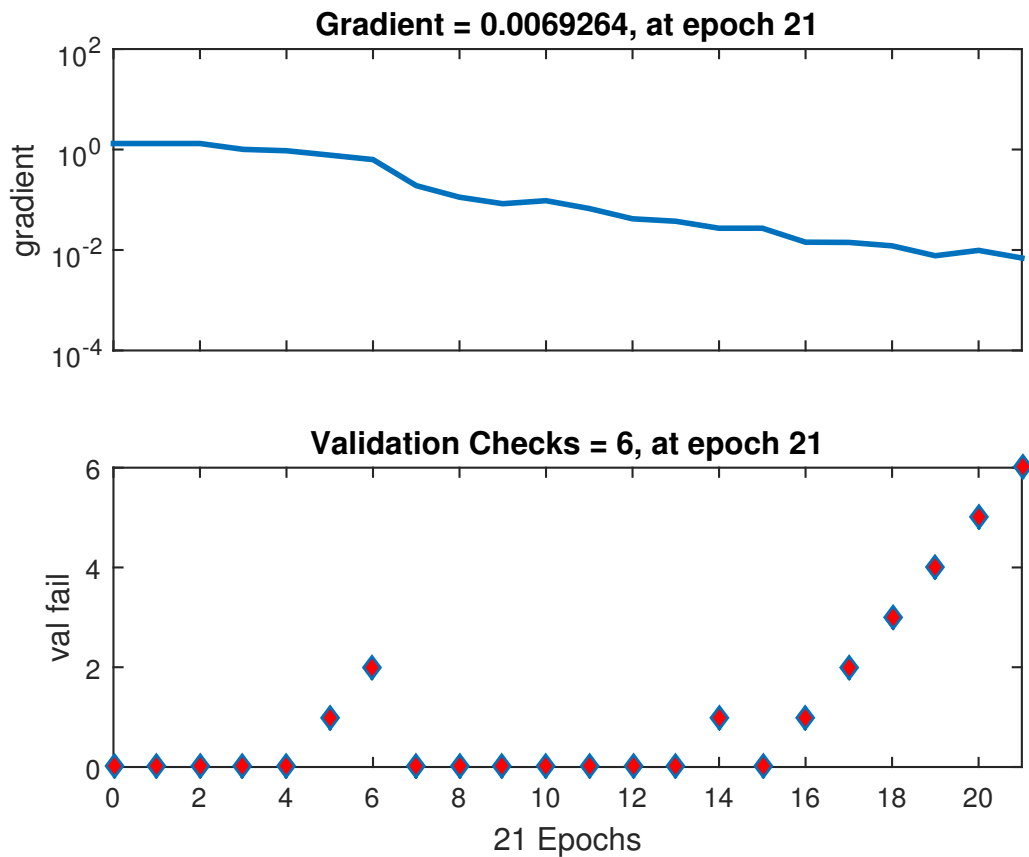
```
1 figure, plotperform(tr)
2 figure, plottrainstate(tr)
3 figure, ploterrhist(e)
4 figure, plotconfusion(t,y)
5 figure, plotroc(t,y)
```

Na grafu na obrázku 4.14 vidíme průběh učícího procesu. Zelená křivka, tedy křivka validace určuje, kdy se má učení ukončit. Výchozí hodnota v MATLABu je nastavena na 6 iterací. To znamená, že pokud od nejmenší chyby v průběhu učení dojde 6 iterací pouze ke zhoršení, ukončí se učení a síť zůstává ve stavu s nejmenší chybou. Z grafu tedy vidíme, že nejlepší hodnota byla získána v 15. iteraci, ale učení skončilo až po 21. iteraci, protože 6 iterací nedošlo k žádnému zlepšení chyby.



Obr. 4.14 Neural Network Toolbox - graf validace neuronové sítě

Dvojitý graf na obrázku 4.15 ukazuje vývoj učení z pohledu validace. Horní část grafu zobrazuje vývoj gradientu a spodní část zobrazuje výsledky validačních kontrol chyby. Jak vidíme ze spodního grafu, až od 15. iterace byla série neúspěšných pokusů o zlepšení chyby, proto bylo učení ve 21. iteraci ukončeno.



Obr. 4.15 Neural Network Toolbox - graf průběhu učícího procesu

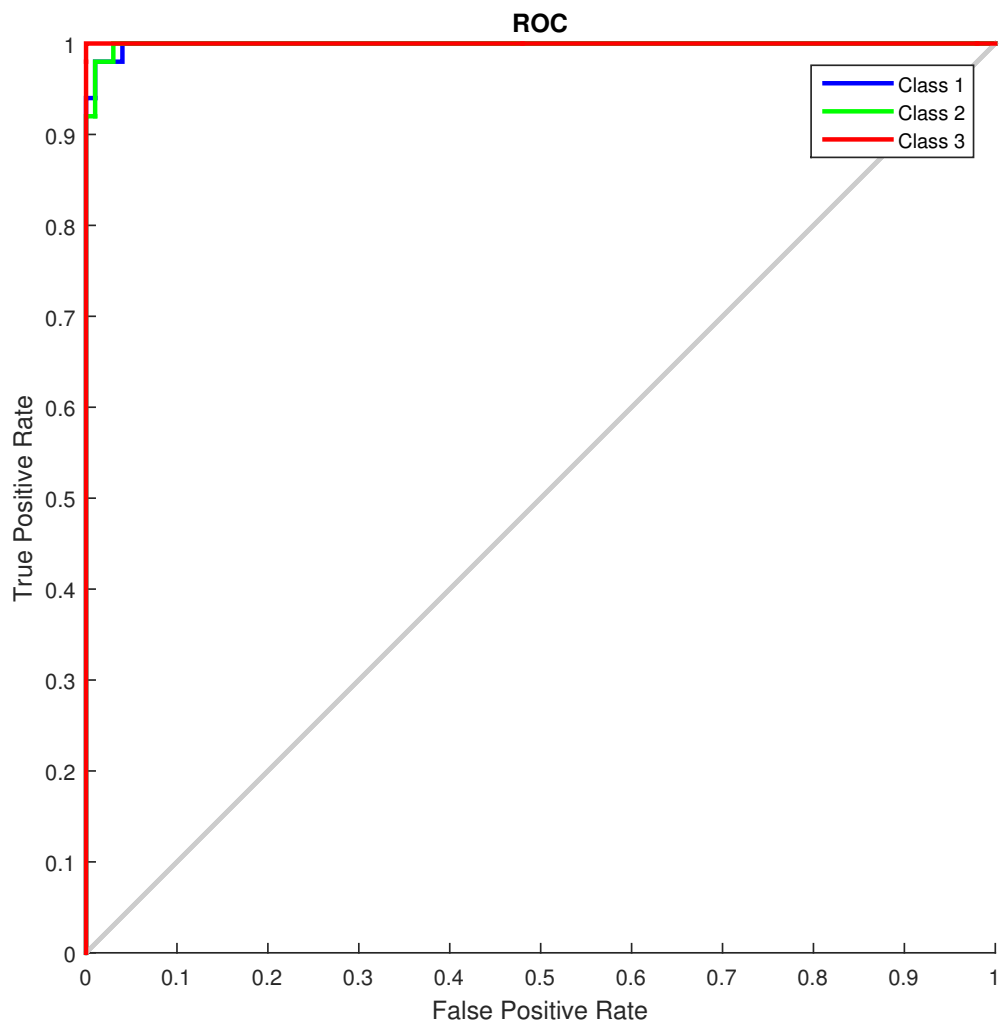
Dalším zajímavým grafem je matice záměny na obrázku 4.16, která nám znázorňuje, kolik vstupních vektorů z učící množiny může naše neuronová síť chybně zařadit, tedy zaměnit třídy. Svislá osa (Output Class) vyjadřuje, jaký je reálný výstup neuronové sítě, tedy klasifikace neuronové sítě. Vodorovná osa (Target Class) vyjadřuje, jaký má být výstup, tedy to, co je definováno v datasetu.

Confusion Matrix

1	49 32.7%	1 0.7%	0 0.0%	98.0% 2.0%
2	1 0.7%	49 32.7%	0 0.0%	98.0% 2.0%
3	0 0.0%	0 0.0%	50 33.3%	100% 0.0%
	98.0% 2.0%	98.0% 2.0%	100% 0.0%	98.7% 1.3%
	1	2	3	
	Target Class			

Obr. 4.16 Neural Network Toolbox - matice záměny

Graf na obrázku 4.17 znázorňuje kvalitu naučení neuronové sítě. Graf obsahuje počet křivek roven počtu tříd, do kterých je možné vstupní vektory přiřazovat. V našem případě tedy tři křivky pro tři třídy. Čím těsněji okolo levého a horního okraje oblasti grafu křivky procházejí, tím kvalitnější je klasifikace touto neuronovou sítí. Zkratka ROC vychází z anglického Receiver Operating Characteristic.



Obr. 4.17 Neural Network Toolbox - ROC graf

4.6 Neuronové sítě v Mathematice

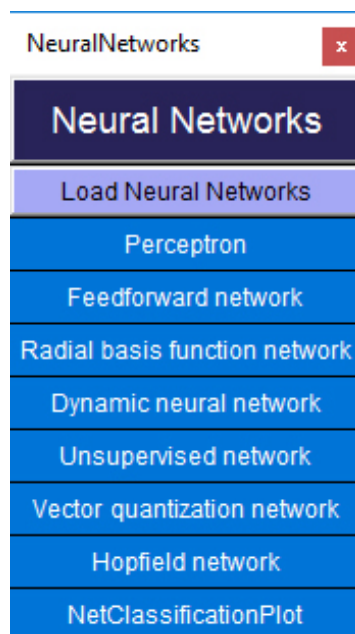
Pro práci s neuronovými sítěmi v softwaru Mathematica máme k dispozici AddOn, tedy přídatný doplněk pro software Mathematica, který se jmenuje Neural Networks. Tento AddOn můžeme zakoupit online přímo na webových stránkách společnosti Wolfram [15].

4.6.1 Instalace AddOnu

Po zakoupení a stažení AddOnu pouhým nakopírováním do určené složky. Jedná se o složku, která slouží přímo pro ukládání doplňků pro tento software. V operačním systému Windows má složka následující umístění:

```
Wolfram Research\Mathematica\10.2\AddOns\Applications\
```

Po nakopírování AddOnu do této složky, můžeme spustit Mathematicu a vytvářet neuronové sítě. Panel tohoto AddOnu spustíme v menu pod položkou Palettes -> Neural Networks. Spustí se nám základní panel neuronových sítí, který vidíme na obrázku 4.18.

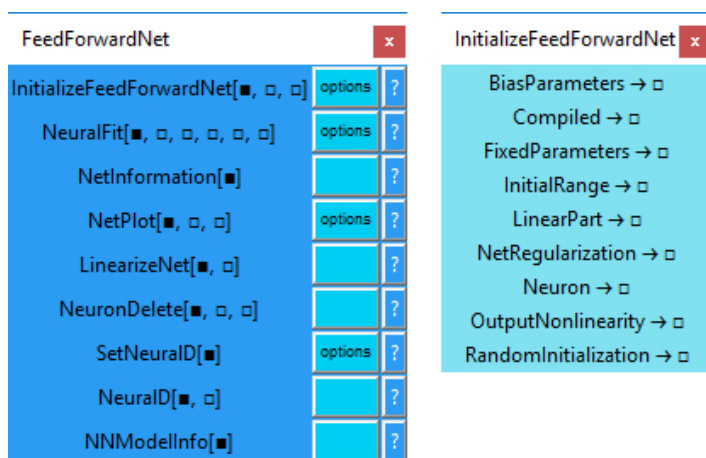


Obr. 4.18 Mathematica
Neural Networks -
základní panel

Základní panel slouží jako rozcestník, na kterém si můžeme zvolit, jakým směrem se vydáme, tedy přesněji k čemu bude naše neuronová síť sloužit. Kliknutím na kteroukoliv

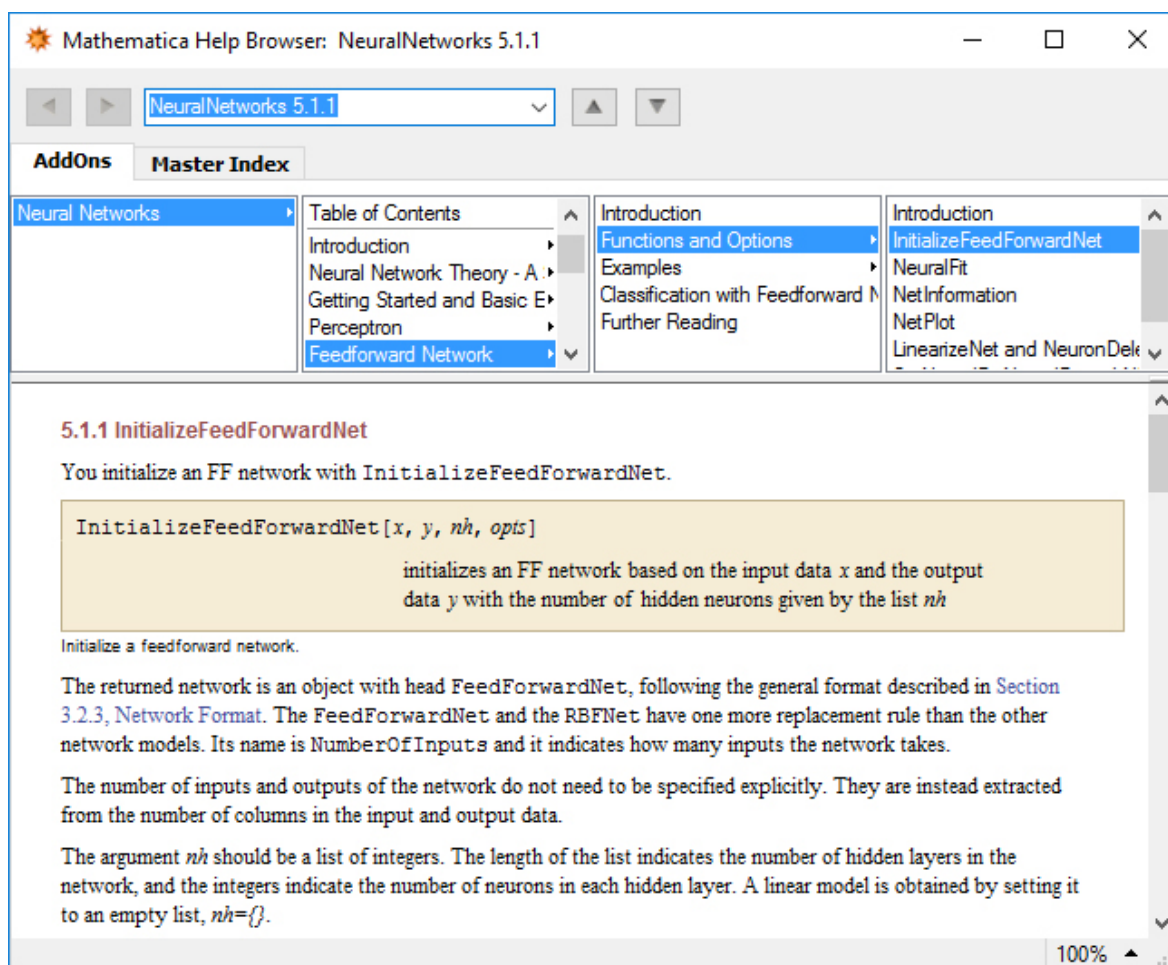
možnost se nám otevře další malý panel s možnostmi pro vybraný typ neuronové sítě, jako vidíme na obrázku 4.19. Na výběr tedy máme z těchto možností:

- **Perceptron** - základní typ neuronové sítě vhodný především pro klasifikační úlohy
- **Feedforward network** - dopředné neuronové sítě, které jsou používány zejména pro aproximace funkcí, klasifikaci a odhady dynamických modelů a časových řad
- **Radial basis function network** - podobně jako dopředné neuronové sítě slouží k aproximaci funkcí, klasifikaci a odhadům dynamických modelů a časových řad
- **Dynamic neural network** - jsou založeny buď na dopředných neuronových sítích nebo na sítích s funkcí radiální báze, jsou používány k odhadu dynamických modelů a časových řad
- **Unsupervised network** - sítě s učením bez učitele, které jsou využívány pro vyhledávání struktur v rozsáhlých datových souborech
- **Vector quantization network** - vektorová kvantizace, využívána zejména pro klasifikační problémy
- **Hopfield network** - Hopfieldovy neuronové sítě, může sloužit jako neuronová síť s autoasociativní pamětí, mohou být též používány ke klasifikaci
- **NetClassificationPlot** - slouží pouze pro vykreslení grafu rozložení klasifikačních vstupních a výstupních dat



Obr. 4.19 Mathematica Neural Networks - panel dopředné neuronové sítě

Na obrázku 4.19 (levý panel) vidíme, že některé funkce, které nám panel poskytuje mají na pravé straně tlačítko Options, které slouží pro otevření dalšího panelu s možnostmi pro tu konkrétní funkci. Panel s možnostmi (pravá část obrázku 4.19) nám poskytuje rozšiřující možnosti manipulace s danou funkcí. U každé funkce je také tlačítko se znakem otazníku. Toto tlačítko nám otevře nápovědu k dané funkci (Obr. 4.20). Nápověda nám poskytuje důležité informace o dané funkci, jako jsou například parametry funkce nebo návratový typ funkce. Nechybí také stručný popis k čemu funkce slouží.



Obr. 4.20 Mathematica Neural Networks - nápověda k funkcím

4.6.2 Postup vytvoření neuronové sítě

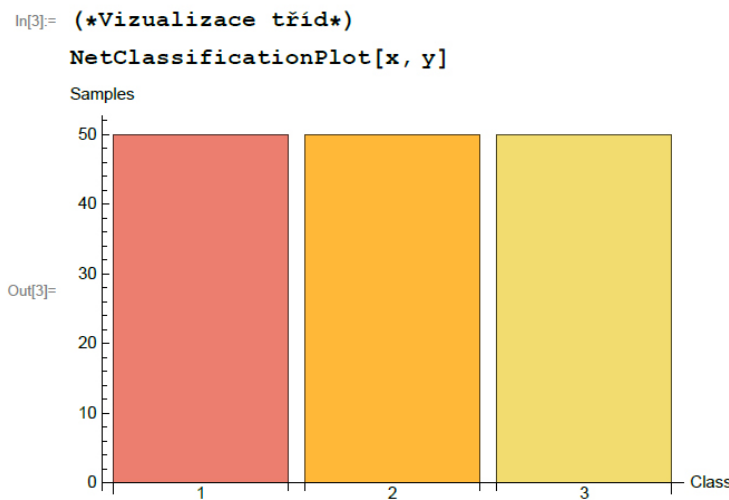
Vytvoříme klasifikační neuronovou síť. Použijeme Fisherův iris dataset, s upravenými výstupy stejně, jako v tabulce 4.5. Použijeme neuronovou síť typu Feedforward network, tedy dopřednou neuronovou síť.

Nejprve dataset rozdělíme na vstupy a výstupy a přiřadíme je proměnným x a y (viz obrázek 4.21).

```
In[1]:= x = {{5.1, 3.5, 1.4, 0.2}, {4.9, 3.0, 1.4, 0.2}, {4.7, 3.2, 1.3, 0.2}, ... };  
In[2]:= y = {{0, 0, 1}, {0, 0, 1}, {0, 0, 1}, ... };
```

Obr. 4.21 Mathematica Neural Networks - dataset

Po přiřazení vstupních a výstupních vektorů do proměnných můžeme data vizualizovat pomocí funkce `NetClassificationPlot`, které předáme jako parametry proměnné x a y obsahující vstupní a výstupní vektory. Máme-li vstupní vektory dvou-rozměrné, zobrazí nám funkce 2D graf s barevnými body podle třídy do které jsou přiřazeny. Fisherův dataset je čtyřrozměrný, a proto máme na grafu (Obr. 4.22) zobrazeny pouze třídy a počet vzorků v datasetu, které do dané třídy spadají.



Obr. 4.22 Mathematica Neural Networks - vizualizace datasetu

Nyní je potřeba definovat architekturu neuronové sítě, tedy topologii. Můžeme ji definovat zvlášť do proměnné jako vektor, nebo tento vektor zapsat přímo do inicializační funkce. Na obrázku 4.23 vidíme, že máme topologii definovanou v proměnné `layers`. Každá hodnota vyjadřuje počet neuronů v dané skryté vrstvě. Vektor tedy představuje 3 skryté vrstvy, kde první skrytá vrstva je složena ze 3 neuronů, druhá ze 4 neuronů a třetí z 5 neuronů. Počet vstupů a výstupů si neuronová síť určí při inicializaci automaticky sama z přiřazené učící množiny, tedy z proměnných x a y .

Nyní jsme tedy schopni funkcí `InitializeFeedForwardNet` inicializovat neuronovou síť. Inicializační funkci předáváme proměnné `x` a `y`, tedy učící množinu, následně proměnnou `layers` definující topologii neuronové sítě a posledním čtvrtým parametrem, jak vidíme na obrázku 4.23, je nastavení parametru `OutputNonlinearity` na hodnotu `Sigmoid`. Tento parametr nastavujeme, pokud potřebujeme nelineární výstup, což u klasifikačního problému potřebujeme. Náš výstup musí jednoznačně určit, do které třídy kombinace vstupních hodnot patří. Sigmoida nikdy nedosáhne hodnot 0 a 1, ale pouze se jim přibližuje a to nám postačuje k tomu, abychom byli schopni určit zda hodnota představuje 0 nebo 1. Inicializovanou neuronovou síť jsme uložili do proměnné `nn`, abychom s ní následně mohli pracovat.

```
In[4]= (*Definice topologie*)
      layers = {3, 4, 5};

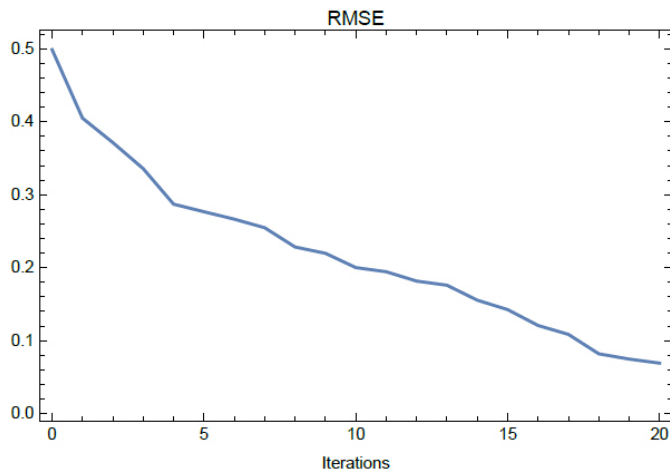
In[5]= (*Inicializace neuronové sítě*)
      nn = InitializeFeedForwardNet[x, y, layers, OutputNonlinearity -> Sigmoid]

Out[5]= FeedForwardNet[{{w1, w2, w3, w4}}, {Neuron -> Sigmoid, FixedParameters -> None,
      AccumulatedIterations -> 0, CreationDate -> {2016, 5, 3, 19, 3, 15.7694198},
      OutputNonlinearity -> Sigmoid, NumberOfInputs -> 4}]
```

Obr. 4.23 Mathematica Neural Networks - inicializace neuronové sítě

Máme tedy inicializovanou neuronovou síť v proměnné `nn`. Nyní neuronovou síť musíme naučit pomocí našeho datasetu, aby byla schopna vstupním vektorům přiřazovat výstupní vektory, tedy třídy. Pro učení neuronové sítě slouží funkce `NeuralFit`, které předáme čtyři parametry jako vidíme na obrázku 4.24. Prvním parametrem je inicializovaná neuronová síť, tedy proměnná `nn`. Následují vstupní a výstupní vektory v proměnných `x` a `y` a posledním parametrem je počet iterací učícího procesu. Tato funkce vrací dvě proměnné, a proto výsledek funkce ukládáme do proměnných `nnTrained` a `fitrecord`. Do první proměnné, tedy do `nnTrained`, se nám uloží naučená neuronová síť, a do druhé proměnné se nám uloží záznam učícího procesu, tedy také vývoj chyby, který se následně vykreslí i do grafu (viz Obr. 4.24).

```
In[6]:= (*Učení neuronové sítě*) {nnTrained, fitrecord} = NeuralFit[nn, x, y, 20];
```



Obr. 4.24 Mathematica Neural Networks - učení neuronové sítě

Naučenou neuronovou síť nyní můžeme otestovat. Do objektu pouze vložíme sadu vstupních vektorů jako je vyobrazeno na obrázku 4.25 a síť spustíme. Výstupem bude sada výstupních vektorů.

```
In[7]:= (*Testování naučené neuronové sítě (2 vstupní vektory)*)
nnTrained[{{5.1, 3.5, 1.4, 0.2}, {5.9, 3.0, 5.1, 1.8}}]
Out[7]= {{0.00576669, 0.00713474, 0.993106}, {0.837411, 0.192043, 0.000836568}}
```

Obr. 4.25 Mathematica Neural Networks - testování neuronové sítě

Jak už bylo řečeno, sigmoida nikdy nedosáhne celočíselných hodnot 0 a 1, což vidíme i na obrázku 4.25, ale z výsledných vektorů jasně vidíme výsledek klasifikace (viz tabulka 4.6).

Tab. 4.6 Výstup neuronové sítě v Mathematice

Výstupní vektor	Zaokrouhleno	Třída
{0,00576669; 0,00713474; 0,993106}	{0; 0; 1}	setosa
{0,837411; 0,192043; 0,000836568}	{1; 0; 0}	virginica

O naší neuronové síti si můžeme zobrazit informace pomocí funkce `NNModelInfo`. Této funkci předáme pouze jeden parametr a tím je naše naučená neuronová síť, kterou

máme uloženu v proměnné `nnTrained`. Funkce mimo jiné vypíše také topologii neuronové sítě. Ve výpisu na obrázku 4.26 vidíme 4 vstupy, 3 výstupy a vektor popisující topologii skrytých vrstev.

```
In[8]:= (*Výpis informací o neuronové síti*)
        NNModelInfo[nnTrained]
Out[8]:= {4, 3, {3, 4, 5}, {LinearPart -> False, Neuron -> Sigmoid,
        OutputNonlinearity -> Sigmoid, BiasParameters -> True}}
```

Obr. 4.26 Mathematica Neural Networks - výpis informací o síti

5 SROVNÁNÍ KNIHOVEN A SOFTWARE

Pro porovnání vybraných knihoven a softwaru byly naprogramovány jednoduché aplikace řešící čtyři předem dané úlohy vycházející ze zvolených datasetů. Aplikace při procesu učení a běhu sítě zapisují data do CSV souboru. Na základě těchto dat knihovny a software porovnám.

5.1 Srovnání knihoven

Pro srovnání byly vybrány tři knihovny z nalezených řešení. Každá je řešena v jiném programovacím jazyce. Jedná se o jazyky C/C++, C# a PHP. Máme možnost použít i knihovnu pro řešení typu klient-server buď v jazyce PHP, nebo v jazyce C#.

5.1.1 FANN C/C++

Knihovna FANN má velmi dobře zpracovanou online dokumentaci. Ke každé funkci lze tedy na webu projektu [7] nalézt všechny potřebné informace pro použití dané funkce. Před použitím knihovny musí programátor věnovat více času studiu dokumentace, než získá povědomí o základních principech knihovny pro její další použití. Nicméně po prostudování dokumentace a pochopení, jak knihovnu používat, je velice jednoduché programovat aplikace využívající tuto knihovnu.

Tab. 5.1 Výhody a nevýhody knihovny FANN C/C++

Výhody	Nevýhody
<ul style="list-style-type: none"> + podrobná dokumentace + vzorové příklady + výběr učících algoritmů + výběr přenosových funkcí + před-připravené datasety + aplikace s grafickým rozhraním + open-source řešení + možnost ukládat vše do souborů + přehledný a dobře komentovaný zdrojový kód 	<ul style="list-style-type: none"> - složitější programování - málo typů neuronových sítí - pouze dopředné neuronové sítě

5.1.2 FANN C#

Tato knihovna vychází z předchozí knihovny FANN. K před-kompilovaným DLL souborům autoři projektu vytvořili tzv. wrapper, který poskytuje rozhraní pro práci s knihovnou v jazyce C#. V tomto jazyce se programuje lépe a rychleji než v C/C++, protože C# už má vyřešeno velké množství často využívaných funkcí a metod ve formě přídavných knihoven. Proto bylo velmi jednoduché manipulovat s učící množinou například v souborech *.xls a *.csv. Celkově je objektový přístup tohoto jazyka jednodušší a lepší než v C/C++. Z mého pohledu se tedy jedná o nejlepší z porovnávaných knihoven.

Tab. 5.2 Výhody a nevýhody knihovny FANN C#

Výhody	Nevýhody
<ul style="list-style-type: none"> + podrobná dokumentace + jednoduché programování + vzorové příklady + výběr učících algoritmů + výběr přenosových funkcí + před-připravené datasety + aplikace s grafickým rozhraním + open-source řešení + možnost ukládat vše do souborů + přehledný a dobře komentovaný zdrojový kód 	<ul style="list-style-type: none"> - málo typů neuronových sítí - pouze dopředné neuronové sítě

5.1.3 ANN for PHP5

Knihovna ANN for PHP5 poskytuje možnost vytvořit aplikaci typu klient-server. Výhodou tohoto jazyka v kombinaci s touto knihovnou je, že ke spuštění výsledné aplikace uživateli stačí internetový prohlížeč, protože veškeré výpočty probíhají na serveru. Bohužel to s sebou přináší i řadu nevýhod. Aplikace využívající neuronových sítí jsou někdy velmi náročné na výpočetní čas a paměť. Výpočetní čas i paměť na serveru, jsou ale pro jeden požadavek omezeny. Proto jsou v této knihovně některé principy zjednodušeny.

Tab. 5.3 Výhody a nevýhody knihovny ANN for PHP5

Výhody	Nevýhody
+ aplikace typu klient-server	- nemá dokumentaci
+ grafické znázornění topologie	- nutno prostudovat zdrojové kódy
+ logování změn vah v průběhu učení	- přenosová funkce pouze sigmoida
+ možnost ukládat vše do souborů	- omezené možnosti v topologii
+ přehledný a dobře komentovaný zdrojový kód	- omezení výpočetního času
	- omezení paměti

5.1.4 Rychlostní testy

U všech vybraných knihoven byly provedeny rychlostní testy. Pro řešení vybraných modelových problémů byly vytvořeny v těchto vybraných knihovnách jednoduché aplikace využívající neuronové sítě. U všech knihoven byla snaha vytvořit totožnou neuronovou síť. Byla nastavena tedy stejná topologie sítě i parametry. Výchozí topologie a parametry jsou v následujícím seznamu:

- **XOR:** 2 vstupy; 1 výstup; 1 skrytá vrstva se 4 neurony; učící koeficient 0,7; požadovaná chyba 0,01; učící algoritmus backpropagation, přenosová funkce sigmoida
- **Složitější log. funkce:** 5 vstupů; 3 výstupy; 1 skrytá vrstva s 10 neurony; učící koeficient 0,7; požadovaná chyba 0,01; učící algoritmus backpropagation; přenosová funkce sigmoida
- **Sinus:** 1 vstup; 1 výstup; 1 skrytá vrstva s 10 neurony; učící koeficient 0,7; požadovaná chyba 0,0000001; učící algoritmus backpropagation; přenosová funkce lineární omezená
- **Iris:** 4 vstupy; 3 výstupy; 1 skrytá vrstva s 20 neurony; učící koeficient 0,7; požadovaná chyba 0,01; učící algoritmus backpropagation; přenosová funkce sigmoida

Tab. 5.4 Testy rychlosti

	FANN C/C++		FANN C#		ANN for PHP5	
	Epochy	Čas [ms]	Epochy	Čas [ms]	Epochy	Čas [ms]
XOR	413	7	92	2	182	153
Log. fce	386	23	6 252	151	21 829	32 425
Sinus	43 509	121	35 329	192	—	—
Iris	1 607	275	1 930	266	—	—

Hodnoty v tabulce testování rychlosti (Tab. 5.4) jsou průměry naměřených hodnot. Uvedené časy jsou v milisekundách. Jak je z tabulky vidět, knihovna FANN v obou

programovacích jazycích pracuje rychle, a dokáže vyřešit všechny vybrané úlohy. Bohužel knihovna ANN for PHP5 se dokázala naučit pouze první dva jednodušší úkoly. U úlohy XOR bylo učení sítě naprosto bez problémů, ale u druhé ho datasetu už byly znát problémy v podobě nedoučené neuronové sítě. Přesto i u druhého úkolu se povedlo naměřit výsledky, ze kterých bylo možno vytvořit závěrečné průměrné hodnoty. U úloh sinus a rozeznávání druhu květiny kosatce nebyla vytvořená neuronová síť vůbec schopna se naučit na požadovanou učící množinu. Knihovna ANN for PHP5 je tedy vhodná pouze pro řešení velmi jednoduchých problémů. Problémy s učením jsou způsobeny nejspíše tím, že není možné nastavit jiný typ přenosové funkce než sigmoidu a také tím, že není možnost volby učícího algoritmu.

Tab. 5.5 Srovnání požadovaných a dosažených chyb

	FANN C/C++		FANN C#		ANN for PHP5	
	Max	Dosažená	Max	Dosažená	Max	Dosažená
XOR	0,01	0,00995	0,01	0,00971	0,01	0
Log. fce	0,01	0,00983	0,01	0,00724	0,01	0
Sinus	$1 \cdot 10^{-7}$	$0,8 \cdot 10^{-8}$	$1 \cdot 10^{-7}$	$0,8 \cdot 10^{-8}$	$1 \cdot 10^{-7}$	—
Iris	0,01	0,00999	0,01	0,00998	0,01	—

V tabulce 5.5 si můžeme prohlédnout rozdíl mezi požadovanou maximální chybou a chybou dosaženou učením neuronové sítě. Z tabulky je vidět, že bylo dosaženo požadované chyby vždy, mimo dva neměřitelné případy u knihovny ANN for PHP5.

5.1.5 Srovnání

Všechny nalezené knihovny jsou schopny pracovat pouze s dopřednými neuronovými sítěmi a učícím algoritmem backpropagation. Jiné typy sítí jsou schopny implementovat pouze pokročilé softwary, jakými jsou například MATLAB a Mathematica.

Tab. 5.6 Srovnání knihoven

	FANN C/C++	FANN C#	ANN for PHP5
Dokumentace	velmi dobrá	velmi dobrá	není
Náročnost programování	náročnější	jednodušší	střední
Open-source	ano	ano	ano
Topologie sítě (graficky)	ne	ne	ano (*.PNG)
Rychlost učení	rychlá	rychlá	nevyhovující

5.2 Srovnání softwaru

Porovnávanými softwary byli původně pouze MATLAB a Mathematica. Při práci s knihovnou FANN [7] byl k těmto dvěma softwarům přidán ještě nástroj FANNTool, což je aplikace s GUI, tedy grafickým rozhraním, založená na knihovně FANN. Tento nástroj má spoustu zajímavých funkcí, a proto byl přidán do srovnání softwaru pro implementaci umělých neuronových sítí.

5.2.1 MATLAB

Neural Network Toolbox pro software MATLAB je velmi propracovaný nástroj. Podrobně zpracovaná online dokumentace pomáhá uživateli vytvořit, uložit a implementovat neuronovou síť společně s grafickým průvodcem. Grafický průvodce uživatele vede krok za krokem s podrobnými popisky nastavitelných parametrů. Vytvořenou síť je možné uložit do skriptu, který lze později pouze opětovně spustit bez nutnosti použít průvodce.

Tab. 5.7 Výhody a nevýhody neuronových sítí v MATLABu

Výhody	Nevýhody
<ul style="list-style-type: none"> + podrobná dokumentace + grafický průvodce + možnost exportovat síť do C/C++ kódu + výběr více typů neuronových sítí + vhodný pro uživatele bez zkušeností s neuronovými sítěmi + generování schémat do Simulinku + vizualizace dat a průběhu učení 	<ul style="list-style-type: none"> - placený software i NN Toolbox - složitější programování bez průvodce

5.2.2 Mathematica

AddOn pro software Mathematica obsahuje spoustu palet pro snadnou implementaci neuronových sítí. Každá paleta obsahuje spoustu funkcí k danému problému, a u každé funkce je možné spustit si nápovědu, kde si uživatel může přečíst, jak funkci použít, jaké přijímá parametry, a jaký je návratový typ funkce. Ke všem funkcím lze nalézt jednoduché příklady použití také v online nápovědě na webu společnosti Wolfram.

Tab. 5.8 Výhody a nevýhody neuronových sítí v Mathematice

Výhody	Nevýhody
<ul style="list-style-type: none"> + podrobná dokumentace + podrobná nápověda ke každé funkci + výběr více typů neuronových sítí než v MATLABu + vizualizace průběhu učení a dat 	<ul style="list-style-type: none"> - placený software i Neural Network AddOn - náročnější pro začátečníky - bez grafického průvodce

5.2.3 FANNTool

Tento nástroj je založen na knihovně FANN. Slouží jako grafické rozhraní ke knihovně pro testovací účely. Tento nástroj byl využit hlavně k testování topologií, učících algoritmů a přenosových funkcí pro konkrétní dataset. Funkce detekce učícího algoritmu a přenosových funkcí vychází ze zadané učící množiny a před samotným učením sítě je schopna navrhnout, který algoritmus nebo přenosovou funkci pro ten daný dataset zvolit.

Tab. 5.9 Výhody a nevýhody neuronových sítí ve FANNTool

Výhody	Nevýhody
<ul style="list-style-type: none"> + intuitivní a jednoduché ovládání + příprava učících dat do souborů + detekce vhodných učících algoritmů + detekce vhodných přenosových funkcí + nástroj je open-source 	<ul style="list-style-type: none"> - k softwaru není dokumentace - omezená manipulace s neuronovou sítí - občasné pády z důvodů přetížení

5.2.4 Srovnání

Srovnání softwaru proběhlo na základě několika kritérií. Kritéria a výsledek srovnání je přehledně vypracován v následující tabulce (Tab. 5.10). V každém z porovnávaných softwarů byly vytvořeny neuronové sítě řešící čtyři problémy vycházejících z vybraných datasetů. Na základě výsledků byla vytvořena srovnávací tabulka z uživatelského pohledu.

Softwary MATLAB a Mathematica jsou na téměř srovnatelné úrovni, ikdyž jsou ve své podstatě velmi odlišné. Nejlépe se pracovalo v softwaru MATLAB. Průvodce vytvořením sítě jednoduše vede uživatele krok za krokem k vytvoření a naučení neuronové sítě. Po vytvoření je možné si vygenerovat skript, který je možné uložit do souboru s příponou *.m a opětovně spustět skript podle potřeby. Skript vygeneruje

spoustu grafů, které hezky názorně ukazují průběh učícího procesu z různých úhlů pohledu. Mathematica má oproti MATLABu velice dobře propracovanu nápovědu k Neural Networks AddOnu z pohledu uživatele typu programátora. Všechny funkce a nápovědy k nim lze spouštět přímo z palety tohoto AddOnu, což ušetří spoustu času s prohledáváním online dokumentace.

Tab. 5.10 Srovnání softwaru

	MATLAB	Mathematica	FANNTool
Dokumentace	velmi dobrá	velmi dobrá	není
Intuitivní ovládání	ano	složitější	ano
Vizualizace dat	ne	velmi dobrá	ne
Vizualizace učení	velmi dobrá	velmi dobrá	základní
Počet typů sítě	4	7	1
Open-source	ne	ne	ano
Rychlost učení	vysoká	vysoká	nízká
Stabilita	stabilní	stabilní	občasné pády

ZÁVĚR

Umělá inteligence je čím dál více populární obor informatiky. Umělé neuronové sítě, které do UI spadají, jsou stále více využívány v praxi v podobě různých počítačových aplikací a softwarů. Tyto softwary v dnešní době řeší různé problémy od rozeznávání obrazů či zvuků, přes detekování různých vzorů a struktur v rozsáhlých datových souborech, až po predikci budoucích stavů systémů na základě předchozích naměřených hodnot.

Většina velkých společností, jako je například Google nebo Microsoft, má pro vývoj neuronových sítí a jejich implementaci svůj vývojový tým, a vyvíjí si je samy. Vznikly však knihovny a softwary, ať už placené či neplacené, které je možné využívat pro různé účely. Mým úkolem bylo nalézt dostupné implementace umělých neuronových sítí a vytvořit jejich přehled a srovnání.

Použitelných a kvalitních řešení není mnoho oproti předpokládanému množství. Některé z nalezených řešení vůbec neobsahovaly dokumentaci, a proto nebylo možné s nimi vůbec pracovat. Jedna ze srovnávaných knihoven (ANN for PHP) dokumentaci nemá, ale do srovnání byla zařazena proto, že má velmi dobře zpracovaný kód s podrobnými komentáři, které posloužily místo dokumentace.

Srovnání bylo provedeno na třech knihovnách a třech softwarech. Každá knihovna i software jsou specifické, a proto bylo podle některých kritérií složité řešení srovnávat. Z tohoto důvodu bylo srovnání rozděleno na dvě části: srovnání knihoven a srovnání softwaru.

Srovnávanými knihovnami byly knihovny FANN C/C++, FANN C# a ANN for PHP5. Díky jednoduchosti a robustnosti jazyka C# je knihovna v tomto jazyce, podle mého srovnání, nejlepší volbou pro implementaci umělých neuronových sítí. Programování je jednodušší, než u dvou dalších knihoven a rychlost učení neuronových sítí je velmi dobrá. Naopak naprosto nevyhovující řešení pro aplikace středního a většího rozsahu je knihovna v jazyce PHP. Aplikace běžící na serveru, který přiděluje výpočetní čas a paměť nejsou schopny řešit pomocí neuronových sítí složitější problémy, pouze ty základní.

Srovnání softwarů ukázalo, že žádná z dostupných knihoven není schopna implementovat umělé neuronové sítě tak dobře, jako softwary MATLAB a Mathematica. Kromě dvou zmíněných softwarů byl do srovnání zařazen také software, který slouží jako vizuální nástroj pro manipulaci s knihovou FANN. Je velmi jednoduše zpracovaný a lze si v něm testovat různé typy neuronových sítí. Naproti tomu softwary MATLAB a Mathematica jsou složitější, ale nepřehledné množství nastavitelných parametrů, typů sítí, učících algoritmů a vykreslování různých grafů dělá z těchto softwarů opravdu vý-

konný nástroj v oblasti neuronových sítí. MATLAB má dokonce možnost vytvořenou neuronovou síť konvertovat do jazyka C/C++ pomocí nástroje MATLAB Code, což přináší velkou výhodu, pokud chceme neuronovou síť použít v aplikaci v tomto jazyce. Oba softwary jsou velmi dobré, nicméně nejlépe se pracovalo s neuronovými sítěmi v softwaru MATLAB.

SEZNAM POUŽITÉ LITERATURY

- [1] Umělá inteligence. In: *Wikipedia: The free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-02-15]. Dostupné z: https://cs.wikipedia.org/wiki/Umělá_inteligence
- [2] ŠÍMA J., NERUDA R.: *Teoretické otázky neuronových sítí*. Vyd. 1. Praha, 1996: Matfyzpress, 390 s., ISBN 80-85863-18-9
- [3] Synapse. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-02-23]. Dostupné z: <https://cs.wikipedia.org/wiki/Synapse>
- [4] ZELINKA, Ivan. *Umělá inteligence I: Neuronové sítě a genetické algoritmy*. 1. vyd. Brno: VUT v Brně, 1998, 126 s. ISBN 80-214-1163-5.
- [5] OPLATKOVÁ, Zuzana. *Neuronové sítě: Úvod do umělé inteligence - 1*. Zlín, 2012.
- [6] *Knihovna (programování)*. In: *Wikipedia: The free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-02-10]. Dostupné z: [https://cs.wikipedia.org/wiki/Knihovna_\(programování\)](https://cs.wikipedia.org/wiki/Knihovna_(programování))
- [7] *FANN: Fast Artificial Neural Network Library* [online]. Dánsko [cit. 2016-02-10]. Dostupné z: <http://leenissen.dk>
- [8] Open NN. *Open NN: Open Source Neural Networks Library* [online]. Barcelona: International Center for Numerical Methods in Engineering, 2014 [cit. 2016-02-10]. Dostupné z: <http://opennn.cimne.com>
- [9] *OpenNN: Neural networks* [online]. Madrid, Španělsko: Artificial Intelligence Techniques, Ltd. [cit. 2016-02-11]. Dostupné z: <http://www.opennn.net>
- [10] *Encog Machine Learning Framework* Heaton Research: Encog [online]. 2008 [cit. 2016-03-16]. Dostupné z: <http://www.heatonresearch.com/encog/>
- [11] *Sourceforge: Neuroph* [online]. 2015 [cit. 2016-03-17]. Dostupné z: <http://neuroph.sourceforge.net>
- [12] *ANN for PHP 5: Artificial Neural Network for PHP 5.x* [online]. Düsseldorf, Německo: Thomas Wien, 2007 [cit. 2016-02-11]. Dostupné z: <http://ann.thwien.de>

- [13] JURÁČEK, Petr. *Umělé neuronové sítě v PHP* [online]. Zlín, 2014 [cit. 2016-02-11]. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky. Vedoucí práce doc. Ing. Zuzana Komínková Oplatková, Ph.D. Dostupné z: <http://theses.cz/id/uy1mj2>
- [14] *Mathematica*. In: *Wikipedia: The free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-02-10]. Dostupné z: <https://cs.wikipedia.org/wiki/Mathematica>
- [15] *Wolfram: Neural Networks* [online]. [cit. 2016-03-18]. Dostupné z: <https://www.wolfram.com/products/applications/neuralnetworks/>
- [16] *Matlab*. In: *Wikipedia: The free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-02-10]. Dostupné z: <https://cs.wikipedia.org/wiki/MATLAB>
- [17] MathWorks. *Neural Network Toolbox* [online]. Massachusetts, USA: MathWorks [cit. 2016-02-15]. Dostupné z: <http://www.mathworks.com/products/neural-network>
- [18] ČÍŽEK, Jakub. *Google uvolnil kód neuronové sítě: Snít může i vaše PC* [online]. 2015 [cit. 2016-03-21]. Dostupné z: <http://www.zive.cz/clanky/google-uvolnil-kod-neuronove-site-snit-muze-i-vase-pc/sc-3-a-179142/>
- [19] *Iris flower data set*. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2016-04-19]. Dostupné z: https://en.wikipedia.org/wiki/Iris_flower_data_set
- [20] *UCI: Machine Learning Repository. Iris Data Set* [online]. [cit. 2016-04-19]. Dostupné z: <http://archive.ics.uci.edu/ml/datasets/Iris>
- [21] KŘIVAN, Miloš. *Úvod do umělých neuronových sítí*. Vyd. 3., přeprac. Praha: Oeconomica, 2014, 44 s. ISBN 978-80-245-2024-7.
- [22] PRATA, Stephen. *Mistrovství v C++*. 4., aktualiz. vyd. Brno: Computer Press, 2013, 1176 s. Bestseller (Computer Press). ISBN 978-80-251-3828-1.
- [23] SCHILDT, Herbert. *Mistrovství - Java*. 1. vyd. Brno: Computer Press, 2014, 1224 s. Mistrovství. ISBN 978-80-251-4145-8.
- [24] VRÁNA, Jakub. *1001 tipů a triků pro PHP*. Vyd. 1. Brno: Computer Press, 2010, 456 s. ISBN 978-80-251-2940-1.

- [25] GUTMANS, Andi, Stig Sæther BAKKEN a Derick RETHANS. *Mistrovství v PHP*
5. Vyd. 1. Brno: CP Books, 2005, 655 s. ISBN 80-251-0799-x.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

UI	Umělá inteligence
UNS	Umělá Neuronová Síť
API	Application Programming Interface (aplikační programové rozhraní)
GUI	Graphical User Interface (grafické uživatelské rozhraní)
LVQ	Learning Vector Quantization (učící vektorová kvantizace)
NARX	Nonlinear Autoregressive exogenous (nelineární autoregresivní exogení síť)
GPU	Graphic Processing Unit (grafický procesor)
DLL	Dynamic Link Library
LGPL	Lesser General Public License
TF	Transfer Function (přenosová funkce)
ROC	Receiver Operating Characteristic

SEZNAM OBRÁZKŮ

Obr. 1.1	Biologický neuron [2]	12
Obr. 1.2	Synapse mezi neurony [3]	14
Obr. 1.3	Umělý neuron [2]	15
Obr. 1.4	Typy přenosových funkcí	17
Obr. 2.1	Příklad topologie neuronové sítě [9]	23
Obr. 2.2	Neuroph Studio [11]	25
Obr. 2.3	Ukázka zobrazení topologie sítě pomocí ANN for PHP	27
Obr. 3.1	Logo Matlab	28
Obr. 3.2	Logo Mathematica 10	29
Obr. 3.3	Ukázka obrazu vygenerovaného pomocí Deep Dream [18]	32
Obr. 4.1	Přenosové funkce knihovny FANN	40
Obr. 4.2	Nástroj FANNTool	44
Obr. 4.3	Ukázka logu vah zobrazeného pomocí PHP	52
Obr. 4.4	Schéma neuronové sítě znázorňující strukturu logu vah	53
Obr. 4.5	Výchozí okno nástroje Neural Network Toolbox	54
Obr. 4.6	Neural Network Toolbox - definování vstupních a výstupních vektorů	56
Obr. 4.7	Neural Network Toolbox - validační a testovací data	57
Obr. 4.8	Neural Network Toolbox - architektura sítě	58
Obr. 4.9	Neural Network Toolbox - učení sítě	59
Obr. 4.10	Neural Network Toolbox - přehled průběhu učení	60
Obr. 4.11	Neural Network Toolbox - zhodnocení neuronové sítě	61
Obr. 4.12	Neural Network Toolbox - nasazení vytvořené neuronové sítě	62
Obr. 4.13	Neural Network Toolbox - uložení výsledků	63
Obr. 4.14	Neural Network Toolbox - graf validace neuronové sítě	64
Obr. 4.15	Neural Network Toolbox - graf průběhu učícího procesu	65
Obr. 4.16	Neural Network Toolbox - matice záměny	66
Obr. 4.17	Neural Network Toolbox - ROC graf	67
Obr. 4.18	Mathematica Neural Networks - základní panel	68
Obr. 4.19	Mathematica Neural Networks - panel dopředné neuronové sítě	69
Obr. 4.20	Mathematica Neural Networks - nápověda k funkcím	70
Obr. 4.21	Mathematica Neural Networks - dataset	71
Obr. 4.22	Mathematica Neural Networks - vizualizace datasetu	71
Obr. 4.23	Mathematica Neural Networks - inicializace neuronové sítě	72
Obr. 4.24	Mathematica Neural Networks - učení neuronové sítě	73
Obr. 4.25	Mathematica Neural Networks - testování neuronové sítě	73
Obr. 4.26	Mathematica Neural Networks - výpis informací o síti	74

SEZNAM TABULEK

Tab. 1.1	Zpracování úlohy pomocí NS a klasického počítače [4]	18
Tab. 4.1	Dataset: Logická funkce XOR	34
Tab. 4.2	Dataset: Složitější logická funkce (5 vstupů, 3 výstupy)	35
Tab. 4.3	Dataset: Goniometrická funkce sinus	35
Tab. 4.4	Dataset: Fisherův Iris dataset [20]	36
Tab. 4.5	Výstupy neuronové sítě pro klasifikaci	55
Tab. 4.6	Výstup neuronové sítě v Mathematice	73
Tab. 5.1	Výhody a nevýhody knihovny FANN C/C++	75
Tab. 5.2	Výhody a nevýhody knihovny FANN C#	76
Tab. 5.3	Výhody a nevýhody knihovny ANN for PHP5	77
Tab. 5.4	Testy rychlosti	77
Tab. 5.5	Srovnání požadovaných a dosažených chyb	78
Tab. 5.6	Srovnání knihoven	78
Tab. 5.7	Výhody a nevýhody neuronových sítí v MATLABu	79
Tab. 5.8	Výhody a nevýhody neuronových sítí v Mathematice	80
Tab. 5.9	Výhody a nevýhody neuronových sítí ve FANNTool	80
Tab. 5.10	Srovnání softwaru	81