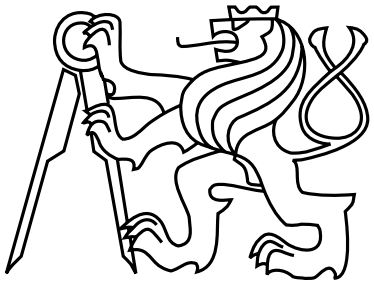
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Demonstrační aplikace pro podporu kurzu neuronových sítí

Adam Činčura

Vedoucí práce: Ing. Zdeněk Buk

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

22. května 2011

Poděkování

Děkuji Ing. Zdeňku Bukovi za spoustu užitečných rad, připomínek a také za pomoc při tvorbě této bakalářské práci.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 24. 5. 2011 .....

Abstract

Demonstrational application created in this bachelor thesis is supposed to complete electronic attachment of lecture notes for the "Neural networks and neurocomputers"course. Application is created in Mathematica program with use of NeuralNetworks library. Every file allows to experiment with specific neural network or its parameteres. All experiments have enclosed explaining commentary. Final application was created by extension and completion of current demonstrational application.

Abstrakt

Demonstrační aplikace vytvořená v rámci této bakalářské práce doplňuje elektronickou přílohu skript pro předmět „Neuronové sítě a neuropočítače“. Aplikace je vytvořena v programu Mathematica s použitím knihovny NeuralNetworks. Každý soubor umožňuje snadno experimentovat s danou neuronovou sítí nebo s jejími parametry. Všechny experimenty jsou doprovázeny vysvětlujícím komentářem. Výsledná aplikace vznikla rozšířením a doplněním stávající demonstrační aplikace.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Problematika neuronových sítí není triviální záležitostí. Pro pochopení principu fungování libovolné neuronové sítě je potřeba porozumět teoretickému fungování sítě. Tohoto porozumění se dosáhne snáze, pokud má student možnost si danou síť vlastnoručně osahat, nastavovat si její parametry, pozorovat výstupy a předkládat síti vlastní data. Z těchto důvodů má předmět „Neuronové sítě a neuropočítače“, v rámci kterého je na FEL ČVUT problematika neuronových sítí vyučována, rozsáhlou elektronickou přílohu skript tzv. Courseware[3].

Součástí Courseware je také demonstrační aplikace v programu Mathematica, kterou jsem v rámci této bakalářské práce přepracoval a rozšířil. Demonstrační aplikace nevyžaduje téměř žádné znalosti programování v systému Mathematica, umožňuje velmi snadno měnit parametry a strukturu zadaných sítí, vizualizovat průběh učení sítě, vizualizovat výstup sítě a upravovat vstupní data. Celá aplikace je doplněna vysvětlujícím komentářem. Věřím, že moje práce pomůže studentům snáze se v neuronových sítích zorientovat.

1.1 Vymezení cíle práce

Cílem práce je provést úpravy a rozšíření stávající demonstrační aplikace. Výstup této práce nahradí stávající aplikaci a bude používán při výuce předmětu „Neuronové sítě a neuropočítače“.

1.2 Struktura práce

V úvodní kapitole popisuji problémy řešitelní pomocí neuronových sítí a uvádím příklady skutečného využití neuronových sítí. V následující kapitole teoreticky popisuji funkci umělého neuronu a věnuji se popisu učicích algoritmů pro neuronové sítě. V nejobsáhlejší teoretické kapitole popisuji vybrané neuronové sítě, konkrétně rozebírám dopřednou neuronovou síť, RBF síť, Hopfieldovu síť, samoorganizující mapu a LVQ síť. V poslední teoretické kapitole rozebírám dělení dat na trénovací a testovací a také zmiňuji křížovou validaci. Následují kapitoly se již věnují popisu demonstrační aplikaci. V první z nich popisuji systém Mathematica a také knihovnu Neural Networks. V druhé již konkrétně rozebírám jednotlivé kapitoly demonstrační aplikace a popisuji experimenty v nich prováděné. V přílohách naleznete pdf verzi vytvoření aplikace a také obsah CD.

Kapitola 2

Umělé neuronové sítě

2.1 Historie

Za počátek vzniku oboru neuronových sítí je považována práce Warrena McCullocha a Waltera Pittse z roku 1943, ve které vytvořili první, velmi jednoduchý model neuronu. V roce 1949 navrhl Donald Hebb učící pravidlo pro neuronové sítě. V roce 1957 vynalezl Frank Rosenblatt tzv. *perceptron*, pro který navrhl také učící algoritmus. Krátce po objevu perceptronu vyvinul Bernard Widrow další typ neuronu ADALINE (ADAptive LINear Element). V této dochází ke značnému rozvoji neuronových sítí, neuronové sítě se začínají vyrábět jako hardwarové prvky. Přesto došlo koncem šedesátých let k výraznému útlumu výzkumu neuronových sítí. Důvodem toho byly nadměrné očekávání jako například vyvinutí umělého mozku. K opouštění výzkumu neuronových sítí vedla také kampaň Marvina Minského a Seymoura Papertema, kteří poukazovali na fakt, že jeden perceptron není schopen rozhodnout jednoduchou logickou úlohu tzv. XOR problém (vylučovací disjunkce). Z toho vyvodili, že výzkum neuronových sítí není perspektivní.

Další výzkum neuronových sítí nastartovala na počátku osmdesátých let americká grantová agentura DARPA (Defense Advanced Reseach Project Agency), který začala výzkum neuronových sítí finančně podporovat. Výsledky se dostavily v podobě objevu algoritmu zpětné propagace chyby (backpropagation) a vynálezu Hopieldovy neuronové sítě. Tyto objevy obnovily zájem veřejnosti i vědců o neuronové sítě. Objev SOM (Self Organizing Map) Teuva Kohonena přinesl novou myšlenku do oblasti neuronových sítí a to učení sítě bez učitele. Další výzkum neuronových sítí probíhá dodnes. Podrobnější historii je možné nalézt v[8].

2.2 Aplikace

- Klasifikace

  Klasifikace, někdy také označována jako rozpoznávání, je jedna z nejjednodušších aplikací neuronových sítí. Úkolem klasifikace je rozhodnout do které ze tříd (kategorií) vstupní vektor patří.

- Predikce

  Predikcí se rozumí předpovídání výstupní hodnoty na základě znalosti jejího průběhu v minulosti. Výhoda neuronových sítí pro predikci spočívá v automatickém naučení pouze z naměřených hodnot, bez nutnosti dodávat další informace a také vysoká schopnost adaptace. Neuronové sítě dokáží odhalit i silně nelineární závislosti. Nevýhodou může být, že se neuronová síť naučí závislost, která je platná pouze v omezeném úseku dat.

- Aproximace

  Aproximace spočívá v přibližném určení hodnoty. Aproximace může například sloužit k určení průběhu funkce na základě navzorkovaných hodnot této funkce. Je výhodné ji použít pokud by výpočet této hodnoty nebyl možný případně velmi neefektivní.

- Shluková analýza

  Cílem shlukové analýzy je nalézt v množině objektů (dat) takové její podmnožiny (shluky), aby si objekty v každém shluku byly vzájemně podobné a zároveň si nebyly příliš podobné s objekty mimo daný shluk. Ke shlukové analýze dat je možné s výhodou použít sítě, které používají učení bez učitele např. samoorganizující se mapy.

- Filtrace slouží

  Filtrace slouží k vyhlazení průběhu signálu, případně k odstranění šumu. Na vstup neuronové sítě se přivede šumem poškozený signál a z výstupu neuronové sítě je získán nezkreslený výstupní signál. Velmi podobnou funkci jako filtrace umožňuje asociace. Na rozdíl od filtrace je při asociaci neuronová síť naučena na datech bez šumu a poté zpracovává zašuměná data.

- Komprese dat

  Komprese dat pomocí neuronové sítě je vhodná především pro kompresi videa například při videokonferencích nebo u televizního vysílání. Ke kompresi jdou nejlépe využít vícevrstvé perceptronové sítě.

2.3 Využití

Neuronové sítě se používají všude tam, kde nám nevadí případná chyba (neuronové sítě dosahují úspěšnosti okolo 95%) a kde je přesný algoritmus náročný na finanční prostředky (hardware) nebo na čas (potřeba rychle rozhodnout). Dále je možné neuronové sítě použít k řešení problému, k jejichž řešení neznáme přesný algoritmus. Následuje několik konkrétních příkladů použití neuronových sítí.

Neuronové sítě byly využity v systému AMT, který optimalizuje rezervace letenek. V první fázi neuronová síť předpovídá poptávku po volných místech v letadlech. Podle těchto předpovědí se navrhuje rozložení letů.

Neuronové sítě jsou používány také ve zdravotnictví k analýze EKG křivky, ve které vyhledává určité složky, čímž dokáže ušetřit velké množství času lékařům, kteří by záznam procházeli ručně. Existuje také systém, který na základě analýzy EKG křivky umožňuje odhadnout nemoc pacienta. Další využití v lékařství je při podávání antibiotik, kdy neuronová

síť pracovala 4× přesněji než lékaři (např. nepředepisovala antibiotika, na které mohl být pacient alergický). Lékařům sloužila jako konzultant.

Další poměrně lákavé využití přináší finančnictví. Pomocí neuronových sítí se predikuje vývoj kurzů měn a také vývoj kurzu akcií. Zatímco při použití klasických metod je úspěšnost predikce cca 55%, při použití neuronových sítí se dosahuje až 75%. Stejným způsobem se pomocí neuronových sítí predikuje spojení a krach podniků. Jiným využitím ve finančnictví je testování, zda daný klient bude schopen splácet půjčku, případně navrhnutí výše půjčky a doby splácení.

Neuronovou síť je možné použít k řízení výrobní linky, případně celého závodu. Nejprve je neuronová síť nainstalována a učí se reagovat na podněty od svého „učitele“ - člověka vykonávajícího stejnou práci, která se požaduje od neuronové sítě. Poté, co je síť dostatečně naučena je schopna reagovat správně i na neznáme podněty. Podobným způsobem byly provedeny pokusy, při kterých se neuronová síť učila řídit automobil. Naučená síť potom řídila podobným způsobem jako její „učitel“ - agresivně brzda, plyn, nebo naopak opatrně.[7]

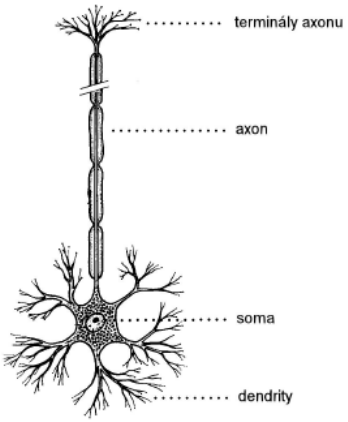
Kapitola 3

Umělý neuron

3.1 Biologický předobraz umělého neuronu

Umělé neuronové sítě vznikly na základě svého biologického předobrazu - nervové soustavy. Nervová soustava se skládá s mnoha vzájemně propojených výkonných buněk - neuronů. Neurony mají za úkol zpracování, přenos a uchování informace. Každý neuron je vzájemně propojen se stovkami jiných neuronů, kterých se v lidském těle nachází přibližně 10^{12}.[9]

Základem biologického neuronu je buněčné tělo s jádrem (soma), z něj vychází jediný rozvětvený výstup zvaný axon. Axon je zakončen takzvanými terminály axonu, ty se dotýkají dendritů ostatních neuronů a slouží k předávání vzruchů ostatním neuronům. Dendrity slouží naopak k přijímání vzruchů od ostatních neuronů. Strukturu biologického neuronu si můžete prohlédnout na obrázku 3.1 Vlastní přenos vzruchů, tedy informací, zprostředkovávají chemické synapse. Tyto synapse mohou vzruch utlumit nebo zesílit. Učení biologické neuronové sítě je založeno na změně reakce synapse na daný vzruch, případně vytvoření nové synapse nebo zánik stávající.

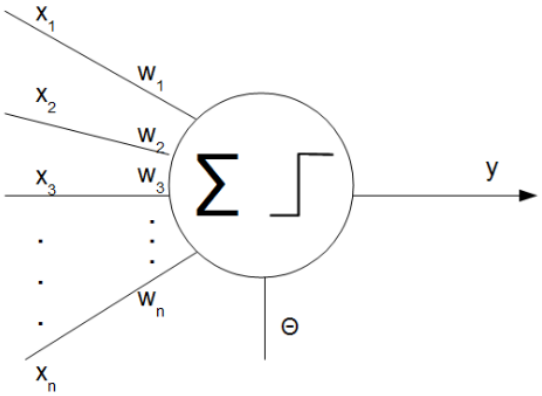


Obrázek 3.1: Biologický neuron[8]

3.2 Struktura umělého neuronu

Základním stavebním kamenem neuronových sítí je model neuronu. Nejrozšířenějším modelem neuronu je McCulloch-Pittsův perceptron, pojmenovaný dle svých vynálezců. Model tohoto neuronu vidíte na obrázku 3.2. Neuron má konečný počet vstupů x_1 až x_n, každý vstup x_i je ohodnocen vahou w_i. Dále je každý neuron vybaven aktivační funkcí S, prahovou hodnotou θ a jedním výstupem y. Výstup neuronu se řídí následující rovnicí:

$$y = S(\sum_{i=1}^{N} w_i x_i + \theta). \tag{3.1}$$



Obrázek 3.2: McCulloch-Pittsův perceptron

Na vstupy neuronu se přivádí data ze vstupní množiny. Data mohou být i binární [0,1], nebo spojitá na intervalu <0,1>. Každý vstup x_i je modifikován vahou daného vstupu w_i. Tyto váhy vstupů se během učení mění, což představuje základ učení neuronu. Významným vstupům, které výrazně ovlivňují výstup neuronu se přiřazuje vyšší váha a naopak méně významným vstupům se váha snižuje.

Práh je hodnota, po jejímž překročení se neuron stává aktivním. Hodnota prahu se spočítá jako vážený součet vstupů. Pokud tento součet nepřekročí prahovou hodnotu, zůstává neuron neaktivní. Práh je často realizován jako další vstup neuronu s konstantní hodnotou $x_0 = 1$, váha w_0 potom označuje samotný práh. Funkce pro výpočet výstupu poté vypadá následovně:

$$y = S(\sum_{i=0}^{N} w_i x_i), \ kde\ x_0 = 1. \tag{3.2}$$

Přenosová funkce je funkce, pomocí které je transformován vnitřní potenciál neuronu do požadovaného oboru hodnot. Pomocí vhodné volby přenosové funkce může neuron produkovat binární nebo spojitý výstup. Pro binární výstup se používají skoková a Heavisideova funkce, pro spojitý výstup se používají saturovaná lineární funkce, Gaussovská funkce, sigmoidní funkce a hyperbolický tangens.[8]

3.3 Proces učení

Při učení dochází v neuronové síti ke změnám, kterými se síť adaptuje na řešení daného problému. Učení je u umělé neuronové sítě realizováno nastavováním vah a prahů jednotlivých neuronů. Během učení nedochází nikdy ke změně struktury sítě (s výjimkou sítě GMDH, která není v tomto textu rozebírána). Před začátkem učení se obvykle nastaví váhy a prahy neuronů na náhodné hodnoty.

U učení umělé neuronové sítě můžeme pozorovat paralelu s učením biologické neuronové sítě, kdy v obou případech dochází k opakovanému předkládání vstupních dat a kontrole výstupu. Tímto opakováním se obě sítě učí dokud nereagují na předávané vstupy správně, nebo jen s akceptovatelnou odchylkou, tedy dokud nejsou naučeny.

Existují dva způsoby učení neuronové sítě:

- Učení s učitelem

  Při učení s učitelem máme k dispozici trénovací množinu, která se skládá ze vstupních vektorů a jim odpovídajícím výstupních vektorů. Učení probíhá postupným předkládáním trénovacích vzorů. Ke každému vzoru ve vypočte odchylka od požadovaného výstupu a na základě této odchylky se upravují hodnoty vah. Váhy se upraví tak, aby při opětovném předložení vzoru byla odchylka od požadovaného výstupu menší. Tato úprava se řídí učicím algoritmem (probrán dále). Změna vah při jednom učicím kroku je obvykle malá. Poté se předloží nový vzor a celý proces se opakuje. Učicí vzory jsou síti předkládány opakovaně a po provedení dostatečného počtu opakování začne síť podávat stabilní výstup v reakci na předkládaný vstup.

- Učení bez učitele

  Při učení bez učitele nemá neuronová síť žádné vnější kritérium správnosti, má k dispozici pouze vstupní vektory. Síť tedy operuje pouze s informacemi, které získala během učení. Algoritmus učení bez učitele je navržen tak, aby hledal ve vstupních datech vzorky se společnými vlastnostmi. Učení bez učitele je také nazýváno samoorganizací.

Existuje také učení jednorázové, kdy je síti předložen vzor a síť si jej zapamatuje. Takováto schopnost učení je u neuronových sítí poměrně neobvyklá. Jednorázového učení je schopna například Hopfieldova síť.

3.4 Proces vybavování

Vybavování je aktivní fází neuronové sítě a zpracovávají se v ní vstupní data. Neuronová síť, stejně jako lidská mysl, pracuje asociativním způsobem.[9] Asociativní paměť pracuje na principu asociovaných prvků např. obličej a jméno člověka, kdy si po spatření člověka vybavíme jeho jméno.

Vybavování je možné rozlišit do dvou variant:

- Autoasociativní vybavování

  Při autoasociativním vybavování dochází k vybavování stejného vektoru, který je předložen. Toto se může zdát nepraktické. Tuto vlastnost ale oceníme pokud je náš vzor

poškozený nebo neúplný a neuronová síť nám ho dovede opravit. K tomuto účelu se také autoasociativní vybavování používá.

- Heteroasociativní vybavování

  Při heteroasociativním vybavování dochází po předložení vzoru k vybavení jiného vzoru, který je předloženým asociován. Typickým použitím této varianty vybavování je klasifikace vektorů do tříd.

3.5 Učicí algoritmy

Nyní již víme že neuronová síť se učí pomocí změn vah vstupů u jednotlivých neuronů. Abychom mohli pozorovat a měřit rozdíly v úspěšnosti sítě v průběhu učení, je potřeba zavést nějakou metriku, pomocí které budeme úspěšnost sítě měřit. V praxi se jako metrika používá střední kvadratická chyba (anglicky Mean Squared Error, odtud skratka MSE). MSE sítě se spočítá pomocí následující rovnice:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - o_i)^2, \tag{3.3}$$

kde n je počet vektorů v trénovací množině, y_i je výstup sítě po přivedení i-tého prvku testovací množiny na vstup a o_i je požadovaný výstup po přivedení i-tého prvku z testovací množiny.

Výše uvedený vzorec je platný pro síť s jedním výstupním neuronem, pokud má síť výstupních neuronů více, sčítá se chyba na všech výstupních neuronech podle následujícího vzorce:

$$MSE = \frac{1}{n}\sum_{j=1}^{m}\sum_{i=1}^{n}(y_{ji} - o_{ji})^2, \tag{3.4}$$

kde m je počet neuronů výstupní vrstvy, n je počet vzorů v trénovací množině, y_{ji} je výstup j-tého neuronu po přivedení i-tého prvku trénovací množiny a o_{ji} je požadovaný výstup j-tého neuronu po přivedení i-tého prvku trénovací množiny.

Ještě výhodnější metrikou pro měření úspěšnosti sítě během trénování je RMSE (Root Mean Squared Error). Její největší předností je možnost přímo ji porovnat s výstupem sítě, protože je ve stejných jednotkách jako výstup. Hodnota RMSE se spočte následovně:

$$RMSE = \sqrt{MSE}. \tag{3.5}$$

Platí, že čím nižší hodnota RMSE, tím lépe je neuronová síť naučena. Učicí algoritmy se v průběhu učení snaží minimalizovat hodnotu RMSE, rozdíl mezi jednotlivými algoritmy tedy spočívá v postupu, jakým je tato hodnota minimalizována. Všechny popisované algoritmy pracují iterativně.

3.5.1 Algoritmus nejvyššího poklesu

Algoritmus začíná s náhodně inicializovaným vektorem vah neuronů a vyžaduje zadat výchozí velikost učicího kroku. V každé iteraci je vektor vah neuronů upraven tak, aby se

posunul ve směru nejvyššího poklesu funkce MSE o velikost učicího kroku, pokud by toto posunutí nesnížilo hodnotu MSE, je velikost učicího kroku snížena na polovinu. Snižobvání probíhá dokud posunutí váhového vektoru nezpůsobí pokles hodnoty funkce MSE. Na závěr iterace je velikost učicího kroku zdvojnásobena, čímž je získána předběžná velikost kroku pro příští iteraci. Algoritmus končí pokud dosáhne minima funkce MSE (může být lokální) nebo vyčerpá přidělený počet iterací.

Algoritmus nejvyššího poklesu je nejjednodušší algoritmus ze zde popisovaných. Potřebuje nejméně výpočetního výkonu a dosahuje mnohem horších výsledků než ostatní popisované algoritmy.[4]

3.5.2 Gauss-Newton algoritmus

Algoritmus začíná s náhodně inicializovaným vektorem vah. Na začátku každé iterace je nastavena velikost učicího kroku na hodnotu 1. Pro určení směru posunu je použita Gauss-Newtonova metoda(rozebrána v[5]), která pro výpočet využívá první a druhé derivace. Ve vypočteném směru je vektor vah posunut o velikost učicího kroku. Posun je uskutečněn pouze pokud způsobí pokles hodnoty funkce MSE, jinak je velikost kroku snížena o polovinu. Poté algoritmus pokračuje další iterací. Algoritmus končí pokud dosáhne minima funkce MSE (může být lokální) nebo vyčerpá přidělený počet iterací.

Gauss-Newton algoritmus je rychlý a spolehlivý algoritmus, který je využívaný v mnoha oblastech k řešení problémů s minimalizací funkcí. Pro řešení problémů v oblasti neuronových sítí nemusí být vždy nejvhodnější volbou, protože může docházet k výpočtům s hodnotami s velkým číselným rozsahem. Pokud k tomuto dojde, bude algoritmus konvergovat velmi pomalu a bude zdržovat celý učicí proces.[4]

3.5.3 Levenberg-Marquardt algoritmus

Algoritmus začíná, stejně jako předchozí algoritmy, s náhodně inicializovaným vektorem vah. Levenberg-Marquardt algoritmus představuje kompromis mezi dvěma předchozími algoritmy. Výsledný směr posunutí vektoru vah je zjednodušeně dán kompromisem mezi směry, které navrhují algoritmus nejvyššího poklesu a Gauss-Newton algoritmus. Namísto velikosti učicího kroku, která je u tohoto algoritmu stále 1, se mění hodnota λ. Na začátku každé iterace se algoritmus pokusí snížit hodnotu λ, pokud by takováto snížená hodnota λ nezaručila pokles MSE, je λ naopak zvětšována dokud nedojde k žádanému poklesu. Hodnota λ určuje, kterému z navrhovaných směrů bude algoritmus přikládat vyšší váhu. Pokud se λ blíží k nule, chová se algoritmus jako Gauss-Newton algoritmus, pokud λ dosahuje vysokých hodnot, chová se jako algoritmus nejvyššího poklesu. Algoritmus končí za stejných podmínek jako Gauss-Newton algoritmus.[4][5]

Levenberg-Marquardt algoritmus představuje pravděpodobně nejlepší volbu trénovacího algoritmu pro neuronové sítě.

3.5.4 Algoritmus zpětné propagace

Algoritmus zpětné propagace začíná stejně jako ostatní algoritmy s náhodně inicializovaným vektorem vah. Před začátkem běhu je algoritmu třeba nastavit délku učicího kroku a

momentum. Algoritmus zpětné propagace se chová velmi podobně jako algoritmus nejvyššího poklesu. Také sleduje směr nejvyššího poklesu funkce MSE, ale liší se v délce učicího kroku, která zůstává po celý běh algoritmu stejná. Momentum slouží jako jakási setrvačnost, která nám říká jak dlouho se budeme držet určitého směru, než zabočíme ve směru nejvyššího poklesu. Díky momentu dokáže algoritmus opustit některá lokální minima.

Algoritmus zpětné propagace je jedním z nejstarších algoritmů pro učení neuronových sítí. I přes použití momenta, které umožní uniknutí z lokálního minima, se nedoporučuje algoritmu zpětné propagace používat. Namísto zpětné propagace se doporučuje použít některý z moderních učicích algoritmů a opakovat trénink pro několik různých, náhodně inicializovaných vektorů vah.[4] Podrobnější popis algoritmu naleznete v[9].

Kapitola 4

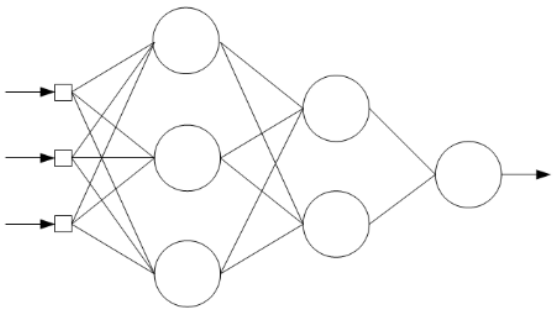
Neuronové sítě

4.1 Dopředná síť

Dopředná neuronová síť se zpětným šířením chyby, někdy také nazývána síť Back-propagation podle pravděpodobně nejznámějšího algoritmu, který je k učení této sítě využíván, patří mezi vícevrstvé neuronové sítě. Dopředná síť má alespoň jednu skrytou vrstvu neuronů. Dopředná neuronová síť, jak již název napovídá, využívá dopředného šíření signálu. Tato neuronová síť je velmi flexibilní v možnostech použití a je poměrně jednoduchá. Díky těmto vlastnostem je jednou z nejpoužívanějších sítí v praxi.

4.1.1 Topologie sítě

Síť je složena z McCulloch-Pittsových neuronů, které využívají jako aktivační funkci sigmoidu. Neurony v dopředné síti jsou tedy spojité. Dopředná neuronová síť se skládá minimálně ze tří vrstev - vstupní, skryté a výstupní. Strukturu sítě skládající se ze čtyř vrstev si můžete vidět na obrázku 4.1. Sousední vrstvy jsou mezi sebou propojeny - každý neuron z vyšší vrstvy je propojen s každým neuronem v nižší vrstvě. První vrstva tzv. vstupní vrstva slouží pouze k rozdělení vstupního signálu a jeho přivedení na neurony první skryté vrstvy. Neurony v ostatních vrstvách sítě jsou již výkonné prvky, na kterých probíhá učení.



Obrázek 4.1: Schéma dopředné neuronové sítě

Počet použitých skrytých vrstev se liší případ od případu. Zvolení vhodného počtu vrstev je většinou provedeno experimentálně. Pro počet neuronů ve skrytých vrstvách existují doporučení. Následující doporučení je pro síť s jednou skrytou vrstvou:

$$N_{skryt} = \sqrt{N_{vst} \cdot N_{vyst}}, \tag{4.1}$$

kde N_{skryt} je doporučený doporučený počet ve skryté vrstvě, N_{vst} je počet neuronů ve vstupní vrstvě a N_{vyst} je počet neuronů ve výstupní vrstvě. Pro síť s dvěma skrytými vrstvami se doporučují následující počty neuronů.

$$N_{skryt-1} = N_{vyst} \cdot \left(\sqrt[3]{\frac{N_{vst}}{N_{vyst}}} \right)^2 \text{ a } \quad N_{skryt-2} = N_{vyst} \cdot \left(\sqrt[3]{\frac{N_{vst}}{N_{vyst}}} \right). \tag{4.2}$$

Tato doporučení představují většinou horní odhady a lépe je určit počty neuronů experimentálně.[9]

Dopředné šíření signálu probíhá následovně. Nejprve je vstupní vektor přiveden na vstupní vrstvu, tato vrstva vstup přenese na první skrytou vrstvu. Na skryté vrstvě dojde k úpravě tohoto signálu podle váhových vektorů jednotlivých neuronů, výstup těchto neuronů je přenesen na další vrstvu. Takto se dostanou data postupně až na výstup poslední (výstupní) vrstvy, odkud je výstup odebrán.

Učení sítě probíhá s učitelem a využívá se k němu jeden z algoritmů popsaných v předchozí kapitole.

4.2 RBF síť

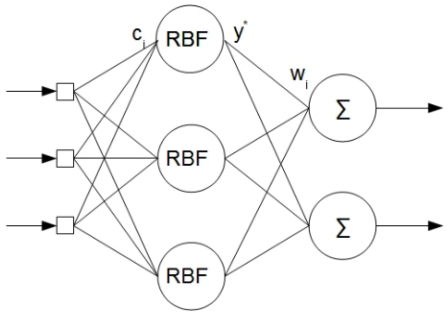
RBF (Radial Basis Function) neuronové sítě jsou známe od 80. let dvacátého století. Jejich rozvoj je spjat s výzkumem tzv. radiálních basických funkcí, které jsou použity jako aktivační funkce neuronů. Síť využívá dopředného šíření signálu a její učení probíhá s učitelem. RBF neuronové sítě představují přímou konkurenci pro dopředné neuronové sítě, protože jsou využívány na stejné úlohy (hlavně klasifikace a aproximace) a jejich učení je výrazně rychlejší než u dopředné sítě.

4.2.1 Topologie sítě

RBF síť se skládá vždy ze tří vrstev neuronů a využívá dopředné šíření signálu. První vrstva je vstupní a slouží pouze k rozvedení signálu do druhé vrstvy. Druhá vrstva je skrytá a obsahuje RBF neurony (popsány níže). Vstupní a skrytá vrstva jsou úplně propojeny, každý vstupní neuron je propojen s každým RBF neuronem. Výstup z RBF neuronů je přiveden na neurony výstupní vrstvy (popsány níže), které z nich vypočtou výstup sítě. RBF vrstva je obvykle úplně spojena s výstupní vrstvou, ale v případě potřeby je možné některý ze spojů vynechat. Topologii sítě přibližuje obrázek 4.2.

4.2.1.1 RBF neuron

RBF neuron se od McCulloch-Pittsova neuronu nejvíce odlišuje v použité aktivační funkci a způsobu výpočtu vnitřního potenciálu neuronu. Vnitřní potenciál neuronu se vypočte ná-



Obrázek 4.2: Schéma RBF neuronové sítě

sledovně:

$$\varphi = \sqrt{\sum_{i=1}^{n}(x_i - c_i)^2}, \tag{4.3}$$

kde φ je vnitřní potenciál neuronu, x_i je i-tý prvek vstupního vektoru a c_i je váha i-tého vstupu RBF neuronu. Vidíme, že vnitřní potenciál udává Eukleidovská vzdálenost předloženého vektoru od vektoru vah RBF neuronu (tento vektor se někdy také nazývá prototyp, protože je reprezentantem určité podmnožiny dat ve formě shluku).

Jako aktivační funkce se nejčastěji používá Gaussova funkce, která je známa ze statistiky. Gaussova funkce má následující rovnici:

$$y^* = e^{-\frac{\varphi^2}{\sigma^2}}, \tag{4.4}$$

kde y^* určuje výstup RBF neuronu, φ je hodnota vnitřního potenciálu neuronu a σ je parametr, který určuje strmost Gaussovy funkce. Gaussova funkce je výhodná protože určuje míru příslušnosti vzoru k prototypu. Pokud je vzor totožný s prototypem, nabývá hodnota Gaussovy funkce hodnoty 1, se vzrůstající vzdáleností vzoru od prototypu hodnota funkce klesá. Kromě Gaussovy funkce se jako aktivační funkce používají ještě další funkce. Jednou z nich je funkce

$$y^* = \varphi \tag{4.5}$$

, kdy se aktivita neuronu se vzrůstající vzdáleností vzoru od prototypu zvyšuje.

4.2.1.2 Neuron výstupní vrstvy

Neurony ve výstupní vrstvě jsou perceptronovského typu a jejich výstup je definován rovnicí:

$$y = \sum_{i=1}^{n}(w_i y_i^*), \tag{4.6}$$

kde w_i je váha daného vstupu, y_i^* je výstup RBF neuronů a y je výstup neuronové sítě.

4.2.2 Učení RBF sítě

Učení RBF sítě se provádí s učitelem. K učení sítě je potřeba stejná množina trénovacích dat, jaká je používána pro učení dopředné sítě. Učení RBF sítě se dělí do dvou fází.[9] V první fázi se nastavují váhové vektory c_i u RBF neuronů, ve druhé fázi se učí neurony výstupní vrstvy.

- 1. fáze

  V této fázi je potřeba určit pozice prototypů. K tomuto účelu se využívá adaptivní K-means algoritmus, používaný především ke shlukové analýze. Tento algoritmus zde bude popsán jen obecně, podrobný popis algoritmu, včetně rovnic, naleznete v[9]. Algoritmus K-means se skládá z následujících čtyř kroků:

  1. Náhodně se inicializují pozice prototypů.
  2. Načte se jeden vzor z trénovací množiny.
  3. Určí se prototyp, který je načtenému vzoru nejblíže. Tento prototyp se posune o η násobek vzdálenosti vzor-prototyp směrem ke vzoru.
  4. Konec pokud $\eta = 0$, nebo po určitém počtu iterací. Jinak se pokračuje bodem 2.

  Velikost η se v průběhu učení snižuje.

  Velikost parametru σ (strmost Gaussovy funkce) se vypočte pro každý shluk (neuron) jako střední kvadratická vzdálenost všech vektorů, patřících do shluku, od prototypu:

  $$\sigma = \sqrt{\frac{1}{Q}\sum_{q=1}^{Q}||\overline{C} - \overline{X}_q||^2}, \tag{4.7}$$

  kde C je ptorotyp daného shluku, Q udává počet prvků patřící do shluku a $\overline{X}_q$ je q/tý vektor shluku. Tato fáze učení probíhá formou samoorganizace tedy učením bez učitele.

- 2. fáze

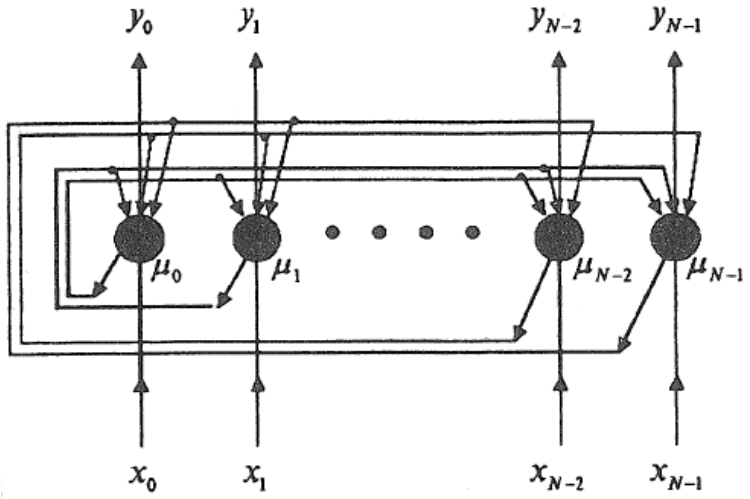
  V této fázi dochází k učení pouze neuronů ve výstupní vrstvě, parametry RBF neuronů zůstávají po 1. fázi nezměněny. V této fázi probíhá učení s učitelem. K učení výstupní vrstvy je použit algoritmus nejvyššího sestupu popsaný v předchozí kapitole.

4.3 Hopfieldova síť

Hopfieldova síť je pojmenována po svém objeviteli Johnu Hopfieldovi. Síť funguje jako autoasociativní paměť a používá se převážné ke zpracovávání obrazových vstupů. Díky svým vlastnostem je síť používána k čištění zašuměných vzorů nebo k rekonstrukci neúplných vzorů. Také je možní ji použít k řešení optimalizačních problémů.

4.3.1 Topologie sítě

Hopfieldova síť se skládá z McCulloch-Pittsových perceptronů, má stejně neuronů jako má vstupů, pokud tedy učíme síť obrázky 8×8 pixelů, má síť 64 neuronů. Každý neuron je pouze jedním svým vstupem napojen na vstup sítě. Výstup z každého neuronu je veden přes synaptické váhy w_{ij} na vstup všech ostatních neuronů, čímž se vytváří zpětnovazební smyčky. Síť má tedy symetrickou strukturu. Protože je Hopfieldova síť autoasociativní, musí mít stejně vstupů jako výstupů, každý neuron v síti má tedy kromě jednoho vstupu také jeden výstup. Hodnoty prahů neuronů jsou nulové, přenosovou funkcí je nejčastěji dvouhodnotová skoková funkce. Strukturu sítě znázorňuje obrázek 4.3.



Obrázek 4.3: Struktura Hopfieldovy neuronové sítě[9]

$x_0, x_1, ..., x_{n-1}$ jsou vstupy sítě, $\mu_0, \mu_1, ..., \mu_{n-1}$ jsou stavy v jednotlivých krocích, které se v následujícím kroku stanou vstupy. Výstupy $y_0, y_1, ..., y_{n-1}$ jsou neaktivní, dokud nedojde k vybavení vzoru. Vybavování je u Hopfieldovy neuronové sítě iterativní proces. Jakmile skončí proces vybavování, jsou výstupy $y_0, y_1, ..., y_{n-1}$ nataveny na hodnoty $\mu_0, \mu_1, ..., \mu_{n-1}$.

4.3.2 Učení Hopfieldovy sítě

Hopfieldova neuronová síť si uchovává informace v podobě matice vah mezi jednotlivými neurony. Matice má dimenzi N×N, kde N je počet neuronů v síti. Tato matice má na diagonále nuly (žádný neuron není spojen sám se sebou) a je diagonálně souměrná protože dva neurony mají mezi sebou vždy nastavenu stejnou váhu (platí že $w_{ij} = w_{ji}$)

Způsob učení a vybavování si ukážeme na jednodušším binárním modelu.

Učení každého vzoru v Hopfieldově síti probíhá jednorázově. Vzor je předložen a síť se ho naučí. Neuronová síť se učí pomocí změny matice vah. Matice vah naučené sítě získáme součtem dílčích matic vah. Pro každý učený vzor získáme dílčí matici vah následujícím způsobem:

$$w_{ij} = \begin{cases} x_i x_j & \text{, pro } i \neq j, \\ 0 & \text{, pro } i = j,\ i \leq 0,\ j \leq N-1 \end{cases}, \tag{4.8}$$

kde w_{ij} je prvek dílčí váhové matice na pozici i, j a x_i resp. x_j je i-tý resp. j-tý prvek na vstupu.

Tímto způsobem jsou předloženy všechny trénovací vzory, kterých je M. Pro získání výsledné matice nám stačí tyto dílčí matice sečíst, čímž získáme výslednou matici vah naučené sítě.

Při učení sítě si můžeme klást otázku kolik vzorů je síť schopna se naučit. Kapacita sítě samozřejmě není neomezená a závisí na počtu použitých neuronů. Experimentálně bylo zjištěno, že počet vzorů musí být nižší než 15% počtu použitých neuronů[9], musí tedy platit vzorec:

$$P \approx 0.15N, \tag{4.9}$$

kde P je počet učených vzorů a N je počet neuronů. Pokud by nebyl tento poměr dodržen, může se stát že si síť bude vybavovat nesmyslné, nenaučené vzory.

Pokud budeme chtít naučit Hopfieldovu síť rozpoznávat všechny znaky české abecedy, kterých je 42 (včetně háčků a čárek), budeme podle předchozího vzorce potřebovat neuronovou síť o minimálně 280 neuronech. Z toho vyplývá, že matice vah bude mít $280^2 = 78400$ hodnot. Tím se dostáváme k největší nevýhodě Hopfieldovy sítě, kterou je nízká kapacita při vysokých paměťových nárocích. Další nevýhodou je nulová odolnost proti natočení nebo posunutí vstupu. Díky těmto vlastnostem se základní Hopfieldova síť téměř nepoužívá.

4.3.3 Vybavování vzoru

Vybavování u Hopfieldovy sítě je založeno na porovnávání vzorů pomocí Hammingovy vzdálenosti. Za správnou odpověď se považuje ten naučený vzor, který má Hammingovu vzdálenost od předloženého vzoru nejnižší. Vybavování v Hopfieldově síti je iterační proces. Po předložení vzoru je tedy nutné počkat, dokud síť nedospěje ke svému výstupu. Vybavování probíhá podle následujícího algoritmu[9]:

1. Nastavení počátečních stavů neuronů podle předloženého vzoru:

$$\mu_i(0) = x_i, 0 \leq i \leq N-1, \tag{4.10}$$

   kde $\mu_i(t)$ je stav i-tého neuronu v čase t a x_i je element vzoru nabývající hodnoty ± 1.

2. Iterování

$$\mu_i(t+1) = f\left[\sum_{j=0}^{N-1} w_{ij}\mu_i(t)\right], 0 \leq i \leq N-1, \tag{4.11}$$

   kde f je skoková aktivační funkce. Tento krok provádíme dokud dochází ke změnám stavů. Jakmile nedojde při iteraci k žádné změně, je algoritmus vybavování ukončen.

3. Na výstupy sítě jsou předány vnitřní stavy jednotlivých neuronů.

$$y_i = \mu_i(t_{posledni}) \tag{4.12}$$

   Nyní můžeme předložit další vzor k vybavování.

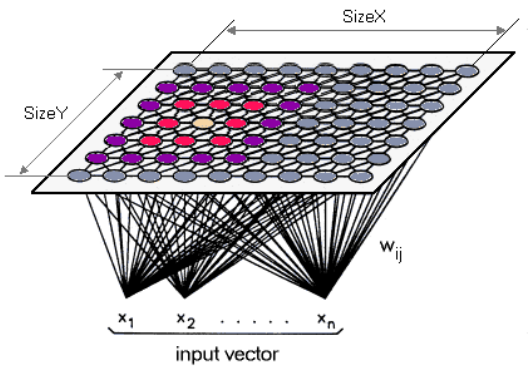
4.4 Samoorganizující mapa

Samoorganizující mapa, zkráceně SOM (Self Organizing Map), byla objevena na počátku osmdesátých let dvacátého století. Jejím objevitelem je finský profesor Teuvo Kohonen, podle něho je také někdy nazývána Kohonenova síť nebo Kohonenova mapa. Samoorganizující mapa pracuje na principu shlukové analýzy. SOM se učí metodou bez učitele a využívá soutěžní strategii učení. V jeden časový okamžik je v síti pouze jeden neuron aktivní a neurony mezi sebou soutěží o to, který bude aktivní. Kohonen také formuloval algoritmus učení samoorganizující mapy, který si klade za cíl vytvoření množiny reprezentantů, kteří mají stejnou pravděpodobnost výběru.[8]

Samoorganizující mapy jsou nejčastěji používány k vizualizováni topologie a hierarchické struktury vícerozměrných dat. Vstupní data jsou pomocí samoorganizující mapy transformována do prostoru s nižší dimenzí (nejčastěji dvoudimenzionálního). Další možností využití je rozpoznávání řeči.[9]

4.4.1 Topologie sítě

Samoorganizující mapa se skládá ze dvou vrstev. První vrstvou je vrstva vstupní, ta pouze distribuuje vstupní vektor na vstupy každého neuronu ve druhé vrstvě. Tyto dvě vrstvy jsou tedy úplně propojeny. Druhá vrstva je někdy nazývána Kohonenova vrstva. Neurony v Kohonenově vrstvě jsou uspořádány (nejčastěji do dvoudimenzionální mřížky). Neurony mají tedy pevně definované sousedy. Každý neuron v Kohonenově vrstvě reprezentuje určitý bod (včetně jistého okolí) ve vstupním prostoru. Neuron v Kohonenově vrstvě má stejný počet vstupů jako samotná SOM. Váhy vstupů u neuronu slouží k uložení pozice bodu, který neuron reprezentuje. Přenosovou funkcí neuronu je Eukleidovská vzdálenost. Výstupem neuronu je Eukleidovská vzdálenost vektoru uloženého ve vahách vstupů a předkládaného vektoru. Výstupem celé sítě je (podle principu vítěz bere vše) ten neuron, který má spočtenou Eukleidovskou vzdálenost nejmenší. Strukturu Kohonenovy sítě ukazuje obrázek 4.4



Obrázek 4.4: Struktura samoorganizující mapy[1]

4.4.2 Učení sítě

Učicí algoritmus se pokouší uspořádat neurony v mřížce do takové podoby, aby mřížka co nejlépe vystihovala vstupní data. Učení probíhá bez učitele a iterativně. Učicí algoritmus samoorganizující mapy je velmi jednoduchý, proto jej zde uvedeme pouze neformálně. Podrobnější popis algoritmu včetně rovnic naleznete v [9].

Nejprve se nastaví váhy vstupů u všech neuronů na náhodné hodnoty. Zvolí se velikost učicího kroku (tímto parametrem je možné ovlivnit jak rychlost učení sítě) a nastaví se velikost okolí neuronu tak, aby do tohoto okolí spadaly všechny ostatní neurony. Okolí neuronu si představme jako skupinu neuronů, které se nachází do určité vzdálenosti od zvoleného neuronu (příkladem okolí mohou být červené neurony na obrázku 4.4). Poté je neuronové síti předložen trénovací vzor. Vypočteme Eukleidovskou vzdálenost všech neuronů od předloženého vzoru a vybereme nejbližší neuron. Tomuto neuronu a všem neuronům v jeho okolí přizpůsobíme váhy směrem k předloženému vzoru o rozdíl odpovídajících si prvků vynásobený velikostí učicího kroku. Zmenšíme okolí a pokračujeme předložením dalšího vzoru.

Proces učení si můžeme představit jako rozprostírání původně zmačkaného papíru do roviny tak, aby pokrýval celý vstupní prostor.

4.4.3 Vybavování vzorů

Vybavování v samoorganizující mapě je velmi snadné, probíhá formou hledání reprezentanta pro předložený vektor. Tímto reprezentantem se stane neuron, jehož váhový vektor nejvíce odpovídá předloženému vzoru. Algoritmus vybavování je následující: Předloží se vstupní vektor, pro všechny neurony se vypočte Eukleidovská vzdálenost od vstupního vektoru a neuron s nejmenší Eukleidovskou vzdáleností vítězí a je vybrán jako reprezentant.

4.5 Learning Vector Quantization (LVQ)

Learning Vector Quantization (LVQ) je rozšířením samoorganizující mapy o učení s učitelem. Toto rozšíření navrhl stejně jako samoorganizující mapu Teuvo Kohonen. LVQ má k dispozici k učicím vzorům jejich správné třídy, bude tedy moci klasifikovat i neznámá data. Struktura LVQ sítě je stejná jako struktura SOM.

Hlavní použití LVQ spočívá v klasifikaci dat.

4.5.1 Učení sítě

V prvním kroku učení se použije standardní algoritmus pro učení samoorganizující mapy. Pomocí tohoto algoritmu dojde k rozmístění reprezentantů (neuronů) v prostoru vstupních dat. V druhém kroku se znovu předloží vstupní data tentokrát i s informací o jejich příslušnosti ke třídě. Pro každý vstupní vektor se určí vítězný (nejbližší) neuron a zapamatuje se informace o příslušnosti vstupního vektoru ke třídě. Po tomto průchodu je pro každý neuron vytvořena tabulka četností tříd, které k tomuto neuronu náleží. Neuronu je přiřazena ta třída, kterou má v tabulce zastoupenu nejčastěji. Posledním krokem učení je zpřesnění pozic neuronů. K tomuto zpřesnění se používají tři algoritmy LVQ1, LVQ2 a LVQ3.

- LVQ1

  U metody LVQ1 dojde k opětovnému předložení vzorů, pro každý vzor je vybrán neuron resp. jeho váhový vektor, který je předloženému vzoru podle Eukleidovské metriky nejblíže. Pouze tomuto vybranému neuronu upravíme váhový vektor podle následující rovnice:

  $$w_i(t+1) = w_i(t) + \eta(t)\left[x_i(t) - w_i(t)\right], \tag{4.13}$$

  pokud je vzor klasifikován dobře(učení),

  $$w_i(t+1) = w_i(t) - \eta(t)\left[x_i(t) - w_i(t)\right], \tag{4.14}$$

  pokud je vzor klasifikován spatně(odučení), kde $0 < i < N-1$, w_i jsou složky váhového vektoru neuronu, x_i jsou složky vstupního vektoru a η je parametr učení. Nejbližší neuron je tedy „přisunut“ k předloženému vzoru nebo je od něj „odsunut“. Ostatním neuronům zůstávají jejich původní váhy.

  Hranice vytvořená pomocí této metody se nachází ve středu spojnice dvou neuronů patřících do rozdílných tříd.

- LVQ2

  Metoda LVQ2 zpřesňuje metodu LVQ1. Na rozdíl od předchozí metody si metoda LVQ2 vybírá dva nejbližší neurony a úpravu vah provádí pouze v případě, kdy jeden neuron klasifikuje správně a druhý špatně. Předpokládejme dva váhové vektory w_i^1 a w_i^2 reprezentující dvě různé třídy T_1 a T_2. Rozhodovací hranice mezi nimi je zde nahrazena okénkem (pásem), které má nenulovou šířku (šířka se určuje experimentálně[9]) a jehož středem prochází hranice. Algoritmus uvažuje modifikaci vektorů pouze pokud vstupní vektor leží v okénku a patří do opačné třídy než vektor, který je mu nejblíže. Váhy neuronů jsou poté upraveny následovně:

  $$w_i^1(t+1) = w_i^1(t) - \eta(t)\left[x_i(t) - w_i^1(t)\right], \tag{4.15}$$

  odsunutí chybně klasifikujícího neuronu od vstupního vektoru,

  $$w_i^2(t+1) = w_i^2(t) + \eta(t)\left[x_i(t) - w_i^2(t)\right], \tag{4.16}$$

  přisunutí správně klasifikujícího neuronu k vstupnímu vektoru, kde x_i jsou složky vstupního vektoru, $0 < i < N-1$ a η je velikost učicího kroku. Ostatní neurony zůstávají beze změny.

  Algoritmus zprvu skutečně zlepšuje pozici hranice mezi třídami, po větším počtu iterací se však začne pozice hranice zhoršovat. Proto je výhodné použít tento algoritmus jen pro menší počat iterací (např. 10000)[3].

- LVQ3

  Z důvodu nestability metody LVQ2 pro pro velký počet iterací vznikl algoritmus LVQ3, který tento nedostatek odstraňuje. Algoritmus LVQ3 rozšiřuje algoritmus LVQ2 o pravidlo, které zajišťuje přibližování správně klasifikujících neuronů k tréningovým vzorům.Rozšiřující pravidlo je následující:

V případě, kdy předložený vzor i oba nejbližší neurony mají stejnou třídu, proveď u obou neuronů následující úpravy:

$$w_i(t+1) = w_i(t) + \lambda\eta(t)\left[x_i(t) - w_i(t)\right], \tag{4.17}$$

kde hodnota koeficientu $\lambda \in \{0.1, 0.5\}$ je přímo úměrná velikosti okénka, její hodnota byla odhadnuta experimentálně.[9]

Algoritmus LVQ3 je již dostatečně stabilní, tzn. že po nalezení optimálního váhového vektoru neuronu se již tato hodnota nebude měnit.

Kapitola 5

Učení a hodnocení sítě

5.1 Rozdělení vstupních dat

Po naučení neuronové sítě je vhodné vědět jak dobře bude síť vykonávat činnost, kterou byla naučena. K tomuto vyhodnocení bychom ideálně potřebovali další vstupní data (pro síť neznámá), pomocí kterých bychom změřili úspěšnost sítě. Tato data většinou k dispozici nemáme, proto je potřeba si je nějak obstarat. Tato testovací data se získají rozdělením množiny dat, kterou máme k dispozici, na dvě menší množiny tzv. trénovací a testovací množinu. V jakém poměru data rozdělit je závislé na mnoha faktorech, v praxi se často tento poměr určuje experimentálně. Učení sítě je poté provedeno na trénovací množině a na testovací množině je určena skutečná chybovost sítě.

Jedním z nebezpečí při učení neuronové sítě je její přeučení. Přeučení sítě znamená, že je síť příliš přizpůsobena trénovacím datům (např. se přizpůsobila šumu v datech). Takto přeučená síť poté podává výborné výsledky na trénovacích datech, v praxi nebo na testovacích datech by již byly její výsledky podstatně horší. Tomuto problému je možné předejít použitím validační množiny. Validační množina slouží k ověření úspěšnosti sítě během učení. V každé iteraci učicího algoritmu je vyhodnocena úspěšnost sítě na validační množině a v případě že by provedením iterace došlo ke zvýšení chyby na validačních datech, není již tato iterace provedena a síť je prohlášena za naučenou. Druhou možností je nechat proběhnout požadovaný počet iterací učicího algoritmy, v jeho průběhu sledovat vývoj chyby na validační množina a za naučenou síť prohlásit síť v té iteraci, kdy byla chyba na validačních datech nejmenší. Jako validační množinu je možné použít testovací množinu. Pokud chceme striktně dodržet postup, že neuronová síť nesmí v průběhu učení přijít do styku s testovací množinou, je možné validační množinu získat jako podmnožinu trénovacích dat.

5.2 Křížová validace

Při učení sítě může mít významnou roli na její úspěšnost náhodné rozdělení dat na trénovací, testovací a případně validační množinu nebo náhodná počáteční inicializace sítě. Křížová validace může sloužit právě k omezení vlivu těchto náhodných jevů na odhad úspěšnosti sítě, nejčastěji je však využívána k nastavení parametrů různých algoritmů. V našem

případě ji tedy můžeme použít k vybrání nejlepší struktury sítě. Křížová validace probíhá následujícím způsobem:

1. Trénovací data se náhodně rozdělí do k stejně velkých množin. Hodnota k určuje stupeň křížové validace. Množiny dat se nazývají foldy.
2. Pro každý fold je vytvořena náhodně inicializovaná neuronová síť, která je na zbývajících foldech naučena a tento jeden fold slouží jako testovací množina, pro určení úspěšnosti sítě.
3. Z úspěšností sítě pro všechny foldy se spočte průměr. Tento průměr je neovlivněným odhadem úspěšnosti dané sítě.

Existuje několik různých typů křížové validace:

- K-stupňová křížová validace je popsána výše.
- Stratifikovaná křížová validace. Při stratifikované křížové validaci jsou data rozdělena do foldů tak, aby v každém foldu byly třídy zastoupeny ve stejném poměru jako v trénovacích datech.
- Leave-one-out křížová validace je extrémní případ k-stupňové křížové validaci, kde k je rovno počtu vektorů v trénovacích datech.

Kapitola 6

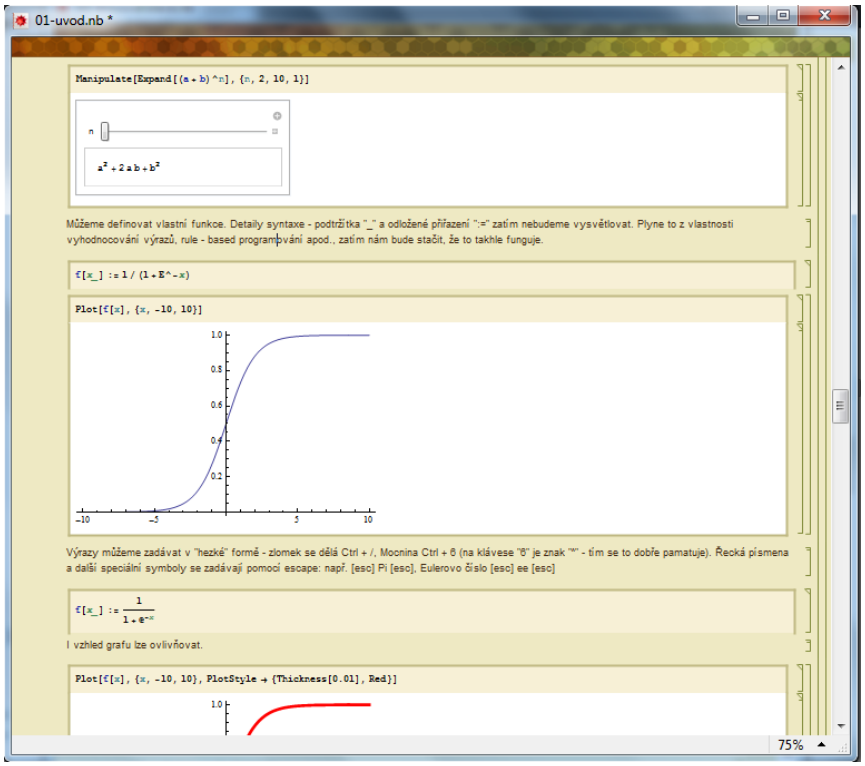
Implementační prostředí

6.1 Wolfram Mathematica

Systém Mathematica od firmy Wolfram Research je sofistikovaný matematický nástroj, který umožňuje provádět numerické i symbolické výpočty. Systém obsahuje velké množství vestavěných funkcí a umožňuje i přidání dalších uživatelsky definovaných funkcí. Dále nabízí široké možnosti vizualizace dat ve 2D i 3D grafech. Prostředí systému vidíte na obrázku 6.1. Neméně podstatnou funkcí je možnost exportu vyhodnoceného notebooku do formátu pdf. Díky tomu může demonstrační aplikaci vytvořenou v systému Mathematica, byť v omezené míře, využít i student, který nemá na svém počítači systém Mathematica nainstalovaný. Systém je možné rozšířit celou řadou knihoven, jednou z nich je i knihovna Neural Networks, kterou jsem využil ve své demonstrační aplikaci. Demonstrační aplikaci jsem programoval ve verzi Mathematica 8.0. Pro studium programování v Mathematice jsem použil knihy[6] a [10].

6.2 Knihovna Neural Networks

Knihovna Neural Networks rozšiřuje systém *Mathematica* o možnost pracovat s neuronovými sítěmi bez nutnosti vytvářet si její vlastní implementaci. Konkrétně knihovna nabízí práci s těmito konkrétními sítěmi: jednoduchý perceptron, dopředná neuronová síť, RBF neuronová síť, Hopfieldova neuronová síť, dynamická neuronová síť, Kohonenova mapa, LVQ a VQ. Knihovna také poskytuje implementaci různých algoritmů učení sítě, konkrétně jsou to tyto algoritmy: Levenberg-Marquardt, Gauss-Newton, gradientní algoritmus a algoritmus zpětné propagace. Knihovna umožňuje široké možnosti natavení parametrů u jednotlivých sítí a pro méně zkušené uživatele nabízí vytvoření sítě s výchozími parametry. Výchozí parametry jsou voleny tak, aby síť s výchozími parametry podávala dobré výkony a její učení a vybavování netrvalo přehnaně dlouho. Knihovna ještě nabízí široké možnosti vizualizace dat, grafické zobrazení průběhu učení sítě a také grafické zobrazení výstupu naučené neuronové sítě. Ke knihovně patří návod k použití a rozsáhlá dokumentace.



Obrázek 6.1: Ukázka prostředí systému Mathematica

Kapitola 7

Experimenty s neuronovými sítěmi

Číslo kapitoly	Název kapitoly	Soubor
1	Úvod	01-uvod.nb
2	Algoritmy učení	02-algoritmy-uceni.nb
3	Křížová validace	03-krossvalidace.nb
4	Dopředná síť a umělá data sin(x)	04-feedforward-sin.nb
5	Dopředná síť a Iris data	05-feedforward-iris.nb
6	Přístupy k učení dopředné sítě	06-feedforward-iris-2.nb
7	Pokrytí dat jednoduchou RBF sítí	07-rbf-neuron.nb
8	Pokrytí dat RBF sítí	08-rbf-neuron-2.nb
9	Síť RBF a umělá data sin(x)	09-rbf-sin.nb
10	Síť RBF Iris data	10-rbf-iris.nb
11	Přístupy k učení RBF sítě	11-rbf-iris-2.nb
12	Hopfieldova síť a umělé vzory	12-hopfield.nb
13	Shlukování dat sítí bez učitele	13-som-clustering.nb
14	Kohonenova síť a Iris data	14-som-iris.nb
15	LVQ a Iris data	15-lvq-iris.nb

Tabulka 7.1: Přehled struktury aplikace

7.1 Úvod

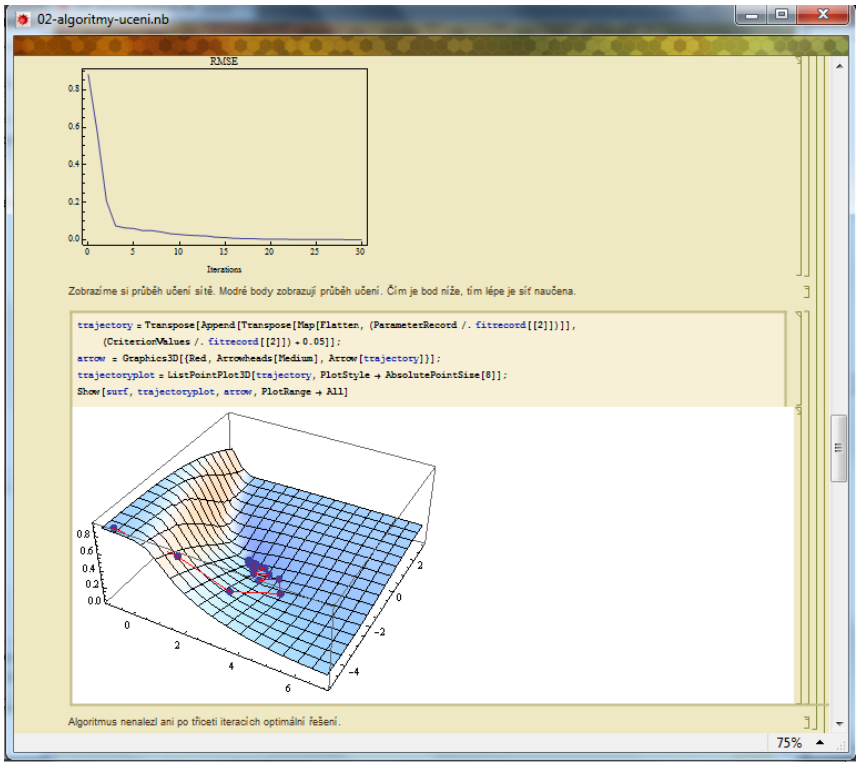
V úvodní kapitole seznamuji studenty s používáním programu *Mathematica* od naprostých základů. Nejprve probírám základní syntaxi a ovládání, dále seznamuji studenty s používáním proměnných, funkcí a s možnostmi vizualizace dat. V další části notebooku probírám základy programování v systému *Mathematica*. Prostudování tohoto úvodu umožní rychleji se orientovat v experimentech s neuronovými sítěmi a také dá studentovi znalosti potřebné k úpravám jednotlivých experimentů. Tato kapitola se nachází v souboru *"01-uvod.nb"*.

Soubor vznikl úpravou souboru Petra Chlumského. Změnil jsem grafický vzhled notebooku, změnil a rozšířil jsem textový komentář k jednotlivým příkazům.

7.2 Algoritmy učení

V notebooku demonstruji použití různých učicích algoritmů. Pro každý učicí algoritmus graficky zobrazuji jeho postup při učení. Demonstruji zde tyto algoritmy: *Levenberg-Marquardt algoritmus*, *Gauss-Newton algoritmus*, *Algoritmus nejvyššího poklesu (Steepest Descent)* a *Algoritmus zpětné propagace (Backpropagation)*.

Učení jsem demonstroval na jednoduché dopředné síti s jedním neuronem, jedním vstupem a jedním výstupem. Nejprve jsem vytvořil potřebnou síť, poté pomocí této sítě vygeneroval data, na kterých jsem učení demonstroval. Poté jsem pro každý algoritmus ukázal jakým způsobem ho použít pro učení sítě a graficky jsem zobrazil průběh učení, jak je vidět na obrázku 7.1. U algoritmu zpětné propagace je potřeba nastavit délku kroku (stepLength) a momentum. Tyto parametry jsem umožnil studentům měnit pomocí interaktivního grafu, kde si studenti mohou pomocí posuvníků měnit hodnotu těchto parametrů, graf se automaticky překresluje podle aktuálně nastavených hodnot parametrů. Tato kapitola se nachází v souboru *02-algoritmy-uceni.nb*.

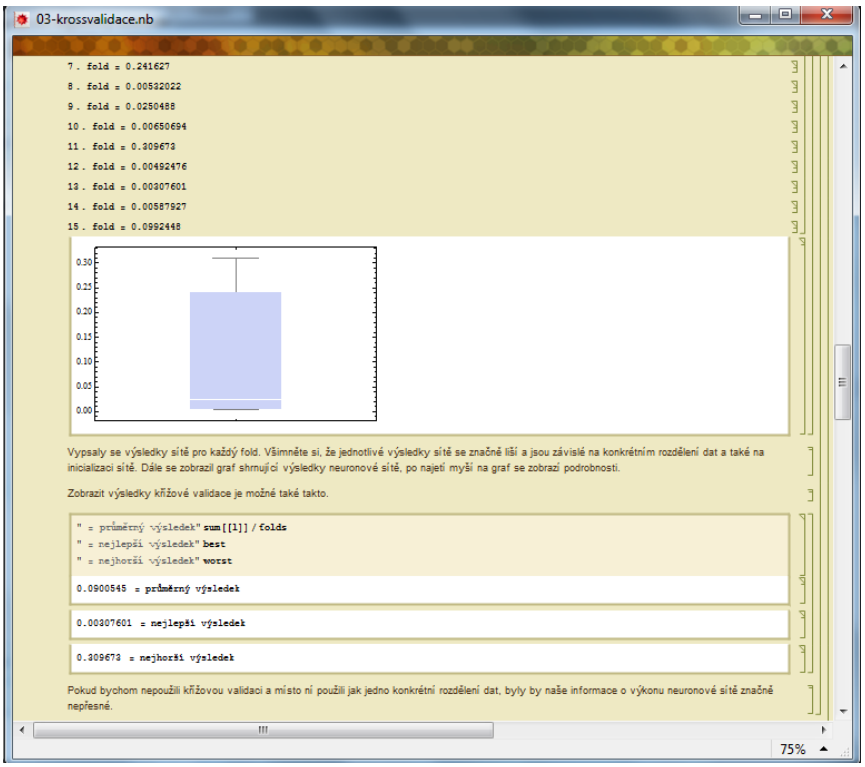


Obrázek 7.1: Ukázka souboru 02-algoritmy-uceni.nb

7.3 Křížová validace

V kapitole křížová validace seznamuji studenty s principem fungování křížové validace. Křížovou validaci ukazuji na Iris datech a jednoduché dopředné síti.

Nejprve jsem načetl data. Iris data jsem rozdělil na vstupní a výstupní vektory, výstupní vektory jsem zakódoval kódem 1 z N. Poté detailně rozebral předzpracování dat pro křížovou validaci. Dále jsem vytvořil neuronovou síť, kterou jsem pomocí křížové validace testoval. Následně jsem naimplementoval samotnou křížovou validaci. V této implementaci jsem vypisoval pro každý fold úspěšnost RMSE, kterou síť dosáhla. Po skončení všech kol křížové validace jsem zobrazil boxový graf s výsledky křížové validace viditelný na obrázku 7.2. Po najetí myší na graf se k němu zobrazí legenda. Studenti jsem dal také možnost nechat si vypsat výsledky křížové validace v textovém formátu. Textový formát vypíše nejlepší výsledek, nejhorší výsledek a aritmetický průměr všech výsledků. Na závěr jsem dal studentům možnost provést si křížovou validaci vlastní sítě s vlastními daty. Vlastní křížové validaci nastaví student parametry jednoduchým přiřazením do proměnných, je potřeba aby zadaná síť odpovídala zadaným vstupním a výstupním datům a data byla předzpracována pomocí postupu uvedeného v této kapitole. Tato kapitola se nachází v souboru *03-krossvalidace.nb*.



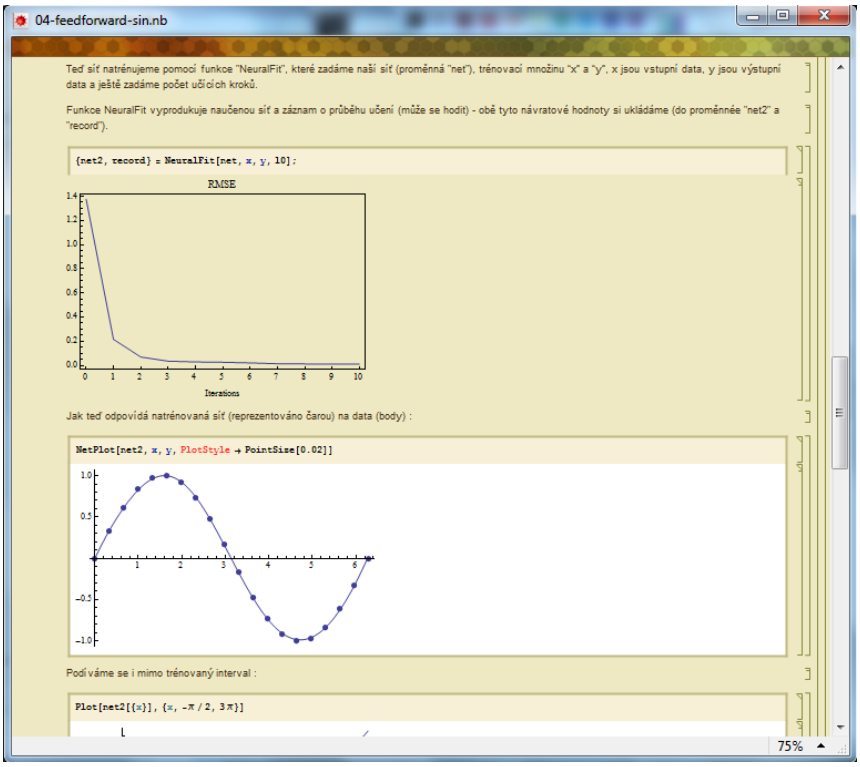
Obrázek 7.2: Ukázka souboru 03-krossvalidace.nb

7.4 Dopředná síť

7.4.1 Jednoduchá data sin(x)

V této kapitole poprvé seznamuji studenty s dopřednou neuronovou sítí. Funkci dopředné neuronové sítě jsem demonstroval na aproximaci funkce sinus.

Nejprve jsem připravil jednoduchá trénovací data navzorkováním funkce sinus na intervalu $<0, 2\pi>$. Navzorkovaná data jsem pro jasnou představu zobrazil v textové i grafické podobě. Poté jsem ukázal jakým způsobem vytvořit neuronovou síť požadované struktury, jak si zobrazit dodatečné informace o síti a jak síť naučit na trénovacích datech. Dále jsem graficky zobrazil výstup sítě před naučením a po naučení jak je vidět na obrázku 7.3. Na závěr jsem ukázal jak převést síť do formy vzorce. Tato kapitola se nachází v souboru *04-feedforward-sin.nb*.



Obrázek 7.3: Ukázka souboru 04-feedforard-sin.nb

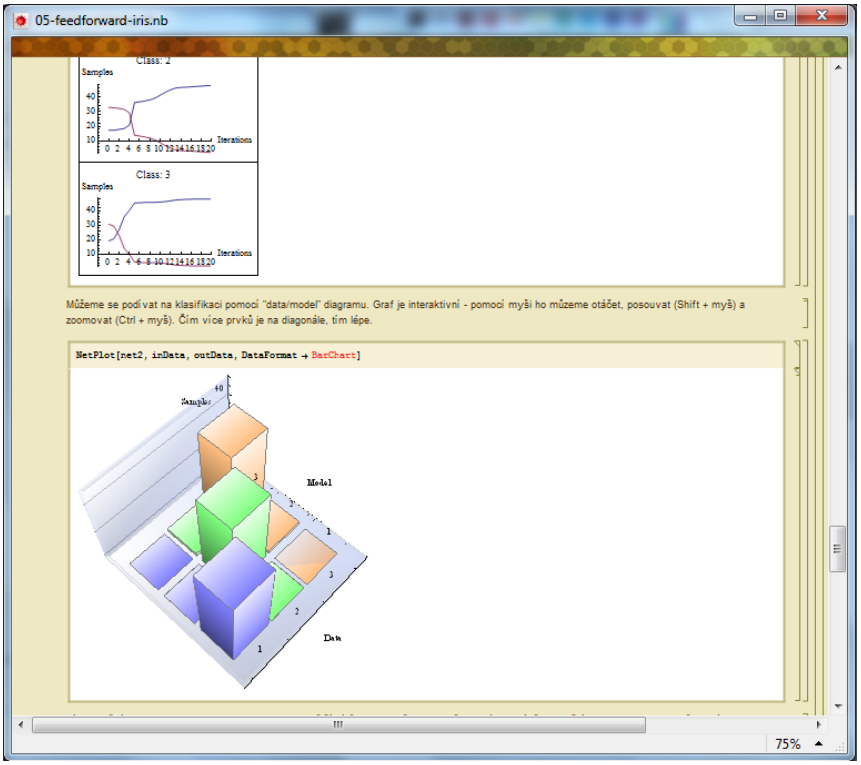
Soubor vznikl úpravou souboru Petra Chlumského. Změnil jsem vzhled notebooku a upravil doprovodný komentář.

7.4.2 Iris data

V této kapitole ukazuji jakým způsobem pomocí dopředné sítě klasifikovat Iris data.

Nejprve jsem načetl data. Studentům jsme dal na výběr zda chtějí načíst data z internetu nebo použít lokální soubor s daty. Poté jsem data předzpracoval. Data jsem rozdělil na

vstupní a výstupní vektory, protože výstupní parametr je textový, provedl jsem jeho překódování metodou 1 z N, kdy je každé třídě přiřazen jeden výstupní vektor. Toto předzpracování jsem prováděl krok po kroku s velmi podrobným komentářem a ještě jsem ho doplnil několika dalšími ukázkovými příklady. Po předzpracování dat jsem vytvořil dopřednou neuronovou síť, a data jsem touto neuronovou sítí zpracoval. Dále jsem zobrazil jak se vyvíjela úspěšnost klasifikace v průběhu učení (zobrazeno na obrázku 7.4). Na závěr jsem použil 3D graf, který ukazuje úspěšnost sítě na trénovacích datech (také ukázáno na obrázku 7.4). Také jsem ukázal možnost jak síť nechat symbolicky vyhodnotit. Tato kapitola se nachází v souboru *05-feedforward-iris.nb*.



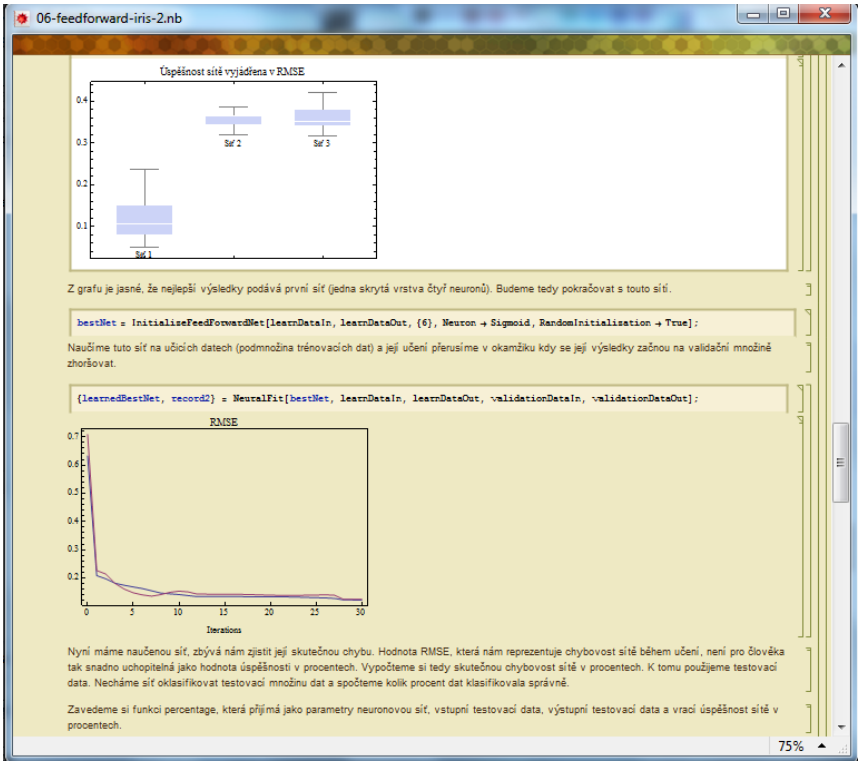
Obrázek 7.4: Ukázka souboru 05-feedforard-iris.nb

Soubor vznikl úpravou souboru Petra Chlumského. Změnil jsem vzhled notebooku. V doprovodném komentáři jsem nahradil sousloví „skupina v datech“ slovem „třída“, které podle mého názoru lépe vystihuje popisovanou skutečnost. Dále jsem komentář rozšířil. V závěru notebooku jsem nezobrazoval vývoj úspěšnosti klasifikace během učení ve 3D grafu, který podle mě nebyl tak přehledný a jasný jako 2D graf, který je zobrazen.

7.4.3 Různé přístupy k učení sítě

V tomto notebooku ukazuji několik různých přístupů k učení dopředné sítě. Tyto přístupy se liší hlavně v dělení dat na trénovací, testovací a validační množinu. Ukazuji také použití křížové validace a rozebírám vyhodnocení úspěšnosti neuronové sítě.

Nejprve jsem načetl Iris data a provedl jejich předzpracování stejně jako v minulé kapitole. Takto předzpracovaná data jsem náhodně rozdělil na trénovací, validační a testovací množinu. Ukázal jsem možnost použití křížové validace k určení vhodné struktury sítě. Při křížové validaci jsem výsledky sítí porovnával podle hodnoty RMSE, výsledky křížové validace jsem zobrazil v boxovém grafu, který je na obrázku 7.5. Síť, která vyšla z křížové validace nejlépe, jsem poté naučil na trénovací množině s použitím validační množiny. Validační množinu jsem použil k zastavení učení sítě, pokud by se úspěšnost klasifikace na validační množině začala v průběhu učení zhoršovat. Poté jsem vyhodnotil úspěšnost klasifikace na testovací množině. Tuto úspěšnost jsem pro snadnou čitelnost uvedl v procentech. Dále jsem síť se stejnou strukturou naučil bez použití validační množiny a taktéž jsem vyhodnotil její úspěšnost. Kapitola se nachází v souboru *06-feedforward-iris-2*.



Obrázek 7.5: Ukázka souboru 06-feedforard-iris-2.nb

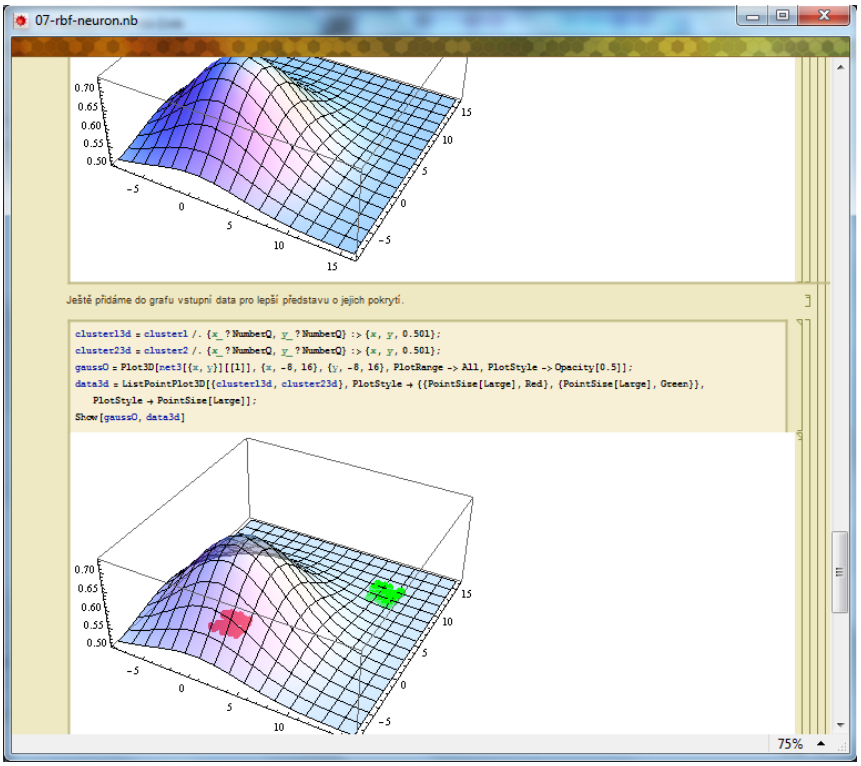
7.5 RBF síť

7.5.1 Ukázka výstupu skrytého neuronu

Tato kapitola není určena přímo pro experimentování se sítí, ale spíše pro pochopení jakým způsobem RBF síť funguje. V notebooku zobrazuji výstup skrytého RBF neuronu.

Nejprve jsem vygeneroval vstupní data. Data jsou tvořena dvěma dobře oddělenými shluky, každý shluk představuje jednu třídu. Poté jsem vytvořil jednoduchou RBF neurono-

vou síť s jedním skrytým RBF neuronem, pomocí které jsem vstupní data klasifikoval. Síť jsem naučil na vygenerovaných datech, poté jsem zobrazil výstup RBF neuronu. Pro lepší představu o tom, jakým způsobem RBF neuron pokryl vstupní data, jsem tato data zobrazil společně s výstupem neuronu do jednoho grafu, který je na obrázku 7.6. Kapitola se nachází v souboru *07-rbf-neuron.nb*.

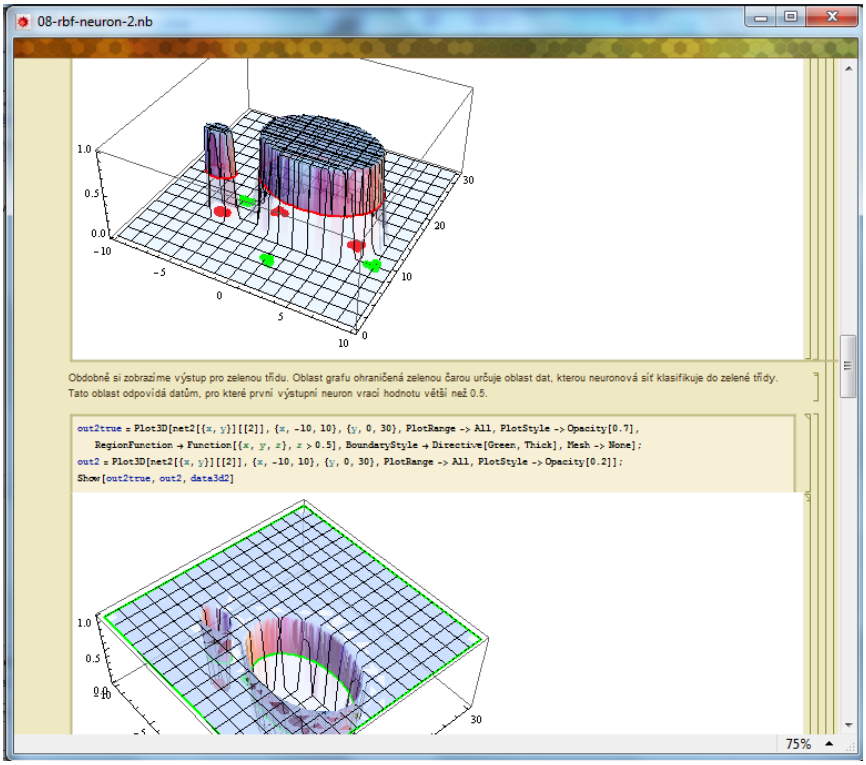


Obrázek 7.6: Ukázka souboru 07-rbf-neuron.nb

7.5.2 Ukázka výstupu tří skrytých neuronů

Tato kapitola, stejně jako předchozí, slouží primárně k pochopení fungování RBF sítě. V notebooku zobrazuji výstup skryté vrstvy neuronů u RBF sítě se třemi RBF neurony, která je naučena na složitějších datech.

Nejprve jsem vygeneroval data. Data se skládají ze šesti oddělených shluků, obsahují dvě třídy, každá třída se skládá ze třech shluků. Data jsem pro představu zobrazil v grafu. Dále jsem vytvořil RBF neuronovou síť, kterou jsem chtěl vygenerovaná data klasifikovat. Síť jsem naučil na vygenerovaných datech. Poté jsem zobrazil výstup neuronové sítě ve 2D i 3D grafu (zobrazeno na obrázku 7.7), na kterých ukazuji jak síť klasifikuje data. Poté jsem zobrazen výstup skryté vrstvy RBF neuronů. Do stejného grafu s výstupem jednotlivých RBF neuronů jsem zobrazil i vstupní data, aby bylo vidět jak se RBF neurony přizpůsobily datům. Kapitola je v souboru *08-rbf-neuron-2.nb*.



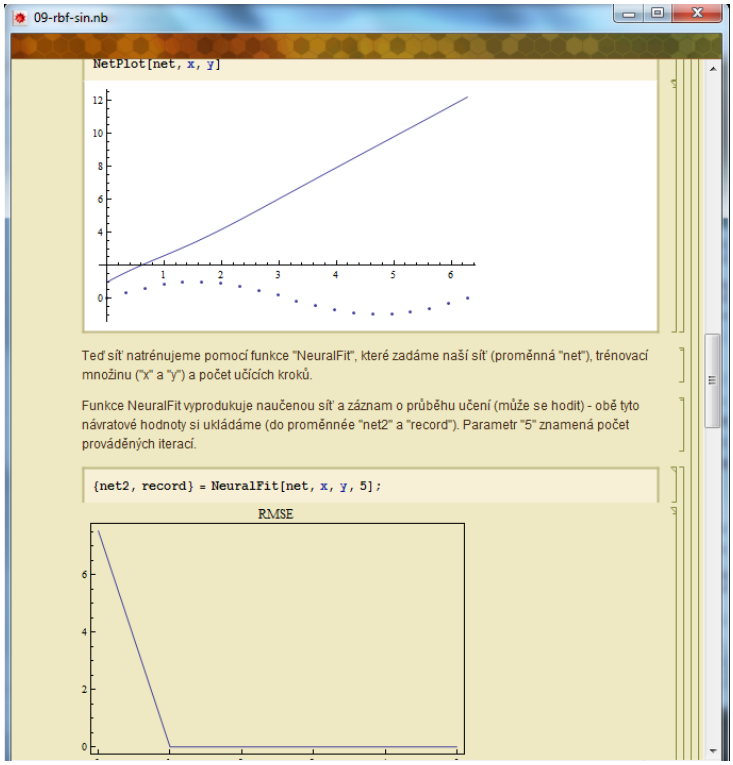
Obrázek 7.7: Ukázka souboru 08-rbf-neuron-2.nb

7.5.3 Jednoduchá data sin(x)

V této kapitole poprvé seznamuji studenty s RBF sítí. Ukazuji jak pomocí RBF sítě zpracovat jednoduchá data, v tomto případě se jedná o aproximaci funkce sinus.

Nejprve jsem vygeneroval data navzorkováním funkce sinus na intervalu $<0, 2\pi>$. Data jsem zobrazil v grafické i textové podobě, aby o nich měli studenti dobrou představu. Dále jsem ukázal jak vytvořit RBF neuronovou síť požadované struktury, jak si o ní zobrazit podrobnější informace a jakým způsobem jí naučit na trénovacích datech. Dále ukazuji jak reagovala nenaučená síť (je vidět na obrázku 7.8) na data a také jak reaguje naučená síť na stejná data. Na závěr jsem ukázal jak je možné nechat síť symbolicky vyhodnotit (zobrazit ji ve formě vzorce). Kapitola je v souboru *09-rbf-sin.nb*.

Soubor vznikl úpravou souboru Petra Chlumského. Změnil jsem vzhled a rozšířil doprovodný komentář. Dále jsem odstranil nepřesné a nepravdivé popisky u vytváření RBF sítě a nahradil je správnou verzí.



Obrázek 7.8: Ukázka souboru 09-rbf-sin.nb

7.5.4 Iris data

Touto kapitolou ukazuji zpracování složitějších dat neuronovou sítí RBF. V notebooku provádím klasifikování Iris dat z UCI databáze [2].

Nejprve jsem načetl Iris data (opět jsem dal na výběr za je načíst ze souboru nebo z internetu), poté jsem provedl rozdělení dat na vstupní a výstupní vektory a výstupní data jsem zakódoval algoritmem 1 z N. Dále jsem vytvořil RBF síť, která jsem následně naučil na datech. Potom jsem zobrazil graf, který ukazuje jak se vyvíjela úspěšnost klasifikace v průběhu učení. Níže jsem ještě vykreslil graf porovnávající zadaná data s výstupem naší naučené RBF sítě. Na závěr jsem ukázal možnost symbolického vyhodnocení sítě. Kapitolu naleznete v souboru *10-rbf-iris.nb*.

Soubor vznikl úpravou souboru Petra Chlumského. Změnil jsem grafický vzhled notebooku. Dále jsem významně zkrátil sekci předzpracování dat. V původním notebooku bylo předzpracování stejně rozsáhlé jako v souboru *05-feedforward-iris.nb*, já ponechal pouze kód, který skutečně zpracovává Iris data a uvedl odkaz na soubor ve kterém je předzpracování popsáno podrobněji. Dále jsem na závěr notebooku nezobrazoval 3D graf s vývojem úspěšnosti klasifikace, který podle mého názoru není tolik vypovídající. Ponechal jsem 2D graf s vývojem klasifikace. Také jsem rozšířil komentář a opravil špatný popis u vytváření RBF sítě.

7.5.5 Různé přístupy k učení sítě

Kapitolou chci demonstrovat různé možnosti jak přistoupit k učení RBF sítě z hlediska rozdělení dat na trénovací, testovací a validační množinu. Také ukazuji možnost využití křížové validace k určení nejvhodnější struktury sítě. Na závěr ukazuji jak vyhodnotit úspěšnost sítě.

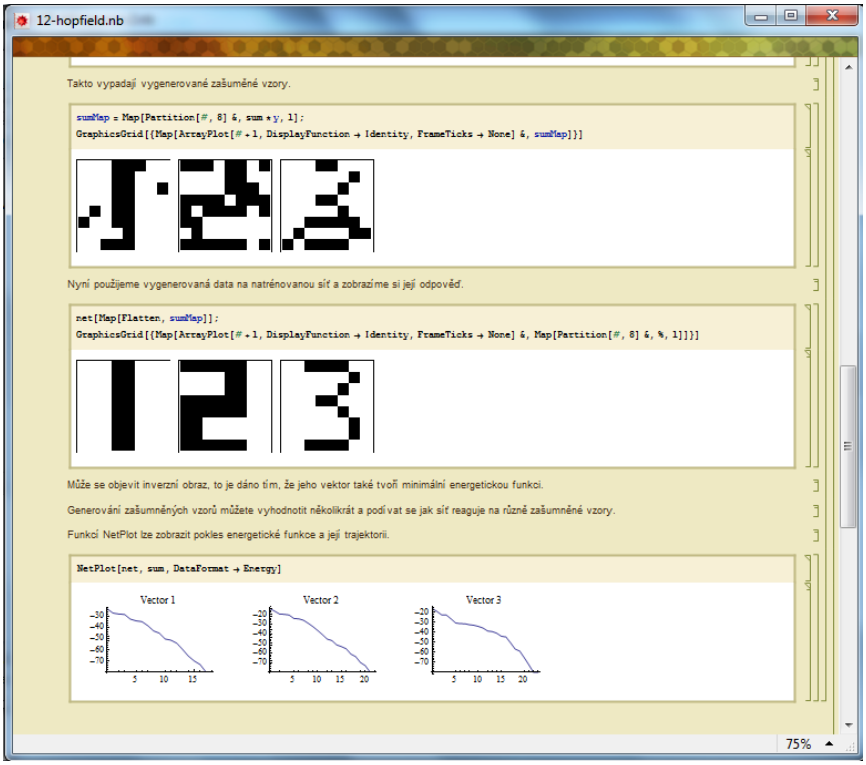
Nejprve jsem načetl Iris data (v notebooku jsem dal na výběr zda ze souboru nebo z internetu). Načtená data jsem rozdělil na vstupní a výstupní vektory, výstupní vektory jsem překódoval kódem 1 z N. Takto zpracovaná data jsem rozdělil na trénovací, testovací a validační množinu. Dále jsem demonstroval možnost použití křížové validace, k určení nejvhodnější struktury sítě (v tomto případě k určení počtu RBF neuronů). Výsledek křížové validace jsem zobrazil v boxovém grafu. Dále jsem ukázal možnost učení RBF sítě s validační množinou. Validační množina slouží ke kontrole úspěšnosti sítě, jakmile se výsledky na validační množině zhorší oproti předchozí iteraci, je učení sítě ukončeno. Na testovacích datech jsem poté provedl určení skutečné úspěšnosti naučené sítě, kterou jsem vyjádřil v procentech. Dále jsem ukázal učení sítě bez použití validační množiny. U takto naučené sítě jsem také určil skutečnou úspěšnost na testovacích datech v procentech. Kapitola se nachází v souboru *11-rbf-iris-2.nb*.

7.6 Hopfieldova síť

V této kapitole představuji studentům Hopfieldovou neuronovou síť a ukazuji její použití na jednoduchých datech.

Jako vstupní data jsem použil obrázky číslic 1, 2 a 3 zobrazené v poli 8x8, kde černá znamená 1 a bílá -1. Zobrazení dat můžete vidět na obrázku 7.9. Tato data jsem zadal přímo v kódu. Poté jsem vytvořil Hopfieldovu síť, která jsem následně na vstupních datech naučil. Pro otestování, jak síť reaguje na data jsem potřeboval vytvořit testovací data, ta jsem vytvořil aplikací šumu na vstupní data. Tato zašuměná data jsem předložil naučené síti a zobrazil její reakci na ně. Na závěr jsem zobrazil graf ukazující jakým způsobem se minimalizovala energetická funkce pro jednotlivé předložené vzory. Kapitola se nalézá v souboru *12-hopfield.nb*.

Soubor vznikl úpravou souboru Petra Chlumského. Mimo úpravy celkového vzhledu notebooku a drobných úprav doprovodného komentáře spočívá hlavní úprava v nahrazení funkce GraphicsArray funkcí GraphicsGrid. Funkce GraphicsArray je v Mathematice 8 zastaralá a nedoporučuje se používat. Také jsem upravil funkci pro aplikaci šumu, která nyní zašumí předložený vzor o trochu více než tomu bylo dříve.



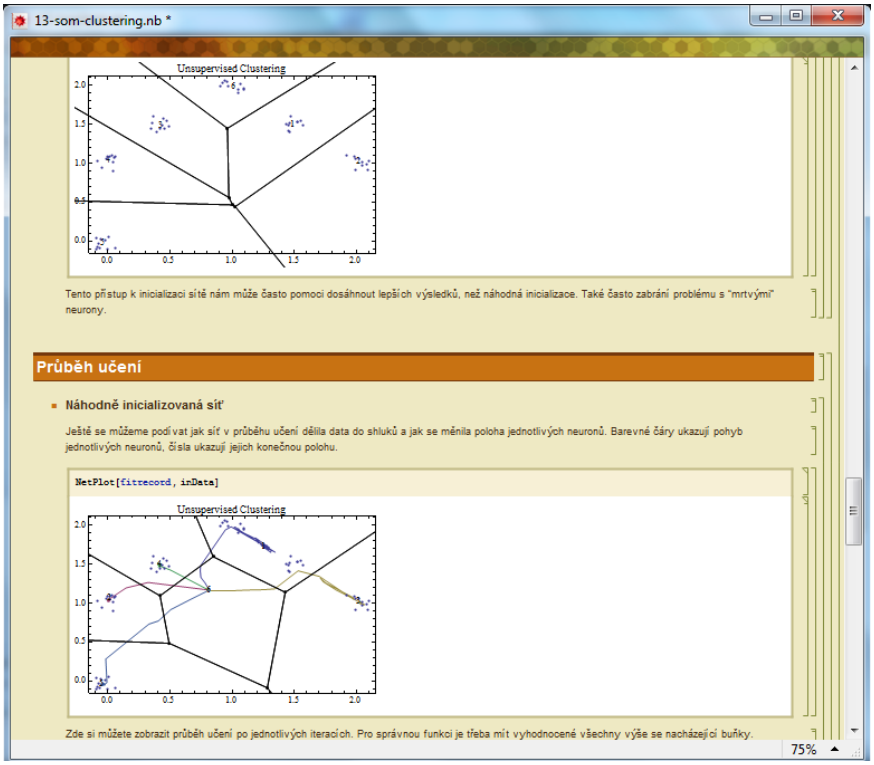
Obrázek 7.9: Ukázka souboru 12-hopfield.nb

7.7 Samoorganizující mapy

7.7.1 Shlukování dat

V této kapitole seznamuji studenty se samoorganizující se mapou a ukazuji její použití na jednoduchých datech. Pomocí samoorganizující se mapy hledám shluky ve vygenerovaných datech.

Na začátku jsem vygeneroval vstupní data. Data obsahují šest dobře oddělených shluků. V těchto datech jsem hledal pomocí samoorganizující se mapy šest shluků. Nejprve jsem vytvořil náhodně inicializovanou samoorganizující se mapu, kterou jsem naučil na vstupních datech. Pomocí grafu jsem zobrazil její výstup (tedy jak síť rozdělila data na shluky). Dále jsem ukázal jak odstranit ze sítě nepotřebný (mrtvý) neuron a zobrazil jsem výstup sítě bez nepotřebného neuronu. Následně jsem ukázal postup umožňující dosažení lepších výsledků. Samoorganizující se mapa jsem neinicializoval náhodně, ale použil jsem k její inicializaci metodou SOM. Poté jsem síť doučil na vstupních datech a graficky zobrazil její výstup, který je zobrazen na obrázku 7.10. Na závěr jsem zobrazil průběh učení obou sítí. Nejprve ve formě statického grafu, který zobrazuje trajektorii jednotlivých neuronů v průběhu učení. Podruhé jako interaktivní graf zobrazující výstup sítě pro danou iteraci. Pomocí posuvníku je možné vybírat, která iterace je zobrazena. Kapitola se nachází v souboru *13-som-clustering.nb*.



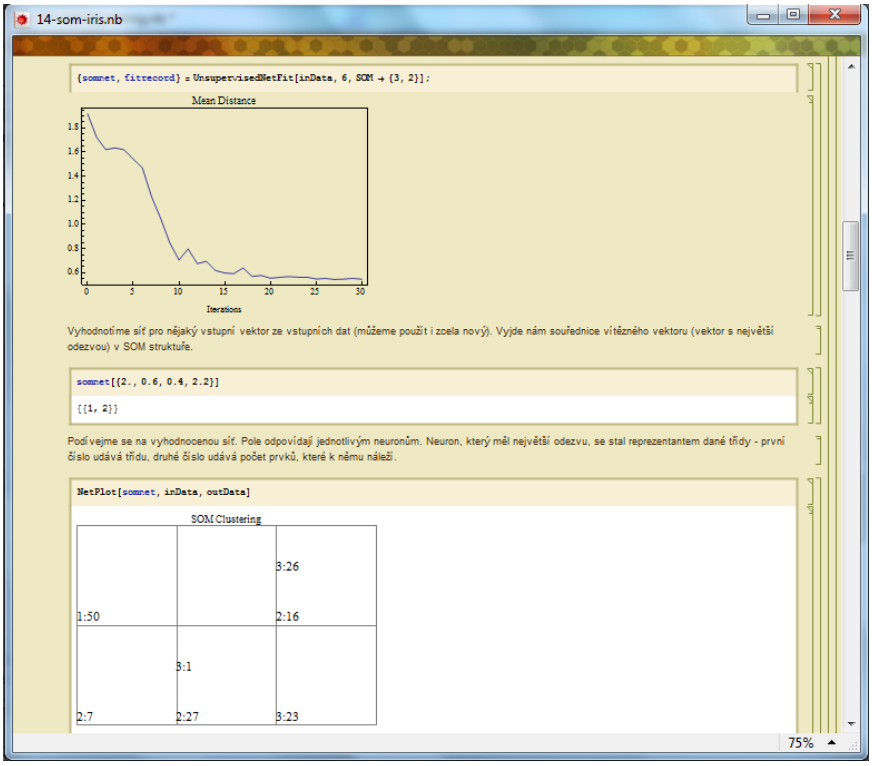
Obrázek 7.10: Ukázka souboru 13-som-clustering.nb

7.7.2 SOM a Iris data

Touto kapitolou demonstruji použití samoorganizující se mapy na složitějších datech. K ukázce použití jsem z zvolil Iris data.

Nejprve jsem načetl Iris data(studentům jsem dal možnost načtení ze souboru nebo z internetu). Data jsem rozdělil na vstupní a výstupní vektory, výstupní vektory jsem překódoval metodou 1 z N. Dále jsem vytvořil samoorganizující se mapu, kterou jsem naučil na vstupních datech. Tento graf můžete vidět na obrázku 7.11. Poté jsem vykreslil graf, který zobrazuje kolik vektorů náleží ke každému neuronu v síti. Stejným způsobem jsem zobrazil i průběh učení sítě. Následně jsem ukázal jakým způsobem lze libovolnému vektoru přiřadit jeho reprezentanta, jak spočítat Eukleidovskou vzdálenost od reprezentanta a také jsem připomněl mazání nepotřebných neuronů. Na závěr jsem rozebral doučování sítě. Kapitola se nalézá v souboru *14-som-iris.nb*.

Soubor vznikl úpravou souboru Petra Chlumského. Změnil jsem celkový vzhled notebooku, v komentáři jsem nahradil slovo „skupina“ slovem „třída“, které podle mě lépe vystihuje popisovanou věc. Udělal jsem ještě několik menších úprav a rozšíření v komentáři. Dále jsem provedl úpravu předzpracování dat, které bylo zbytečně rozsáhlé a tím pádem zdlouhavé. Ponechal jsem pouze minimální potřebný kód a uvedl odkaz na notebook ve kterém je předzpracování dat detailně popsáno.



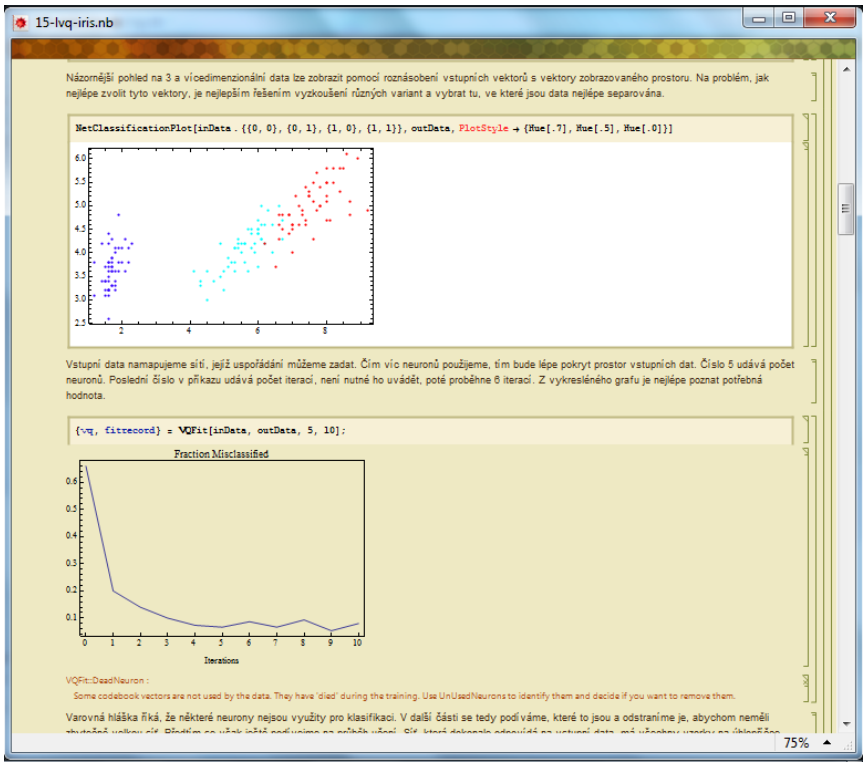
Obrázek 7.11: Ukázka souboru 14-som-iris.nb

7.8 LVQ

V poslední kapitole seznamuji studenty s použitím LVQ sítě. Použití LVQ sítě je ukázáno na klasifikaci Iris dat.

Nejprve jsem načetl Iris data (dávám možnost načtení ze souboru nebo z internetu), data jsem rozdělil na vstupní a výstupní vektory a výstupní vektory jsem překódoval metodou 1 z N. Nejprve jsem nabídl několik pohledů na vstupní data (jeden z nich je na obrázku 7.12, společně s vytvořením LVQ sítě), poté jsem vytvořil LVQ síť, kterou jsem naučil na vstupních datech. Poté jsem zobrazil vizualizaci výstupu sítě v průběhu učení. Následně jsem ukázal jakým způsobem smazat nepotřebné neurony. Dále jsem demonstroval jak je možné nechat oklasifikovat libovolný vektor. Poté jsem zobrazil procentuální úspěšnost klasifikace. Na závěr jsem rozebral možnosti doučování sítě a ukázal několik možných grafických výstupů sítě. Tato kapitola se nachází v souboru *15-lvq-iris.nb*.

Soubor vznikl úpravou souboru Petra Chlumského. Změnil jsem grafickou podobu notebooku. V doprovodném komentáři jsem nahradil slovo „skupina“ slovem „třída“, které lépe vystihuje popisovanou skutečnost. Komentář jsem ještě drobně rozšířil a upravil. Upravil jsem a výrazně zkrátil předzpracování dat. Ponechal jsem pouze kód, který skutečně manipuluje s daty a uvedl odkaz na notebook, ve kterém se nachází detailně popsaný proces předzpracování dat.



Obrázek 7.12: Ukázka souboru 15-lvq.nb

Kapitola 8

Závěr

V rámci své bakalářské práce jsem rozšířil demonstrační aplikaci pro podporu výuky neuronových sítí na FEL ČVUT. Původní aplikace se skládala z 8 souborů, já jsem ji rozšířil na 15 souborů a soubory původní aplikace jsem upravil a částečné přepracoval. Původní aplikace pokrývala základní práci s dopřednou, RBF, Hopfieldovou, Kohonenovou a LVQ sítí. Já jsem ji rozšířil o demonstraci různých trénovacích algoritmů, křížovou validaci, ukázal jsem různé přístupy k učení sítě z pohledu dělení dat na validační, trénovací a testovaní množinu a podrobně jsem rozebral „vnitřnosti“ RBF neuronové sítě. Součástí práce je také teoretická část, ve které jsem popsal základní principy a strukturu vybraných neuronových sítí.

Díky bakalářské jsem si značně rozšířil své znalosti neuronových sítí a také jsem rozšířil své schopnosti programovat v systému Mathematica, se kterým jsem měl do té doby pouze pasivní zkušenosti.

Cíle bakalářské práce se mi podařilo naplnit. Výstup této práce bude použit ve cvičeních předmětu „Neuronové sítě a neuropočítače“.

Literatura

[1] *SDL Component Suite* [online]. 2011. [cit. 18. 5. 2011]. Dostupné z: <http://www.lohninger.com/helpcsuite/nrofinsens_property.htm>.

[2] *UCI - Machine Learning Repository* [online]. 2011. [cit. 22. 4. 2011]. Dostupné z: <http://archive.ics.uci.edu/ml/>.

[3] *Courseware* [online]. 2011. [cit. 19. 4. 2011]. Dostupné z: <http://neuron.felk.cvut.cz/courseware/>.

[4] *Neural Networks Documentation* [online]. 2011. [cit. 16. 5. 2011]. Dostupné z: <http://reference.wolfram.com/applications/neuralnetworks/>.

[5] HAYKIN, S. *Neural Networks and Learning Machines*. New Jersey : Pearson Education, 2009. ISBN 0-13-147139-2.

[6] HOSTE, J. *Mathematica DeMySTiFied*. New York : McGraw-Hill Professional, 1st edition, 2008. ISBN 0-07-159145-1.

[7] KAČENKA, P. *Neuronové sítě* [online]. 1998. [cit. 14. 5. 2011]. Dostupné z: <http://mks.mff.cuni.cz/library/NeuronoveSitePK/NeuronoveSitePK.pdf>.

[8] ŠÍMA, J. – NERUDA, R. *Teoretické otázky neuronových sítí*. Praha : Matfyzpress, 1st edition, 1996. ISBN 80-85863-18-9.

[9] ŠNOREK, M. *Neuronové sítě a neuropočítače*. Praha : Vydavatelství ČVUT, 2002. ISBN 80-01-02549-7.

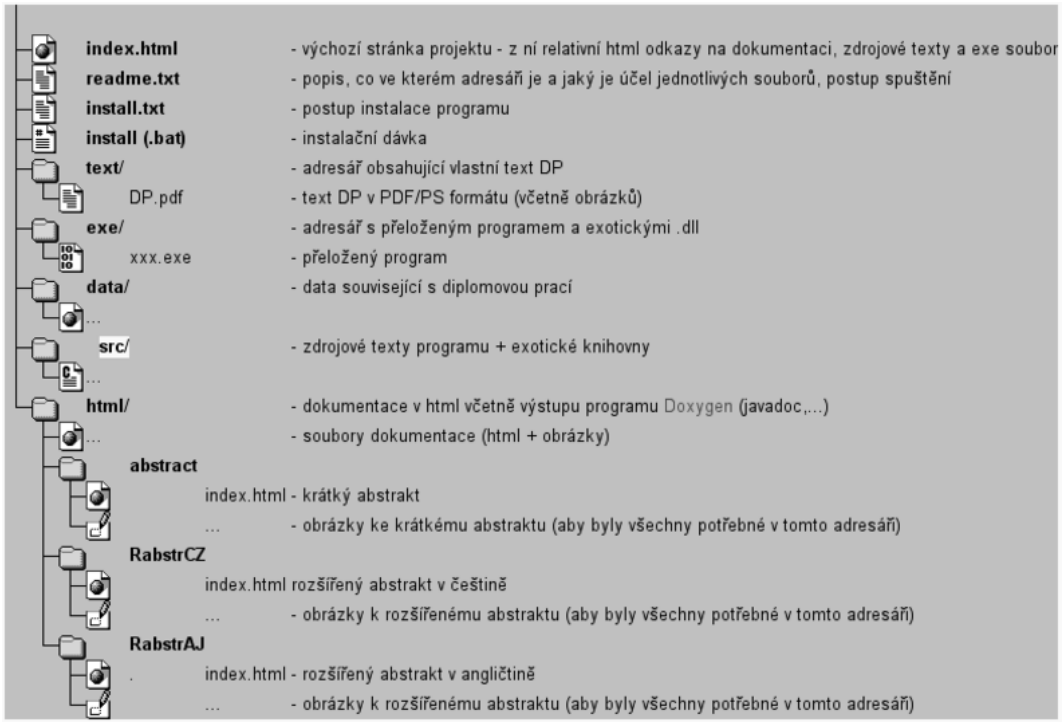
[10] WELLIN, P. R. – GAYLORD, R. J. – KAMIN, S. N. *An Introdution to Programming with Mathematica*. New York : Cambridge University Press, 3rd edition, 2005. ISBN 0-07-159145-1.

Příloha A

Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [**?**]):



Obrázek A.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případne index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.