Sprawozdanie

1.Syntetyczny opis budowy oraz użytego algorytmu uczenia.

Perceptron został zaprojektowany na wzór sieci neuronowych jakie występują w naszych mózgach. Jest to matematyczne przedstawianie w jaki sposób komórki mózgowe przeliczają informacje na odpowiednie wyniki np. to że rozróżniamy kolory, albo potrafimy dodawać. Dzieje się tak za pomocą obliczania prostych funkcji i sprawdzania podanych warunków. Najpierw jest obliczana suma iloczynów wag i wejść do naszego sztucznego neuronu, a następnie wynik jest przepuszczany przez funkcję aktywacji. Najczęściej funkcjami aktywacji są proste funkcje takie jak sigma, liniowa, bipolarna, unipolarna. Nasze wejścia są to wartości dla jakich chcemy, aby nasz neutron „przetrawił” informacje, natomiast wagi są to współczynniki przez jakie przemnażamy wejścia. Następnie sprawdzany jest wynik aktywacji i w zależności od sposobu uczenia podejmowane są decyzje. Na naszych laboratoriach użyliśmy uczenia z nauczycielem czyli takiego, gdzie po każdym uczeniu jest obliczana różnica pomiędzy wynikiem otrzymanym, a prawdziwym i w zależności od przypadku zwiększamy, albo zmniejszamy wagi. Zmiana wag zależy od przyjętego algorytmu. W metodzie perceptronowej, najbardziej podstawowej, zmiana wag następuje poprzez dodanie bądź odjęcie wejść. W metodach bardziej zaawansowanych ustala się odpowiedni krok.

Odnośnie samej budowy dla naszego zadania nasz perceptron miał 3 wejścia (2 dla argumentów funkcji logicznej i 1 dla biasu, który na wejściu zawsze ma 1) oraz 3 wagi dla odpowiednich wejść. Funkcją aktywacji natomiast przyjmowała 1 dla y >= 0 oraz 0 dla y < 0. Jako krok po wielu próbach ustaliłem 1 gdyż wtedy algorytm wydawał się najstabilniejszy, lecz dobrze sprawowało się również dla 0,5.

2 - 3 Zestawienie otrzymanych wyników oraz analiza:

Dla kroku przyjętego 1 było to około 25 – 60 powtórzeń

Dla kroku 0,5 to okolice 20 – 25 oraz 50 – 50 powtórzeń

Dla kroku 0,1 to wyniki z przedziału 0 – 100 powtórzeń

Stosowałem również takie kroki jak 1,5 albo 3 ale wyniki były zazwyczaj lekko gorsze, niż dla kroku 1. Nie utworzyłem niestety żadnych wykresów, albo dokładnie liczonych średnich, gdyż tylko pierwsze losowanie wag(zawsze na początku losuję wagi dopiero dla nich uczę perceptron) było jakieś racjonalne, natomiast potem otrzymywałem wyniki np., 5 razy z rzędu nauczył się za pierwszym razem, następnie 2 razy, że za 3 razem i tak przez 1000 powtórzeń, więc zrezygnowałem z takiego liczenia i odpalałem program wiele razy dla różnych wag (odnośnie nauczania się za pierwszym razem, parę razy i tak się przytrafiło w trakcie testów). Podejrzewam, że błąd krył się w pseudolosowości programu. Natomiast odnośnie danych uczących praktycznie niemożliwe było nauczenie perceptronu tak prostej funkcji logicznej bez podania mu wszystkich możliwych wartości (11,10,01,00), gdyż pula ucząca była za mała. W przypadku, gdy pozbywałem się z uczenia 11, który jako jedyny dawał 1 na wyjściu brakowało w metodzie uczenia czegoś co by przeważało na drugą stronę osi wyników, perceptron uczył się, że kiedykolwiek wyjdzie mu 0 jako wynik jest to prawda.

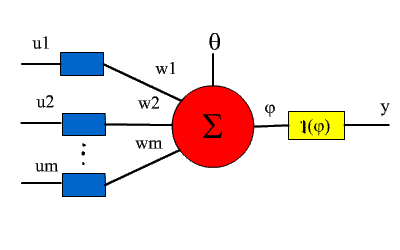
4. Wnioski:

- algorytm funkcji logicznej AND uczy się w okolicach 15 – 50 powtórzeń,

- głównym warunkiem szybkości nauczania się jest krok,

- odpowiednio dobrany krok pozwala na uczenie się przy bardzo podobnej ilości kroków np. dla kroku 1.5, 3, albo 1, w przeciwnym razie dla takich wartości jak 0,5 wyniki były albo bardzo zbliżone do 20 powtórzeń albo 50, a dla kroku 0,1 były to przeróżne wartości do około 100 powtórzeń, skupiające się głównie na liczbie 70,

- duże znaczenie mają również wagi początkowe, gdyż może się zdarzyć, że już dla pierwszej iteracji algorytm się skończy i perceptron będzie nauczony oraz, że wagi są tak rozproszone, że nawet najlepszy krok sobie z tym średnio poradzi.



Gdzie:

u1, u2, um wejścia perceptronu,

w1, w2, wm wagi perceptronu,

∑ suma iloczynów wag i wejść,

θ bias,

ϕ wynik sumy,

f(ϕ) = y wynik naszej funkcji aktywacji,

Listing kodu:

Plik Tester.h

#pragma once

class Subtester {

public:

double x1, x2, wynik;

};

class Tester {

public:

Subtester tab[4];

Tester();

};

Plik Tester.cpp

#include "Tester.h"

Tester::Tester() {

tab[0].x1 = 0;

tab[0].x2 = 0;

tab[0].wynik = 0;

tab[1].x1 = 1;

tab[1].x2 = 0;

tab[1].wynik = 0;

tab[2].x1 = 0;

tab[2].x2 = 1;

tab[2].wynik = 0;

tab[3].x1 = 1;

tab[3].x2 = 1;

tab[3].wynik = 1;

}

Plik Perceptron.h

#pragma once

#include <iostream>

#include <cstdlib>

#include <ctime>

#include "Tester.h"

class Perceptron {

private:

double w[3];

double x[2];

double wspolczynnik;

public:

Perceptron();

void ustal\_wagi(double, double, double);

void wypisz\_wagi();

void ustal\_wejscie(double, double);

double aktywacja();

void licz\_ext(double, double, double);

void licz(double, double, double);

void licz(Subtester);

void sprawdz(double,double);

bool sprawdz\_bool(double, double);

double \*losuj();

bool zrob\_test();

};

Plik Perceptron.cpp

#include "Perceptron.h";

Perceptron::Perceptron(){

w[0] = 0;

w[1] = 0;

w[2] = 0;

x[0] = 0;

x[1] = 0;

wspolczynnik = 1.0;

}

void Perceptron::ustal\_wagi(double w1, double w2, double w3) {

w[0] += w1;

w[1] += w2;

w[2] += w3;

}

void Perceptron::wypisz\_wagi() {

std::cout << "Waga 1 = " << w[0] << std::endl;

std::cout << "Waga 2 = " << w[1] << std::endl;

std::cout << "Waga 3 = " << w[2] << std::endl << std::endl;

}

void Perceptron::ustal\_wejscie(double x1, double x2) {

x[0] = x1;

x[1] = x2;

}

double Perceptron::aktywacja() {

return w[0] + w[1] \* x[0] + w[2] \* x[1];

}

void Perceptron::licz\_ext(double x1, double x2, double oczekiwany\_wynik) {

double obecny\_wynik;

ustal\_wejscie(x1, x2);

std::cout << "oczekiwany wynik: " << oczekiwany\_wynik << std::endl;

std::cout << "x1: " << x[0] << std::endl << "x2: " << x[1] << std::endl;

std::cout << "aktywacja: " << aktywacja() << std::endl;

if (aktywacja() >= 0)

obecny\_wynik = 1;

else

obecny\_wynik = 0;

std::cout << "obecny wynik:" << obecny\_wynik << std::endl;

if (obecny\_wynik > oczekiwany\_wynik) {

std::cout << "zmniejszam wagi" << std::endl;

ustal\_wagi(-1\*wspolczynnik, -x[0]\*wspolczynnik, -x[1]\*wspolczynnik);

}

else if (obecny\_wynik < oczekiwany\_wynik) {

std::cout << "zwiekszam wagi" << std::endl;

ustal\_wagi(1 \* wspolczynnik, x[0] \* wspolczynnik, x[1] \* wspolczynnik);

}

}

void Perceptron::licz(double x1, double x2, double oczekiwany\_wynik) {

double obecny\_wynik;

ustal\_wejscie(x1, x2);

if (aktywacja() >= 0)

obecny\_wynik = 1;

else

obecny\_wynik = 0;

if (obecny\_wynik > oczekiwany\_wynik)

ustal\_wagi(-1 \* wspolczynnik, -x[0] \* wspolczynnik, -x[1] \* wspolczynnik);

else if (obecny\_wynik < oczekiwany\_wynik)

ustal\_wagi(1 \* wspolczynnik, x[0] \* wspolczynnik, x[1] \* wspolczynnik);

}

void Perceptron::sprawdz(double x1, double x2) {

double tmp1 = x1, tmp2 = x2;

ustal\_wejscie(x1, x2);

if (aktywacja() >= 0)

std::cout << "TRUE" << std::endl;

else

std::cout << "FALSE" << std::endl;

ustal\_wejscie(tmp1, tmp2);

}

bool Perceptron::sprawdz\_bool(double x1, double x2) {

double tmp1 = x1, tmp2 = x2;

ustal\_wejscie(x1, x2);

if (aktywacja() >= 0) {

ustal\_wejscie(tmp1, tmp2);

return true;

}

else {

ustal\_wejscie(tmp1, tmp2);

return false;

}

}

double\* Perceptron::losuj() {

srand(time(NULL));

double \*tab = new double[3];

tab[0] = ((std::rand() % 20)-10)/10.;

tab[1] = ((std::rand() % 20)-10)/10.;

tab[2] = ((std::rand() % 20)-10)/10.;

return tab;

}

void Perceptron::licz(Subtester test) {

double obecny\_wynik;

ustal\_wejscie(test.x1, test.x2);

if (aktywacja() >= 0)

obecny\_wynik = 1;

else

obecny\_wynik = 0;

if (obecny\_wynik > test.wynik)

ustal\_wagi(-1 \* wspolczynnik, -x[0] \* wspolczynnik, -x[1] \* wspolczynnik);

else if (obecny\_wynik < test.wynik)

ustal\_wagi(1 \* wspolczynnik, x[0] \* wspolczynnik, x[1] \* wspolczynnik);

}

bool Perceptron::zrob\_test() {

if(sprawdz\_bool(1, 1));

else

return false;

if(!sprawdz\_bool(1, 0));

else

return false;

if (!sprawdz\_bool(0, 1));

else

return false;

if (!sprawdz\_bool(0, 0));

else

return false;

return true;

}

Plik Źródło.cpp

#include <iostream>

#include "perceptron.h"

using namespace std;

int main() {

Perceptron p = Perceptron();

Tester t = Tester();

double \*tab = p.losuj();

p.ustal\_wagi(tab[0], tab[1], tab[2]);

int z = 0, i = 0;

while (true) {

z = rand() % 4;

p.licz(t.tab[z]);

if (p.zrob\_test())

break;

i++;

}

cout << "Udalo sie za " << i << " podejsciem!" << endl;

p.wypisz\_wagi();

p.sprawdz(1, 1);

p.sprawdz(1, 0);

p.sprawdz(0, 1);

p.sprawdz(0, 0);

system("pause");

}

Bibliografia:

<https://pl.wikipedia.org/wiki/Funkcja_aktywacji>

<http://www.cs.put.poznan.pl/rklaus/assn/percep.htm>

<http://aragorn.pb.bialystok.pl/~gkret/SSN/Ssn_w2.PDF>

Maciej Truczka