Projekt do NEU – Dokumentace

Vypracoval: **Zbyněk Křivka (xkrivk01)**
Datum: **10. 12. 2003**

Zadání:

Demonstrace využití neuronové sítě typu BP a RCE při analýze jazyků v teoretické informatice.

Analýza problému:

V teoretické informatice se pro analýzu jazyků používá tzv. syntaktického analyzátoru, což bývá matematický stroj určité obtížnosti – závisející na složitosti gramatiky (případně do jaké třídy Chomského hierarchie patří).

Třída jazyků	Potřebný stroj pro analýzu	Využití
3. regulární	konečný automat	sekvenční popis textu (kontrola výskytu)
2. bezkontextové	zásobníkový automat	zanoření (kontrola struktury), programovací jazyky
1. kontextové	turingův stroj	přirozený jazyk?
0. třída 0	turingův stroj	vyčíslitelné funkce; přirozený jazyk?

Motivace:

Protože analýza většiny jazyků je dosti nákladná (především kontextových jazyků a jazyků typu 0, kam patří i přirozený jazyk - nejasná sémantika, náhodné prvky), tak by se mohlo v některých konkrétních případech uvažovat o jakési předselekci vět, které má smysl dále analyzovat. K tomuto účelu by se možná mohly hodit právě neuronové sítě, jejichž výpočet (ne učení) je sám o sobě už poměrně rychlý proces.

Představme si, že máme zadaný jazyk $a^ib^ic^i$, kde i je přirozené číslo, a dále máme velké množství vět, o kterých se má rozhodnout, zda patří do tohoto jazyka. Proč si tedy prostřednictví neuronové sítě nevytipovat kandidáty na členství v zadaném jazyce a naopak.

Problém zadání jazyka a věty neuronové síti:

- Většina jazyků potenciálně obsahuje věty nekonečné délky; NS však neumí pracovat s nekonečnými vstupními vektory (nebo alespoň proměnné délky), tak budu nucen stanovit horní mez délky vstupního řetězce.
  Jiným řešením by bylo rozsekání věty na několik „okének“, ale nejsem si jist, zda by byla neuronová síť schopna reflektovat jejich úzkou návaznost.
- Konečná gramatika popisuje nekonečný jazyk, ale neuronovou síť, jež by měla na vstupu gramatiku si nedokáži dost dobře představit resp. myslím, že by nebyla schopna příliš dobře odvozovat věty právě z gramatiky (k tomu účelu máme právě ty zmiňované exaktní analyzátory).

Poznámka: Námět na další experiment: Rozpoznávat třídu zadaného jazyka, ať už podle vět nebo spíš podle pravidel.

PDF byl vytvořen zkušební verzí FinePrint pdfFactory http://www.fineprint.cz

Vybraný přístup k řešení a výběr typu neuronové sítě:
Pro experimenty s výše nastíněnou problematikou jsem vybral 2 druhy neuronových sítí:

1. **Back Propagation** Neural Network (LBF, skoková aktivační fce) – síť se širokým spektrem použití
2. **Restricted Coulomb Energy** = RCE (RBF, skoková aktivační fce) – klasifikace do tříd pomocí omezování prostoru hyperkoulemi.

Reprezentace vstupů:
Všechny jazyky přijímané implementovaným programem jsou podmnožinou jazyka {a,b,c}*. To znamená, že jsem použil pouze čtyři terminály s následujícím binárním zakódováním:
ε = 00 a = 001 b = 010 c = 100
Délka řetězce je omezena na maximálně 35 znaků => 70 vstupních neuronů.
ε … značí prázdný znak (pro případy řetězců kratších než maximum)

Druhý přístup je zakódovat terminály do celých čísel: ε = 3; a = 0; b = 1; c = 2; …atd
Tento přístup využívám v příkladech.

Algoritmy:
BP i RCE viz. přednášky předmětu NEU.

Implementace

Program by napsán v jazyce Squeak Smalltalk 3.6. Grafická implementace využívá standardně dodávaných kolekcí tříd Morphic.

Smalltalk je jazyk pro objektově orientované prototypování, takže byl možnou využít již vytvořený základní framework pro práci s neuronovými sítěmi s vytvořenou implementaci Back Propagation algoritmu. Takže já osobně jsem programoval pouze implementaci RCE, což si však vyžádalo podrobné prostudování celého frameworku (včetně pár oprav a úprav) i vzorové implementace BP.

Všechny třídy projektu je nachází v kolekci tříd **Neural-Networks**.

1. Neural-Networks framework & BP síť:

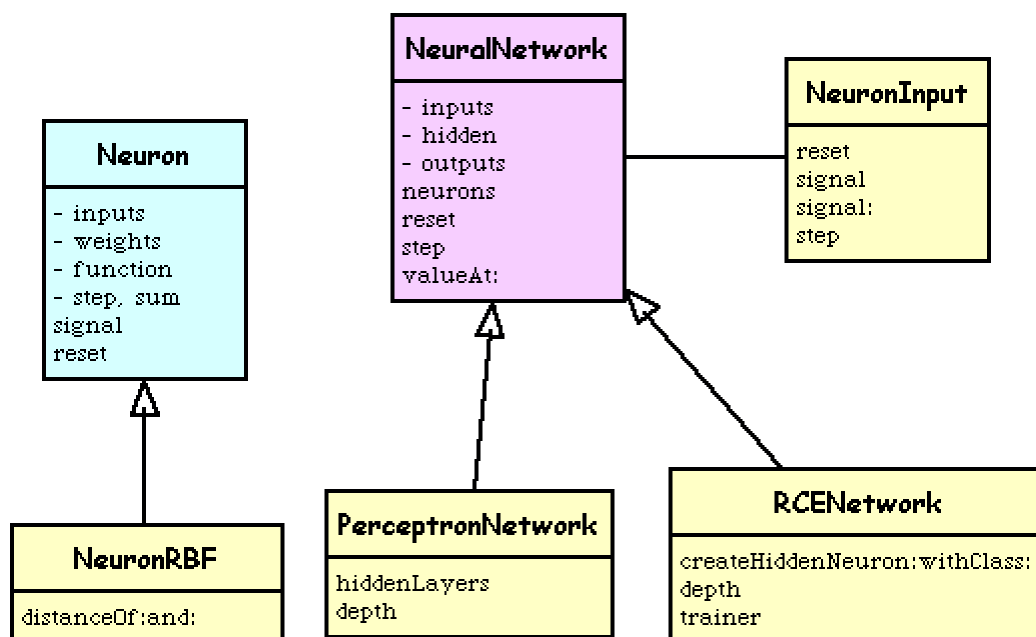
Framework se skládá z těchto základních tříd:

§ **Neuron** – jednoduchý model počítačového neuron, který obsahuje odkazy na vstupy a váhy. Odkazy na vstupy mohou ukazovat také na speciální třídu NeuronInput, která již nemá vlastní vstupy (jedná se o neurony vstupní vrstvy, které nemají veškerou funkčnost jako plnohodnotné neurony). Vstupní hodnotu neuronu uchovává proměnná *signal*. Pro výpočet bázové funkce slouží u LBF metoda *sum* a u RBF *distanceOfCenterAndInputVector*. Pro výpočet výsledku aktivační funkce slouží odkaz na funkční objekt odvozený od abstraktní třídy *ActivationFunction* (viz obr. UML1). Metoda *step* slouží pro synchronní výpočet výsledku celé neuronové sítě (po krocích).

§ **NeuronInput** – zjednodušený neuron (pouze signál a reset).

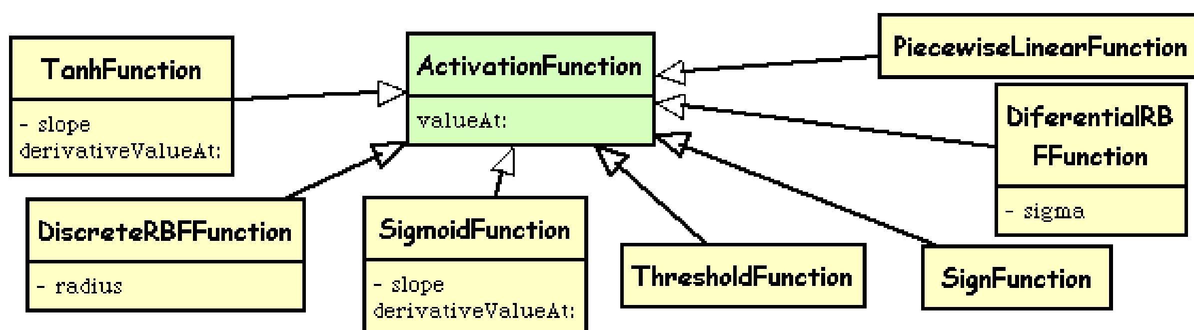
§ **NeuralNetwork** – model neuronové sítě (sice ne pouze dopředné, ale implementace v některých pasážích nepočítá s cykly apod.). Obsahuje seznamy vstupní, skrytých a výstupních neuronů. Každý neuron zná své vstupy/vstupní neurony, ale neznají do jakých neuronů směřuje jejich výstup (u backpropagation nevadí, protože se stejně provádí výpočet od výstupní vrstvy směrem ke vstupní; u RCE je to také jedno, protože pracuje pouze s 1 skrytou vrstvou).

PDF byl vytvořen zkušební verzí FinePrint pdfFactory http://www.fineprint.cz



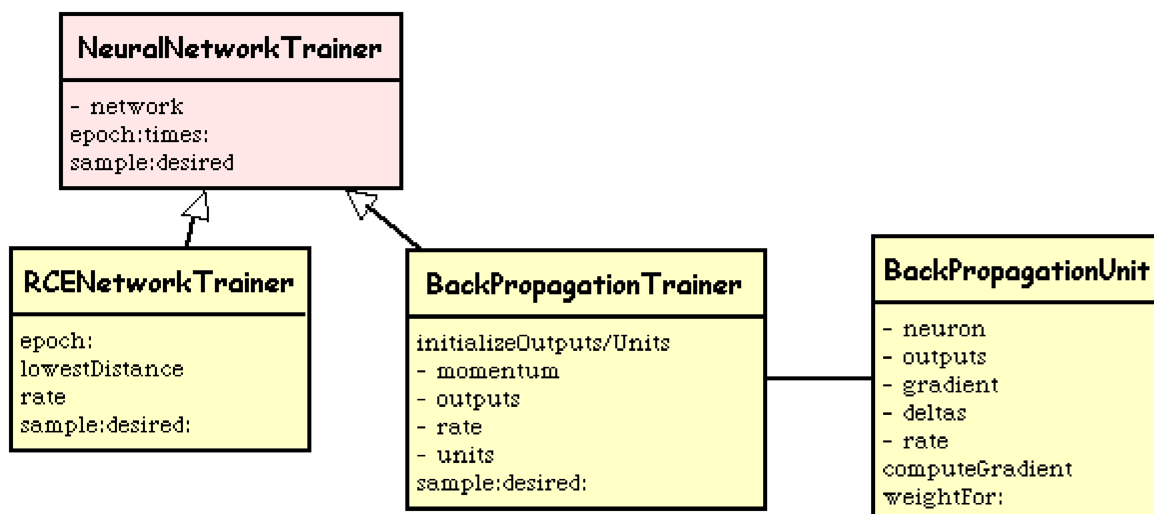
obr. UML1

§ **SignFunction, StochasticFunction, TanhFunction, ThresholdFunction, SigmoidFunction, PiecewiseLinearFunction,** – třídy pro vytvoření funkčních objektů, které jsou přiřazený každému neuronu a ten je používá jako **aktivační funkci**. V projektu se využívají pouze *DiscreteRBFFunction* pro skokovou aktivační RBF funkci (pro simulaci logického OR na výstupních neuronech RCE je použita *SignFunction*) a *SigmoidFunction* u BP.



obr. UML2

PDF byl vytvořen zkušební verzí FinePrint pdfFactory http://www.fineprint.cz



obr. UML3

- **PerceptronNetwork** – ukázka sítě perceptronů. Implementuje pouze pomocné metody.

- **NeuralNetworkTrainer** – třída starající se o učení neuronové sítě, na kterou obsahuje odkaz.
  Hlavní 3 metody:

  - *sample*: vstupníVektor *desired*: výstupníVektor … provede 1 krok učení na 1 vzorku
  - *epoch*: seznamDvojic_Vzor+Vystup … provede 1 krok učení pro všechny vzorky
  - *epoch*: seznam *times*: početOpakování … udává počet kroků učení

  Není implementováno učení až do určité chyby nebo procenta klasifikace.

Backpropagation třídy:

- BackPropagationUnit - stará se o výpočet 1 vrstvy zpětně
- BackPropagationTrainer - koordinuje BackPropagationUnit(s)

Doplnění:

- **Transformer** – třída se stará o převod řetězce na vstupní hodnoty akceptované NeuronInput. Existují 2 základní varianty (viz analýza). Implementovány jsou obě. Každá neuronová síť má svůj *transformer*.
- **StringTransformerNumeric** – převod znaku na signál velikosti 0, 1, … N. Tento transformer je využit v demonstrační aplikaci RCEApp.
- **StringTransformerBinary** – převod znaku na více binárních signálů (0 nebo 1).

2. RCE síť:

Implementováno je pouze nezbytně nutné chování odvozených tříd.

- **DiferentialRBFFunction, DiscreteRBFFunction** – slouží jako aktivační funkce (Radial Base): spojitá a skoková.

PDF byl vytvořen zkušební verzí FinePrint pdfFactory http://www.fineprint.cz

- **NeuronRCE** – neuron s Radial Based Funkcí (tj. nahrazuje sumu součinů vstupů a vah, eukleidovskou vzdáleností vektoru vstupu a středového vektoru skrytého neuronu), RCE neurony jsou pouze v 1 skryté vrstvě RCE sítě.
- **RCENetwork** – nově obsahuje pouze vytvoření RCE neuronu, jeho navázání na vstupní vrstvu a výstupní neuron (jen jeden; simulace log. fce OR pomocí vah 1.0 u všech vstupu neuronů výstupní vrstvy). Pokud odpovídající výstupní neuron (třída klasifikace) neexistuje, tak je vytvořen.

Atribut *rate* slouží pro míru snižování poloměru hyperkoule při špatném zásahu.

Poznámka: RCENetwork používá pro **klasifikaci do tříd** kolekci Array, která umožňuje indexovat pouze **od 1 výše**, takže nelze vytvářet klasifikaci do třídy 0. Také nedělá dobrotu tvořit očíslování tříd nesouvislé (např. třídy označené 1, 2, 4, 7 označte raději 1, 2, 3, 4).

- **RCENetworkTrainer** – učení probíhá pouze v 1 kroku (nemá smysl *epoch*:*times*:). Hlavním problémem je nastavení správných konstant zmenšování se hyperkoulí, aby síť dobře klasifikovala a zároveň byla dost obecná.

3. Aplikace:

Je možno si nechat vypisovat průběh učení RCE sítě zapnutím debug výpisů:

RCENetworkTrainer enableDebug.

Výpisy do pomocného textového okna, které značně zpomalují výpočet vypnete:

RCENetworkTrainer disableDebug.

Vykreslení 2D grafu RCE sítě (vhodné pro vstupy dimenze 2) lze:

RCEPlotMorph plotNN: neuronNetwork

nebo *RCEPlotMorph plotNN: trainer network*.

Celá aplikace se nachází ve třídě *RCEApp*.

Spuštění aplikace se provede následovně:

1. otevřete si workspace (kontextové menu -> open -> workspace)
2. napíšete výraz:

   RCEApp new openInWorld
3. výraz označíte a stisknete Alt + D (nebo kontextové menu->Do It)
4. aplikace je jednoduchá a ovladatelná intuitivně.

Aplikace využívá kolekci tříd *PlotMorph*, které slouží k tvorbě 2D grafů (podobně jako GNU Plot), takže je možno u sítí RCE zobrazovat první 2 rozměru vstupní vektorů a středových vektorů skrytých neuronů.

Experimenty a výsledky:

Logický OR: Ukázky učení jednoduchého OR s nahlédnutím na 2D graf vytvořených neuronů a učebních vstupů je dostatečně demonstrativní, takže nejsou nutné testovací prvky.

PDF byl vytvořen zkušební verzí FinePrint pdfFactory http://www.fineprint.cz

Vytvořil jsem 3 příklady učení, které mění pořadí učebních vstupních vektorů, takže jsou vytvářeny mírně odlišné konfigurace RCE sítě.
Spuštění pomocí výrazu (jedná se o třídní metody):

RCENetworkTrainer exampleOR1; exampleOR2; exampleOR3.

Příklady demonstrují citlivost RCE na pořadí vstupních vektorů při učení:

- nemusí vždy prospět náhodný výběr, vhodnější může být začít nejobecnějšími vektory (ty s největší vzájemnou vzdáleností)
- střídání pozitivních a negativních neuronů je velmi vhodné! Zmenší to počet nutných plných cyklů.
- snížení počtu skrytých neuronů RCE často vede k nižší obecnosti sítě.

Optimalizace RCE pro klasifikaci ANO/NE – tj. 1 třída nebo neúspěch (porovnat a zjistit zda se zlepšily vlastnosti nebo ne)

Ještě lepší přístup: 2 výstupní neurony (ANO, NE), pokud nefajruje ani jeden, tak síť neví a je potřeba provést klasickou SA => využití: předvoj syntaktické analýzy (předrozhodnutí).

Maximální délka řetězce je 12 znaků. Způsob kódování: 1 znak na 1 číslo (0 – 2), žádný znak označuje číslo 3.

Trénovací množiny jsou v příkladech sítí ve *workspace* (pro BP) nebo v jednotlivých metodách *RCEApp>>testingData1,2,3,4* a je jich přibližně stejně jako testovacích vzorků tedy kolem 15.

Spuštění: tento výraz vložím do workspace, označím a z kontextového menu vyberu DoIt.

RCEApp new testingData1; openInWorld.

Testovací množina měla vždy po 10-20 řetězcích, z nichž polovina patřila do jazyka a polovina ne.

BP: počet epoch = 100 – 1000 (pro 15 – 100 neuronů), tabelován je vždy nejlepší výsledek.
RCE: rate = 1.0 (nezměněn)

Testovaný jazyk	BP	RCE
$\mathbf{a^i b^i c^i}$ (kontextový) (testingData1)	30 neuronů => **85%** (2 chyby z trén.množiny)	**100 %**
$\mathbf{a^i b^i}$ (bezkontextový) (testingData2)	50 neuronů => **81%** (2 chyby z trén. množiny)	**100%** (dokonce i zobecněný vzorek)
$\mathbf{(ab)^*}$ (regulární) (testingData3)	50 neuronů => **76%** (4 chyby z trén. mny)	**82%** 1 chyba, 1 neví z trén. množiny
a*b* (regulární) (testingData4)	nerozeznala žádný prvek trénovací množiny – nepodařilo se správně nastavit parametry	**77%** 2 chyby, 2 neví z trén. mny.

PDF byl vytvořen zkušební verzí FinePrint pdfFactory http://www.fineprint.cz

Diskuse výsledků:

§ pro tak malé vstupní problémy je RCE PODSTATNĚ rychlejší než BP, protože nepracuje na epochy. Nevýhodou je však to, že RCE trénování zastaví, až začne správně klasifikovat všechny vzorky (pro malé vstupy to ale nemělo význam).
§ trénovací množiny by měli nejlépe obsahovat všechny přijímané řetězce a pak jejich nejbližší okolí reprezentující řetězce již nepřijímané => náročnost RCE na počet trénovacích vzorků
§ přestože jsem zvolil jazyky podobné struktury z různých tříd jazyků, tak překvapivě směrem k jednodušším jazykům (regulárním) se výsledky zhoršovali
§ podle intuice by se řeklo, že obecnost je horší u RCE (především kvůli náhodnému pořadí trénovacích vzorků), ale podle testů vychází RCE oproti BP překvapivě vítězně.
§ nastavení RCE je také kapitola sama pro sebe, protože do značné míry určuje míru obecnosti; další optimalizaci pro konkrétní problémy (jazyky) by mohlo být stanovení maximálního poloměru hyperkoule (v projektu uvažuji nekonečno: *Float infinity*) => problém překrytí hyperkoulí pro různé třídy (na kolik povolit/omezit).

Závěr:

Vyzkoušel jsem si jednoduchou implementaci neuronové sítě RCE a ověřil její funkčnost.

Bohužel omezení velikosti vstupního vektoru značně zmenšuje využitelnost neuronových sítí v naznačované oblasti analýzy jazyků.

Také přesnost sítí pro složitější jazyky je nevalná a množství nutných trénovacích řetězců se blíží výčtu celého omezeného jazyka.

Pro konsolidovanější závěry bychom samozřejmě museli provést mnohem více pokusů pro různé modifikace (vytvořené prostředí by se muselo doplnit o možnost automatické mutace sítě – např. upravování snižování vzdálenosti nebo změny pořadí vzorků, také by bylo nutno mít k dispozici automatizované statistické nástroje), to však nebylo cílem tohoto projektu (návrh spíše na ročníkový projekt).

PDF byl vytvořen zkušební verzí FinePrint pdfFactory http://www.fineprint.cz