Fakulta matematiky, fyziky a informatika
Univerzita Komenského

Projekt na Neuronové siete

Spojitý perceptrón

Student : Martin Slavkovsky
Studies : Aplikovaná informatika
Semester : 2
Email : martinslavkovsky@gmail.com

12.2.2015 Bratislava

1 Úvod

Cieľom projektu bolo implementovať jednoduchý spojitý perceptrón v ľubovoľnom nami vybratom programovacom jazyku, preto som si v projekte vybral Javu s použitím matematickej knižnice *apache-commons-math3* [1]. Na zobrazovanie dát som využil *gnuplot* [2], ktorý je potrebný na to, aby aplikácia bola schopná vykresliť krivku priebehu učenia a separujúcu 3D rovinu. K tomuto textu je teda priložený aj balíček so skompilovateľným zdrojovým kódom. (treba mať však maven!)

Kedže gnuplot nie je bežna súčasť systému treba ho nainštalovať. *Na* linuxe je veľmi jednoduché a npr. na Ubuntu alebo Debian je len len príkaz “sudo apt-get install gnuplot”.

1 Vzťah rýchlosti učenia a rýchlosti konvergencie pre spojitý perceptrón

Obrázky 5. - 9. nám ukazujú vplyv alfy(rýchlosť učenia) na stochastické učenie, kde s rastúcou alfou sa postupne kráti čas, za ktorý algoritmus skonverguje. V projekte som pre spojitý perceptrón naimplementoval *online* aj *batch learning* s tým, že stačí nastaviť príslušnú hodnotu *LearningType* pre inštanciu triedy Perceptron. Tiež som pridal možnosť, aby bolo možné vybrať aký typy chyby (enum *ErrorType*) bude algoritmus učenia vyhodnocovať.

Ako aktivačná funkcia bol použitý sigmoid, pričom výsledná zmena váh po chybovom vyhodnotení trénovacieho príkladu bude :

$$\Delta w = \alpha(d - y)(\frac{s(y)}{1 - s(y)})\overline{x}$$

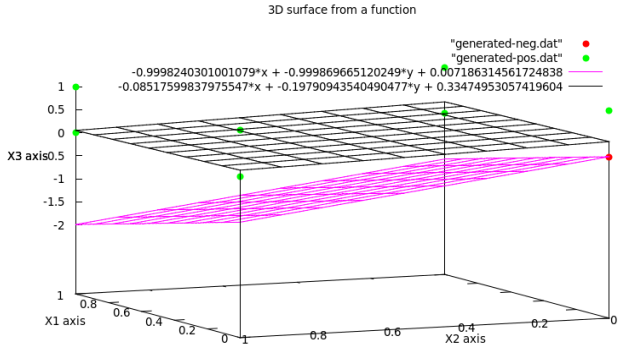
Kde s(x) je sigmoid, α je rýchlosť učenia, d cieľová hodnota pre vstup $\overline{x}$ a $f'(s(x)) = \frac{s(x)}{1 - s(x)}$.

Zvyšok trénovacieho algoritmu je takmer totožný s Diskrétnym perceptrónom.

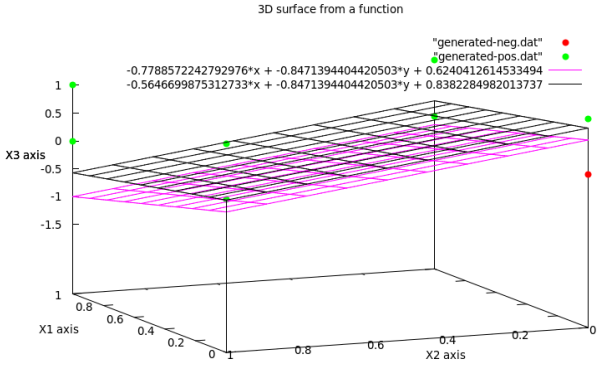
Aproximačná vs. klasifikačná chyba

Pri vyhodnocovaní aproximačnej chyby trénovanie skončí pri nejakej hodnote chyby $E > \varepsilon$. Rýchlosť konvergencie bude oveľa nižšia ako pri použití klasifikačnej, pretože pri vyhodnocovaní aproximačnej chyby nám algoritmus skonveruje v globálnom minime chybovej funkcie, čo nemusí byť 0. To znamená, že budeme pokračovať v trénovaní aj potom ako sme všetky príklady klasifikovali správne a snažíme sa nájsť "najlepší"vektor váh. Výhodou vyhodnocovania aproximačnej chyby by ale bolo oveľa presnejšie odhadnutie oddeľujúcej nadroviny za čo však budeme musieť zaplatiť aj stonásobným spomalením trénovania.

Na obrázku 1. a 2. môžeme vidieť porovnanie výslednej roviny pri použití trojrozmérných vstupov pre perceptrón. Pri vyhodnocovaní aproximačnou chybou sa algoritmus snaží nájsť rovinu, ktorá sa takmer pretína s vektorom z triedy 0 (červené) pričom na druhom obrázku vidíme len miernu zmenu natočenia roviny a mierne posunutie smerom nadol. Podobné výsledky som pozoroval pri 3D-OR a viacnásobnom trénovaní toho istého neurónu.



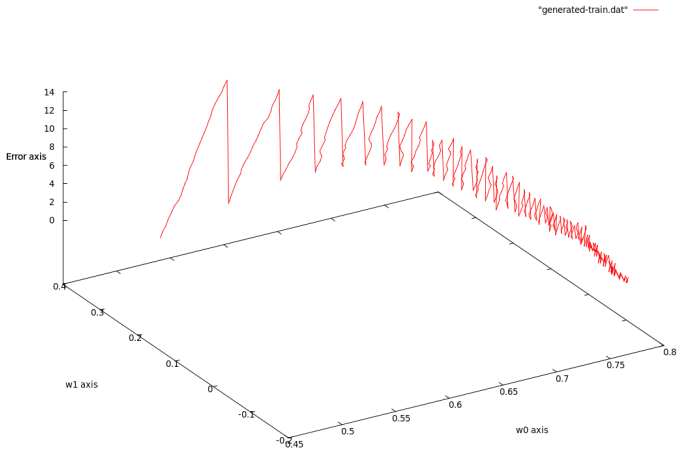
Obr. 1 - 3D-OR Aproximačná chyba.

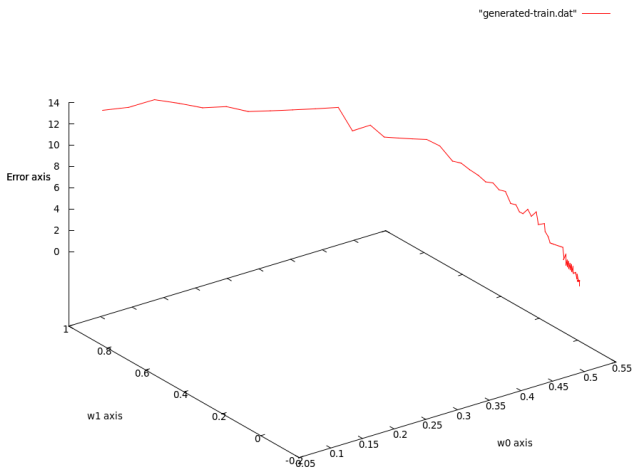


Obr. 2 - 3D-AND Klasifikačná chyba

Online vs batch learning

Na obrázku 3. - 4. je znázornený graf chybovej funkcie v závislosti od váh. Môžeme vidieť, že zatiaľ čo online learning postupuje k minimu veľkými skokmi, batch learning ide plynule ku globálnemu minimu. Z experimentovania som tiež odpozoroval, že stochastické učenie je menej náchylné na to, že algoritmus zdiverguje pri vyššej rýchlosti učenia. Napríklad pri použití tých istých dát (dvojrozmerné vstupy z druhého cvičenia "hard.mat") mal batch learning problém skonvergovať pri alfa = 0.1.





Obr. 3 - 3D-OR Online learning.

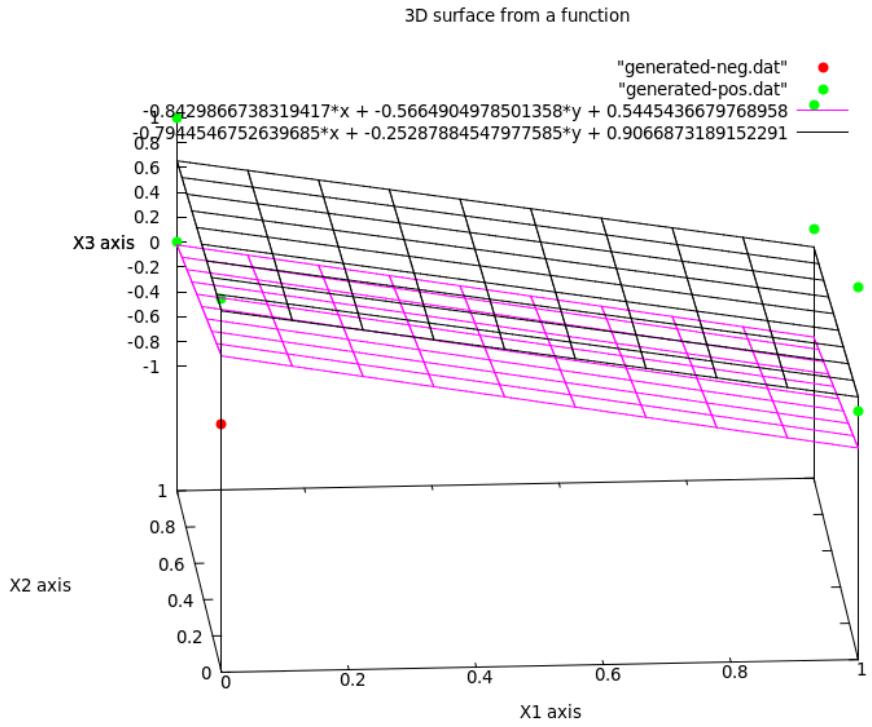
Obr. 4 - 3D-AND Batch learning

3 Simulácia učenia

Rovina oddeľúcej {0,1} triedy pre 3D-OR pred začiatkom učenia a po konci učenia je obrázku 3. zobrazená fialovou farbou pričom počiatočna je zobrazená čiernou. Na ďalšej strane je výpis zo simulácie trénovia sigmoidného perceptrónu pre alfa = 0.1 a 3D-OR dáta. Pre každú epochu je vypísaný nesprávne klasifikovaný vstup spolu s príslušnými váhami

===== Phase : Simulacia ucenia =====
x = {0; 1; 0}, expected = 1.000, predicted c(0.342) = 0.000, w = {0.793; 0.269; 0.998; 0.887}
x = {1; 0; 0}, expected = 1.000, predicted c(0.476) = 0.000, w = {0.816; 0.269; 0.998; 0.864}
---------Epoch 1, E = 1.0---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.356) = 0.000, w = {0.816; 0.287; 0.998; 0.846}
x = {1; 0; 0}, expected = 1.000, predicted c(0.493) = 0.000, w = {0.841; 0.287; 0.998; 0.821}
---------Epoch 2, E = 1.0---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.369) = 0.000, w = {0.841; 0.305; 0.998; 0.803}
---------Epoch 3, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.378) = 0.000, w = {0.841; 0.32; 0.998; 0.784}
---------Epoch 4, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.387) = 0.000, w = {0.841; 0.344; 0.998; 0.764}
---------Epoch 5, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.396) = 0.000, w = {0.841; 0.363; 0.998; 0.745}
---------Epoch 6, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.406) = 0.000, w = {0.841; 0.384; 0.998; 0.724}
---------Epoch 7, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.416) = 0.000, w = {0.841; 0.404; 0.998; 0.704}
---------Epoch 8, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.426) = 0.000, w = {0.841; 0.426; 0.998; 0.682}
---------Epoch 9, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.436) = 0.000, w = {0.841; 0.448; 0.998; 0.661}
---------Epoch 10, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.447) = 0.000, w = {0.841; 0.47; 0.998; 0.638}
---------Epoch 11, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.458) = 0.000, w = {0.841; 0.49; 0.998; 0.615}
---------Epoch 12, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.469) = 0.000, w = {0.841; 0.516; 0.998; 0.592}
---------Epoch 13, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.481) = 0.000, w = {0.841; 0.540; 0.998; 0.568}
---------Epoch 14, E = 0.5---------
x = {0; 1; 0}, expected = 1.000, predicted c(0.493) = 0.000, w = {0.841; 0.565; 0.998; 0.543}
---------Epoch 15, E = 0.5

InitWeights = {0.79; 0.252; 0.998; 0.905}
Weights   = {0.841; 0.565; 0.998; 0.543}
MinWeights = {0.841; 0.565; 0.998; 0.543}
Error   = 0.0
MinError   = 0.0
Epoch   = 16
Converged   = true



Obr. 4 - Simulácia online učenia pre 3D-OR

3 Diskrétny vs spojitý perceptrón

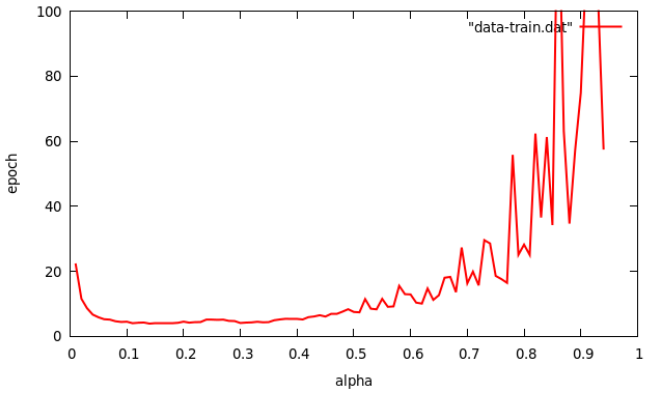
V projekte sú naimplementované dva druhy perceptrónov : spojitý a diskrétny pričom mierny rozdiel je v metóde trénovania. Aj keď je to v je kód pre diskrétny a spojitý perceptrón v kóde oddelený, trénovanie diskrétného perceptrónu sa od spojitého líši len v použití inej funkcie resp. jej derivácie pri zmene váh. Preto by tento kód by bolo veľmi ľahké a správne spojiť pomocou dedenia.

Obrázky na ďalšej strane ukazujú rozdiely medzi správaním dvoch typov perceptrónov. Vidíme, že disk. perceptrón má pri trénovaní 3D-AND stabilnejšie výsledky s rastúcou rýchlosťou učenia ako sigmoidný. 3D-AND je pravdepodobne náchylnejšie na nekonzistentnú klasifikáciu vysokých hodnotách alfy. Priebeh 3D-OR je v oboch prípadoch takmer totožný.

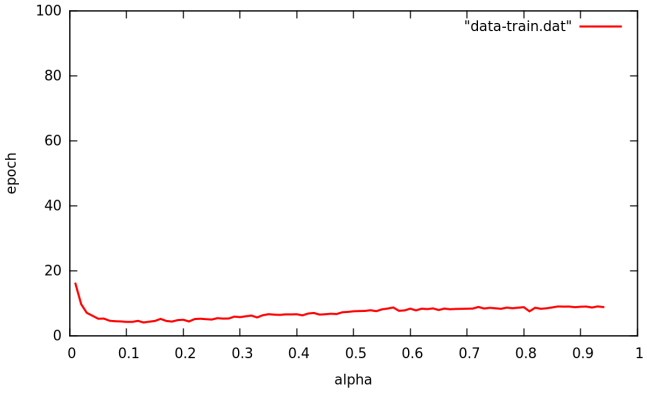
To isté môžeme pozorovať pri zložitejších lineárne separovatelných množinách a na obrázkoch 5. a 6. som použil dvojrozmerné vstupy z druhého cvičenia (“hard.mat”). Pre každú

hodnotu alfa sa natrénovalo 200 perceptrónov a vypočítal sa priemer rýchlostí konvergencie (y osa - epoch). Inak povedané všetky udávané grafy priemernú rýchlosť konvergencie pri danej rýchlosti učenia.

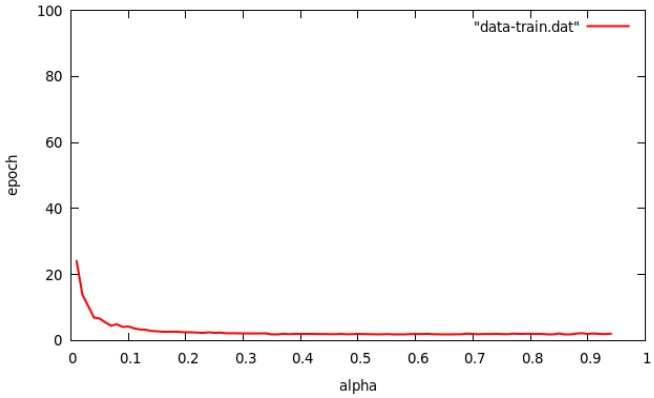
Na základe expertimentovania usudzujem, že veľký rozdiel medzi perceptrónmi nie je, aspoň čo sa týka tohoto testu. Pri testovaní generalizácie (mali by sme veľkú množinu a testovacie dáta) by pravdepodobne diskrétny vyšiel v priemere horšie.



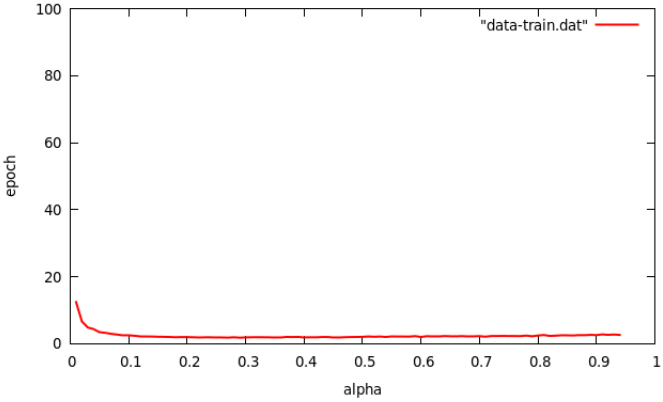
Obr. 5 - 3D-AND Sigmodiný p.



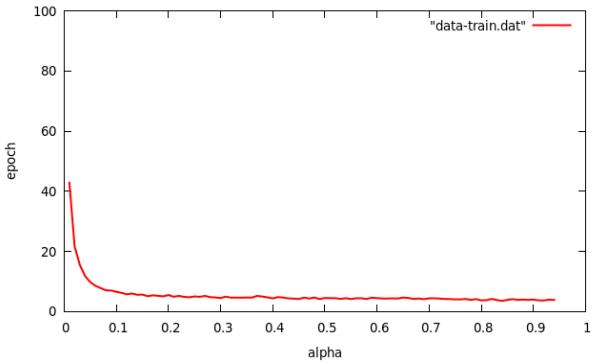
Obr. 6 - 3D-AND Diskrény p.



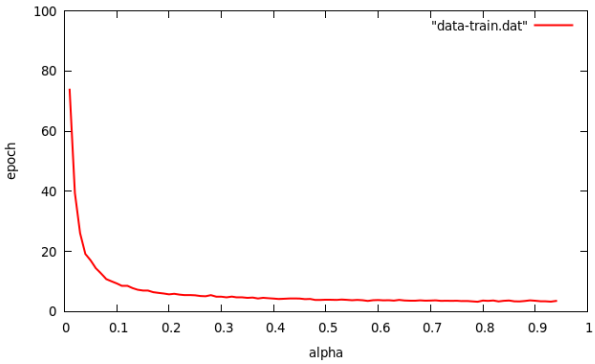
Obr. 6 - 3D-OR Sigmodiný p.



Obr. 7 - 3D-OR Diskrétny p.



Obr. 8 - 2D-HARD Sigmodiný p.



Obr. 9 - 2D-HARD Diskrétny p.

7 Odkazy

[1] **http://commons.apache.org/proper/commons-math/**
[2] **http://www.gnuplot.info/**