VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUÁLNÍ SIMULÁTOR A LADÍCÍ PROGRAM PRO NEURONOVÉ SÍTĚ

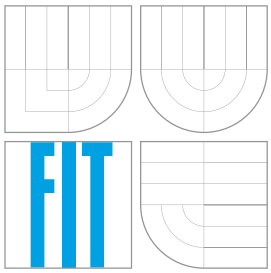
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDREJ BELUSKÝ

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUÁLNÍ SIMULÁTOR A LADÍCÍ PROGRAM PRO NEURONOVÉ SÍTĚ

VISUAL SIMULATOR AND DEBUGGER OF NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDREJ BELUSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. DAVID MARTINEK
SUPERVISOR

BRNO 2008

Abstrakt

Umelé neurónové siete predstavujú paralelné systémy. Táto časť umelej inteligencie je stále skúmaná. Neurónové siete sú definované ako malý výpočtový procesor. Tieto siete disponujú známkami ľudskej inteligencie, pretože majú schopnosť učenia sa. Ich cieľom je simulácia ľudského mozgu, avšak vytvorenie siete so všetkými schopnosťami je vzhľadom k počtu neurónov a prepojení nemožné. Preto sú simulované len niektoré aspekty ľudského myslenia. Mojou úlohou bolo vytvoriť simulátor a ladiaci nástroj pre tieto siete. Program mal umožňovať krokovanie algoritmu pre učenie siete. Taktiež cieľom bolo implementovať editor, ktorý by umožnil vytvorenie a editáciu neurónovej siete.

Klíčová slova

umelé neurónové siete, simulátor, objektovo orientované programovanie, Java, krokovanie algoritmu, učenie, umelá inteligencia

Abstract

Artificial neural networks represent computational parallel systems. This part of artificial intelligence is still researched. Basically neural network is the set of neurons, which are connected together. These networks have aspects of human intelligence, because they have an ability to learn. Their main goal is to simulate human brain, but creating such a network with large number of neurons and connections between them is impossible. They simulate some parts of human thinking. My task was to create visual simulator and debugger of these networks. The program was supposed to have a choice to debugging and stepping the algorithm for learning. My goal was also to implement editor, which allows creating and editing neural networks.

Keywords

artificial neural networks, simulator, object-oriented programming, Java, debugging of algorithm, learning, artificial intelligence

Citace

Ondrej Beluský: Vizuální simulátor a ladící program pro neuronové sítě, bakalářská práce, Brno, FIT VUT v Brně, 2008

Vizuální simulátor a ladící program pro neuronové sítě

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Davida Martinka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondrej Beluský
12. května 2008

Poděkování

Ďakujem Ing. Davidovi Martinkovi za cenné rady a pripomienky pri návrhu, implementácii a pri písaní tejto práce. Taktiež ďakujem všetkým tím, ktorí ma pri práci podporovali a poradili.

Obsah

Kapitola 1

Úvod

Projekt sa zaoberá problematikou neurónových sietí. Sú to matematické modely nervovej sústavy živých organizmov. Modely majú napodobniť paralelizmus a výpočtovú silu ľudského mozgu. Spadajú do odvetvia umelej inteligencie.

Mojou úlohou v tomto bakalárskom projekte bolo navrhnúť a implementovať knižnicu pre prácu s neurónovými sieťami. Preštudoval som rôzne druhy topológií a typy. Pre moju prácu som si vybral viacvrstvové dopredné siete.

Nad knižnicou bola realizovaná aplikácia s grafickým rozhraním. Aplikácia slúži ako editor a simulátor. Simulácia umožňuje animovať alebo krokovať výpočet siete alebo algoritmus učenia siete. Metódu učenia som zvolil najpoužívanejší algoritmus - metódu spätného šírenia chyby a jeho tri modifikácie. Aplikácia bola vyvinutá hlavne pre výukové účely. Vďaka krokovaniu algoritmov a grafickému znázorňovaniu jednotlivých úkonov je možné demonštrovať ako tieto siete fungujú. V editore je poskytnutá značná voľnosť pri vytváraní siete, takže je možné s týmto rôzne experimentovať.

V ďalších kapitolách som zhrnul poznatky a teóriu viacvrstvových neurónových sietí. Popísaný je aj návrh a implementácia simulátora.

Kapitola 2

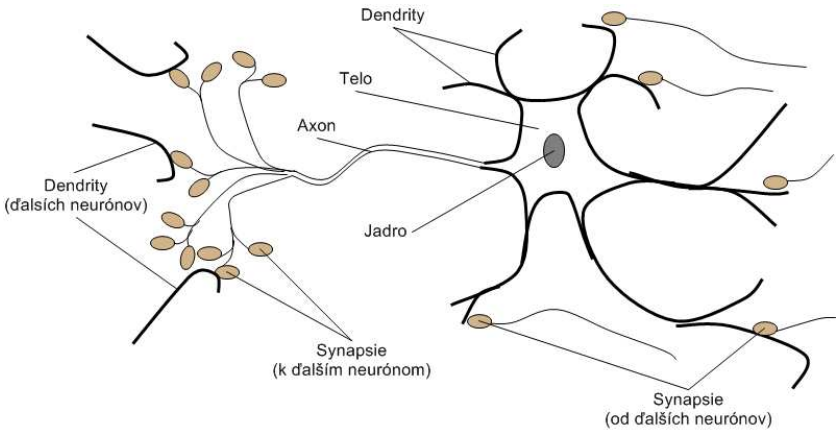
Neurón

V tejto kapitole bude popísané, čo je to neurón, ako sa modeluje biologický neurón a aktivačné funkcie neurónu. Pre popis neurónu som vychádzal z [6] a popis jednotlivých funkcií som čerpal z prednášiek [8].

2.1 Model neurónu

Neurón je špeciálnym typom nervovej bunky v organizme. Tieto bunky sú prevažne určené na efektívne ovládanie organizmu. Neurón sa skladá z tela bunky, ktoré je zabalené v membráne. Ďalej obsahuje vstupy (*dendrites*) a výstupy (*axons*). Výstup neurónu je rozvetvený a je spojený so vstupmi druhých neurónov pomocou synoptických váh. Na obrázku 2.1 je znázornený biologický neurón s popisom jeho častí.

Vstupné signály sa limitujú váhou jednotlivých vstupov a následne vstupujú do tela neurónu. V tele bunky sa signál spracuje. Po spracovaní je generovaný výstupný signál, ktorý ide cez *axon* až k synaptickým váham. Cez synaptické váhy je signál transformovaný do nového vstupného signálu ďalších neurónov. Tento signál môže byť pozitívny alebo negatívny, závisí na type synapsie.

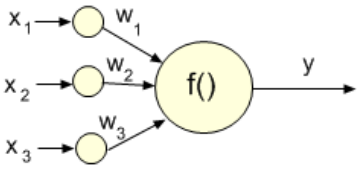


Obrázek 2.1: Biologický neurón

Na podnet biologických neurónov boli navrhnuté ich matematické modely. Tieto modely majú schopnosť učiť sa a zobecňovať predchádzajúce skúsenosti. Vykazujú teda známky

ľudskej inteligencie. Toto je zrejme dôvod, prečo sa stali pozornosťou mnohých laikov ale i odborníkov. [2].

- Prvý formálny model uviedli McCulloch a Pitts (1943), ktorý skombinovali neurofyziológiu a matematiku. Ukázali ako môže byť pomocou *excitačných*, *inhibičných* hrán a prahu zostrojená široká škála neurónov. Model pracuje s diskrétnym časom. Na obrázku 2.3 je ukázané ako môžu byť nastavené váhy a prah neurónu, aby realizovali logické funkcie AND, OR a NOT [1].
- Umelý neurón (obrázok 2.2) príjme množinu vstupných signálov, ktoré sú zvyčajne výstupmi iných neurónov. Každý vstup má určitú váhu, ktorý určitým spôsobom ovplyvňuje vstupný signál – analógia neurónových synapsií. Tieto hodnoty sú ďalej spracované v tele neurónu, kde sa zvyčajne nachádzajú funkcie. Prvá funkcia je tzv. bázová funkcia. Druhou funkciou je aktivačná funkcia (kapitola 2.4). Hodnota je spracovaná a výstupom tejto funkcie je tzv. aktivita neurónu (jeho hodnota).



Obrázek 2.2: Umelý neurón

Ďalej by som doplnil, že podľa [8] existujú dva druhy bázových funkcií:

- lineárne bázové funkcie LBF, napríklad $u = \sum_{i=1}^{n} x_i w_i$
- radiálne bázové funkcie RBF, napríklad $u = \|\vec{x} - \vec{w}\| = \sqrt{\sum_{i=1}^{n} (x_i - w_i)^2}$,

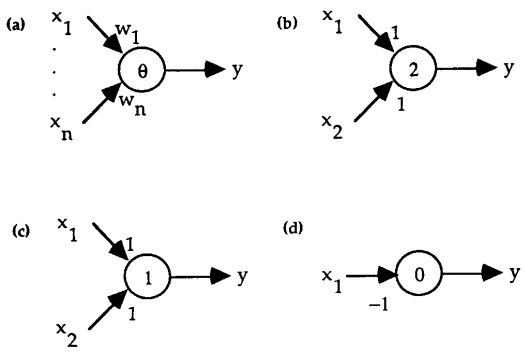
kde u je aktivita počítaného neurónu, n je počet vstupných signálov vstupujúcich do neurónu a w_i je i-tá váha prepojenia. Premenná x_i potom udáva hodnotu vstupného signálu.

2.2 Aktivačné funkcie

Neuróny sú základnou výpočtovou jednotkou v neurónových sieťach. Je to druh jednoduchého procesoru, ktorý príjme váhovaný súčet vstupných signálov, aplikuje na ne nelineárnu funkciu (nie nevyhnutne lineárnu), ktorá sa nazýva aktivačná. Je to uskutočnené predtým ako neurón odošle výstupný signál k ďalším neurónom [3].

Aktivačné funkcie sa delia na:

- nespojité (sekcie 2.2.3, 2.2.4)
- po častiach spojité (sekcie 2.2.1, 2.2.2)
- spojité



Obrázek 2.3: McCullochov a Pittsov neurón. *(a)* Neurón "vystrelí" v čase $t+1$ iba v prípade, že jeho aktuálna hodnota je väčšia alebo rovná prahu v čase t. *(b)* Nastavenie váh a prahu, aby neurón fungoval ako hradlo AND - výstup je 1 v prípade, že oba vstupy sú 1 *(c)* Hradlo OR - výstup je 1 ak jeden zo vstupov je 1 *(d)* Hradlo NOT - výstup neurónu je 1 len ak vstup je 0. Obrázok bol prevzatý z [1].

V tejto podkapitole si predstavíme aktivačné funkcie, ktoré som použil v aplikácii. Existuje viacero aktivačných funkcií, ktoré je možné použiť. V prípade použitia lineárnej bázovej funkcie sú najvhodnejšími tie, ktoré splňujú nasledujúce podmienky:

$$\lim_{x \to -\infty}(x) = 0 \tag{2.1}$$

$$\lim_{x \to \infty}(x) = 1 \tag{2.2}$$

kde x je hodnota vstupného signálu.

V ďalších častiach budeme predpokladať, použitie lineárnej bázovej funkcie,

$$u = \sum_{i=1}^{n} x_i w_i \tag{2.3}$$

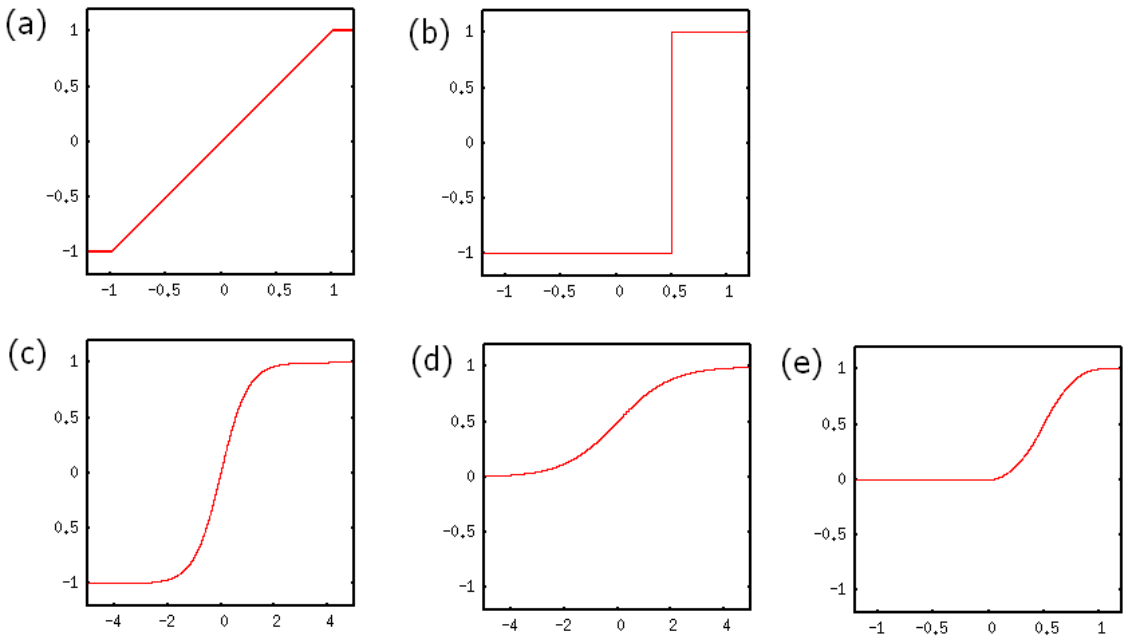
kde n je počet vstupov konkrétneho neurónu, x_i je hodnota vstupu i a w_i je i-tá váha.

2.2.1 Po častiach lineárna funkcia

Predpis tejto funkcie je daný vzťahom 2.4. Na obrázku 2.4 (a) je znázornený priebeh. Nazýva sa tiež rampová, pretože ako z obrázka vidieť, má tvar rampy.

$$y^{nové} = \begin{cases} a & \text{pre} \quad u < c \\ b & \text{pre} \quad u > d \\ a + \frac{(b-a)(u-c)}{d-c} & \text{pre} \quad c \leq u \leq d \end{cases} \tag{2.4}$$

Parameter u je výstup lineárnej bázovej funkcie (rovnica 2.3), a je minimálna hodnota grafu, b je maximálna hodnota grafu, c je hodnota x-ovej osi, od ktorého začína funkcia stúpať a parameter d je hodnota, po ktorú funkcia stúpa.



Obrázek 2.4: Aktivačné funkcie: *(a)* po častiach lineárna funkcia, *(b)* signum (skoková), *(c)* sigmoidálna, *(d)* hyperbolický tangens a *(e)* čiastočne lineárna/čiastočne spojitá

2.2.2 Skoková funkcia (signum)

Skoková funkcia (*signum*) je daná vzťahom 2.5. Na obrázku 2.4 (b) je znázornený priebeh s hodnotami konštánt: $a = -1, b = 1, \theta = 0{,}5$.

$$y^{nové} = \begin{cases} a & \text{pre} \quad u < \theta \\ b & \text{pre} \quad u > \theta \\ y^{staré} & \text{pre} \quad u = \theta \end{cases} \tag{2.5}$$

Parameter funkcie a je spodná hodnota grafu, parameter b naopak udáva hodnotu hornej časti grafu. Symbol θ je hodnota x-ovej osy, odkiaľ funkcia zmení svoju hodnotu a u je opäť definované ako v rovnici 2.3.

2.2.3 Sigmoidálna funkcia

Sigmoidálna funkcia patrí medzi spojité funkcie. Je to jedna z najpoužívanejších aktivačných funkcií pre neurónové siete. Jej predpis má tvar 2.6. Avšak premenné a, b, c majú často hodnoty $a = 0, b = 1, c = 0$ (zjednodušený predpis 2.7). Pre tieto hodnoty má graf tvar uvedený na obrázku 2.4 (c). Parameter λ určuje ostrosť (*sklon*) funkcie. Tieto parametre sú nejaké vhodne zvolené konštanty.

$$y = a + \frac{b - a}{1 + e^{(-\lambda u + c)}} \tag{2.6}$$

$$y = \frac{1}{1 + e^{(-\lambda u)}} \tag{2.7}$$

2.2.4 Hyperbolický tangens

Hyperbolický tangens je opäť ďalšou spojitou funkciou, používanou v neurónových sieťach. Funkcia $\tanh(u)$ je definovaná ako 2.8, jej graf je vykreslený na obrázku 2.4 (d).

$$y = \tanh(u) = \frac{\sinh(u)}{\cosh u} = \frac{e^u - e^{-u}}{e^u + e^{-u}} \tag{2.8}$$

2.2.5 Po častiach lineárna/spojitá funkcia

V aplikácii bola použitá ešte jedna aktivačná funkcia 2.9 (obrázok 2.4 (e)), ktorá bola uvedená v zdroji [4], v príkladoch na precvičenie.

$$y^{nové} = \begin{cases} 0 & \text{pre} \quad u \leq 0 \\ 2u^2 & \text{pre} \quad 0 \leq u \leq 0.5 \\ 1 - 2(u-1)^2 & \text{pre} \quad 0.5 \leq u \leq 1 \\ 1 & \text{pre} \quad u \geq 1 \end{cases} \tag{2.9}$$

Kapitola 3

Umelá neurónová sieť

Táto kapitola má za cieľ predstaviť neurónové siete. Popisuje štruktúru neurónových sietí a využitie v praxi. Podklad k textu som čerpal zo zdroju [4]. Prípadné iné použité zdroje boli citované priamo za textom, ktorý bol prebratý.

3.1 Neurónové siete obecne

Počítače sú programované, aby riešili určité problémy, používajúc sekvenčné algoritmy. Narozdiel od nich ľudský mozog používa masívnu sieť paralelných výpočtových elementov zvaných neuróny(2.1). Veľký počet prepojení spájajúcich tieto elementy produkuje veľmi mocnú schopnosť učenia sa. Vedci motivovaný týmto veľmi efektívnym biologickým výpočtovým modelom sa desaťročia pokúšali zostrojiť podobné výpočtové systémy. Systémy, ktoré by umožňovali spracovať informácie založené na tomto princípe. Tieto systémy sa nazývajú umelé neurónové siete[3].

Neurónové siete sú považované za tzv. "čierne skrinky" (*black boxes*). Ich hlavnou výhodou oproti von neumanovskej architektúry je schopnosť učiť sa a adaptovať. Na rozdiel od algoritmov a funkcií, nie je definovaný presný postup riešenia problému. Sieť sama prispôsobuje svoje váhy v adaptačnom procese, pri ktorom sa učí zo vzorových príkladov [2].

3.2 Štruktúra neurónových sietí

Štruktúra (*topológia*) odpovedá zoradeniu a organizácii uzlov zo vstupnej vrstvy k výstupnej vrstve. Spôsob akým sú neuróny a prepojenia usporiadané vo vrstvách danej neurónovej siete určuje jej topológiu. Výber použitia určitej topológie je založený na probléme, ktorý má riešiť.

Základné delenie sietí je na jednovrstvové a viacvrstvové. Jednovrstvové siete (3.2.2.a) dokážu riešiť problémy, ktoré sú lineárne oddeliteľné. Oveľa väčšiu výpočtovú silu majú viacvrstvové siete (3.2.2.b,c). Tieto siete obsahujú jednu alebo viac skrytých vrstiev (teoreticky stačí jedna skrytá vrstva). Nazývajú sa skrytými, pretože ich vstupy a výstupy nie sú prístupné z vonkajšieho sveta. Definované sú dva vektory. Vstupný vektor pre vstupnú vrstvu a požadovaný výstupný (poslednú vrstvu siete).

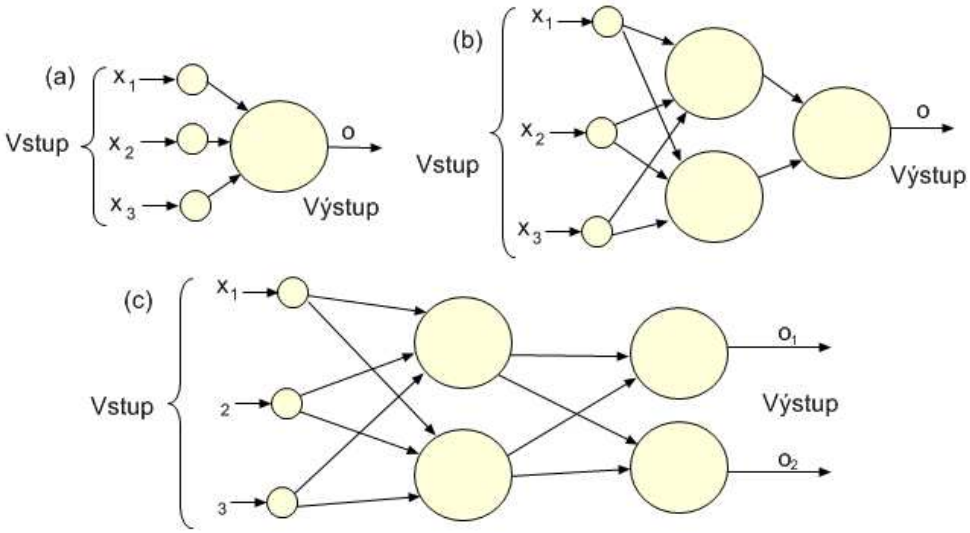
Ďalej sa budem zaoberať len viacvrstvovými topológiami, pretože boli predmetom mojej práce. Známe štruktúry viacvrstvového rozloženia sú dopredné a rekurentné architektúry (3.2.3).

3.2.1 Acyklická sieť

Táto sieť je podmnožinou vrstvových sietí. Acyklické siete nemajú prepojenia neurónov v rámci jednej vrstvy. Inými slovami prepojenia môžu existovať len medzi uzlom vo vrstve i a druhým uzlom vo vrstve j pre $i < j$, ale prepojenia pre $i = j$ nie sú povolené (tak je to vo vrstvových sieťach).

Výpočtový proces v acyklickej sieti je oveľa jednoduchší než s tými sieťami, ktoré obsahujú cykly a vzájomné prepojenia vo vrstve. Siete, ktoré nie sú acyklické sú považované za rekurentné siete 3.2.3.

3.2.2 Dopredná sieť



Obrázek 3.1: Typické štruktúry dopredných neurónových sietí, (a) viacvstupová, jednovrstvová s jedným výstupom, (b) viacvstupová, jeden výstup, jedna skrytá vrstva (c) viacvstupová, viacvýstupová s jednou skrytou vrstvou

Dopredné siete sú podmnožinou acyklických sietí. Sieť sa skladá z prepojených uzlov z vrstvy i len s uzlami vo vrstve $i + 1$.

Výstup siete vždy závisí len od konkrétneho vstupu, neberie ohľad nad predchádzajúce vstupy. Používa sa aj modifikácia týchto sietí, ktorých výstup závisí na predchádzajúcich hodnotách. Vstup z predchádzajúceho výpočtu sa priloží opäť na vstup spolu s novým vstupným vektorom. Viac o týchto sieťach je možné naštudovať v týchto knihách [4, 6, 3].

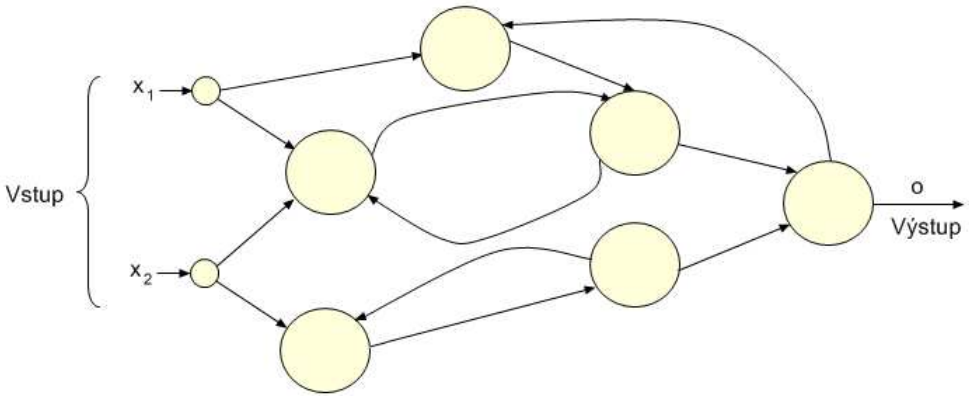
Tieto siete sú zvyčajne stručne popísané sekvenciou čísiel, ktoré určujú počet uzlov v každej vrstve. Napríklad táto sekvencia čísiel 3-2-3-2 reprezentuje doprednú sieť, ktorá pozostáva z troch vstupných uzlov vo vstupnej vrstve, dvoch neurónov v prvej skrytej vrstve, troch neurónov v druhej skrytej vrstve a dvoch neurónov vo výstupnej vrstve.

Dopredné siete zvyčajne s nie viac ako štyrmi takýmito vrstvami patria medzi bežne používané neurónové siete. Pod pojmom "neurónové siete" väčšina užívateľov rozumie práve len dopredné siete. Na obrázku 3.2.2 sú znázornené viacvrstvové dopredné siete (b, c).

V nasledujúcom texte, budeme považovať za dopredné siete všetky siete, ktoré sa nedajú považovať za rekurentné (teda aj acyklické). Tento pojem vyjadruje smer šírenia sa signálu (smerom vpred od vstupu k výstupu).

3.2.3 Rekurentná sieť

Táto topológia je jednou zo sietí, ktoré sa používajú na predikciu. Rekurentné siete môžu pozostávať z prepojení z výstupnej vrstvy do skrytých vrstiev a/alebo do vstupnej vrstvy. Povolené sú prepojenia v rámci jednej vrstvy a taktiež spojenia medzi skrytými vrstvami. Veľa druhov rekurentných sietí bolo navrhnutých, závisiac na podstate problému, ktorému boli adresované.



Obrázek 3.2: Príklad rekurentnej neurónovej siete

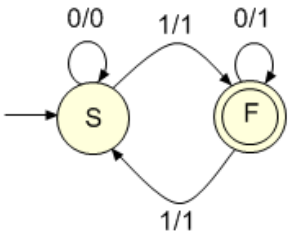
Nazačiatku má sieť definovaný počiatočný stav. Cez spätné väzby, ktoré sa vyskytujú v tejto topológii, sa šíria v čase t hodnoty neurónov z času $t-1$. Týmto je dosiahnuté, že súčasný stav siete je ovplyvnený predchádzajúcimi vstupnými vektormi.

Jednoduchý príklad, kedy potrebujeme, aby sme pre konkrétny vstup mali viacero výstupov je jednoduchý konečný automat[1]. Na obrázku 3.3 je príklad takéhoto automatu. Stav S je štartovací bod a F je naopak koncový stav. Číslo pred lomítkom značí vstup, číslo za lomítkom korešpondujúci výstup. Ak sa nachádzame v stave S a vstupom je 0 výstup je 0. Avšak ak sa dostaneme do stavu F a príde na vstup 0 výstupom je už 1. Ak by sme chceli naučiť neurónovú sieť, aby dokázala simulovať takýto automat, museli by sme použiť niektorú vhodnú z rekurentných topológií, pretože pre jeden vstup môžeme mať viac výstupov (konkrétne 2).

3.3 Využitie v praxi

- Klasifikácia objektov do určitej triedy (jednou z viacerých definovaných) je dôležitým základom rozpoznávania čísiel z obrázku, rozpoznávania reči. K dispozícii máme trénovaciu množinu, ktorá obsahuje ukážkové vzory, ktoré sú zástupcami jednotlivých tried. Použitím tejto množiny si neurónová sieť vydedukuje pravidlá, ktoré rozhodnú o členstve objektu danej triedy. Napríklad každý výstup siete predstavuje jednu triedu. Vstupná vzorka je priradená k triede i, ak i-ty výstupný neurón vypočítal najväčšiu hodnotu so všetkých výstupných neurónov, keď bola táto vzorka priložená na vstup siete. V niektorých sieťach je dôležité, aby hodnota výstupného neurónu prekročila

[1] Jednoduchý neformálny popis: Konečné automaty pozostávajú zo stavov a z hrán spájajúce tieto stavy. Podľa vstupu a konkrétneho stavu sa automat presunie do ďalšieho stavu. Viac o automatoch a problémoch, ktoré riešia je možné naštudovať v literatúre, ktoré sa nimi zaoberajú.



Obrázek 3.3: Príklad jednoduchého konečného automatu

zadaný prah. Neurónové siete boli úspešne použité vo veľkom množstve praktických rozpoznávacích úlohách, ako napríklad rozpoznávanie ručne písaných znakov alebo analýza sonarových a radarových dát na zistenie pôvodu zdroja signálu

- Mnoho výpočtových modelov môže byť popísaných ako funkcia mapujúca vstupné vektory na vektory výstupné. Výstupy korešpondujúce k nejakým vstupným vektorom môžu byť známe z trénovacích dát, ale nemusíme vedieť matematickú funkciu popisujúcu aktuálny proces, ktorý generuje výstupy zo vstupným vektorov. Aproximácia funkcií je úlohou zostrojovania funkcie, ktorá generuje približne rovnaké výstupy ako proces, ktorý je modelovaný, založený na dostupných trénovacích dátach.

- Existuje veľa skutočných úloh, ktoré potrebujú predpovedať udalosti v budúcnosti na základe udalostí minulosti. Príkladom môže byť vývoj ceny na marketingovom trhu. Perfektná predikcia nie je skutočne možná, ale neurónové siete môžu byť použité na získanie rozumne dobrej predikcie v mnohých prípadoch.

- A ešte veľa ďalších ako ovládanie aplikácií, optimalizácia, riadenie automobilu, prehľadávanie . . .

Kapitola 4

Učenie

4.1 Učenie s učiteľom a bez učiteľa

Za učenie neurónových sietí sa považuje proces, pri ktorom sa menia hodnoty váh jednotlivých prepojení. Po vytvorení siete je dobré inicializovať všetky váhy na náhodné hodnoty (v rozmedzí od $-0,5$ až $0,5$). Učením sa tieto váhy upravujú, aby sa dosiahli požadované výsledky (napríklad určitým vstupom produkovali korešpondujúce výstupy). Neurónové siete akceptujú určitú tolerovateľnú chybu. Veľkosť tejto chyby závisí na probléme, ktorému sú vystavené.

Schopnosť učiť sa je typickou známkou ľudskej inteligencie. Najväčším problémom pri hodnotení generalizačných schopností neurónových sietí je určiť čo je to správna generalizácia. Napríklad pri určení ďalšieho členu rady $1, 2, 3 \cdots$, by väčšina ľudí doplnila číslo 4. Avšak nejaký matematik by si mohol všimnúť, že číslo 3 je súčtom predchádzajúcich dvoch čísiel a doplnil by ako ďalšie číslo 5. Nie je vôbec jasné, ktoré z uvedených doplnení je správnou generalizáciou postupnosti. [2]

Algoritmy pre učenie sietí sa rozdeľujú do dvoch hlavných skupín[5]:

1. kontrolované učenie vyžaduje "učiteľa", ktorý určí želaný výstup k priloženému vstupu. Algoritmy pre učenie potom prispôsobujú všetky potrebné váhy (môže byť veľmi komplikované) a celý proces sa opakuje pokiaľ sú dáta sieťou správne analyzované.

2. nekontrolované učenie, ktoré nevyžaduje "učiteľa", pretože vyprodukuje svoj vlastný výstup, ktorý je ešte ďalej vyhodnotený a spracovaný.

Príklad [4]:

1. Archeológ nájde ľudskú kostru a musí určiť, či patrila žene alebo mužovi. Vďaka predchádzajúcim skúsenostiam z minulosti je možné určiť komu kostra patrí. Skúmanie týchto znalostí (zvaných trénovacia množina) dovoľuje archeológovi rozpoznávať rozdiel medzi mužskou a ženskou kostrou. Toto je príklad učenia sa s učiteľom a výsledok tohto procesu môže byť použitý na určenie, či novoobjavená kostra patrí mužovi.

2. Majme inú situáciu, archeológ musí rozhodnúť či skupina úlomkov z kostry patrí tomu istému dinosaurovi alebo patria rôznym druhom. Pre túto úlohu nemusia existovať žiadne predchádzajúce dáta, ktoré jasne určia živočíšny druh pre každý úlomok kosti. Archeológ rozhodne, či tieto úlomky sú dostatočne podobné na to, aby patrili jednému druhu. Ak sú veľmi rozdielne rozdelí ich do odlišných druhov. Tento príklad popisuje učenie sa bez učiteľa, ktorá odhaduje stupeň rozdielov medzi jednotlivými úlomkami.

Jeden archeológ môže byť presvedčený, že úlomky patria k viacerým druhom, zatiaľ čo iný nemusí súhlasiť a nie je tu absolútne kritérium na určenie toho, kto má pravdu.

4.2 Metóda spätného šírenia chyby (backpropagation)

Najznámejší a najpoužívanejší algoritmus pre učenie viacvrstvových dopredných sietí je metóda spätného šírenia chyby, ktorú používa 80 % všetkých aplikácií, ktoré používajú neurónové siete [2].

Pri ďalšom popise algoritmu som sa inšpiroval z týchto zdrojov [3, 4].

Algoritmus je založený na výpočte kumulatívnej chyby E_c na výstupnej vrstve, výpočtu gradientu a šírenia chyby do skrytých vrstiev. Chyba E_c reprezentuje sumu n štvorcových chýb $E(k)$, kde

$$E(k) = \sum_{i=1}^{q} [(t_i(k) - o_i(k))^2] \tag{4.1}$$

Predstavuje chybu medzi k-tým želaným výstupným vektorom $t(k)$ a k-tým aktuálnym výstupným vektorom $o(k)$ siete. Index i definuje i-tý neurón vo výstupnej vrstve, ktorá pozostáva z q neurónov.

Po vypočítaní gradientu sa pre každý neurón spočítajú delty a jednotlivé váhy sú modifikované korešpondujúcou deltou. Tento postup je vykonávaný iteratívne až po prekročení určitého času, či počtu iterácií alebo dosiahnutie chyby akceptovateľne malej chyby E_c, ktorá je definovaná ako

$$E_c = \frac{1}{2} \sum_{k=1}^{n} E(k) \tag{4.2}$$

Počet vzorov v trénovacej množine určuje n. Aby sme minimalizovali chyby E_c alebo $E(k)$ musíme nájsť správne hodnoty váh. Existujú dva prípady minimalizácie:

- aktualizácia váh prebieha až po naučení celej trénovacej množiny *(offline učenie)*
- váhy sú aktualizované po každom trénovacom vzore z množiny *(online učenie)*

Pre *online* učenie platí:

$$\triangle \mathbf{w}^{(l)} = -\eta \frac{\partial E(k)}{\partial \mathbf{w}^{(l)}} \tag{4.3}$$

kde $\mathbf{w}$ ja taký vektor všetkých váhových prepojení siete, ktorý vedie k minimalizácii chyby $E(k)$ a $\frac{\partial E(k)}{\partial w^{(l)}}$ je gradient chyby $E(k)$. Symbol $\mathbf{w}^{(l)}$, ktorý zodpovedá všetkým prepojeniam medzi vrstvou (l) a predchádzajúcou vrstvou $(l-1)$. Koeficient učenia je daný symbolom η, je to malé pozitívne číslo (obvykle medzi 0 a 1). Koeficient η sa odzrkadľuje na konvergencii a stabilite učiaceho algoritmu. Symbol $\triangle \mathbf{w}^{(l)}$ označuje rozdiel medzi vektorom $\mathbf{w}^{(l)}(k+1)$ a $\mathbf{w}^{(l)}(k)$, reprezentujúca váhové prepojenia smerujúce k vrstve (l) po, respektíve pred použitím trénovacieho vzoru k.

Prírastok, respektíve úbytok váhy prepojenia spájajúceho bunku j z vrsvty $(l-1)$ s bunkou i nachádzajúcou sa vo vrstve l označíme ako $\triangle w_{ij}^{(l)}$. Ďalej ak $o_j^{(l-1)}$ reprezentuje výstup bunky j a hodnota $\mathrm{tot}_i^{(l)}$ predstavuje sumu výstupov všetkých predchádzajúcich buniek z vrstvy $(l-1)$, ktoré vstupujú do neurónu i, môžeme definovať:

$$\triangle \mathbf{w}^{(l)} = \triangle w_{ij}^{(l)} = -\eta \Big[\frac{\partial E(k)}{\partial o_i^{(l)}}\Big] \Big[\frac{\partial o_i^{(l)}}{\partial \mathrm{tot}_i^{(l)}}\Big] \Big[\frac{\partial \mathrm{tot}_i^{(l)}}{\partial w_{ij}^{(l)}}\Big] \tag{4.4}$$

Pre výstupnú vrstvu (L) je definícia 4.4 vyjadrená ako

$$\triangle w_{ij}^{(l)} = \eta[t_i - o_i^{(L)}][f'(\text{tot}_i^{(L)})]o_j^{(L-1)} \tag{4.5}$$

kde $f'(\text{tot}_i^{(L)}) = \frac{\partial f(\text{tot}_i^{(L)})}{\partial \text{tot}_i^{(L)}}$.

Keď označíme $\delta_i^{(L)} = [t_i - o_i^{(L)}][f'(\text{tot}_i^{(L)})]$ ako chybu signálu i-teho neurónu výstupnej vrstvy, dostaneme výraz pre zmenu váh vo výstupnej vrstve (L):

$$\triangle w_{ij}^{(L)} = \eta\delta_i^{(L)} o_j^{L-1} \tag{4.6}$$

V prípade, že aktivačná funkcia f je sigmoidálna funkcia, chybový signál má tvar:

$$\delta_i^{(L)} = (t_i - o_i^{(L)})o_i^{(L)}(1 - o_i^{(L)}) \tag{4.7}$$

Ďalej sa chyba šíri spätne pre ostatné vrstvy. Ak vrstva (l) reprezentuje skrytú vrstvu $(l < L)$, výpočet pre $\triangle w_{ij}^{(l)}$ je daný vzťahom:

$$\triangle w_{ij}^{(l)} = \eta\delta_i^{(l)} o_j^{l-1} \tag{4.8}$$

kde chybový signál $\delta_i^{(l)}$ je teraz vyjadrený ako funkcia výstupu predchádzajúcej počítanej vrstvy $(l+1)$:

$$\delta_i^{(l)} = f'(\text{tot}_i^{(l)}) \sum_{p=1}^{n_l} \delta_p^{(l+1)} w_{p_i}^{(l+1)} \tag{4.9}$$

Znova, ak aktivačná funkcia je sigmoidálna, $\delta_i^{(l)}$ je vyjadrená ako:

$$\delta_i^{(l)} = o_i^{(l)}(1 - o_i^{(l)}) \sum_{p=1}^{n_l} \delta_p^{l+1} w_{p_i}^{(l+1)} \tag{4.10}$$

Index n_l v rovniciach 4.9 a 4.10 značia celkový počet neurónov vo vrstve $(l+1)$, index i reprezentuje neurón vrstvy (l) a index p je neurón vrstvy $(l+1)$. Názov algoritmu je odvodený od toho, že $\delta_i^{(l)}$ je počítaná z $\delta_p^{(l+1)}$ ďalšej vrstvy, teda je spätne šírená.

V prvom spomínanom prípade, zvaným aj ako *offline* trénovanie, sa pravidlo pre výpočet úbytku delty počíta ako:

$$\triangle \mathbf{w}^{(l)} = -\eta \frac{\partial E_c}{\partial \mathbf{w}^{(l)}} \tag{4.11}$$

Všetky predchádzajúce kroky spomínané pre výpočet *online* trénovania sú reprodukované na *offline* učenie tak, že $E(k)$ je nahradené E_c. V každom cykle ako je možné vidieť z definicií 4.5 a 4.6, hodnoty váh sú šírené spätne, začínajúc vo výstupnej vrstve pokračujúc cez všetky skryté vrstvy až po vstupnú vrstvu siete. Zatiaľ čo veľká hodnota η môže zrýchliť proces učenia, takisto to môže viesť k oscilácii.

4.3 Modifikácia algoritmu

Predchádzajúci popisovaný algoritmus pre *online učenie* sa taktiež nazýva *vanilla backpropagation*. Pre tento algoritmus sa prírastok, respektíve úbytok váhy prepojenia od bunky i

k bunke j $\triangle w_{ij}(t+1)$ vypočíta podľa vzťahu 4.8. *Batch backpropagation* je naopak názov algoritmu pre *offline učenie*.

Zdokonalený backpropagation (*enhanced backpropagation*) je modifikáciou *vanilla* algoritmu. Prírastok $\triangle w_{ij}(t+1)$ sa počíta ako:

$$\triangle w_{ij}(t+1) = \eta \delta_j o_i + \alpha \triangle w_{ij}(t) \tag{4.12}$$

kde α je tzv. momentum. K aktuálnemu prírastku v čase t sa pripočíta prírastok z času $(t+1)$, ktorý je limitovaný momentom α. Touto úpravou je možné neurónovú sieť rýchlejšie adaptovať.

Ďalšou modifikáciou algoritmu *vanilla* je metóda rozpadania váh (angl. weight decay):

$$\triangle w_{ij}(t+1) = \eta \delta_j o_i - d w_{ij}(t) \tag{4.13}$$

Rozpadanie váh pridáva určitú penaltu každej váhe. Táto penalta je daná predošlou hodnotou váhy, ktorá je limitovaná konštantou d. Tento algoritmus je podmnožinou regularizačných metód. Pravidlo udeľovania penált penalizuje hlavne váhy, ktoré majú veľké hodnoty. Tieto penalty spôsobujú konvergenciu váh k malým absolútnym hodnotám. Viac o tomto algoritme je popísané v [7].

Kapitola 5

Návrh aplikácie

5.1 Súčasný stav

Bolo navrhnutých a zrealizovaných viacero simulátorov pre neurónové siete. Väčšina z nich je zložitá na ovládanie a pochopenie samotnej aplikácie. Tieto simulátory simulujú konkrétny druh sietí. Pre vytvorenie siete sa špecifikuje počet vrstiev a počet neurónov. Nestretol som sa so simulátorom, ktorý by umožňoval krokovať výpočet učenia.

Mojím cieľom bolo vytvoriť aplikáciu, ktorej ovládanie bude takmer intuitívne. Vytvorenie siete bude pripomínať kreslenie v grafickej aplikácii. Výsledný produkt by mal slúžiť ako výuková pomôcka pre neurónové siete. Hlavnou úlohou simulátoru je krokovanie algoritmu pre učenie siete a grafické znázornenie jednotlivých úkonov pre ľahšie pohopenie toho ako siete fungujú.

5.2 Implementačný jazyk

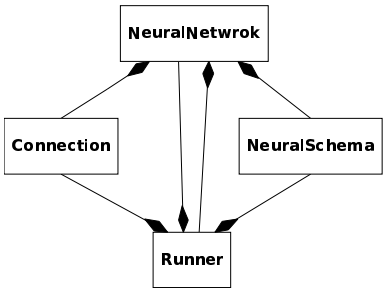
Pre implementáciu bol vybratý programovací jazyk *Java*. Java je čisto objektovo orientovaný jazyk. Zadanie požadovalo prenositeľnosť a tento jazyk je platformovo nezávislý. Pre tvorbu užívateľského rozhrania bola zvolená technológia *swing*. Swing je celkom nezávislý na hosťovskom operačnom systéme, pretože svoje komponenty vykresľuje pomocou *Java 2D api*.

5.3 Návrh knižnice

Knižnica bola navrhnutá tak, aby poskytovala možnosť vytvorenia ľubovoľnej viacvrstvovej siete. Knižnica funguje ako jeden celok. Obsahuje informácie o prepojeniach, jednotlivých váhach týchto prepojení a dátových buniek 5.1. Knižnica obsahuje metódy pre prácu s jednotlivými bunkami, vrstvami, spojeniami. Navrhnutá knižnica neobsahuje radiálne bázové funkcie, to môže byť prípadným rozšírením.

5.3.1 Neuróny a vstupy (*bunky*)

Neuróny museli byť patrične odlíšené od vstupných jednotiek. Vstupy nesú informáciu len o ich aktuálnej hodnote. Neurón predstavuje väčšiu výpočtovú jednotku v sieti. Obsahuje hodnotu výstupu, prípadne je možné ukladať výstupy z rôznych časových jednotiek (pri použití rekurentnej siete). Každý neurón taktiež nesie informáciu o svojej aktivačnej



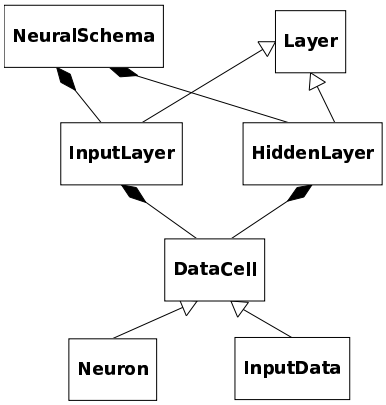
Obrázek 5.1: Neurónová sieť (`NeuralNetwork`) obsahuje prepojenia (`Connection`), schému (`NeuralSchema`) a algoritmus na učenie siete (`Runner`).

funkcii, ktorá modifikuje výstup z lineárnej bázovej funkcie. Pri učení je možné ukladať hodnotu delty, ktorou sa budú meniť jednotlivé váhy.

Bunky neobsahujú informácie o prepojeniach s inými neurónmi. Každý neurón má svoje jedinečné identifikačné číslo medzi ostatnými neurónmi. Taktiež všetky vstupy sú rozlíšené svojím *id*.

5.3.2 Vrstvy

Sieť vždy obsahuje vstupnú vrstvu, ktorá obsahuje len vstupy. Táto vrstva sa nemusí vytvárať, po vytvorení siete je automaticky vytvorená. Po vstupnej vrstve nasleduje jedna alebo niekoľko skrytých vrstiev, pričom posledná skrytá vrstva sa vždy považuje za výstupnú. Skryté vrstvy a výstupná vrstva obsahujú už len jednotlivé neuróny.



Obrázek 5.2: Reprezentácia vrstiev a buniek v `NeuralSchema`

Podobne ako neuróny aj vrstvy majú svoje *id*. *Id* vrstvy a *id* bunky sú nezáporné číselné hodnoty, ktoré sú automaticky generované. Jednoznačná identifikácia neurónu (alebo vstupu) v konkrétnej vrstve je daná identifikátorom "i:j". kde i je *id* vrstvy a j je *id* neurónu.

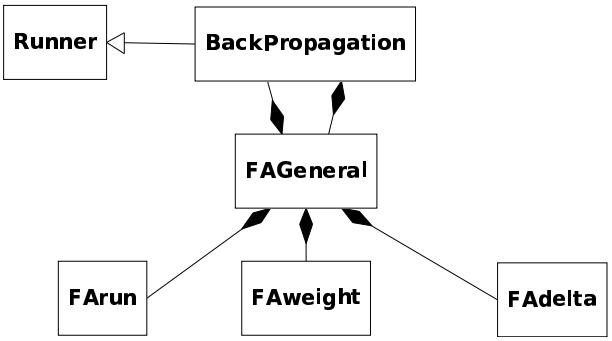
5.3.3 Prepojenia

Ako som už spomínal, informácie o prepojení siete nie sú súčasťou daných buniek. Je možné spájať nielen vrstvu i s vrstvou $i+1$, ale aj s ľubovoľnou skrytou alebo výstupnou vrstvou,

aby bolo možné vytvárať acyklické a rekurentné siete.

Spojenie je určené reťazcom "id1,id2", kde $id1$ je identifikátor zdrojovej bunky a $id2$ je cieľový uzol. Každé prepojenie je potom jednoznačne odlíšiteľné od ostatných, pričom nemôžu existovať dve alebo viac spojení medzi tými istými bunkami.

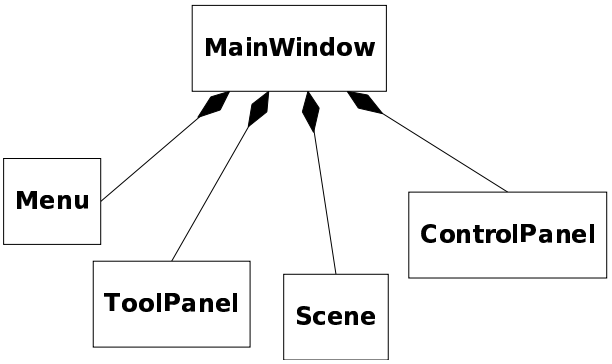
5.3.4 Učenie



Obrázek 5.3: UML diagram algoritmu `BackPropagation`, využitie štyroch konečných automatov pre krokovanie výpočtu a učenia siete. (Konečné automaty sú reprezentované triedami s predponou FA - finite automata)

Učenie je najdôležitejšou súčasťou knižnice. Knižnica bude umožňovať rozšírenie a ďalšie algoritmy učenia. Učenie bude možné krokovať po jednotlivých úkonoch alebo sa sieť vypočíta bez krokovania kvôli rýchlosti. Pre učenie dopredných sietí bol zvolený algoritmus spätného šírenia chyby a jeho tri modifikácie popisované v kapitole 4. 4.2.

5.4 Aplikácia



Obrázek 5.4: Hlavné okno (`MainWindow`) obsahuje menu (`Menu`), panel nástrojov (`ToolPanel`), pracovnú plochu (`Scene`) a panel zobrazujúci informácie o bunkách (`ControlPanel`).

Hlavné okno obsahuje pracovnú plochu obsahujúcou neurónovú sieť. Aplikácia má dva režimy. Prvý je editačný mód, ktorý je implicitne spustený pri štarte aplikácie. Druhým

režimom je simulačný mód, kde prebieha samotná simulácia výpočtu a učenia siete.

5.4.1 Grafický editor

V editačnom móde obsahuje aplikácia prvky pre vytvorenie a editáciu siete. Pri vytváraní a editácii siete sa vytvára nielen grafická podoba, ale na pozadí sa tvorí dátová (fyzická) podoba siete.

Vrstvy

V tomto móde aplikácia využíva funkcie knižnice pre vytváranie alebo mazanie jednotlivých vrstiev. Po vytvorení vrstvy je možné pridávať ľubovoľný počet neurónov. Prvou vrstvou je vstupná a posledná je vždy výstupná vrstva.

Bunky

Kvôli väčšej flexibilite je umožnené pridávať vstupy na ploche aj do skrytých vrstiev. Na pozadí sú však vstupy pridané do vstupnej vrstvy, kam patria. Neuróny je možné presúvať z jednej vrstvy do druhej, pričom sa zachovajú pôvodné prepojenia s ostatnými neurónmi a vstupmi. Je možné presúvať aj vstupy, tie sú však presúvané len graficky, fyzicky sú stále vo vstupnej vrstve. Po presunutí neurónu z jednej vrstvy do druhej sa zmení aj jeho fyzická poloha v sieti, nielen pozícia na pracovnej ploche.

Každému neurónu sa po pridaní do siete implicitne nastaví aktivačná funkcia, ktorá bude modifikovať výstup z bázovej funkcie. Po zmene tejto funkcie bude konkrétny neurón používať nastavenú funkciu.

Vstupy sú rozlišované ako klasické alebo konštantné. Klasickým vstupom sa mení hodnota a tieto vstupy sú počítané ako vstupy od užívateľa. Konštantné vstupy predstavujú stále hodnoty. Po nastavení ich konštantnej hodnoty v editačnom móde, nie je možné v simulačnom túto hodnotu meniť.

Prepojenia

Ďalej je tu interaktívna možnosť vytvárania prepojení medzi jednotlivými uzlami. Vytvorením spojenia sa nastaví náhodná hodnota váhy tohto prepojenia. Každé prepojenie sa viaže na určité dva uzly. Prepojenie je vektor, ktorý určuje z ktorého uzlu sa vychádza a do ktorého uzlu sa vstupuje.

Ukladanie a načítanie

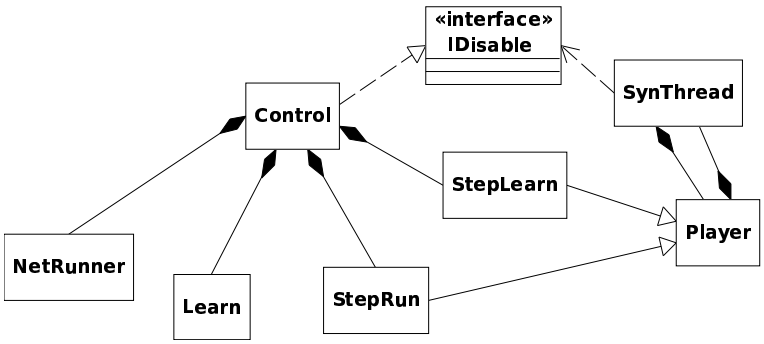
Po ukončení editácie sa dá neurónová sieť exportovať do formátu XML. Taktiež je možné kedykoľvek v editačnom móde sieť z XML načítať. Editor umožňuje interaktívnu tvorbu siete, kde je možné odstrániť jednotlivé vrstvy a ľubovoľne vrstvy umiestňovať. Preto pred uložením sieťe alebo prepnutím sa do simulačného módu je sieť optimalizovaná. Optimalizácia spočíva v odstránení nadbytočným vrstiev, ktoré neobsahujú neuróny a zmena identifikátorov neurónov a vrstiev, aby boli zoradené vzostupne.

5.4.2 Simulácia

Trénovacie množiny

Prepnutím do simulačného módu sa predpokladá vytvorenie alebo načítanie trénovacích množín. Trénovacie množiny obsahujú jednotlivé vzory, zložené zo vstupného vektoru a želaného výstupného vektoru. Množina môže obsahovať ľubovoľný počet vzorov a taktiež aplikácia môže definovať viacero trénovacích množín. Každá množina má svoje jedinečné pomenovanie, ktorým je určená. Je to aj hlavne kvôli prehľadnosti, vhodným pomenovaním množiny je zrejmé, čo obsahuje. Trénovacie dáta sú tvorené číslami v plávajúcej desatinnej čiarke.

Krokovanie



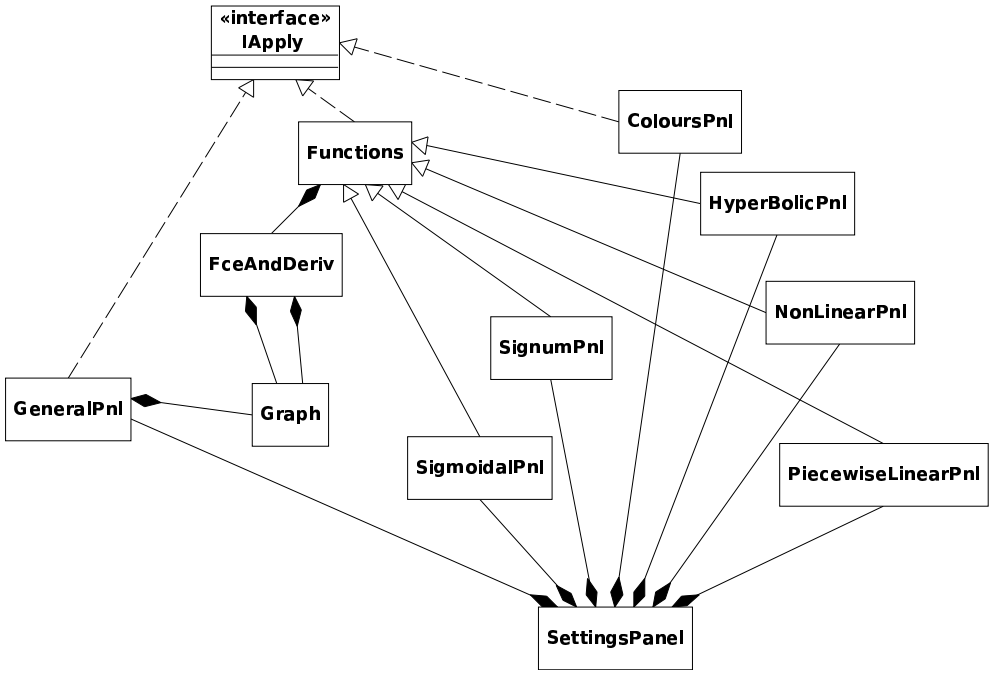
Obrázek 5.5: Diagram znázorňujúci triedy a ich vzťahy pre krokovanie a spustenie výpočtov.

Aplikácia v simulačnom móde umožňuje krokovať učenie alebo krokovať výpočet siete. Ďalej je tu možnosť zrýchliť krokovanie učenia predbehnutím jednej iterácie. Toto predbehnutie sa vypočíta na pozadí a nijako sa neanimuje. Táto možnosť bola zahrnutá z dôvodu, že učenie siete je v mnohých prípadoch veľmi časovou náročnou operáciou, a tak pri krokovaní jednotlivých úkonov by sa požadovaný výsledok nedosiahol v prijateľnom čase. Krokovanie jednotlivých úkonov učenia, krokovanie výpočtu iterácií a aj krokovanie výpočtu siete je možné spustiť ako animáciu.

Učenie alebo výpočet siete je taktiež možné spustiť na pozadí. Tento spôsob učenia, respektíve výpočtu siete je najrýchlejší pretože nie je nutné si pamätať jednotlivé vykonané kroky.

Nastavenia

V nastaveniach sú taktiež aktivačné funkcie, kde je sprístupnená zmena jednotlivých parametrov funkcie. Aby bolo jasne vidieť ako sa funkcia zmení po prestavení parametrov, je priebeh funkcie a jej derivácia vykreslená na grafe. Veľkosť prijateľnej chyby, koeficient učenia, úbytok koeficientu učenia a ďalšie parametre ako aj rýchlosť animácie je možné meniť v nastaveniach. Ďalšou drobnosťou je zmena farieb jednotlivých položiek na pracovnej ploche.



Obrázek 5.6: Triedy prezentujúce panel nastavení.

Simulácia

Samotná simulácia algoritmu učenia je graficky znázorňovaná. Jednotlivé časti algoritmu sú vykonané a znázornené na pracovnej ploche, pričom informácie o aktuálnom kroku sú užívateľovi vypisované. Ako napríklad rovnica, ktorá bola práva spočítaná, aké hodnoty boli do rovnice dosadené a výsledok rovnice. Po prejdení jedného vzorku sa vypíše aktuálna chyba. Po aplikovaní celej trénovacej množiny sa vypočíta globálna chyba, ktorej priebeh v čase sa zobrazuje na grafe, aby bolo vidieť ako sa mení stav siete.

Parametre

Naučené parametre siete (hodnoty jednotlivých váhových prepojení) je taktiež možné exportovať do XML súboru. Aby pre jednu danú sieť bolo možné definovať viacero parametrov, závisiac na tom čo predstavujú, sieť a parametre sú ukladané zvlášť. Majme napríklad sieť, ktorá obsahuje dva vstupy a jeden neurón vo výstupnej vrstve. Túto sieť môžeme naučiť, aby prezentovali funkčnosť jednotlivých hradiel (AND, OR, XOR). Pre každé hradlo budú hodnoty parametrov rôzne, takže pre jednu konkrétnu sieť máme viacero nastavení. Ukladaním zvlášť do súborov sa vyhneme redundancii dát, ktorá by vznikla viacnásobným uložením siete. Aby potom bolo možné načítať parametre pre tú sieť, pre ktorú boli vytvorené, vypočíta sa heš konkrétnej siete. Heš sa počíta z umiestnených neurónov v konkrétnej vrstve a z prepojení medzi uzlami s rešpektom na poradie siete.

Kapitola 6

Implementácia

Štruktúra aplikácie sa skladá z troch častí:

1. knižnica pre neurónové siete
2. balíček pre grafické užívateľské rozhranie (*GUI*)
3. simulátor, ktorý okrem riadenia simulácie obsluhuje GUI a využíva funkcie knižnice

Pri implementácii knižnice a aplikácie bol často použitý návrhový vzor zvaný unikát (*singleton*). Trieda, ktorá využíva tento model, obsahuje inštanciu sama na seba. Konštruktor triedy je privátny, takže nie je možné vytvoriť inštanciu triedy pomocou obvyklého spôsobu. Tento vzor definuje statickú metódu, ktorá vracia inštanciu danej triedy. Metóda pri prvom zavolaní vytvorí inštanciu, uloží si jej odkaz do statickej premennej a pri ďalšom zavolaní sa vracia už len inštancia danej triedy. Toto umožňuje sprístupnenie triedy v programe na ktoromkoľvek mieste, bez potreby prenášať referenciu na túto triedu. Taktiež je zaručené, že trieda nebude vytvorená viac ako jeden krát.

6.1 Knižnica

6.1.1 Neurónová sieť

Celá knižnica je zabalená v triede `NeuralNetwrok`. Po vytvorení inštancie tejto triedy je možné s knižnicou pracovať. V hlavnej triede sú vlastné a delegované metódy z iných tried. Konkrétne sú to `Connection`, ktorá definuje prepojenia celej siete a trieda `NeuralSchema`, v ktorej sú definované všetky vrstvy, neuróny a vstupy.

6.1.2 Vrstvy

Skrytá vrstva `HiddenLayer` a vstupná vrstva `InputLayer` dedia od abstraktnej triedy `Layer`. Jedna vstupná vrstva a množina skrytých vrstiev sú súčasťou triedy `NeuralSchema`. Skryté vrstvy obsahujú neuróny a vstupy sú uložené vo vstupnej vrstve. Za výstupnú vrstvu sa považuje posledná skrytá vrstva. Je to kvôli tomu, že tieto vrstvy majú spoločné rysy. Vrstvy je možné pridávať na koniec zoznamu alebo na ľubovoľnú pozíciu. Prácu s vrstvami poskytuje trieda `NeuralSchema`.

6.1.3 Vstupy a neuróny

Trieda `InputData` predstavuje vstup a `Neuron` je trieda, ktorá zabaľuje funkcie neurónu. Obe tieto triedy sú potomkovia abstraktnej triedy `DataCell`. Táto trieda definuje aké metódy by mala bunka obsahovať.

Bunka obsahuje informáciu o svojom aktuálnom výstupe, prípadne pri použití rekurentných sietí je možné ukladať všetky výstupy od času t_0 až po čas t, kde t_0 je počiatočný čas a t je aktuálny čas. Vstup neobsahuje žiadne ďalšie informácie. Neurón obsahuje odkaz na triedu, ktorá implementuje rozhranie `ActivationFce`. V knižnici bolo zahrnutých päť aktivačných funkcií. Tieto triedy implementujú rozhranie `ActivationFce` a boli umiestnené ako statické triedy v tomto rozhraní.

6.1.4 Prepojenie siete

Všetky prepojenia siete ukladá, modifikuje a maže trieda `Connection`. Prepojenia sú definované na základe identifikátorov jednotlivých neurónov, respektíve vstupov. Prepojenia sú uložené v hešovacej tabuľke. Kľúčom je uzol id_{key}, z ktorého spojenie vychádza a hodnota je zoznam všetkých $id_0, id_1, \cdots id_n$ uzlov, do ktorých uzol d_{key} vstupuje. Táto informácie je ukladaná aj opačným spôsobom v druhej tabuľke, kvôli rýchlemu vyhľadaniu všetkých prepojení daného neurónu. Trieda `Connection` poskytuje prácu pre manipuláciu s týmito tabuľkami. Hodnoty váh prepojení sú ukladané v tabuľke, kde kľúčom do tabuľky je *id* prepojenia.

6.2 Učenie

Abstraktná trieda `Runner` predstavuje rozhranie pre algoritmy učenia siete. Obsahuje metódy na krokovanie výpočtu a na okamžitý výpočet. Taktiež obsahuje koeficient učenia η, maximálnu povolenú chybu výpočtu a maximálny počet iterácií. Pri krokovaní vracia metóda inštanciu triedy `InfoData`, ktorá obsahuje informácie o prevedenom kroku. Obsahuje názov správy, detail správy a akciu. Na základe akcie je okrem iného možné zistiť, či sa už výpočet úspešne skončil, alebo naopak bol prekročený počet iterácií.

Algoritmus *backpropagation* predstavuje trieda `BackPropagation`, ktorá dedí z triedy `Runner` a je teda použitá pre učenie vytvorenej siete. Krokovanie algoritmu bolo definované pomocou štyroch konečných automatov. Hlavnou úlohou prvého automatu `FAGeneral` je volanie ostaných troch automatov. Jeden má na starosť krokovanie výpočtu siete, druhý počíta spätným šírením delty pre každý neurón. Posledný konečný automat slúži na aktualizáciu váhových prepojení siete, na základe vypočítaných delt. Algoritmus bol rozdelený do štyroch automatov kvôli prehľadnosti a jednoznačnej funkcii každého z nich. Typ algoritmu *backpropagation* sa určí pri vytváraní triedy. Tento typ určuje, či bude trieda pracovať ako *vanilla*, *enhanced*, *weight decay* alebo *batch backpropagation*, ktoré boli popísané v kapitole 4.

Algoritmus pre učenie rekurentných neurónových sietí nebol implementovaný. Tieto siete umožňujú simulovať len výpočet siete.

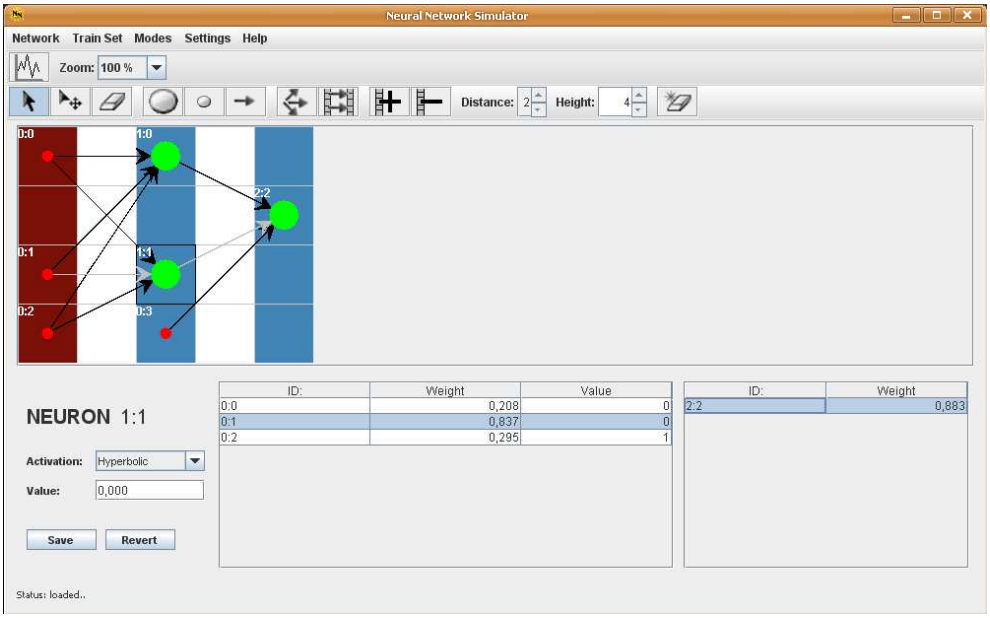
6.2.1 Trénovacie množiny

Trénovaciu množinu predstavuje trieda `TrainSet`. V tejto triede sú ďalej ukladané jednotlivé trénovacie vzorky, ktoré predstavuje trieda `TrainPattern`. `TrainSet` poskytuje funkčnosť, ktorá umožní prácu so vzorkami. Vzorky je možné pridávať a uberať. Sú dve

možnosti výberu vzoriek z množiny. Prvou je na základe indexu uloženého vzorku. Druhou možnosťou je náhodný výber, ktorý sa musí inicializovať pred prvým požadavkom na trénovaciu vzorku. Tieto vzory sú používané ako parametre v metódach pre učenie siete, podľa nich sa určuje želaný výstup.

6.3 Grafické užívateľské rozhranie

Druhým balíkom je skupina grafických prvkov, ktoré tvoria vizuálnu časť aplikácie. Hlavné okno je definované v `MainWindow`. Toto okno obsahuje menu, panel nástrojov, hlavný panel obsahujúci pracovnú plochu a panel zobrazujúci informácie o zvolenej bunke a jej prepojeniach.



Obrázek 6.1: Hlavné okno aplikácie v editačnom móde.

6.3.1 Pracovná plocha

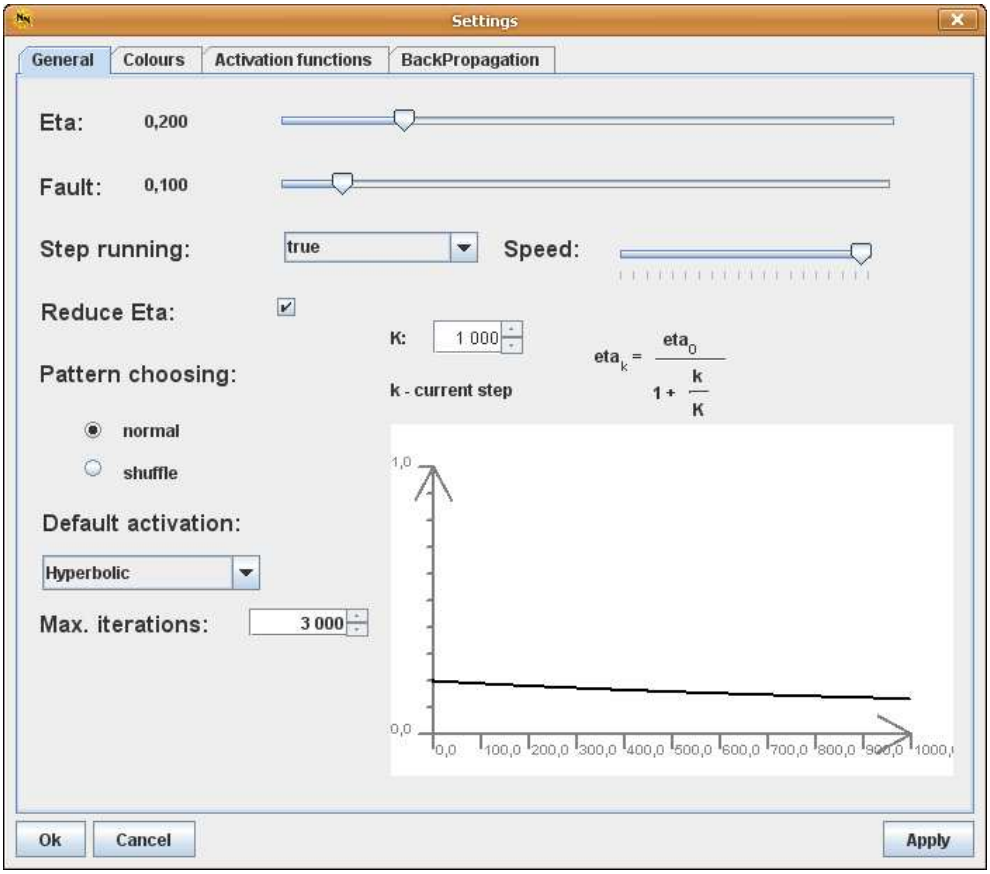
Najdôležitejšou súčasťou grafického rozhrania je pracovná plocha, kde sa vykresľuje a edituje celá neurónová sieť. Plocha je definovaná v `Scene`. Trieda `Scene` predstavuje komplexnú funkčnosť. V nej sú uložené pozície jednotlivých neurónov a vstupov. V tejto triede nie sú uložené prepojenia, využíva sa vyššie spomenutá trieda `Connection`, ktorá spojenia definuje.

`Scene` obsahuje množstvo funkcií na zmenu veľkosti plochy, na prekreslenie plochy, pridávanie a mazanie neurónov, respektíve vstupov. Celá scéna je najprv vykreslená do odkladacieho priestoru zvaným *buffer*. Je to hlavne kvôli tomu, aby sa pri každom prekreslení plochy nemusela znova načítavať celá sieť. Pri vykreselní do *buffera* sa zisťujú pozície uzlov a ich prepojení a na základe toho sú vykreslené. Tento výpočet a prechod všetkými uzlami je pri veľkých sieťach náročný. Ak je celá sieť uložená v *bufferi*, pri vykreslení sa jednoducho buffer vykreslí a nie je potrebné nič počítať. *Buffer* je nutné prekresliť v prípade ak je pri-

daný alebo zmazaný uzol, čím sa zmenila podoba siete a jej podoba v bufferi nezodpovedá konkrétnemu stavu.

Pri presúvaní uzlov na ploche sa zistí pozícia uzlu ktorý sa bude presúvať. *Buffer* sa musí prekresliť a to tak, že sa vykreslí celá sieť, okrem presúvaného uzlu a jeho prepojení. Presúvaný uzol a jeho prepojenia k ostatným neurónov sú prekreslované cez *buffer* na ploche, pri hýbaní kurzorom myši. Po vložení uzlu na konkrétne miesto sa terajší stav aktualizuje a *buffer* je opäť modifikovaný.

6.3.2 Nastavenia



Obrázek 6.2: Okno pre rôzne nastavenia aplikácie.

Okno s nastaveniami obsahuje viacero záložiek. V hlavnej záložke, ktorá sa zobrazí po zobrazení okna obsahuje hlavné nastavenia aplikácie. V ďalších záložkách sa nastavujú farby prvkov a parametre aktivačných funkcií. Každý panel, ktorý sa nachádza v nastaveniach implementuje rozhranie `IApply`. Toto rozhranie definuje dve metódy, ktoré sú volané pri aplikovaní prevedených zmien, alebo naopak pre zrušenie týchto zmien.

6.3.3 Panel nástrojov

Na paneli nástrojov sa nachádzajú rôzne komponenty. V editačnom režime sa tu nachádzajú najmä nástroje pre vytvorenie a úpravu siete. Skupina tlačidiel pre kreslenie a editovanie sa

kliknutím aktivuje, a ďalej pri klikaní na plochu sa vykonáva zvolená akcia. Nachádzajú sa tu aj dve komponenty, ktorým je možné nastaviť určitú celočíselnú hodnotu. Prvá komponenta slúži na nastavenie vzdialenosti medzi jednotlivými vrstvami. Druhá definuje výšku pracovnej plochy. Šírka plochy je daná počtom vrstiev a vzdialenosťou medzi nimi.

Panel nástrojov obsahuje taktiež komponentu, ktorou je možné približovať alebo oddiaľovať pracovnú plochu. Implicitné zobrazenie je 100 %. Plochu je možné priblížiť až na 200 %, alebo oddialiť na 10 %. Keďže je celá plocha vykresľovaná do `bufferu`, nie je vhodné príliš veľké siete maximálne približovať. Explicitne je možné spustiť program s parametrami pre navŕšenie pamäte. Viac je popísané v sekcii 6.5.

Pri simulačnom režime sa nachádzajú nové tlačidla na paneli nástrojov. Tieto slúžia na spustenie, zastavenie alebo krokovanie simulácie.

6.3.4 Informačné okno

Pri prepnutí programu do simulačného režimu sa zobrazí vpravo na obrazovke nové okno. Po zavretí okna, je ho možné znova otvoriť príslušným tlačidlom na paneli nástrojov. Na tomto okne sa zobrazujú informácie pri simulácii.

Navrchu sa nachádzajú dve komponenty zobrazujúce globálnu a aktuálnu chybu. Pod nimi je zobrazená aktuálna trénovacia vzorka, ktorá je priložená na vstupnom vektore siete. Nižšie sa vypisuje práve vykonávaná akcia. Pod ňou v textovej komponente je vypísaný detail tejto akcie. Na spodku okna sa v grafe vykresľuje globálna chyba od času 0 po doterajší čas. Toto umožňuje špeciálna komponenta `Graph`, popísaná nižšie v 6.3.6. Ak graf zaplňuje celú plochu, jednotky grafu sú zväčšené, graf sa zmenší. Graf sa ďalej posúva v čase až pokiaľ nenarazí opäť na koniec, kedy je graf opäť zmenšený.

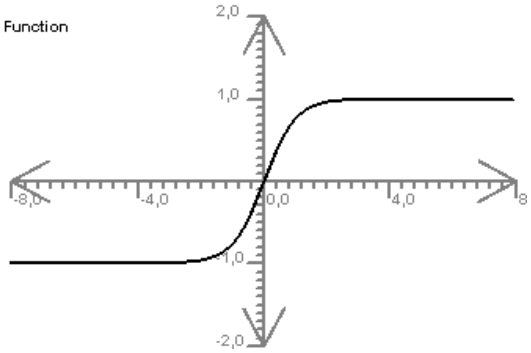
6.3.5 Okno pre definovanie trénovacích dát

Trieda `TrainSetDialog` definuje rozmiestnenie a logiku zadávania trénovacích dát do programu. Je možnosť vytvorenia alebo editácie viacerých množín obsahujúcich trénovacie vzorky. Dáta sú vkladané do tabuľky, ktorá farebne odlišuje stĺpce, ktoré určujú vstupy a stĺpce definujúce želané výstupy. Po definícii alebo úprave sú dáta pripravené k použitiu.

6.3.6 Užitočné nástroje

Pre grafickú reprezentáciu neurónov, vstupov, prepojení, vrstiev a mriežky na pracovnej ploche bola vytvorená trieda `GraphicElements`. Pomocou grafických primitív sú vykresľované jednotlivé prvky. Trieda je využívaná hlavne triedou `Scene`.

Uplatnenie má aj v ďalšom užitočnom nástroji `Graph`. `Graph` je trieda, ktorá reprezentuje dvojdimenzionálny (2D) graf. Podľa konštruktoru je určené či bude graf vykresľovať len prvý kvadrant alebo sa vykreslia všetky štyri kvadranty. Komponenta umožňuje pridávať jednotlivé body grafu pomocou čísiel s plávajúcou desatinnou čiarkou. V Jave je súradnica $[0 : 0]$ v ľavom hornom rohu komponenty pričom hodnota y rastie smerom dole a x smerom doprava. Aby bolo možné graf správne vykresliť, súradnice sa musia prepočítavať vzhľadom na zobrazované súradnice. Každý bod je ukladaný s pôvodnými súradnicami, pri vykresľovaní grafu sa bod prepočíta na súradnice komponenty. Pri zmene veľkosti komponenty `Graph` sa zmení jej súradný systém, takže pri prepočítaní sú body zobrazené správne. Nástroj umožňuje vykresľovať graf ako jednotlivé body, alebo sú tieto body spojené čiarou. `Graph` umožňuje nastavovať počet zobrazovaných jednotiek súradníc zvlášť pre x a zvlášť pre y. Taktiež sa dá nastaviť koľko jednotiek zobrazuje jedno políčko grafu.



Obrázek 6.3: Komponenta graf, ktorá umožňuje vykreslenie 2D grafu v jednom alebo všetkých štyroch kvadrantoch. Na obrázku je príklad vykreslenia hyperbolického tangensu.

Tento nástroj je použitý v nastaveniach aplikácie, kde zobrazuje priebehy aktivačných funkcií a pri procese učenia sa na nej animuje priebeh globálnej chyby.

Trieda `Colours` obsahuje všetky nastaviteľné farby aplikácie. Týka sa to hlavne prvkov pracovnej plochy. Implicitné farby je možné v nastaveniach meniť.

Boli vytvorené aj ďalšie triedy, ktoré majú charakter nástroja. Príkladom je trieda `Picture`, ktorá načítava obrázky aplikácie, ďalej triedy pre zjednodušenú prácu s dialógmi a dialógmi pre ukladanie a načítavanie súborov z pevného disku.

6.4 Simulátor

Poslednou časťou programu je samotný simulátor. Väčšina udalostí grafického užívateľského rozhrania je odchytávaných a spracovávaných priamo v tomto balíčku. Niektoré konkrétne udalosti sú spracovávané priamo v triede, ktorá definuje aj vzhľad komponenty.

Simulátor má na starosť spúšťanie vlákien na vykonávanie simulácií. Je hlavným vstupným bodom celého programu. Definuje triedu ktorá odchytáva nezachytené výnimky programu.

6.4.1 Udalosti

Udalosti sú spracovávané pomocou viacerých tried. Každá grafická časť má svoju protiľahlú triedu, ktorá sa stará o udalosti. Napríklad trieda `ToolPanel` predstavuje grafickú podobu panelu nástrojov a trieda `ToolControl` odchytáva udalosti kliknutí tlačidiel a zmenu stavu komponenty, ktorá predstavuje trénovacie prvky.

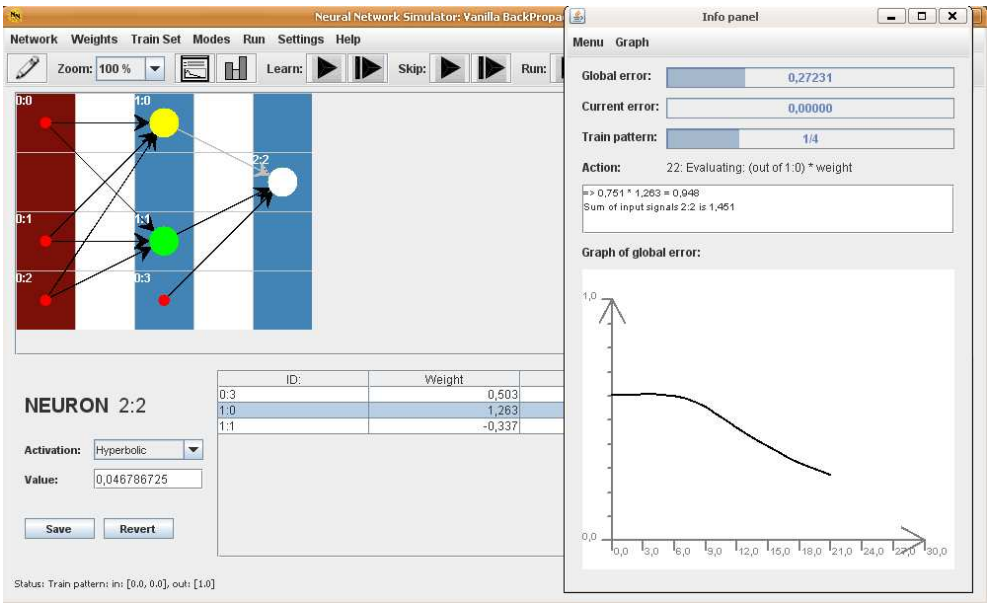
Trieda `MouseControl` spracováva udalosti pri kliknutí kurzora myši na pracovnú plochu. Zisťuje aké tlačidlo na paneli nástrojov je momentálne stlačené a podľa toho je vykonaná akcia. Po kliknutí sa musia prerátať súradnice s rešpektom na priblíženie alebo oddialenie plochy. Touto triedou je realizované kreslenie a editovanie siete. Pri pridávaní, odoberaní a o všetky ďalšie úkony sú oznamované triede `Scene`, ktorá ich patrične spracuje. `Scene` sa stará o grafickú podobu siete, pričom v `MouseControl` sa modifikuje podoba skutočnej siete.

Ďalšie triedy sa starajú o udalosti menu a spodného panelu zobrazujúceho informácie a prepojenia uzlu. Niektoré triedy odchytávajúce udalosti zdieľajú svoju funkčnosť pre viaceré grafické triedy. Je to v prípade, že tlačidlo menu má rovnakú funkčnosť ako tlačidlo

na paneli nástrojov. Potom udalosť pre obe je rovnaká a spracovaná jednou triedou.

6.4.2 Simulácia

V balíčku simulátora sa nachádzajú aj triedy, ktoré súvisia priamo s behom simulácie. Trieda `Control` odchytáva udalosti pri spúšťaní alebo zastavovaní simulácie. Taktiež je možnosť resetovania simulácie. Trieda `NetRunner` a `Learn` predstavujú vlákna, ktoré spočítajú sieť, respektíve učia sieť okamžite. Vytvorí sa inštancia triedy, následne je vlákno spustené, aby bola možná ďalšia komunikácia s aplikáciou.



Obrázek 6.4: Okno aplikácie v simulačnom režime. Na popredí je okno zobrazujúce informácie o simulácii.

Trieda `SynThread` predstavuje vlákno, ktoré je použité v ďalších triedach. Táto trieda umožňuje spustenie vlákna, ktoré je spustené pri animácii a pri zastavení je zastavené. Pri znovu spustení je vytvorené ďalšie vlákno. Obsahuje metódy na spustenie a zastavenie vlákna, pri behu je volaná metóda z triedy, ktorá dedí abstraktnú triedu `Player`. Táto trieda definuje rozhranie pre spustenie vlákna `SynThread` pri spustení animácie, pri zastavení animácie oznámi vláknu, že sa má v ďalšom cykle zastaviť a vlákno je odstránené. Triedu `Player` rozširujú triedy `StepLearn` a `StepRun`. Prvá krokuje výpočet učenia a druhá výpočet siete. Implementujú dané metódy, ktoré definuje otcovská trieda `Player`. Potom je možné algoritmy krokovať pomocou konečných automatov implementovaných v knižnici.

6.4.3 XML

Ukladanie a načítavania z XML majú na starosť viaceré triedy. Práca s váhami, sieťou a trénovacími množinami je každá zvlášť v inej triede. Každá obsahuje dve metódy pre uloženie, respektíve načítanie dát z a do súboru. XML je technológia na štruktualizáciu dát v dokumente, slúži na zjednodušenie práce s čítaním a zápisom dát. Je platformovo nezávislé, preto som sa rozhodol túto technológiu použiť. Knižnica `dom4j` zabaľuje jednoduchú prácu s XML. Je to verejne dostupná knižnica.

6.4.4 Ďalšie triedy

Užitočnou triedou je `TrainHolder`, ktorá obsahuje všetky definované alebo načítané trénovacie množiny. Na základe počtu vstupov a počte výstupov sa určí "dimenzia" siete. Podľa tejto dimenzie sú množiny ukladané do hešovacej tabuľky. Umožňuje získanie aktuálne zvolenej trénovacej množiny, takisto pridávanie nových. Ďalšou triedou v tomto balíčku je `Refactor`. Táto trieda má za úlohu minimalizovať a upraviť identifikátory siete, tak aby boli vzostupne smerom od vstupnej k výstupnej vrstve. Minimalizácie spočíva v odstránení nadbytočným vrstiev a veľkosti pracovnej plochy.

6.5 Odporúčania a obmedzenia programu

Program bol testovaný pod operačným systémom Windows XP od firmy Microsoft a taktiež pod unixovým systémom Ubuntu 8. Keďže je aplikácia napísaná v Jave, nie je závislá na platforme, ale určité rozdielnosti sú. Napríklad bolo treba zmeniť veľkosti určitých komponent, pretože pod unixom boli niektoré komponenty o niečo väčšie.

Program je vhodné spúšťať s parametrami -XmsNN -XmxNN, kde NN je hodnota. Táto hodnota určuje veľkosť priradenej pamäte programu. Toto je vhodné pri simulácii veľkých sietí. Ak program využije všetky svoje zdroje, vedie to k chybe, pretože program nemá dostatok pamäte pre ďalší beh. V tomto prípade je zobrazené chybové hlásenie a program sa pokúsi terajšie hodnoty a sieť uložiť do XML.

6.6 Návod na použitie programu

Návod je možné nájsť priamo v aplikácii. Nachádza sa v menu **Help** > **Help**. Návod aj program je písaný v anglickom jazyku. Najprv sú predstavené všetky okná použité v aplikácii a ich jednotlivé časti. Popis je doplnený obrázkami, ktoré boli vytovrené pri písaní tohto návodu. Demonštračný príklad je zahrnutý ako posledná kapitola. Príklad demonštruje použitie editačného a simulačného režimu. Najskôr je podrobne popísané a znázornené vytváranie neurónovej siete, jej editácia a nakoniec práca s ňou.

Kapitola 7

Záver

Počas práce na tomto projekte som sa zoznámil s novým odvetvím umelej inteligencie. S neurónovými sieťami som sa predtým nestretol. Preto bolo nutné si zohnať a naštudovať rôzne materiály. Ako som zistil neurónové siete sú ešte stále skúmaním odvetvím. Avšak ich použitie v rôznych aplikáciách stále rastie. Neurónové siete tvoria istú časť v aplikáciach, netvoria teda celý program.

V mojej práci som sa snažil zhrnúť naštudované poznatky, navrhnúť aplikáciu a následne ju implementovať. Snažil som sa vytvoriť program, ktorý by umožňoval jednoduchú interaktívnu prácu s neurónovými sieťami. Dbal som na zrozumiteľné vypisovanie a znázorňovanie informácií pri simulácii.

Pôvodne mojím plánom bolo zahrnúť aj učenie rekurentných neurónových sietí. Toto sa mi bohužiaľ nepodarilo najmä kvôli ich zložitosti a časovým podmienkam. Preto je toto jedna z možností na rozšírenie aplikácie. Aplikáciu by bolo vhodné rozšíriť tak, aby sa nové algoritmy pre učenie sietí mohli pridávať formou zásuvných modulov (*pluginov*). Každý takýto modul by obsahoval triedu, ktorá by dedila z triedy `Runner`. Druhou triedou by bol panel, ktorý by sa pridal do nastavení. Tento panel by umožňoval nastaviť parametre algoritmu.

Ďalším rozšírením alebo zmenou by bolo zadávanie trénovacích vzoriek. Vzorky by sa nezadávali ako číselná informácia, ale v podobe obrázku. Od počtu vstupných jednotiek by sa určilo rozlíšenie obrázku, užívateľ by mohol nakresliť napríklad jednotlivé číslice, ktoré by sa binárne zakódovali. Týmto by sa dalo implementovať rozpoznávanie ručne písaných znakov.

Práca na bakalárskom projekte mala pre mňa veľký význam, pretože som bol nútený naučiť sa niečo nové a pokročiť s dobou. Táto téma ma zaujala a chcem sa ďalej zaoberať týmito sieťami, vyskúšať naprogramovať nejaké aplikácie, ktoré budú priamo využívať neurónové siete. Veľkým prínosom neboli len teoretické znalosti, ale aj zlepšenie sa v programovacom jazyku Java.

Literatura

[1] Arbib, M. A.: *The Handbook of Brain Theory and Neural Networks*. USA: The Mit Press, druhé vydání, ©2003, ISBN 0-262-01197-2.

[2] Jiří Šíma, R. N.: *Teoretické otázky neuronových sítí*. Praha: Matfyzpress, 1996, ISBN 80-85863-18-9.

[3] Karray, F. O.; de Silva, C.: *Soft computing and Intelligent Systems Design*. England: Pearson Education Limited, ©2004, ISBN 0-321-11617-8.

[4] Kishan Mehrota, S. R., Chilukuri K. Mohan: *Elements of artifictial neural networks*. USA: Masachusetts Institute of Technology, ©1997, ISBN 0-262-13328-8.

[5] Matthews, J.: *An Introduction to Neural Networks*. [online], [citované 20.4.2008]. URL http://www.generation5.org/content/2000/nnintro.asp

[6] R. A. Aliev, R. R. A.: *Soft computing and its applications*. Singapore: World Scientific Publishing Co. Pte Ltd., ©2001, ISBN 981-02-4700-1.

[7] Sarle, W. S.: *Neural Networks: FAQ, Papers, and Various Other Things*. [online], [citované 25.4.2008].
URL ftp://ftp.sas.com/pub/neural/index.html

[8] Zbořil, F. V.: *Soft Computing*, 2007/08, prednášky na VUT FIT v Brně.
URL http://www.fit.vutbr.cz/