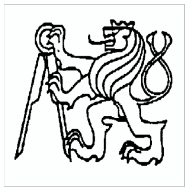


Středoškolská technika 2009

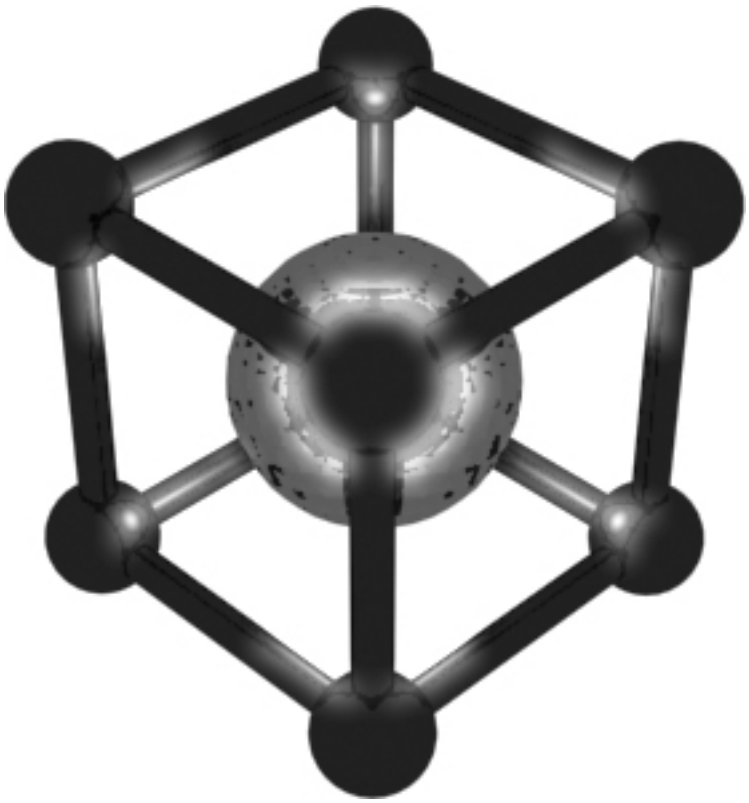
Setkání a prezentace prací středoškolských studentů na ČVUT

Neural Networks Studio

Jiří Hýbek

SPŠ a VOŠ Písek, 2. ročník

březen 2009



Prohlašuji tímto, že jsem práci vypracoval samostatně a uvedl v seznamu literatury veškerou použitou literaturu a další informační zdroje včetně internetu.

Vřelé díky za pomoc a spolupráci Mgr. Miroslavu Širokému.

V Písku dne 1. 5. 2009

Anotace

Neuronové sítě mají široké využití v mnoha oblastech, například v robotice či bezpečnostních systémech. Umožňují například rozpoznávat obrazy, činit predikce, či řešit logické situace. Cílem této práce je vytvořit nástroje pro efektivní a jednoduché navrhování neuronových sítí a jejich trénování. Aplikaci, kterou jsem napsal v jazyce C++, využívám k vývoji neuronových sítí a doufám, že bude mít v budoucnu širší využití nejen pro mě, ale i pro začátečníky, kteří se o tuto oblast zajímají.

Neural networks have wide usage in many areas – for example in robotics or security systems. It can recognize images, make a prediction and solve logic problems. The objective of this work was to create a tool, which provides simple and effective tool for designing and testing neural networks. I am using the application, which I wrote in C++ language, for developing neural networks and I hope it will have wide usage not only for me but for many beginners who are interested in this problematics.

Klíčová slova

Neuronové sítě – Neural networks
Umělá inteligence – Artificial intelligence
Rozpoznávání obrazů – Image recognition
Vývojové nástroje – Development tools
Zpětné šíření chyby – Backpropagation

Obsah

Úvod

Člověk je schopen řešit efektivně a rychle různorodé situace. Stejné schopnosti mají i jiní tvorové například mravenci. Proto si z živých tvorů při řešení problémů výpočetní technikou bereme příklad. Co nám umožňuje přemýšlet a řešit situace? Je to mozek. Mozek, který se skládá z neuronových sítí. Jedná se o propojení milionů malých buněk, které se nazývají neurony. Neuron samotný toho příliš nevyřeší. Ale pokud je součástí správně sestavené sítě, je schopen spolu s ostatními neurony řešit velice složité úlohy, a to naprosto rychle a efektivně. Nezanedbatelnou výhodou neuronu je schopnost učení. Pokud neuron ví, že zareagoval špatně, je schopen se velice rychle přizpůsobit tak, aby příště k chybě nedošlo.

Simulací neuronové sítě je umělá neuronová síť, která se chová podle daných matematických vzorců. Existují genetické algoritmy, které simulují chování neuronových sítí po několika vývojových generacích, simulují křížení genů a tzv. výběr nejsilnějších jedinců. Více o neuronových sítích v první části.

Aplikace Neural Network Studio psaná v jazyce C++, zkráceně jen NN Studio, slouží k návrhu jednoduchých neuronových sítí, jejich vypočítávání, trénování a analýze. Pomocí jednoduchého uživatelského prostředí lze síť téměř doslova nakreslit. Dále pomocí trénovacích sad správně nastavit. To vše doplňují například funkce vypočítávání v reálném čase, nástroj na rozpoznávání obrazů a konzolové utility.

Tato publikace je rozdělena na čtyři základní části. První část popisuje obecně umělé neuronové sítě, jejich algoritmy a využití. Druhá část pojednává o technickém řešení aplikace NN Studio a hlavní struktuře aplikace. Třetí část je věnována nástroji Recognition Wizard, který slouží k rozpoznávání obrazů.

Část I

Neuronové sítě

Kapitola 1

Popis neuronové sítě

Neurony jsou specializované nervové buňky, které zpracovávají elektrické signály. Reagují na vstupní signály a dále předávají výstupní signály ostatním neuronům. Jsou základním stavebním prvkem nervové sítě a umožňují organismům reagovat na podněty, na jejich základě řešit situace, které nastaly, a v neposlední řadě přemýšlet.

1.1 Popis biologického neuronu

Organický neuron je velmi malá buňka, která se skládá z těla, ve kterém je obsaženo jádro buňky a z krátkých a dlouhých výběžků. Krátké výběžky, nazývané dendrity, slouží k přenosu vstupních signálů. Jsou na ně připojeny ostatní neurony. Dlouhé výběžky, které se jmenují neurity neboli axony, předávají zpracovaný výstupní signál dál do sítě. Neuron má vždy pouze jeden tento výběžek, který se však může dále větvit. Tyto výběžky mohou zanikat a degenerovat, pokud nejsou využívány. Neurony mají schopnost se podle potřeby připojovat na ostatní buňky a naopak i odpojovat, což způsobuje například zapomínání nebo umožňuje hledaní nových řešení.

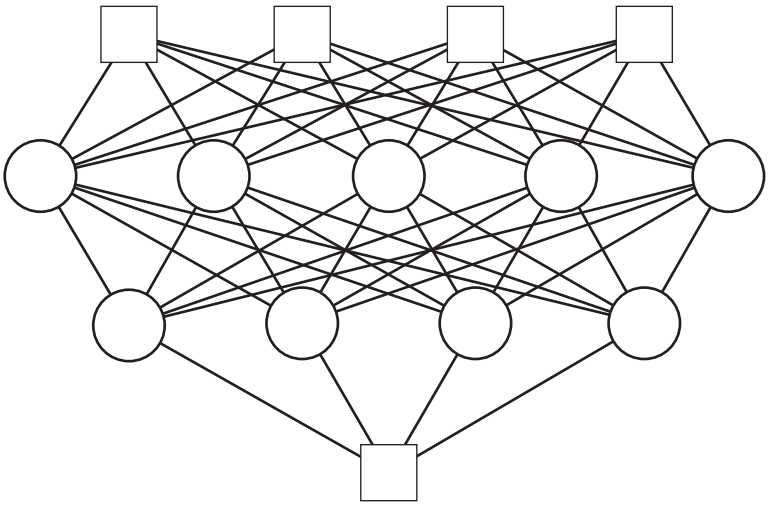
Součástí těla neuronu je tzv. neuronální membrána, která ovlivňuje velikost výstupního signálu. Chemické složení každého dendritu ovlivňuje tzv. váhu vstupního signálu. Pokud všechny vstupní signály jsou dostatečně silné, dojde k podráždění membrány, což způsobí silný výstupní puls.

1.2 Umělý neuron

Simulovaný neuron má stejnou strukturu, jako biologický. Skládá se ze vstupů, které mají svou váhu (**Weight**), z hodnoty prahu (**Threshold**), která slouží jako membrána, což při použití prahové výstupní funkce způsobí, že pokud je vstupní signál větší nebo roven této hodnotě, výstup bude pozitivní, a z genetické konstanty (**Bias**), se kterou je pracováno jako s dalším vstupem.

1.3 Umělá neuronová síť

Stavba umělé neuronové sítě ze skládá ze tří hlavních objektů. Ze vstupů, neuronů a výstupů. Vstupy obsahují prvotní informace, které celá neuronová síť zpracovává. Jsou připojeny k neuronům, které jsou posléze připojeny buď k dalším neuronům nebo výstupům. Výstup může být připojen pouze k jednomu neuronu a slouží k zobrazení výstupní hodnoty. Neurony zpravidla tvoří vrstvy, které postupně zpracovávají vstupní signál (viz obr. č. 1). Mohou být zapojeny i do tzv. zpětné vazby, což se využívá například při předpovídání vývoje nebo rozpoznávání obrazů či zvuku. Správně nastavený neuron může fungovat například jako logické funkce AND, OR nebo NOT.



Obr. č. 1 - Čtyřvrstvá neuronová síť

Kapitola 2

Výstupní funkce neuronů

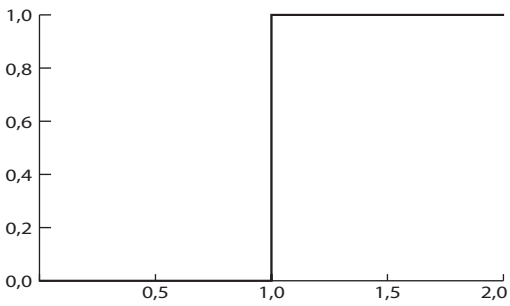
Při výpočtu výstupní hodnoty neuronu se nejprve provede sumarizace vstupů, což je suma součinů všech vstupních hodnot s jejich vahami. Tato suma je pak dále zpracována výstupní (filtrační) funkcí, která ovlivní celkový výstup neuronu. Nejčastěji je používána prahová funkce (**Threshold function**) nebo sigmoida (**Sigmoid function**).

2.1 Prahová funkce

Výstupem této funkce může být logická jednička nebo logická nula. V případě, že sumarizace vstupů je větší nebo rovna hodnotě prahu (**Threshold**), výstup bude pozitivní nebo v opačném případě negativní (viz obr. č. 2).

Toto chování můžeme popsat následující nerovnicí, kde x je vstupní hodnota (**Input**), w je váha vstupu (**Weight**), T je hodnota prahu (**Threshold**) a y je výstupní hodnota (**Output**):

$$\sum x_j \cdot w_{ji} - T \geq 0 \rightarrow y = 1 \tag{2.1}$$



Obr. č. 2 - Prahová funkce (Threshold function)

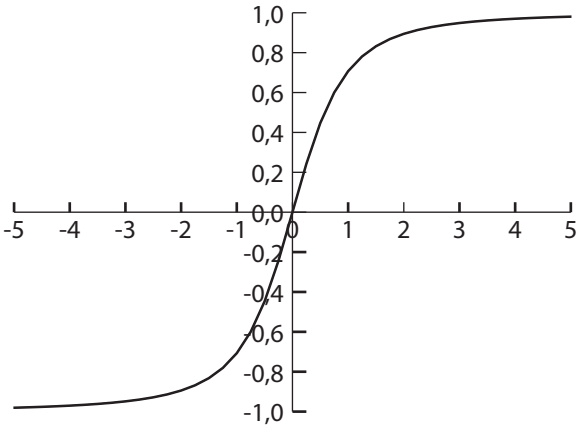
2.2 Sigmoida

Jedná se o logistickou funkci, která nejprve roste přibližně exponenciálně a později s rostoucím nasycením zpomaluje až se asymptoticky zastaví. Tato logistická funkce se často používá v empirických vědách například pro modelování růstu populace. Parametrem s je sumarizace vstupů neuronu.

Výstupní hodnota této funkce je $y \in R \in (-1, 1)$ a lze ji znázornit grafem (viz obr. č. 3) a popsat následujícími rovnicemi, kde x je vstupní hodnota (**Input**) a w je váha vstupu (**Weight**):

$$s = \sum x_j \cdot w_{ji} \tag{2.2}$$

$$o(s) = \frac{s}{\sqrt{1 + s^2}} \tag{2.3}$$



Obr. č. 3 - Sigmoida (Sigmoid function)

Kapitola 3

Učení neuronů

Učení neuronu spočívá v úpravě vah tak, aby příští výstup byl správný. Abychom mohli správně upravit váhy vstupů, je třeba znát, zda došlo k chybě a k jak velké. Výpočet chybové hodnoty (**Error**) je závislý na použité metodě učení.

3.1 Samostatný neuron

Pro samostatný neuron je chybovou hodnotou E rozdíl hodnoty, která by měla být na výstupu, a hodnoty, která je na výstupu. Velikost chyby pro samostatný neuron vypočítáme pomocí rovnice, kde o je hodnota, která by měla být na výstupu (**True output**) a O je výstupní hodnota neuronu (**Output**):

$$E = (o - O) \tag{3.1}$$

Správnou hodnotu váhy vypočítáme úpravou hodnoty staré a to o součin velikosti chyby a vstupní hodnoty. Tuto úpravu popisuje následující rovnice, kde W je váha vstupu (**Weight**), α je konstanta ovlivňující rychlost a kvalitu učení, E je velikost chyby (**Error**) a x hodnota na vstupu (**Input**):

$$W_{ji} = W_{ji} + \alpha \cdot E \cdot x_j \tag{3.2}$$

3.2 Trénovací sady

Schopnost neuronu učit se je největší výhodou neuronových sítí. Trénování spočívá v tom, že vytvoříme sadu vzorových situací. K různým kombinacím vstupů přiřadíme požadované výstupy. Čím více vzorových příkladů tím lépe. Poté použijeme takto vytvořenou sadu k trénování neuronu (např. viz obr. č. 4). Ne vždy se to však podaří napoprvé. Občas je třeba provést korekci sady podle situace nebo zopakovat trénink. Rychlost učení je závislá na konstantě α. Pokud je konstanta příliš nízká, učení trvá déle, avšak pokud je naopak příliš vysoká, nemusí se vůbec podařit neuron správně vytrénovat. Existují samozřejmě situace a kombinace vstupů, na které nelze neuron zcela vytrénovat.

x3	x2	x1	x0	y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1

Obr. č. 4 - Trénovací sada (Training set)

3.3 Zpětné šíření chyby

Tento systém učení spočívá v určení chyby na výstupní vrstvě sítě a na jejím následném šíření zpět až do vstupní vrstvy. Hodnota chyby (**Error**) se vypočte z rovnice, kde o je hodnota, která by měla být na výstupu (**True output**), O je výstupní hodnota neuronu (**Output**), n je počet vrstev a l je číslo vrstvy (**Layer**):

$$E_l = (O - o) \cdot \frac{1}{n} \cdot l \cdot 100 \tag{3.3}$$

Váhy se následně upraví pomocí rovnice, kde W je váha vstupu (**Weight**), α je konstanta ovlivňující rychlost a kvalitu učení, E je velikost chyby (**Error**) a x hodnota na vstupu (**Input**):

$$W_{jil} = W_{jil} + \alpha \cdot E_l \cdot x_{jl} \tag{3.4}$$

Kapitola 4

Využití

4.1 Logické operace

Pomocí neuronových sítí můžeme provádět logické operace. Například, pokud nastavíme váhy tak, aby byly stejné a jejich součet byl roven hodnotě prahu, vytvoříme logický člen AND. Výstup bude pozitivní pouze v případě, že všechny vstupy jsou také pozitivní.

4.2 Rozpoznávání obrazů

Pokud pro každý pixel budeme mít jeden vstup, můžeme neuron vytrénovat tak, aby výstup byl pozitivní pouze v případě, že na vstupu je některá z trénovaných kombinací nebo jim podobná.

4.3 Predikce

Při použití vícevrstvé sítě a učení pomocí zpětného šíření chyby je možno předpovídat přibližný vývoj křivek. Síť se postupně učí z trénovacích sad, které jsou vytvořeny pro toto předpovídání. Využívá se tzv. oken. Jedná se o hodnoty v určitých časových úsecích vývoje, které se částečně překrývají tak, aby měla síť k dispozici i část předchozích dat.

4.4 Jiné využití

Neuronové sítě lze také využít například k řízení robotů, topných soustav nebo k analýze velkých databází či k řízení datového toku v datových sítích.

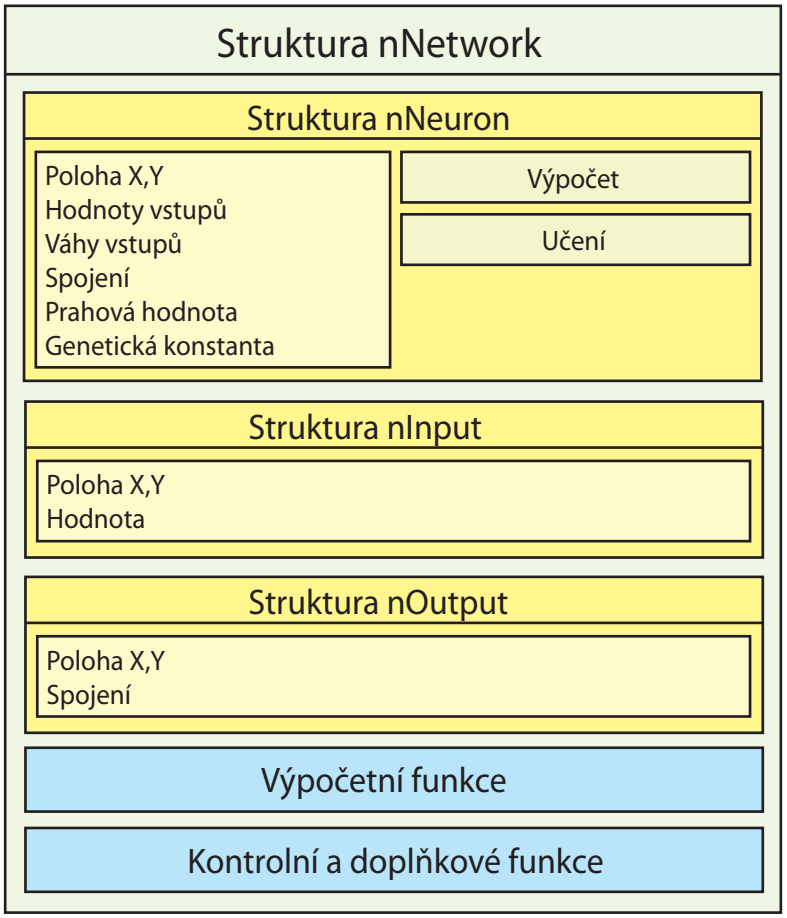
Část II

NN Studio

Kapitola 5

Struktura neuronové sítě

Hlavní strukturu aplikace popisuje obr. č. 5. Zdrojový kód – viz příloha č. 1.



Obr. č. 5 - Struktura nNetwork

Kapitola 6

Uživatelské prostředí

Aplikace má sloužit k jednoduchému a efektivnímu návrhu neuronových sítí. Z toho důvodu je třeba, aby bylo uživatelské prostředí co nejpřívětivější. Proto byla použita komponenta Ribbon a celá aplikace byla navržena v moderním stylu.

6.1 Ribbon

Komponenta Ribbon se skládá z hlavního tlačítka, panelu rychlého přístupu a z pásu karet. Autorem tohoto stylu je společnost Microsoft a poprvé jsme jej mohli vidět v sadě kancelářských aplikací Microsoft Office 2007.

Toto uspořádání nám umožňuje přehledný přístup ke všem důležitým funkcím. Pásy karet nám oddělují jednotlivé funkční sekce. Například v aplikaci NN Studio se jedná o oddělení funkcí pro návrhové zobrazení a pro práci s trénovací sadou. Každý pás karet dále obsahuje skupiny, které nám umožňují další oddělení funkcí a tím jejich zpřehlednění.

Všechny prvky komponenty Ribbon jsou připojeny ke komponentě ActionManager, která se stará o zpracování všech funkcí. Dále napojení na tuto komponentu umožňuje, aby si uživatel mohl přizpůsobit všechny akce svým potřebám.

Kromě jiného má tato komponenta moderní vzhled, který zaujme a dodá uživatelskému prostředí jistý přívětivý styl.

6.2 Návrhové zobrazení

Prostředí pro návrh neuronové sítě se skládá z mřížky, do které se vkládají objekty. Tato mřížka je vykreslována pomocí vlastní funkce, která je popsána dále.

V návrhovém zobrazení lze objekty vytvářet, označovat, přesouvat a mazat. A to pomocí nástrojů:

- Výběr
- Neuron
- Vstup
- Výstup
- Smazat

Dále je možno objekty spojovat a rozpojovat pomocí následujících nástrojů, které je třeba stručně popsat:

- Spojit (Vytvoří spojení mezi objekty.)
- Rozpojit (Smaže spojení mezi objekty.)
- Rozpojit neuron (Smaže **pouze** spojení, na kterých je **závislý** označený neuron.)
- Rozpojit vše (Rozpojí celou síť.)

Spoje objektů jsou vykreslovány barevným přechodem. Modrá barva značí pozitivní výstup z objektu a červená negativní.

Akce myši byly vytvořeny tak, aby bylo navrhování co nejjednodušší. Pravý klik vždy zruší označení objektu a změní nástroj na Výběr.

Při označení objektu se v horní liště objeví jeho název a vlastnost. U neuronů se jedná o hodnotu prahu, u vstupu o vstupní hodnotu a u výstupu o výstupní hodnotu, ta ovšem nelze měnit. Každou změnu je potřeba potvrdit stisknutím klávesy Enter. Pokud označíme spoj mezi dvěma objekty, zobrazí se název, který měnit nelze. Dále se zobrazí vlastnost spoje, což je jeho váha. Ukázka návrhového zobrazení – viz příloha č. 2.

Doplňkovou funkcí je možnost automatického vypočítávání neuronové sítě během návrhu. Tato funkce ovšem není vhodná pro velké sítě, jelikož může značně zpomalit celou aplikaci.

6.3 Trénovací zobrazení

Tento režim umožňuje vytvářet a upravovat trénovací sady pro neurony. Pracovat s nimi a v neposlední řadě spustit samotný trénink.

Skládá se z tabulky, ve které jsou ve sloupcích zobrazeny jednotlivé vstupy neuronu a na konci jeho požadovaný výstup. Řádky se podle potřeby přidávají automaticky. Pomocí této tabulky se vytváří a upravuje trénovací sada, s kterou pak lze provádět tyto operace:

- Uložit sadu
- Přiřadit sadu
- Spustit trénink

Včetně těchto funkcí jsou k dispozici v trénovacím zobrazení tyto akce:

- Vytvořit novou sadu
- Otevřít sadu
- Uložit sadu
- Uložit sadu jako...
- Přiřadit sadu
- Načíst sadu
- Spustit trénink

Pomocí akce Přiřadit sadu lze přiřadit danou sadu k vybranému neuronu. K provedení této akce je potřeba aktivní návrhový režim. Pokud má označený neuron přiřazenou sadu, lze pomocí akce Trénovat spustit trénink. Přiřazenou trénovací akci neuronu je možno otevřít kliknutím na tlačítko Načíst sadu z kategorie Neuron.

Ukázka trénovacího zobrazení – viz příloha č. 3.

Kapitola 7

Vykreslovací funkce

Návrhové zobrazení je vykreslováno vlastní funkcí. Tato funkce obsahuje vykreslování spojů pomocí částečně vyhlazených hran, vykreslování objektů a tzv. duchů (gridGhost).

7.1 Částečně vyhlazená čára

Základem je vykreslení čáry z jednoho bodu do druhého nejprve bez vyhlazení. Pozice pixelů jsou vypočítávány pomocí lineární závislosti vektorů. Výpočet popisují následující vztahy, přičemž X_s a Y_s jsou zdrojové souřadnice, X_d a Y_d jsou cílové souřadnice, X_i je pozice vykreslovaného pixelu na ose X a $Y_?$ je pozice vykreslovaného pixelu na ose Y, kterou je třeba vypočítat.

$A = [X_s; Y_s]$
$B = [X_d; Y_d]$
$C = [X_i, Y_?]$
$\overrightarrow{a} = \overrightarrow{AB} = (B_x - A_x; B_y - A_y)$
$\overrightarrow{b} = \overrightarrow{AC} = (C_x - A_x; C_y - A_y)$

$\overrightarrow{a} = k \cdot \overrightarrow{b}$
$a_x = k \cdot b_x$
$k = \frac{a_x}{b_x}$
$a_y = k \cdot b_y$
$a_y = \frac{a_x}{b_x} \cdot b_y$
$b_y = \frac{a_y}{\frac{a_x}{b_x}}$

Po vypočtení souřadnic je vykreslen ostrý pixel. Poté jsou okolo něj vykresleny pixely, které již mají vypočítanou barvu podle okolních bodů, takže ve výsledku vytvářejí efekt částečně vyhlazené čáry. Tu lze také vykreslit barevným přechodem s průhledností.

Výpočet barvy podle okolních pixelů

```
int calcRGB(TColor color, float *r, float *g, float *b, int q){
  long K;
  int R,G,B;
  K = ColorToRGB(color);
  R = K & 255;
  G = (K >> 8) & 255;
  B = (K >> 16);
  *r+=R/q;
  *g+=G/q;
  *b+=B/q;
  return 0;
}

long TNNetForm::calcColor(int x, int y){
float rgb[3]={0,0,0};
calcRGB(x,y,&rgb[0],&rgb[1],&rgb[2],2);     //Center
calcRGB(x-1,y,&rgb[0],&rgb[1],&rgb[2],8);   //Left
calcRGB(x+1,y,&rgb[0],&rgb[1],&rgb[2],8);   //Right
calcRGB(x,y-1,&rgb[0],&rgb[1],&rgb[2],8);   //Top
calcRGB(x,y+1,&rgb[0],&rgb[1],&rgb[2],8);   //Bottom
return RGB(rgb[0],rgb[1],rgb[2]);
}
```

7.2 gridGhost a gridLine

Tzv. duchové jsou ikony objektů vykreslované nad veškerou ostatní grafiku. Zobrazují se například při vytváření nebo přesunu objektů. Jedná se pouze o vykreslení obrazu daného objektu na aktuální pozici kurzoru.

GridLine je téměř to samé jako gridGhost, akorád s tím rozdílem, že se nejedná o ikonu objektu ale o čáru, která se zobrazuje při vytváření spoje.

Obojí je pouze dočasně zobrazovaná grafika, sloužící k usnadnění návrhu.

7.3 Mapa obrazu

Systém vykreslování postupuje celou mřížkou a vykresluje dané objekty a jejich spoje. Ovšem pokud by se vykreslil vždy spoj a poté ikona, překrývaly by jiné spoje a ikony. Z toho důvodu jsou nejprve vykresleny jen spoje a indexy ikon se zapisují do tzv. mapy obrazu. Poté, co algoritmus dokončí procházení pole, je vykreslena mapa obrazu, tj. objekty jsou vykresleny nad spoje.

7.4 Použití bufferu

Jelikož vykreslovat vždy veškerou grafiku, obzvláště částečně vyhlazené čáry, je výpočetně náročné, je používán buffer statické grafiky, což umožňuje při vykreslování duchů, nevykreslovat veškeré objekty a spoje, čímž se celý proces urychlí o více než 90 %. Celá grafika se znovu vykresluje jen v případě, že se změní struktura sítě.

Část III

Recognition Library

Kapitola 8

Úvod

Neuronové sítě mají široké využití, avšak v rozpoznávání vzorků mají výhodu v tom, že jsou rychlé, snadněji se nastavují a jsou výpočetně méně náročné. Toho lze využít například v zabezpečovacích systémech. Externí aplikace Recognition Wizard slouží k rozpoznávání obrazů pomocí neuronových sítí. Jejím základem je vytvoření sady vzorových obrazů s pozitivním a negativním výstupem, následné vytrénování neuronové sítě a nakonec samotné rozpoznávání.

8.1 Použití

Aplikaci Recognition Wizard lze použít k vytrénování knihovny vzorků. Výsledný soubor s vahami lze poté použít v kombinaci s utilitou RcgnProc například k rozpoznávání obrazu z webové kamery. Ukázka nástroje – viz příloha č. 4.

Kapitola 9

Struktura knihovny

Knihovna RecognitionLibrary se skládá z těchto hlavních částí: knihovna vzorků, neuronová síť a výpočetní funkce.

9.1 Neuronová síť

Neuronová síť se skládá ze vstupů, jejichž počet odpovídá počtu pixelů daného obrazu, z jednoho neuronu a jednoho výstupu. Síť se trénuje na základě vzorových příkladů. Tento trénink je většinou potřeba několikrát zopakovat, aby síť měla správně nastavené váhy vstupů. Neuronová síť aplikace Recognition Wizard pracuje s černobílými obrazy velikosti (200 × 150) px, což je 30 000 pixelů.

9.2 Knihovna vzorků

V této knihovně jsou uloženy váhy a zdrojová data všech vzorků, které jsou použity při porovnávání vstupního obrazu. Dále název vzorku, autor, datum pořízení a index filtru.

9.3 Trénování

Při trénování neuronové sítě a následné úpravě vah je možno použití dvě varianty. Částečný trénink, který provede úpravu vah pouze aktuálního vzorku a úplný trénink, který zároveň upraví váhy i všechny ostatních vzorků. Dále je možnost určit s kolika procenty vzorků bude proveden úplný trénink.

Kapitola 10

Úprava obrazu

Aby byl obraz co nejpoužitelnější, je třeba ho upravit. Při úpravě dochází k použití filtrů v následujícím pořadí:

1. Zmenšení obrazu
2. Kontrast
3. Negativ
4. Vytažení obrysů
5. Kontrast
6. Převedení na černobílý obraz
7. Negativ

Kapitola 11

Utility

Součástí NN Studia jsou konzolové utility RcgnProc a nTrain, které slouží k rozpoznávání obrazů.

nTrain

Utilita, sloužící k vytrénování neuronové sítě pro rozpoznávání obrazů.

Syntaxe příkazu

```
nTrain <input image> <weights file> <true value> <alpha> [nobalance]
```

RcgnProc

Utilita, sloužící k rozpoznávání obrazů.

Syntaxe příkazu

```
nTrain <input image> <weights file> [nobalance]
```

Část IV

Závěr

Kapitola 12

Plán vývoje a využití

Vývoj aplikace NN Studio je teprve na začátku a do budoucna mám v plánu přidat mnoho dalších funkcí a nástrojů, které by umožnili řešit úlohy pomocí neuronových sítí. Aplikace je nyní ve stádiu alpha testování verze 2.0.

12.1 Genetické algoritmy

Do budoucna je možné přidat podporu genetických algoritmů. To znamená možnost generačního vývoje neuronových sítí, jejich genetické křížení a nebo samo-rozvíjejících se síťí.

12.2 Import a export

Užitečnou funkcí je import a export do různých formátů. Například export testovací sady do formátu aplikace Microsoft Excel nebo naopak import z této aplikace. S touto funkcí je do budoucna počítáno.

12.3 Prostředí pro testování

Přínosné by bylo vytvořit nástroj, který by například umožňoval simulaci chování robota, který je řízený navrženou neuronovou sítí.

Kapitola 13

Použité informační zdroje a SW

13.1 Elektronické zdroje

AI Junkie *Neural Networks* [online] [31. 3. 2009]
URL: <http://www.ai-junkie.com/>
Vikram Pudi *Neural Networks Tutorial* [online] [31. 3. 2009]
URL: <http://www.iiit.net/ vikram/nn_intro.html>
Simple Neural Network as Robot Brain [online] [31. 3. 2009]
URL: <http://www.generation5.org/content/2005/neuroLego.asp>
Wikipedia *Neuron* [online] [31. 3. 2009]
URL: <http://cs.wikipedia.org/wiki/Neuronová_síť>

13.2 SW vybavení

LaTeXv2.5
Adobe Design Premium CS3 Student

Část V

Přílohy

Seznam příloh:

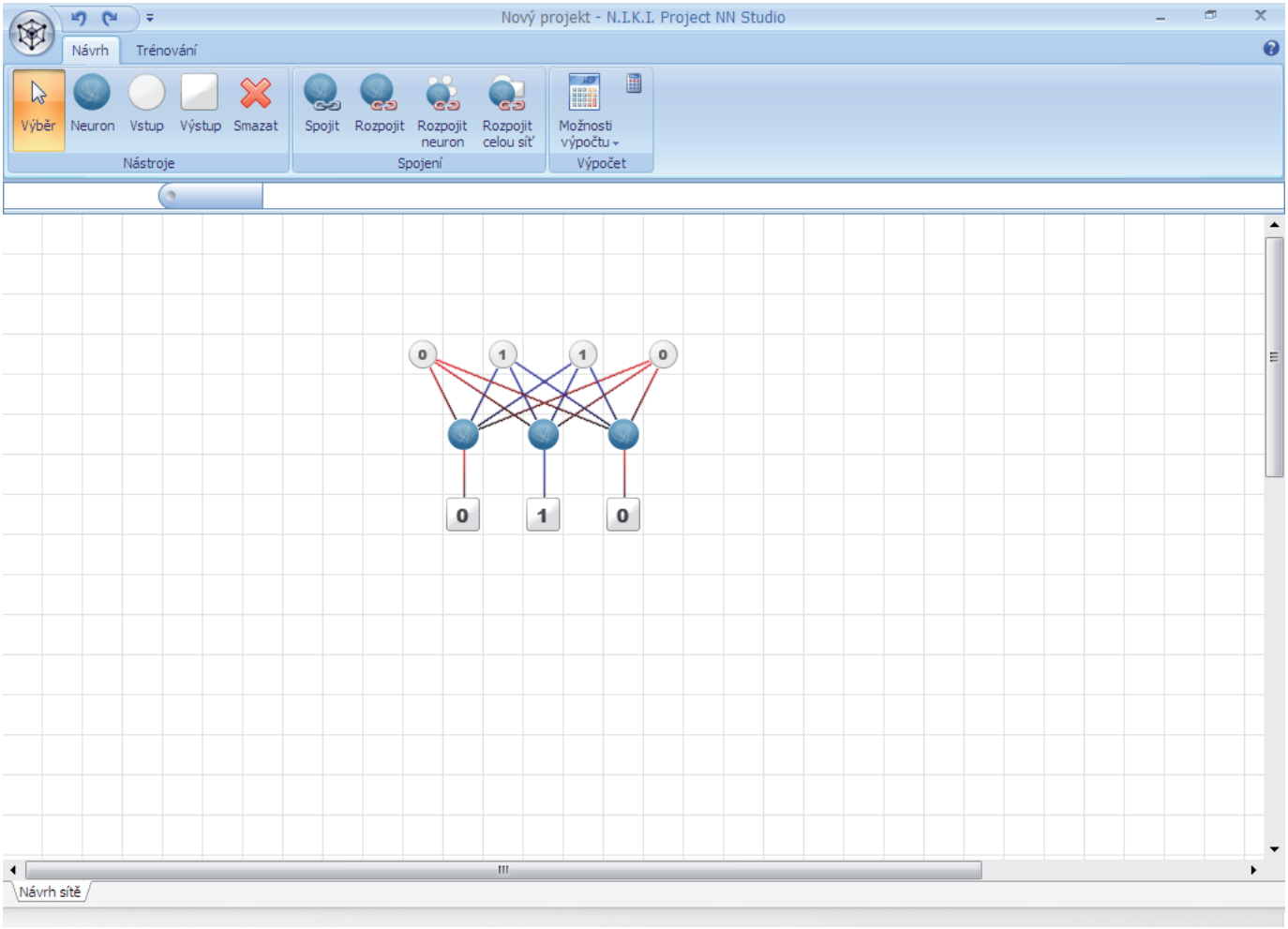
1. Ukázka zdrojového kódu struktury nNetwork
2. Ukázka návrhového zobrazení aplikace NN Studio
3. Ukázka trénovacího zobrazení aplikace NN Studio
4. Ukázka nástroje Recognition Wizard

Příloha č. 1 – Ukázka zdrojového kódu struktury nNetwork

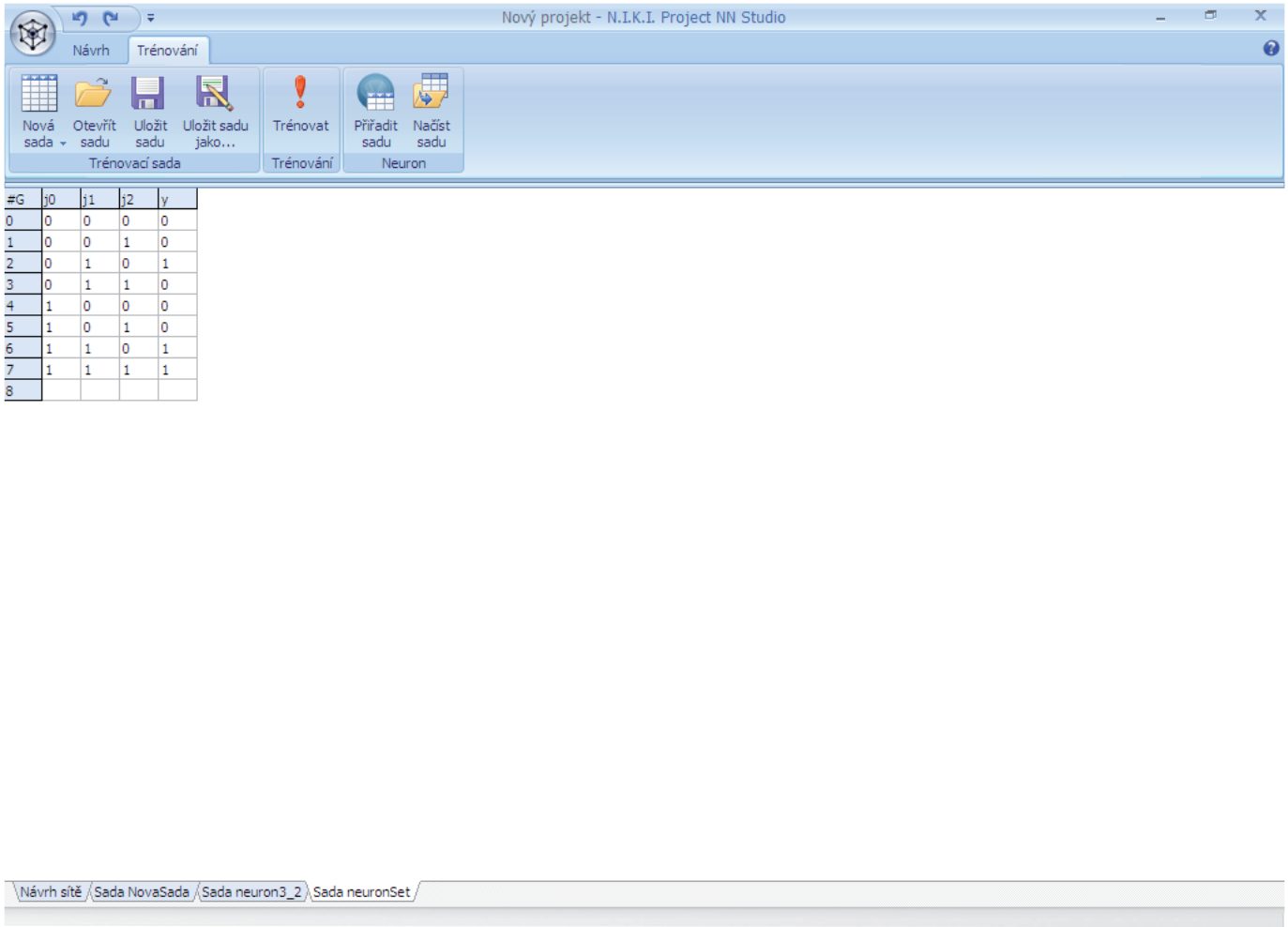
```
//Compute neuron
int compute(){
 if(computed==false){
  float summary=0.0;
  for(int i=0;i<inputsCount;i++)
   summary+=inputs[i]*weights[i];
  if(summary+bias-threshold>=0.0){
   output=1;
  } else {
   output=0;
  }
  computed=true;
 }
 return output;
}
//Train neuron
int train(float trueOutput, float alpha){
 //Nejprve je nutne spocitat vystup,
 //abychom mohli vypocitat velikost chyby
 compute();

 computed=false;
 for(int i=0;i<inputsCount;i++)
  weights[i]=weights[i]+alpha*(trueOutput-output)*inputs[i];
 return output;
}

//Network function - compute neuron in network
int computeNeuron(int index){
 //Zkontrolujeme, zda neuron existuje,
 //pokud ne skoncime s navratovou hodnotou 0
 if(!neurons[index].exists)
  return 0;
```

```
 //Zkontrolujeme, zda se jiz dany neuron nepocita,
 //slouzi jako ochrana proti zacykleni
 if(neurons[index].processing==true){
  return neurons[index].output;
 }
 neurons[index].processing=true;
 //Pripravime vstupy
 for(int i=0;i<neurons[index].inputsCount;i++){
  if(neurons[index].linkage[i]>0){
   //pokud je ID objektu > 0, jedna se o neuron
   neurons[index].inputs[i]=computeNeuron(neurons[index].linkage[i]-1);
  } else if(neurons[index].linkage[i]<0) {
   //pokud je ID objektu < 0, jedna se o vstup
   neurons[index].inputs[i]=inputs[abs(neurons[index].linkage[i])-1].value;
  }
 }
 //Spocitame neuron a vratime vysledek
 neurons[index].processing=false;
 return neurons[index].compute();
}

//Compute network
void compute(){
 //Zkontrolujeme konzistenci
 checkConsistency();
 //Nastavime vsechny neurony jako nespocitane
 for(int i=0;i<neuronsCount;i++)
  neurons[i].computed=false;
 //Pro kazdy vystup spocitame rekurzivne hodnotu
 for(int i=0;i<outputsCount;i++){
  if(outputs[i].linkage>0)
   outputs[i].value=computeNeuron(outputs[i].linkage-1);
 }
}
```

Příloha č. 2 - Ukázka návrhového zobrazení aplikace NN Studio



Příloha č. 3 - Ukázka trénovacího zobrazení aplikace NN Studio



#G	j0	j1	j2	y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1
8				

Příloha č. 4 - Ukázka nástroje Recognition Wizard