Jacek Pozowski

Scenariusz 1

Temat ćwiczenia : Budowa i działanie perceptronu

1.​ ​Cel​ ​ćwiczenia:

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

2. Wstęp teoretyczny:

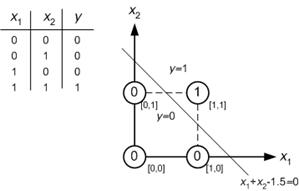
**Neuron McCullocha-Pittsa**  – jeden z matematycznych modeli [neuronu](https://pl.wikipedia.org/wiki/Neuron). Posiada wiele wejść i jedno wyjście. Każdemu z wejść przyporządkowana jest liczba rzeczywista - waga wejścia.

Prosty perceptron jest zwykłym modelem McCullocha-Pittsa o odpowiednio przyjętej strategii uczenia. Wykorzystując tzw. metodę perceptronu dobieramy losowe wagi dla danych wejściowych.

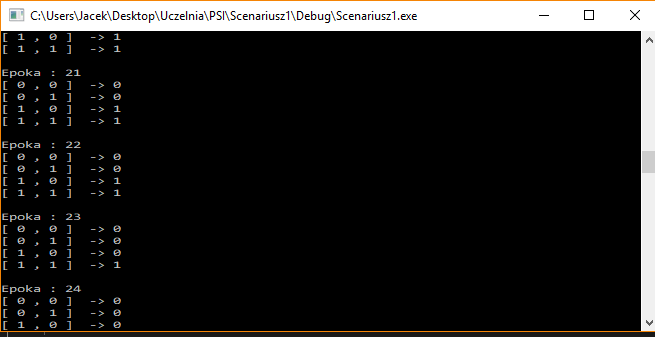
Następnie obliczamy sygnał wyjściowy y i porównujemy wynik z oczekiwanym.

- jeżeli wynik jest zgodny, wagi pozostają bez zmian

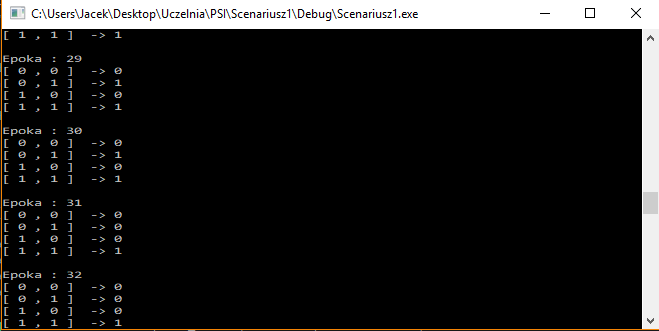
-jeżeli wynik oczekiwany jest różny waga zostaje zmieniona o współczynnik uczenia przemnożony przez różnice wyniku oczekiwanego i wyliczonego(jeżeli oczekujemy 1 a wynikiem jest 0

\begin{figure}
\centerline {\epsfysize=2.0in \epsfbox{./figures/figLTU.epsi}}\end{figure}

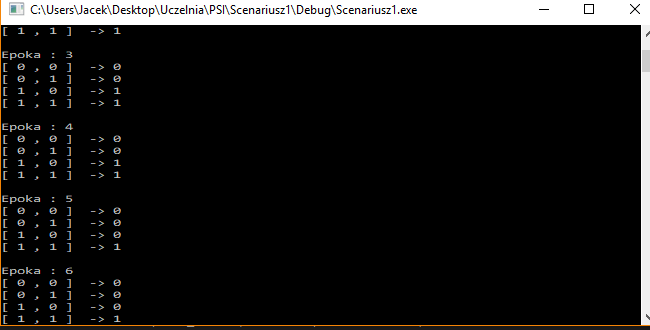
Training rate = 0.01 Liczba potrzebnych epok - 23



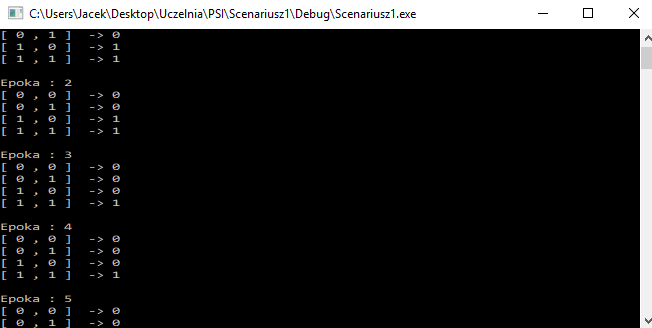
Training rate = 0.02 Liczba potrzebnych epok - 31



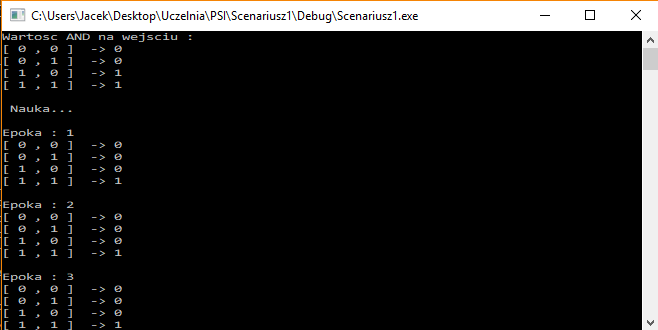
Training rate = 0.05 Liczba potrzebnych epok -5



Training rate = 0.1 Liczba potrzebnych epok -3

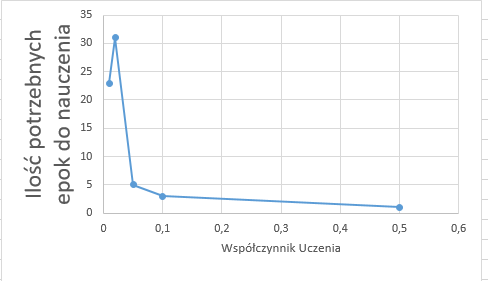


Training rate = 0.1 Liczba potrzebnych epok - 1



Wykresy Ilości potrzebnych epok w zależności od Danych uczących:

Wykres zależności ilości epok od współczynnika uczenia



**Analiza:**

Na podstawie otrzymanych wyników można stwierdzić, że szybkość uczenia zależna jest od współczynnika uczenia oraz liczby danych. Sama efektywność uczenia znacząco zwiększa się dla małego współczynnika uczenia oraz większej ilości danych uczących- w ten sposób otrzymujemy większy stosunek poprawnych odpowiedzi podczas dania do nauki czterech danych uczących. Można również zaobserwować, że gdy wybierze się skrajną wartość tylko jednego z nich (współczynnik uczenia lub liczba danych uczących) to efektywność już nie jest stosunkowo tak dobra. Dla czterech danych uczących niezależnie od współczynnika uczenia mamy praktycznie od razu poprawnie działający perceptron, który nie popełniał błędów w kolejnych testach.

Wnioski:

* Skuteczność procesu uczenia zależy od liczby danych uczących, które zostaną dostarczone do naszego programu. Kiedy ich ilość rośnie wtedy proces ten wykazuje większą skuteczność. Natomiast kiedy liczba ta będzie za mała to perceptron nie będzie działał poprawnie i nie będzie w stanie przeprowadzić udanej nauki
* Kolejnym aspektem wpływającym na skuteczność jest współczynnik uczenia(TRAINING RATE). Wraz z jego wzrostem proces uczenia jest poprawniejszy.
* Możliwe jest takie manipulowanie współczynnikiem uczenia, aby nie było potrzebna duża liczba danych uczących, co za tym idzie proces uczenia przebiegał by szybciej a jego efektywność ciągle byłaby zachowana.
* Jednowarstwowe sieci neuronowe można stosować do wykonywania prostych funkcji logicznych. Po odpowiednio przeprowadzonym procesie uczenia otrzymano bezbłędnie działający perceptron wykonujący funkcję logiczną AND dwóch zmiennych.
* Ostatnią rzeczą mającą wpływ na szybkość uczenia są przyjęte dane testowe. Przez to że się różnią dla kroku 0.01 nauka była szybsza niż dla 0.02

**Listing:**

**„perceptron.cpp”**

#include <stdlib.h>

#include <time.h>

#include "perceptron.h"

//Tworzenie nowego perceptronu

Perceptron \* Perceptron\_create(unsigned inputs, double trainingRate)

{

unsigned i;

srand(time(NULL));

Perceptron \*newOne = (Perceptron\*)malloc(sizeof(Perceptron));

newOne->inputs\_ = inputs;

newOne->trainingRate\_ = trainingRate;

newOne->weights\_ = (double\*)malloc(newOne->inputs\_ \* sizeof(double));

for (i = 0; i < newOne->inputs\_; i++) {

newOne->weights\_[i] = randValue();

}

newOne->threshold\_ = 1;

return newOne;

}

//funkcja ucząca

void learn(Perceptron \*perceptron, const double inputs[], int expectedResult) {

int result = getReasult(perceptron, inputs);

if (result == expectedResult) {

return;

}

changeWeight(perceptron, result, expectedResult, inputs);

}

//Funkcja porównująca wartość wynikową z progową(Funkcja aktywacji)

int getReasult(const Perceptron \*perceptron, const double inputs[]) {

if (getValue(perceptron, inputs) >= perceptron->threshold\_)

return 1;

else

return 0;

}

//pobieranie wag

const double \*getWeight(const Perceptron \*perceptron) {

return perceptron->weights\_;

}

//Funkcja sumująca

double getValue(const Perceptron \*perceptron, const double inputs[]) {

unsigned i;

double ans = 0;

for (i = 0; i < perceptron->inputs\_; i++) {

ans += perceptron->weights\_[i] \* inputs[i];

}

return ans;

}

//Ustawianie wag

void setWeight(Perceptron \*perceptron, const double \*weights) {

unsigned i;

for (i = 0; i < perceptron->inputs\_; i++) {

perceptron->weights\_[i] = weights[i];

}

}

//zmiana wag

void changeWeight(Perceptron \*perceptron, int actualResult, int desiredResult, const double inputs[]) {

unsigned i;

for (i = 0; i < perceptron->inputs\_; i++) {

perceptron->weights\_[i] += perceptron->trainingRate\_ \* (desiredResult - actualResult)\*inputs[i] ;

}

}

//Losowanie double z przedziału <0.1>

double randValue() {

double randValue = 0;

randValue = (double)rand()/(RAND\_MAX);

return randValue;

}

„Źródło.cpp”

#include <stdio.h>

#include "perceptron.h"

const int iteration =40, numberOfImputs = 2;

const double learnRate = 0.5;

void wait()

{

while (true)

{

}

}

int main() {

//tworzenie funkcji logicznej

double case1[] = { 0, 0 };

double case2[] = { 0, 1 };

double case3[] = { 1, 0 };

double case4[] = { 1, 1 };

Perceptron \*AndGate = Perceptron\_create(numberOfImputs, learnRate);

//wartość na wejściu i wynik przed nauczaniem

printf("Wartosc AND na wejsciu :\n");

printf("[ 0 , 0 ] -> %d\n", getReasult(AndGate, case1));

printf("[ 0 , 1 ] -> %d\n", getReasult(AndGate, case2));

printf("[ 1 , 0 ] -> %d\n", getReasult(AndGate, case3));

printf("[ 1 , 1 ] -> %d\n", getReasult(AndGate, case4));

printf("\n Nauka... \n\n");

//nauczanie i wyświetlenie wyników

for (int i = 0; i < iteration; i++) {

printf("Epoka : %i \n", i + 1);

learn(AndGate, case1, 0);

printf("[ 0 , 0 ] -> %d\n", getReasult(AndGate, case1));

learn(AndGate, case2, 0);

printf("[ 0 , 1 ] -> %d\n", getReasult(AndGate, case2));

learn(AndGate, case3, 0);

printf("[ 1 , 0 ] -> %d\n", getReasult(AndGate, case3));

learn(AndGate, case4, 1);

printf("[ 1 , 1 ] -> %d\n", getReasult(AndGate, case4));

printf("\n");

}

wait();

return 0;

}