Temat: Budowa​ ​i​ ​działanie​ ​perceptronu

Data oddania: 17.10.2017

Prowadzący: dr inż. Dorota Wilk - Kołodziejczyk

Przedmiot: Podstawy sztucznej inteligencji

Autor: Przemysław Adamus 286084

Kierunek: Inżynieria Obliczeniowa sem.V

Spis treści:

1. Cel projektu;
2. Wstęp i opis wykonanych zadań/etapów;
3. Budowa perceptronu;
4. Syntetyczny​ ​opis​ ​budowy​​ ​wykorzystanego​ ​algorytmu​ ​uczenia w projekcie C++;
5. Opis projektu MatLab;
6. Wyniki;
7. Wnioski i analiza wyników;
8. Źródła;
9. Załączniki.
10. **CEL:**

Celem​ ​ćwiczenia​ ​jest​ ​poznanie​ ​budowy​ ​i​ ​działanie​ ​perceptronu​ ​poprzez​ ​implementację​ ​oraz uczenie​ ​perceptronu​ ​realizującego​ ​wybraną​ ​funkcję​ ​dwóch​ ​zmiennych.

1. **WSTĘP:**

Zanim usiadłem do rozwiązania problemu przedstawionego w scenariuszu 1, zapoznałem się z ogólnymi zasadami działania sieci neuronowej. Poszerzyłem swoją wiedzę na bazie artykułów i tutoriali internetowych (ważniejsze adresy źródłowe w pkt 7). Następnie przystąpiłem do tworzenia programu .

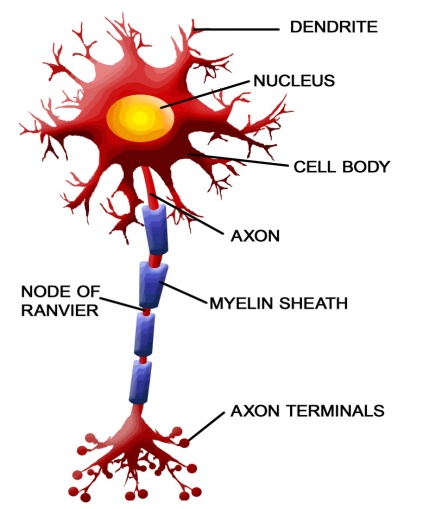
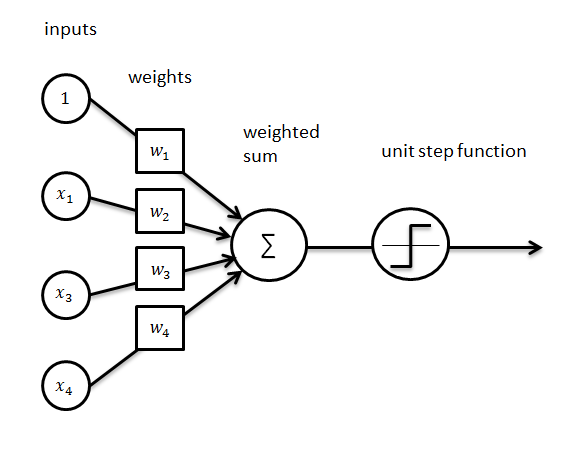
Zadaniem mojego programu było podanie wyniku funkcji logicznej – koniunkcja – funkcji logicznej AND. Na wejściu program otrzymywał dwie wartości logiczne 0 lub 1. Spodziewaną odpowiedzią jest również wartość logiczna, np. 0 ˄ 1 => 0 lub 1 ˄ 1 => 1

W sprawozdaniu zostały umieszczone rozwiązania, kody i wyniki zadania z środowiska MatLab oraz C++. W programie MatLab użyłem z gotowych funkcji do tworzenia sieci neuronowych – załącznik 3. Skorzystałem z biblioteki narzędziowej „Neural Network Toolbox”. Jednak chcąc lepiej zrozumieć zasady jej działania zaimplementowałem pojedynczy neuron w języku C++. Stworzyłem klasę neuron wraz z całą gamą metod do działania na obiektach tej klasy. Znajduje się tam również funkcja do nauki neuronu. Klasę umieściłem w pliku Neurone.h – załącznik 1.

Przygotowałem zestaw uczący – załącznik 4. Jest on zapisany w rozszerzeniu .csv. Dane są ułożone w odpowiedniej kolejność – najpierw dwie wartości wejścia , a następnie spodziewany wynik funkcji logicznej. Program wyposażony jest w funkcję wczytującą dane z pliku .csv.

Po zakończonej implementacji przetestowałem mój neuron.

1. **BUDOWA PERCEPTRONU:**



Perceptronem nazywamy prosty element obliczeniowy, który sumuje ważone sygnały wejściowe i porównuję tę sumę z progiem aktywacji - w zależności od wyniku perceptron może być albo wzbudzony (wynik 1), albo nie (wynik 0).

Funkcja aktywacji w sztucznej inteligencji określa funkcje, według której obliczana jest wartość wyjścia neuronów sieci neuronowej. Do najczęściej używanych funkcji aktywacji należą: funkcja liniowa, funkcja progowa unipolarna, funkcja progowa bipolarna, funkcja sigmoidalna unipolarna, funkcja sigmoidalna bipolarna.

1. **ALGORYTM UCZENIA W PROJEKCIE C++:**

  Algorytm uczenia perceptronu jest to automatyczny doboru wag na podstawie napływających przykładów. W moim programie C++ za uczenie neuronu odpowiada metoda klasy Neurone typu void o nazwie „teach\_neurone”. Przyjmuje ona tablice wejść i oczekiwaną odpowiedź. W moim projekcie C++ możemy pozwolić sobie na zainicjowanie wagi losowo. Najlepiej z przedziału od -0.5 do 0.5, a współczynnik nauczania dobieramy dowolny. Ja zdecydowałem się na 0,1.

* Dla każdego przykładu uczącego obliczamy odpowiedź perceptronu.
* Jeśli odpowiedź perceptronu jest nieprawidłowa, to modyfikujemy wagi według reguły uczenia Widrowa – Hoffa:

w1 += n \* (d-y) \* x1   
w2 += n \* (d-y) \* x2   
…

gdzie:

n - współczynnikiem uczenia (n > 0);

d - oczekiwana odpowiedź;

y - odpowiedź neuronu.

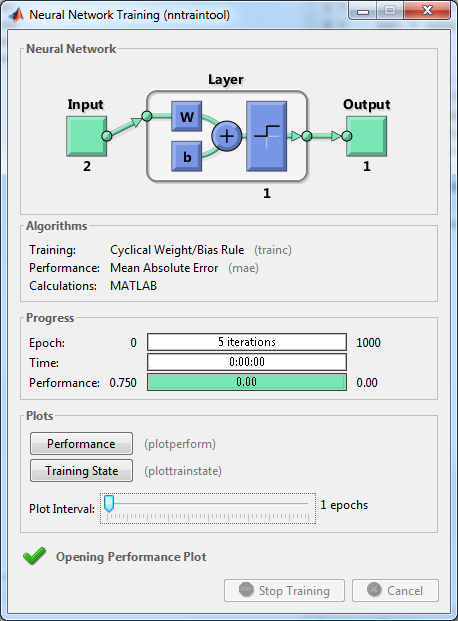
Po wyczerpaniu przykładów zaczynamy proces uczenia od początku, do momentu kiedy liczba cykli uczenia (liczba epok) przekracza zdefiniowaną z góry liczbę epok lub średniokwadratowy błąd sieci maleje poniżej progu błędu ε.

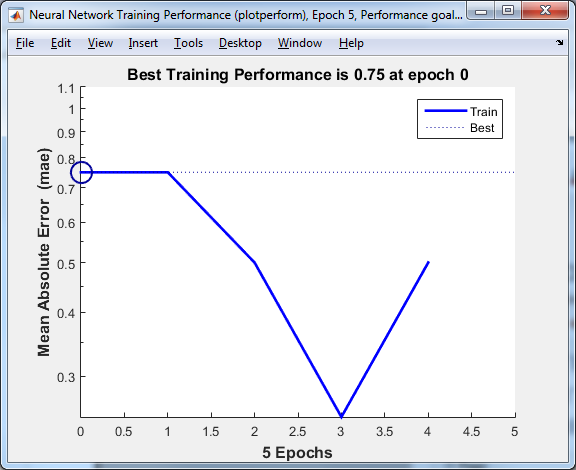
1. **OPIS PROJEKTU MATLAB:**

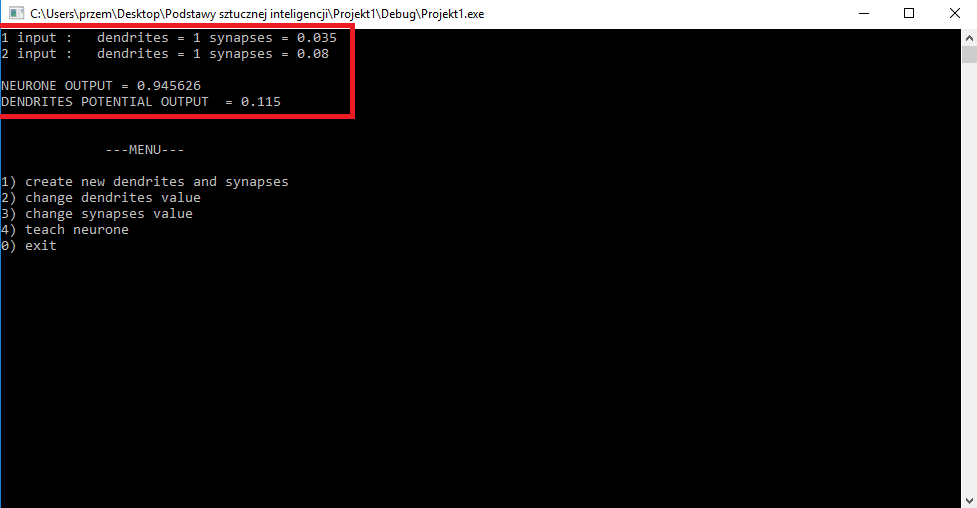
Program w MatLab (załącznik 3) również liczy rezultat bramki logicznej AND. Z podanej wcześniej biblioteki użyłem 3 funkcji:

* newp – tworzy jednowarstwową sieć perceptronów;
* sim – symulacja rezultatu dla podanych danych wejściowych;
* train – trening/nauka sieci neuronowej;

Środowisko MatLab automatycznie ustawia początkowe wagi na wartość zero, a współczynnik uczenia równa się 0,001;

1. **WYNIKI:**

*Projekt MatLab*

**

*Testowanie perceptronu: np. dla wejścia 1 i 1 -> odpowiedź neuronu 0.945626 ~ 1*

*Projekt C++*

1. **WNIOSKI I ANALIZA WYNIKÓW**:

W obu projektach wagi zmieniają się po każdej iteracji. W programie MatLab nauka zawsze zakończyła się po 3 iteracjach. Natomiast w projekcie C++ błąd osiąga zadawalający poziom już po pierwszym cyklu nauki. Rozbieżność może być to spowodowane większym współczynnikiem uczenia, lub różnymi startowymi wartościami wag. Jednak w obu przypadkach efekt jest zadowalający, choć zastosowano różne zestawy uczące.

W podpunktach wyżej nie znalazł się opis wszystkich metod i możliwości projektu C++. Osobiście uważam, że najlepszym komentarzem do niego jest jego kod. Starałem się aby sam się komentował, a jego budowa i nazewnictwo odnosiły się do prawdziwej naturalnej sieci neuronowej.

Podstawowym wnioskiem, nasuwającym się po zakończonym projekcie, jest spostrzeżenie, że środowisko MatLab jest łatwiejsze i lepiej przystosowane do tworzenia sieci neuronowych niż język c++. W tym języku nie znalazłem dobrej biblioteki przygotowanej do pracy z sztuczną inteligencją. Jest to prawdopodobnie spowodowane istnieniem wielu bardziej zaawansowanych narzędzi do rozwiązywania tych zagadnień. Lecz nie oznacza to, że nie jest możliwe osiągnięcie poprawnych wyników.

1. **ŹRÓDŁA:**

* <https://edux.pjwstk.edu.pl/mat/273/lec/wyklad3/w3.htm>;
* <https://pl.wikipedia.org/wiki/Perceptron>
* <https://forum.pasja-informatyki.pl/73877/sztuczne-sieci-neuronowe>
* <http://www.if.uz.zgora.pl/~mdudek/siecineuronowe.pdf>
* <https://www.youtube.com/watch?v=Wa_9S20SkKw&t=995s>
* Materiały udostępnione przez dr inż. Dorota Wilk - Kołodziejczyk

1. **ZAŁĄCZNIKI:**
   1. Neurone.h – klasa neuron, projekt C++;
   2. Main.cpp – funkcja main, projekt C++;
   3. Program MatLab z opisem funkcji;
   4. Zestaw uczący w rozszerzeniu .csv

**Załącznik 2. – Main.cpp**

#include <iostream>

#include <vector>

#include <conio.h>

#include <windows.h>

#include <cstdlib>

#include <fstream>

#include <string>

#include "Neurone.h"

using namespace std;

void LoadData(double\* tab, double\* tabs)

{

double a = 0.0;

fstream file;

file.open("zestaw.csv", ios::in);

int no\_line = 1;

string line;

int i = 0;

if (file.good() == false)

{

cout << "Error 0" << endl;

exit(0);

}

cout << "DANE UCZĄCE: " << endl;

while (getline(file, line, ';'))

{

a = atoi(line.c\_str());

no\_line++;

if (i % 3 == 2)

{

tabs[i] = a;

cout << tabs[i] << endl;

}

else

{

tab[i] = a;

cout << tab[i] << " ";

}

i++;

}

cout << endl << "++++++++++++++++++++++++++++++" << endl;

file.close();

}

int main()

{

int n = 2; // ILOŚ WEJŚC - ustalamy sami

double teachCoefficient = 1.0; // WSPóŁCZYNNIK NAUKI - ustalamy sami

double Data[40] = { 0 }; // DANE WEJŚCIOWE

double Score[20] = { 0 }; // OCZEKIWANE WYNIKI

double tab[2] = { 0.0,1.0 };

LoadData(Data, Score); // WCZYTANIE DANYCH UCZĄCYCH

Neurone neurone; // TWORZYMY NEURON

neurone.create\_elements(n);

neurone.set\_neurone(); //USTAWIAMY JEGO WAGI NA LOSOWO

neurone.teach\_neurone(teachCoefficient, Data, Score); //NAUKA NEURONU

//cout << endl << neurone.result\_neurone(tab);

system("pause");

return 0;

}

**Załącznik 1. - Neurone.h**

#pragma once

#include <iostream>

#include <vector>

#include <conio.h>

#include <windows.h>

#include <cstdlib>

#include <cstdio>

#include <ctime>

using namespace std;

/// KLASA NEURONU ///

class Neurone

{

private:

vector<double>dendrites; //WEJŚCIA

vector<double>synapses; //WAGI

public:

void create\_elements(int n) //TWORZYMY ZADANĄ LICZBĘ WAG I WEJŚĆ Z POCZĄTKOWĄ WARTOŚCIĄ 0

{

for (int i = 0; i < n; i++)

{

dendrites.push\_back(0);

synapses.push\_back(0);

}

}

void create\_new\_element\_of\_dendrites() //TWORZENIE NOWEGO DENDRYTU DO SYMULATORA

{

dendrites.push\_back(0);

}

void create\_new\_element\_of\_synapses() //TWORZENIE NOWEJ SYNAPSY DO SYMULATORA

{

synapses.push\_back(0);

}

//double membranePotential(double membrane\_potential) //ZWRACA POTENCIAŁ MEMBRANOWY

//{

// return membrane\_potential;

//}

int input\_size() //ZWRACA ILOŚĆ DENDRYT(WEJŚĆ DO NEURONU)

{

return dendrites.size();

}

double get\_input(int index) //ZWRACA WARTOŚĆ WYBRANEGO NUMERU WEJŚCIA

{

return dendrites[index];

}

void set\_input(int index, double value) //USTAWIA WARTOŚĆ WYBRANEGO NUMERU WEJŚCIA

{

dendrites[index] = value;

}

double get\_synapses(int index) //ZWRACA WARTOŚĆ WYBRANEJ WAGI

{

return synapses[index];

}

void set\_synapses(int index, double value) //USTAWIA WARTOŚĆ WYBRANEJ WAGI

{

synapses[index] = value;

}

double processCell(int index) //MNOŻENIE ODPOWIEDNIEJ DENDRYTY PRZES ODPOWIADAJĄCĄ JEJ WAGĘ

{

return dendrites[index] \* synapses[index];

}

double getMembranePotential() //OBLICZANIE SUMY MEMBRANOWEJ

{

if (input\_size() == 0) //JEŚLI WEJŚCIA NIE ISTNIEJĄ

return -1;

else

{

double sum = 0.0;

for (int i = 0; i<input\_size(); i++)

{

sum += processCell(i);

}

return sum;

}

}

//long double power(int p) //POTĘGOWANIE NIEZBĘDNE DO WLICZNIA FUNKCJI AKTYWACJI UNIPOLARNEJ

//{

// const double w = getMembranePotential();

// double n = 1;

// for (int i = 0; i<w; i++)

// {

// n \*= p;

// }

// long double x = getMembranePotential() / n; //BO POTENGUJEMY PRZEZ -5 WIĘDZ DZIELIMY

// return x;

//}

//double function\_use() //FUNKCJA AKTYWACJI

//{

// if (input\_size() == 0) //JEŚLI WEJŚCIA NIE ISTNIEJĄ

// return -1;

// else

// {

// float e = 2.71; //WARTOŚĆ LOGARYTMU

// float f = (1 / (1 + (power(e)))); //AKTYWACJA UNIPOLARNA

// return f;

// }

//}

double function\_use(double sum) //FUNKCJA AKTYWACJI

{

if (input\_size() == 0) //JEŚLI WEJŚCIA NIE ISTNIEJĄ

return -1.0;

else

{

if (sum >= 0.0) return 1.0;

else return 0.0;

}

}

void set\_neurone() // USTAWIA WARTOŚCI NEURONU NA LOSOWE

{

srand(time(NULL));

for (int i = 0; i < input\_size(); i++)

{

set\_synapses(i, (rand() % 100) / 100.0 - 0.5);

//cout << get\_synapses(i)<<" ";

}

//cout << endl<<endl;

}

double result\_neurone(double\* valuet) // UCZENIE NEURONU

{

double sum = 0.0; // SUMA MEMBRANOWA

double result = 0.0; // WYNIK NEURONU

for (int i = 0; i < input\_size(); i++)

{

set\_input(i, valuet[i]);

}

sum = getMembranePotential();

//cout << function\_use(sum) << endl;

return result = function\_use(sum);

// sum;

}

void teach\_neurone(double teachCoefficient, double\* value, double\* score)

{

double result = 0.0; // WYNIK NEURONU

double epsilon = 0.0; // BŁĄD

double\* valuet = new double[input\_size()];

double error = 0.0;

do

{

error = 0;

for (int i = 0; i < 40; i = i + input\_size())

{

valuet[0] = value[i];

valuet[1] = value[i + 1];

result = getMembranePotential();

epsilon = score[i / input\_size()] - result;

//cout << epsilon << endl;

error = (epsilon\*epsilon) + error;

for (int j = 0; j < input\_size(); j++)

{

set\_synapses(j, get\_synapses(j) - epsilon\*teachCoefficient\*value[j]);

}

}

error = error / 2;

cout << error << endl;

} while (error > 0.4);

}

};

**Załącznik 4. - Program MatLab**

n = newp([0 1,0 1]), 1,'hardlim');

P = [0 0 1 1 ;

0 1 0 1];

L = [0 0 0 1];

plotpv(P,L);

Y = sim(n, P);

n = train(n, P, L);

plotpc(n.iw{1,1}, n.b{1})

Y = sim(n, P);

testPer = randi([0 1],2,randi(5));

rezultat = sim(n, testPer);

**Załącznik 5. - Zestaw\_Uczący.csv**

0;1;0;

1;1;1;

0;0;0;

0;0;0;

1;1;1;

0;0;0;

0;0;0;

0;1;0;

1;1;1;

1;0;0;

0;0;0;

1;1;1;

1;0;0;

0;0;0;

0;0;0;

0;1;0;

1;1;1;

0;0;0;

1;0;0;

1;1;1