**Jacek Pozowski**

Scenariusz 2

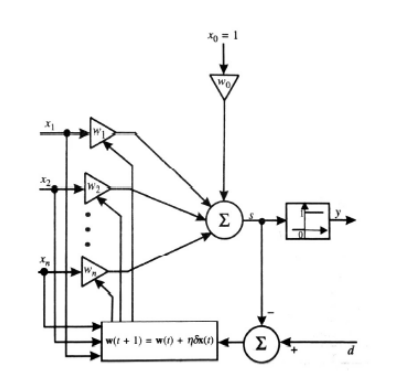
**Cel ćwiczenia:**

Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

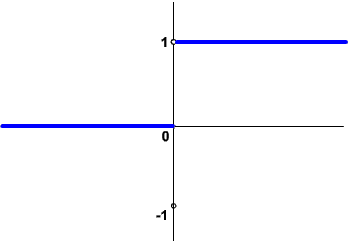
**Opis budowy sieci i algorytmów uczenia.**

Zadaniem algorytmu jest rozpoznawanie wielkości liter A-J oraz a-j reprezentowanych w postaci tablic 5x7 pikseli. Wykorzystane modele sieci to Adaline oraz DeltaRule.

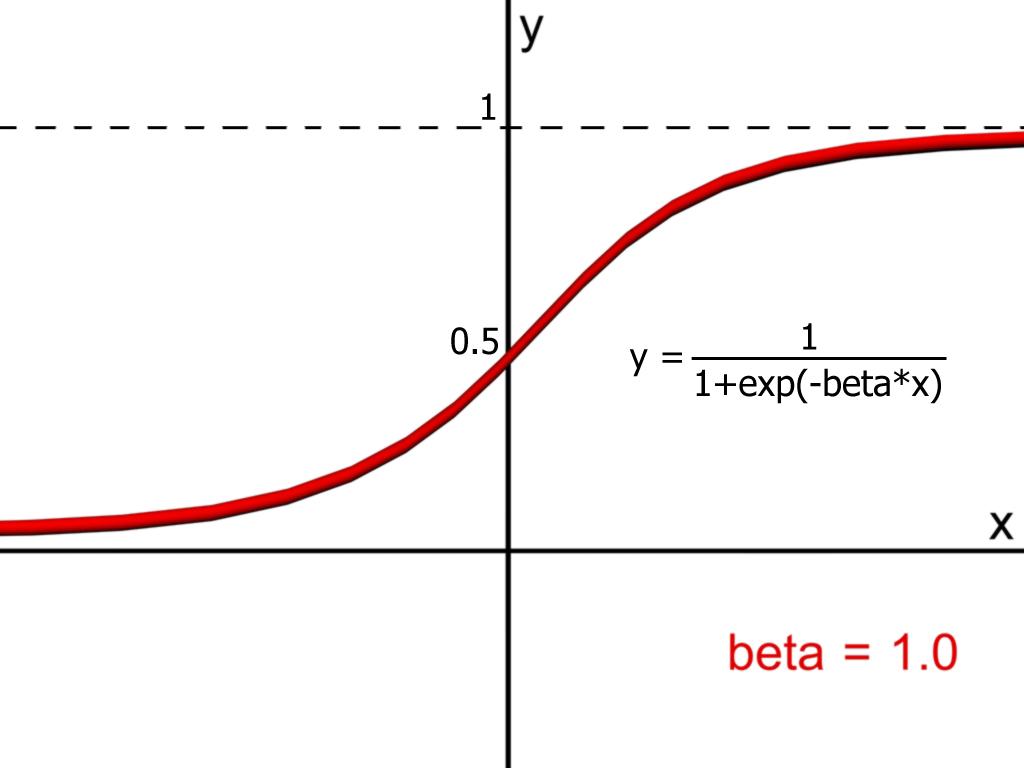
**ADALINE** (**Adaptive Linear Neuron)** - budowa tego neuronu jest bardzo podobna do modelu perceptronu, a jedyna różnica dotyczy algorytmu uczenia. Sygnał wyznacza się w ten sam sposób, co w przypadku uczenia perceptronu. Jednak w przypadku neuronu typu Adaline porównuje się sygnał wzorcowy d z sygnałem s, na wyjściu części liniowej neuronu (nie uwzględnia się funkcji aktywacji).



Źrudło: <http://pracownik.kul.pl/files/31717/public/Model_Adaline.pdf>



Funkcja Unipolarna

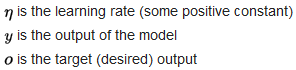


Sigmoidalna Funkcja Unipolarna

Uaktualnienie wag:

a) dla Adaline:

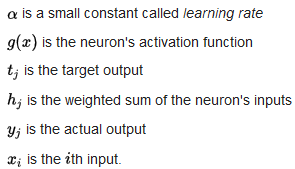
, gdzie:



Źródło : https://en.wikipedia.org/wiki/ADALINE

b) dla DeltaRule:

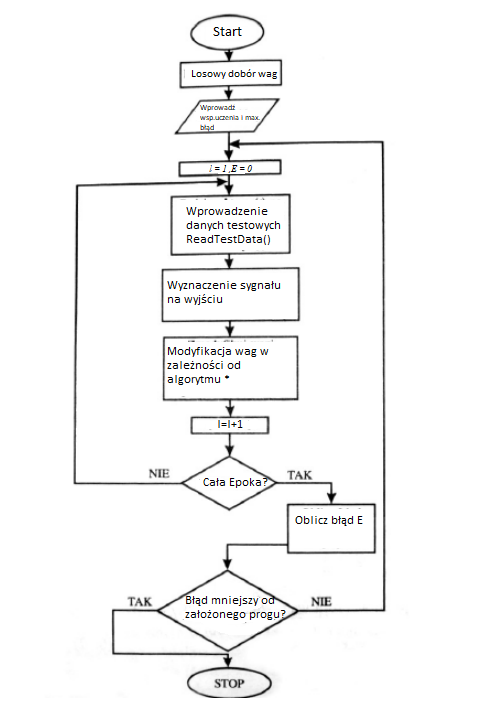
, gdzie:



Źródło : https://en.wikipedia.org/wiki/Delta\_rule

Model DeltaRule ma analogiczną budowę do modelu Adaline, jednakże funkcją aktywacji jest funkcja sigmoidalna, a przy aktualizacji wag uwzględnia się pochodną tejże funkcji, obliczaną w funkcji derivativeActivationFunction().

Schemat blokowy:



**Wyniki:**

|  |  |  |
| --- | --- | --- |
| **Rate** | **Adaline** | **Delta** |
| 0,001 | 463 | 4039 |
| 0,01 | 48 | 405 |
| 0,02 | 26 | 203 |
| 0,05 | 13 | 82 |
| 0,075 | 11 | 55 |
| 0,1 | 11 | 42 |
| 0,25 | ~~101~~ | 17 |
| 0,5 | ~~36~~ | 9 |
| 1 | ~~22~~ | 5 |

W powyższej tabeli przedstawiono wyniki nauczania.

Zielone - najbardziej optymalne

Czerwone – najmniej optymalne

Przekreślone – nauczanie nie przebiegło poprawnie

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Stały współczynnik uczenia h = 0.01 | | | | | |
| Współczynnik  | 0.15 | 0.25 | 0.5 | 0.75 | 1.0 |
| Liczba epok | 20360 | 11782 | 1581 | 656 | 405 |

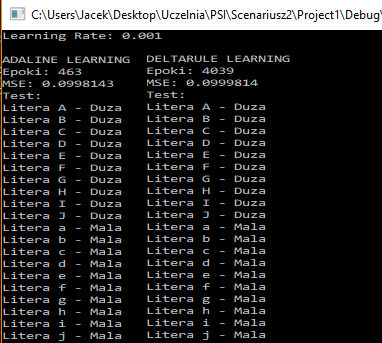
Analizując powyższą tabelę można zauważyć że najbardziej optymalna wartość współczynnika to 1.

Widać również drastyczną różnicę między wynikami dla 0.25 i 0.5 dlatego wartości poniżej 0.5 nie są optymalne.

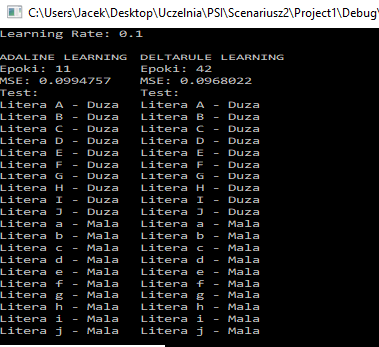
|  |  |  |
| --- | --- | --- |
| **Błędne litery** | | |
| **Emax** | **Adaline** | **Delta** |
| 0,0001 | 0 | 0 |
| 0,001 | 0 | 0 |
| 0,01 | 0 | 0 |
| 0,1 | 0 | 1 |
| 0,25 | 1 | 2 |
| 0,5 | 2 | 2 |
| 0,75 | 2 | 4 |
| 1 | 3 | 5 |

Dane z tabeli pokazują że oba algorytmy działają bezbłędnie dla Emax<=0.01. Przy wartości 0.1 tylko DeltaRule nie działa poprawnie. Natomiast dla większych wartości obie funkcje przestają uczyć poprawnie. Wraz z wzrostem błędu zwiększa się ilość błędnie rozpoznanych liter.

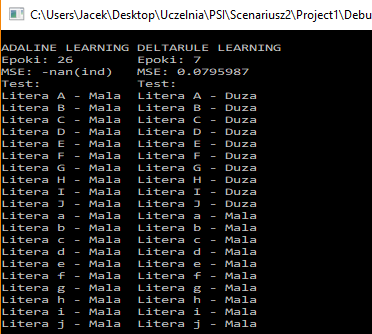
Najbardziej nieoptymalny przypadek



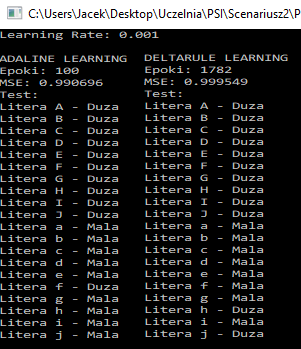
Najbardziej Optymalny przypadek



Błąd Adaline



Wynik przy zwiększonym Emax



Proces uczenia sieci jednowarstwowej przebiega analogicznie do uczenia perceptronu. Analizując wykres można zauważyć, że do osiągnięcia pewnej wartości im wyższy współczynnik uczenia, tym liczba potrzebnych epok była niższa. Dla współczynnika 0.1 sieć została wyćwiczona już po 11 epokach. Dla współczynnika uczenia = 0.001 przy algorytmie DeltaRule proces uczenia zajął ponad 4000 epok gdy Adaline potrzebowało prawie dziewięć razy mniej.

W przypadku manipulacji Emax możemy zauważyć że błąd dla Adaline zawsze był mniejszy bądź równy błędowi Delta.

Możemy zauważyć że po przekroczeniu wartości 0.5 ilość epok przestaje się tak dynamicznie zmieniać.

**Wnioski:**

* Skuteczność procesu uczenia zależy od współczynnika uczenia. Wzrost współczynnika przyspiesza proces uczenia.
* W przypadku Adaline po przekroczeniu pewnej wartości współczynnika uczenia przestaje ono przebiegać poprawnie.
* Przy projektowaniu sieci neuronowej trzeba wybrać odpowiedni jej model. Oprócz samego modelu na jej efektywność ma wpływ sama jej struktura, np. zastosowana funkcja aktywacji.
* Model Adaline pozwala na o wiele sprawniejsze uczenie sieci w porównaniu do modelu DeltaRule natomiast w przeciwieństwie do tego drugiego przestaje działać dla większych współczynników.
* W modelu DeltaRule wpływ na efektywność uczenia sieci ma również współczynnik β w sigmoidalnej funkcji aktywacji. Dla rozważanego przez nas problemu jego akceptowalna wartość wynosiła powyżej 0.5 lecz najlepsze wyniki zostały osiągnięte dla β=1.

**Listing:**

**„Adaline.h”**

#include <iostream>

#include <ctime>

#include <fstream>

using namespace std;

#define LETTER\_AMOUNT 20

#define PIXEL\_AMOUNT 35

class Adaline {

public:

double \* weights; //przechowuje wagi liter

double learningRate; //wspolczynnik uczenia

double EMax; //maksymalna wartosc bledu

int inputData[LETTER\_AMOUNT][PIXEL\_AMOUNT]; //tablica przechowująca litery [20] i ich reprezentancje bitowe [35]

int expectedResults[LETTER\_AMOUNT]; //tablica przechowująca oczekiwane wyniki [1 - duża liczba, 0 - mała liczba]

int numberOfSets; //liczba liter = 20

int numberOfWeights; //liczba wag = 35

double error; //blad globalny (po podniesieniu do kwadratu, podzieleniu przez 2 daje MSE)

double delta; //jest to wyliczona roznica (dodawana do wag)

int setTest[LETTER\_AMOUNT][PIXEL\_AMOUNT] = { //tablica danych testowych

{ 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0 }

};

//char'owe reprezentacje tablicy bitow

char setTestLetters[20] = { 'A','B','C','D','E','F','G','H','I','J','a','b','c','d','e','f','g','h','i','j' };

Adaline(double \_learningRate); //konstruktor (przyjmuje learningRate)

void readTestData(); //funkcja wczytuje dane do nauki z pliku

double getSum(int letter[], double \*weights); //funkcja zwraca sume wejscia danej litery

double getRandomDouble(); //funkcja losuje double z przedzialu <0; 1>

bool activationFunction(double sum); //funkcja aktywacji - funkcja progowa unipolarna

void learn(); //funkcja uczaca

void test(); //funkcja testujaca

};

„Adaline.cpp”

#include "Adaline.h"

Adaline::Adaline(double \_learningRate)

{

delta = 0;

numberOfWeights = PIXEL\_AMOUNT;

numberOfSets = LETTER\_AMOUNT;

learningRate = \_learningRate;

EMax = 0.1;

error = 0.0;

weights = new double[numberOfWeights];

for (int i = 0; i < PIXEL\_AMOUNT; i++)

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

readTestData();//wczytanie danych uczących

}

//wczytanie danych uczących z pliku

void Adaline::readTestData()

{

fstream file;

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

if (!file.good()) {

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

system("PAUSE");

exit(0);

}

//wczytuj z pliku dopóki są dane

while (!file.eof())

for (int i = 0; i < LETTER\_AMOUNT; i++) { // i oznacza indeks litery

for (int j = 0; j < PIXEL\_AMOUNT; j++) // j oznacza ilosc pikseli na daną litere

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

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

}

file.close();

}

//losuje double'a z przedzialu <0;1>

double Adaline::getRandomDouble()

{

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

return randValue;

}

//funkcja aktywacji - funkcja progowa unipolarna

bool Adaline::activationFunction(double sum)

{

if (sum > 0.5)

return true;

else

return false;

}

//zwraca sume danego wejscia

double Adaline::getSum(int letter[], double \* weights)

{

double sum = 0.0;

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

sum += letter[i] \* weights[i];

return sum;

}

//funkcja uczaca

void Adaline::learn()

{

cout << endl << "ADALINE LEARNING" << endl;

bool acceptableError = false; //zmienna, stwierdzajaca czy blad jest mozliwy do zaakceptowania

int epoch = 0; //numer epoki

/\*

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

cout << "Weights are: w" << i+1 << " = " << weights[i] << endl;;

cout << endl;

\*/

do {

epoch++; // zwiekszenie epoki

error = 0.0; //zerowanie glownego bledu w celu sprawdzenia bledow podczas jednej iteracji

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

//obliczanie roznicy pomiedzy wynikiem oczekiwanym a wynikiem otrzymanym

delta = expectedResults[i] - getSum(inputData[i], weights);

//aktualizowanie wag

for (int j = 0; j < numberOfWeights; j++)

weights[j] += learningRate \* delta\*inputData[i][j];

//aktualizowanie bledu glownego

error += delta \* delta;

}

error /= 2;

//porownywanie bledu z progiem

if (error>EMax) { //EMax

acceptableError = false;

}

else {

acceptableError = true;

}

} while (!acceptableError);

cout << "Epoki: " << epoch << endl;

cout << "MSE: " << error << endl;

}

//funkcja testujaca

void Adaline::test()

{

cout << "Test:" << endl;

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

cout << "Litera " << setTestLetters[i] << " - ";

if (activationFunction(getSum(setTest[i], weights))) {

cout << "Duza";

}

else {

cout << "Mala";

}

cout << endl;

}

}

„DeltaRule.h”

#include <iostream>

#include <ctime>

#include <fstream>

using namespace std;

#define LETTER\_AMOUNT 20

#define PIXEL\_AMOUNT 35

class DeltaRule {

public:

double \* weights; //tablica wag

double learningRate; //wspolczynnik uczenia

double EMax; //maksymalna wartosc bledu

int inputData[LETTER\_AMOUNT][PIXEL\_AMOUNT];

int expectedResults[LETTER\_AMOUNT]; //tablica oczekiwanych wynikow

int numberOfSets; //liczba liter

int numberOfWeights; //liczba wag

double error; //blad globalny (po podniesieniu do kwadratu, podzieleniu przez 2 daje MSE)

double delta; //jest to wyliczona roznica(dodawana do wag)

double output; //przechowuje aktualna wartosc (porownywana z wartoscia oczekiwana)

int setTest[LETTER\_AMOUNT][PIXEL\_AMOUNT] = { //tablica danych testowych

{ 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0 }

};

//char'owe reprezentacje tablicy bitow

char setTestLetters[20] = { 'A','B','C','D','E','F','G','H','I','J','a','b','c','d','e','f','g','h','i','j' };

DeltaRule(double \_learningRate); //konstruktor

void readTestData(); //wczytuje z pliku dane uczące

double getRandomDouble(); //losuje double z przedzialu <0; 1>

double activationFunction(double sum); //funkcja aktywacji - funkcja sigmoidalna

double derivativeActivationFunction(double sum); //pochodna funkcji aktywacji

double getSum(int letter[], double \* weights); //zwraca sume wejscia danej litery

void learn(); //funkcja uczaca

void test(); //funkcja testujaca

};

„DeltaRule.cpp”

#include "DeltaRule.h"

//konstruktor

DeltaRule::DeltaRule(double \_learningRate) {

delta = 0;

numberOfWeights = PIXEL\_AMOUNT;

numberOfSets = LETTER\_AMOUNT;

learningRate = \_learningRate;

EMax = 0.1;

error = 0;

output = 0;

weights = new double[numberOfWeights];

for (int i = 0; i < PIXEL\_AMOUNT; i++)

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

//wczytanie z pliku danych uczacych

readTestData();

}

//losuje double'a z przedzialu <0;1>

double DeltaRule::getRandomDouble()

{

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

return randValue;

}

//wczytanie danych uczących z pliku

void DeltaRule::readTestData()

{

fstream file;

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

if (!file.good()) {

cout << " I can't open the file" << endl;

system("PAUSE");

exit(0);

}

//wczytuj z pliku dopoki są dane

while (!file.eof())

for (int i = 0; i < LETTER\_AMOUNT; i++) { // i oznacza indeks litery

for (int j = 0; j < PIXEL\_AMOUNT; j++) // j oznacza ilosc bitow na dana litere

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

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

}

file.close();

}

//funkcja aktywacji - funkcja sigmoidalna

double DeltaRule::activationFunction(double sum) {

//Wspolczynnik beta = 1.0

return (1 / (1 + exp(-1.0 \* sum)));

}

//pochodna funkcji aktywcji

double DeltaRule::derivativeActivationFunction(double sum)

{

return (1.0\*exp(-1.0\*sum)) / (pow(exp(-1.0\*sum) + 1, 2));

}

//zwraca sume danego wejscia

double DeltaRule::getSum(int letter[], double \* weights)

{

double sum = 0.0;

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

sum += letter[i] \* weights[i];

return sum;

}

//funkcja uczaca

void DeltaRule::learn() {

cout << endl << "DELTARULE LEARNING" << endl;

bool acceptableError = false; //zmienna, stwierdzajaca czy blad jest mozliwy do zaakceptowania

int epoch = 0; //numer epoki

do {

epoch++; // zwiekszenie numeru epoki

error = 0.0; //zerowanie glownego bledu w celu sprawdzenia bledow podczas jednej iteracji

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

//wynik otrzymany

output = activationFunction(getSum(inputData[i], weights));

//obliczanie roznicy pomiedzy wynikiem oczekiwanym a wynikiem otrzymanym

delta = expectedResults[i] - output;

//aktualizowanie wag

for (int j = 0; j < numberOfWeights; j++)

weights[j] += learningRate \* delta\*inputData[i][j] \* derivativeActivationFunction(getSum(inputData[i], weights));

//aktualizowanie bledu glownego

error += delta \* delta;

}

error /= 2;

//porownywanie bledu z progiem

if (error > EMax)

acceptableError = false;

else

acceptableError = true;

} while (!acceptableError);

cout << "Epoki: " << epoch << endl;

cout << "MSE: " << error << endl;

}

//funkcja testujaca

void DeltaRule::test()

{

cout << "Test: " << endl;

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

cout << "Litera " << setTestLetters[i] << " - ";

if (activationFunction(getSum(setTest[i], weights)) > 0.5) {

cout << "Duza";

}

else {

cout << "Mala";

}

cout << endl;

}

}

„Źródło.cpp ”

#include "Adaline.h"

#include "DeltaRule.h"

int main()

{

double learningRate = 0.001; //wspolczynnik uczenia

cout << "Learning Rate: " << learningRate << endl;

//Adaline

Adaline adaline(learningRate);

adaline.learn();

adaline.test();

//DeltaRule

DeltaRule deltaRule(learningRate);

deltaRule.learn();

deltaRule.test();

while (true)

{

}

}