**Scenariusz 2**

Białek Tomasz, gr. 1

**Temat ćwiczenia:** Budowa i działanie sieci jednowarstwowej

**1. Cel ćwiczenia**

Celem ćwiczenia jest poznanie budowy i działanie jednowarstwowych sieci neuronowych orazuczenie rozpoznawania wielkości liter.

**2. Opis budowy sieci i algorytmów uczenia.**

Celem budowanej sieci jest rozpoznawanie wielkości liter. Do tego celu utworzony został zestaw liter (10 dużych i 10 małych), które są reprezentowane w postaci dwuwymiarowej tablicy 5x7 pikseli dla jednej litery.

Ćwiczenie zostało wykonane w języku C++ (własna implementacja), w którym zastosowano uczenie nadzorowane (z pomocą nauczyciela). Funkcja aktywacji jest funkcją sigmoidalną unipolarną (stały współczynnik β = 0,5).

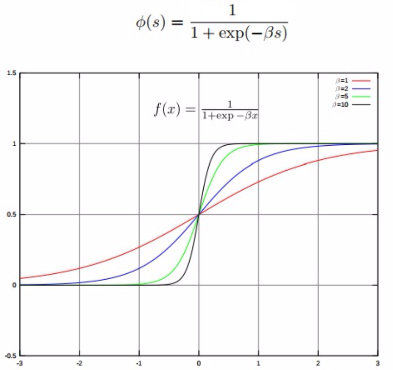
Duże litery wykorzystane w ćwiczeniu: ABCDEFGHIJ

Małe litery wykorzystane w ćwiczeniu: abcdefghij

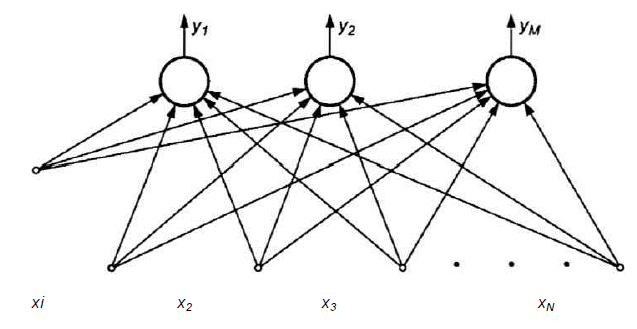
Tab. 1. Litery i ich reprezentacja

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 1 1 1 0  1 0 0 0 1  1 0 0 0 1  1 1 1 1 1  1 0 0 0 1  1 0 0 0 1  1 0 0 0 1 |  | 0 0 0 0 0  0 0 0 0 0  0 1 1 1 0  0 0 0 0 1  0 1 1 1 1  1 0 0 0 1  0 1 1 1 1 |
|  | 1 1 1 1 0  1 0 0 0 1  1 0 0 0 1  1 1 1 1 0  1 0 0 0 1  1 0 0 0 1  1 1 1 1 0 |  | 0 0 0 0 0  0 0 0 0 0  1 0 0 0 0  1 0 0 0 0  1 1 1 0 0  1 0 0 1 0  1 1 1 0 0 |
|  | 0 1 1 1 0  1 0 0 0 1  1 0 0 0 0  1 0 0 0 0  1 0 0 0 0  1 0 0 0 1  0 1 1 1 0 |  | 0 0 0 0 0  0 0 0 0 0  0 1 1 1 0  1 0 0 0 1  1 0 0 0 0  1 0 0 0 1  0 1 1 1 0 |
|  | 1 1 1 1 0  1 0 0 0 1  1 0 0 0 1  1 0 0 0 1  1 0 0 0 1  1 0 0 0 1  1 1 1 1 0 |  | 0 0 0 0 0  0 0 0 0 0  0 0 0 0 1  0 0 0 0 1  0 0 1 1 1  0 1 0 0 1  0 0 1 1 1 |
|  | 1 1 1 1 1  1 0 0 0 0  1 0 0 0 0  1 1 1 1 0  1 0 0 0 0  1 0 0 0 0  1 1 1 1 1 |  | 0 0 0 0 0  0 0 0 0 0  0 1 1 1 0  1 0 0 0 1  1 1 1 1 0  1 0 0 0 0  0 1 1 1 0 |
|  | 1 1 1 1 1  1 0 0 0 0  1 0 0 0 0  1 1 1 1 0  1 0 0 0 0  1 0 0 0 0  1 0 0 0 0 |  | 0 0 0 0 0  0 0 0 0 0  0 1 1 1 0  0 1 0 0 0  1 1 1 0 0  0 1 0 0 0  0 1 0 0 0 |
|  | 1 1 1 1 1  1 0 0 0 1  1 0 0 0 0  1 0 1 1 1  1 0 0 0 1  1 0 0 0 1  0 1 1 1 0 |  | 0 0 0 0 0  0 0 0 0 0  0 1 1 1 1  1 0 0 0 1  0 1 1 1 1  0 0 0 0 1  1 1 1 1 0 |
|  | 1 0 0 0 1  1 0 0 0 1  1 0 0 0 1  1 1 1 1 1  1 0 0 0 1  1 0 0 0 1  1 0 0 0 1 |  | 0 0 0 0 0  0 0 0 0 0  1 0 0 0 0  1 0 0 0 0  1 1 1 1 0  1 0 0 0 1  1 0 0 0 1 |
|  | 0 1 1 1 0  0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 1 1 1 0 |  | 0 0 0 0 0  0 0 0 0 0  0 0 1 0 0  0 0 0 0 0  0 0 1 0 0  0 0 1 0 0  0 0 1 1 0 |
|  | 1 1 1 1 1  0 0 0 0 1  0 0 0 0 1  0 0 0 0 1  0 0 0 0 1  1 0 0 0 1  0 1 1 1 0 |  | 0 0 0 0 0  0 0 0 0 0  0 0 0 0 1  0 0 1 1 1  0 0 0 0 1  0 1 0 0 1  0 0 1 1 0 |

Ciąg przeznaczony do nauki składa się z 20 wektorów (tablic) zawierających 35 pozycje oraz ciąg wyjściowy (zawierający 1 – duża litera, 0 – mała litera). Sieć neuronowa używa funkcji sigmoidalnej unipolarnej ze współczynnikiem β = 0,5 jako funkcji aktywacji.



Rys. 1 Funkcja sigmoidalna unipolarna z różnymi współczynnikami β



Rys. 2 Schemat sieci neuronowej sigmoidalnej jednowarstwowej

Sieć wykorzystuje unipolarną, logistyczną(sigmoidalną) funkcję aktywacji, natomiast proces uczenia przebiega według następującego schematu w danej epoce uczenia:  
1. Wybór początkowych wartości wag, jako niewielkich liczb losowych (z przedziału 0 – 1)  
2. Ustawienie sumy błędu kwadratowego na zero (E = 0)  
3. Obliczana jest suma ilorazów sygnałów wejściowych i wag-   
4.

a) Obliczana jest wartość wyjściowa za pomocą unipolarnej, sigmoidalnej funkcji aktywacji według wzoru:

gdzie jest z zakresu (0, 1>

b) Obliczana jest wartość wyjściowa za pomocą unipolarnej, progowej funkcji aktywacji według wzoru:



5. Uaktualnienie wag za pomocą reguły delty według zależności:

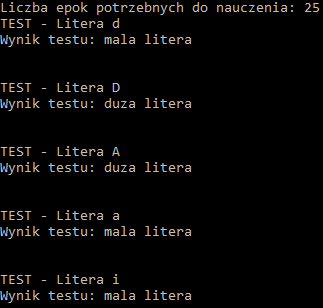
6. Obliczenie łącznego błędu epoki:

7. Uczenie uważamy za zakończone, jeżeli E < Emax. W przeciwnym razie rozpoczynamy nową epokę uczenia.

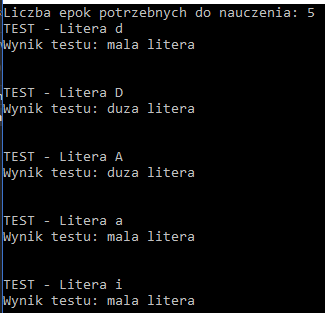
**3. Otrzymane wyniki i ich analiza**

*Przykładowe działanie programu*

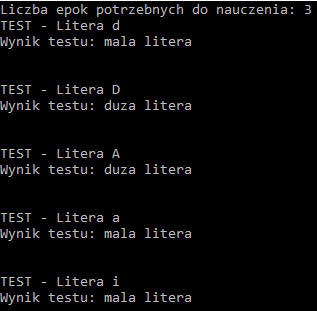
Współczynnik uczenia = 0.01



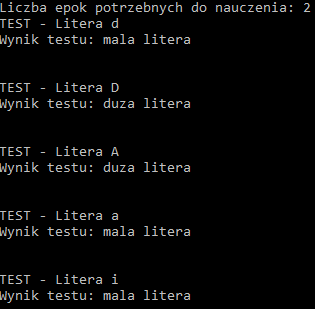
Współczynnik uczenia = 0.05



Współczynnik uczenia = 0.1



Współczynnik uczenia = 0.5



*Analiza*

Proces uczenia sieci jednowarstwowej przebiega analogicznie do uczenia perceptronu. Analizując wykres można zauważyć, że im wyższy współczynnik uczenia, tym liczba potrzebnych epok była niższa. Dla współczynnika uczenia > 0.25 sieć została wyćwiczona już po 2-3 epokach. Dla współczynnika uczenia = 0.0001 proces uczenia zajął aż 2400 epok.

Dla przypadku z funkcją progową unipolarną sytuacja dla bardzo małych współczynników uczenia wygląda trochę lepiej, jednakże nadal jest to ponad 1700 epok.

Dla współczynnika uczenia = 0.001 liczba epok uczenia praktycznie się wyrównała.

Analizując powyższy wykres można zauważyć, że na efektywność uczenia sieci ma wpływ funkcja aktywacji. Im mniejsza wartość współczynnika β, tym potrzeba więcej epok nauki. Akceptowalne wartości współczynnika β wynoszą powyżej 0.1, gdzie uczenie przebiega w miarę na tym samym poziomie.

Z powyższego wykresu można odczytać, że współczynnik uczenia silnie wpływa na błąd średniokwadratowy tego uczenia. Im wyższy współczynnik uczenia, tym powstały błąd jest mniejszy.

Dla mniejszych współczynników uczenia błąd jest większy, co może wpływać na samo uczenie się sieci (nauka może trwać dłużej właśnie ze względu na powstały błąd).

Dla przypadku z funkcją progową unipolarną błąd uczenia nie zmniejsza się jak przy funkcji sigmoidalnej.

**4. Podsumowanie**

Skuteczność procesu uczenia zależy od współczynnika uczenia. Wraz z jego wzrostem proces uczenia jest poprawniejszy. Jest to wytłumaczalne z tego względu, że im ta wartość jest większa tym przyrost wag, które na samym początku są niewielkie jest szybszy, więc proces uczenia przebiega szybciej. Sieć jednowarstwowa jest odporna na zaszumienie, jednakże tylko dla kilku bitów. Zbyt duże zaszumienie powoduje błędne odpowiedzi dawane przez sieć. Przy projektowaniu sieci neuronowej trzeba wybrać odpowiedni jej model. Na jej efektywność ma wpływ sama jej struktura, np. funkcja aktywacji. Funkcja sigmoidalna unipolarna powoduje szybsze uczenie sieci niż funkcja progowa unipolarna, a dla samej funkcji sigmoidalnej unipolarnej ogromne znaczenie ma współczynnik β (dla wyższych współczynników - >=0.5 sieć zachowuje się stabilnie i uczy z taką samą efektywnością, jednakże poniżej tych wartości następuje drastyczny spadek tej efektywności). Dla naszego przypadku wartość współczynnika, wynosząca 0.5, była wystarczająca, lecz dla bardziej zaawansowanych sieci współczynnik ten musi być wyższy.

**5. Kod programu**

Perceptron.h

#pragma once

#include <iostream>

#include <ctime>

#include <fstream>

using namespace std;

#define HOW\_MANY\_LETTERS 20

#define BITS\_OF\_ONE\_LETTER 35

class Perceptron

{

double \* weights; //przechowuje wagi dla każdego wejścia perceptronu

double learningRate; //współczynnik uczenia

int inputData[HOW\_MANY\_LETTERS][BITS\_OF\_ONE\_LETTER]; //tablica przechowująca litery i ich reprezentacje bitowe

int expectedResults[HOW\_MANY\_LETTERS]; //tablica zawiera oczekiwane wyniki (1 - duża litera lub 0 - mała litera) dla danych uczących

int numberOfInputs; //ilosc wejść

bool resultOfTest; //przechowuje wynik testu (1 - duża litera lub 0 - mała litera)

public:

void learn1(); //funkcja ucząca <sigmoidalna>

void learn2(); //funkcja ucząca <progowa unipolarna>

void test(int tab[]); //funkcja przeznaczona do testu (tab[] - reprezentacja bitowa litery do testowania)

void readTestData(); //wczytanie danych do nauki z pliku

double getRandomDouble(); // losowanie wag z przedziału <0; 1>

double getSumOfInput(int i); //funkcja sumująca perceptronu (i - indeks litery)

double func(double sum); //funkcja sigmoidalna (przyjmuje getSumOfInput)

double der(double sum); //pochodna func

double func2(double sum); //funkcja progowa unipolarna (przyjmuje getSumOfInput)

Perceptron(unsigned numOfInputs, double trainingRate);

~Perceptron();

};

Perceptron.cpp

#include "Perceptron.h"

//dla kazdej wagi perceptronu będą losowanie liczby z zakresu od 0 do 1

double Perceptron::getRandomDouble()

{

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

return randValue;

}

//oblicza sumę wszytkich wejście\*waga, przyjmuje indeks litery

double Perceptron::getSumOfInput(int i)

{

double sum = 0;

for (int j = 0; j < BITS\_OF\_ONE\_LETTER; j++)

sum += this->weights[j] \* this->inputData[i][j];

return sum;

}

//wczytanie danych uczących z pliku

void Perceptron::readTestData()

{

fstream file;

file.open("data\_for\_learning.txt");

if (!file.good()){

cout<< "--------- I can't open the file with learning data ---------" << endl;

system("PAUSE");

exit(0);

}

//wczytuj z pliku dopóki są dane

while (!file.eof())

for (int i = 0; i < numberOfInputs; i++){ // i oznacza indeks litery

for (int j = 0; j < BITS\_OF\_ONE\_LETTER; j++) // j oznacza ilość bitów na daną literę

file >> this->inputData[i][j]; //wczytywanie do tablicy z wejściami

file >> this->expectedResults[i]; //wczytanie z pliku czy dana litera jest duża (1) lub mała (0)

}

file.close();

}

//ucz się

void Perceptron::learn1()

{

double sumOfInput = 0.; //zmienna przechowująca wynik getSumOfInput(i)

double errorOfOnePerceptron = 0.; //błąd lokalny

double globalError = 0.; //błąd globalny

double threshold = 1.5; //próg określający kiedy sieć jet już nauczona

int epoch = 0; //epoka uczenia się (na start = 0)

double MAPE = 0., MSE = 0.;

do{

globalError = 0.;

for (int i = 0; i < this->numberOfInputs; i++) //zmienna i oznacza indeks litery w tablicy

{

sumOfInput = getSumOfInput(i);

errorOfOnePerceptron = this->expectedResults[i] - func(sumOfInput);

//zmiana wag

for (int j = 0; j < BITS\_OF\_ONE\_LETTER; j++) { //zmienna j oznacza bity danej litery

this->weights[j] += this->learningRate \* errorOfOnePerceptron \* this->inputData[i][j];

//this->weights[j] += this->learningRate \* (this->expectedResults[j] - func2(sumOfInput)) \*this->inputData[i][j];

}

////////////////////////////////////////

globalError += pow(errorOfOnePerceptron, 2);

MSE = pow(globalError, 2) / (BITS\_OF\_ONE\_LETTER);

//MAPE = (globalError \* 10 / BITS\_OF\_ONE\_LETTER);

//cout << " MSE: " << MSE << "\tMAPE: " << MAPE << "%\n";

}

epoch++;

} while (globalError >= threshold); //zapętla uczenie dopóki błąd nie będzie niższy od progu

cout << "Liczba epok potrzebnych do nauczenia: " << epoch << endl;

cout << "MSE: " << globalError << endl;

}

//ucz się

void Perceptron::learn2()

{

double sumOfInput = 0.; //zmienna przechowująca wynik getSumOfInput(i)

double errorOfOnePerceptron = 0.; //błąd lokalny

double globalError = 0.; //błąd globalny

double threshold = 1.5; //próg określający kiedy sieć jet już nauczona

int epoch = 0; //epoka uczenia się (na start = 0)

double MAPE = 0., MSE = 0.;

do {

globalError = 0.;

for (int i = 0; i < this->numberOfInputs; i++) //zmienna i oznacza indeks litery w tablicy

{

sumOfInput = getSumOfInput(i);

errorOfOnePerceptron = this->expectedResults[i] -func2(sumOfInput);

//zmiana wag

for (int j = 0; j < BITS\_OF\_ONE\_LETTER; j++) { //zmienna j oznacza bity danej litery

this->weights[j] += this->learningRate \* errorOfOnePerceptron \* this->inputData[i][j];

//this->weights[j] += this->learningRate \* (this->expectedResults[j] - func2(sumOfInput)) \*this->inputData[i][j];

}

////////////////////////////////////////

globalError += pow(errorOfOnePerceptron, 2);

//MSE = pow(globalError, 2) / (BITS\_OF\_ONE\_LETTER);

//MAPE = (globalError \* 10 / BITS\_OF\_ONE\_LETTER);

//cout << " MSE: " << MSE << "\tMAPE: " << MAPE << "%\n";

}

epoch++;

} while (globalError >= threshold); //zapętla uczenie dopóki błąd nie będzie niższy od progu

cout << "Liczba epok potrzebnych do nauczenia: " << epoch << endl;

cout << "MSE: " << globalError << endl;

}

//funkcja testująca czy podane bity tworzą małą czy dużą literę

void Perceptron::test(int tab[])

{

double sum = 0.;

double errorOfOnePerceptron = 0.;

double globalError = 0.;

double threshold = 1.5;

double MAPE = 0., MSE = 0.;

for (int i = 0; i < BITS\_OF\_ONE\_LETTER; i++)

{

sum += tab[i] \* this->weights[i]; //oblicza sume dla zadanego elementu

//MAPE += ((fabs(tab[i] \* this->weights[i] - tab[i])\*(tab[i] \* this->weights[i]) \* 100) / (tab[i] \* this->weights[i]);

//MSE += pow((tab[i] \* weights[i]), 2);MSE = pow(globalError,2)/(SIZE);

}

errorOfOnePerceptron = 1 - sum; //błąd lokalny

globalError = pow(2, errorOfOnePerceptron); //błąd globalny

if (globalError >= threshold)

this->resultOfTest = true; //podana litera jest mała

else

this->resultOfTest = false; //podana litera jest duża

//cout << "MAPE: " <<MAPE << endl;

//cout << "MSE: " << MSE << endl;

//wypisanie w konsoli ostatecznego wyniku testu

if (this->resultOfTest == true)

cout << "Wynik testu: mala litera" << endl << endl;

else

cout << "Wynik testu: duza litera" << endl << endl;

}

Perceptron::Perceptron(unsigned numberOfInputs, double learningRate)

{

srand(time(NULL));

this->numberOfInputs = numberOfInputs;

this->learningRate = learningRate;

//stworzenie dynamicznej tablicy wag i losowanie wag dla każdego wejścia

this->weights = new double[BITS\_OF\_ONE\_LETTER];

for (int i = 0; i < BITS\_OF\_ONE\_LETTER; i++)

this->weights[i] = getRandomDouble();

//wczytanie danych z pliku tekstowego

readTestData();

}

Perceptron::~Perceptron()

{

delete weights;

}

//wykorzystywane przy aktualizacji wag <funkcja sigmoidalna>

double Perceptron::func(double sum)

{

return 1 / (1 + exp(-0.5\*sum)); //współczynnik beta = 0.5

}

//wykorzystywane przy aktualizacji wag <funkcja progowa unipolarna>

double Perceptron::func2(double sum)

{

return (sum > 0)? 1 : 0;

}

//pochodna func

double Perceptron::der(double sum)

{

return (0.5\*exp(-0.5\*sum))/(pow(exp(-0.5\*sum)+1, 2));

}

Source.cpp

#include "Perceptron.h"

# define A 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1

# define B 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0

# define C 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0

# define D 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0

# define E 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1

# define F 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0

# define G 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0

# define H 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1

# define I 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0

# define J 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0

# define a 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1

# define b 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0

# define c 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0

# define d 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1

# define e 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0

# define f 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0

# define g 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0

# define h 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1

# define i 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0

# define j 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0

int main()

{

int numberOfInputs = HOW\_MANY\_LETTERS;

double learningRate = 0.25;

//pierwszy sposób

Perceptron perceptron1(numberOfInputs, learningRate);

int testLetter[BITS\_OF\_ONE\_LETTER] = { d }; // bity litery do testowania

int testLetter2[BITS\_OF\_ONE\_LETTER] = { D };

int testLetter3[BITS\_OF\_ONE\_LETTER] = { j };

int testLetter4[BITS\_OF\_ONE\_LETTER] = { a };

int testLetter5[BITS\_OF\_ONE\_LETTER] = { J };

cout << "Pierwszy sposob - funkcja sigmoidalna" << endl;

perceptron1.learn1();

cout << "TEST - Litera d" << endl;

perceptron1.test(testLetter);

cout << "TEST - Litera D" << endl;

perceptron1.test(testLetter2);

cout << "TEST - Litera j" << endl;

perceptron1.test(testLetter3);

cout << "TEST - Litera a" << endl;

perceptron1.test(testLetter4);

cout << "TEST - Litera J" << endl;

perceptron1.test(testLetter5);

//drugi sposób

Perceptron perceptron2 = Perceptron(numberOfInputs, learningRate);

cout << "Drugi sposob - funkcja progowa unipolarna " << endl;

perceptron2.learn2();

cout << "TEST - Litera d" << endl;

perceptron2.test(testLetter);

cout << "TEST - Litera D" << endl;

perceptron2.test(testLetter2);

cout << "TEST - Litera j" << endl;

perceptron2.test(testLetter3);

cout << "TEST - Litera a" << endl;

perceptron2.test(testLetter4);

cout << "TEST - Litera J" << endl;

perceptron2.test(testLetter5);

perceptron1.~Perceptron();

perceptron2.~Perceptron();

system("PAUSE");

return 0;

}

data\_for\_learning.txt

0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1

1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1

0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0 1

1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1

1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1

1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1

1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 1 0 1 1 1 0 1

1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1

0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0 1

1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 1

0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 0 1 0 1 1 1 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 0

0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 0

0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 0

0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0