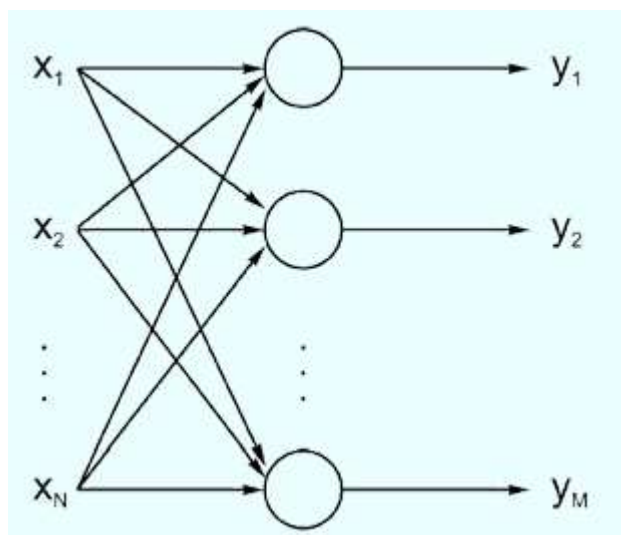


19.12.2017 r.	Izabela Musztyfaga	Podstawy Sztucznej Inteligencji
Scenariusz 2 - Budowa i działanie sieci jednowarstwowej		

Jednowarstwowa sieć neuronowa to sieć zbudowana z jednej warstwy neuronów. Każde z wielu wejść ($x_1 - x_N$) wchodzi do każdego z neuronów, natomiast każdy z neuronów przekazuje jedną wartość na wyjście ($y_1 - y_M$).

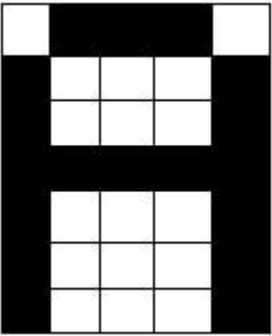
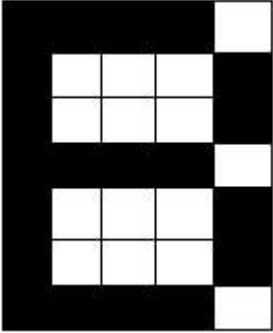
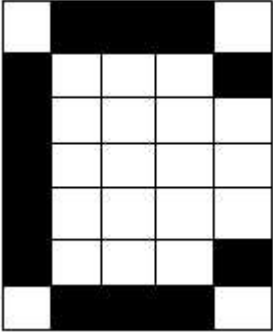


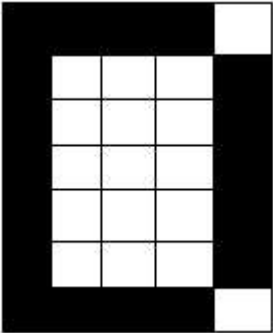
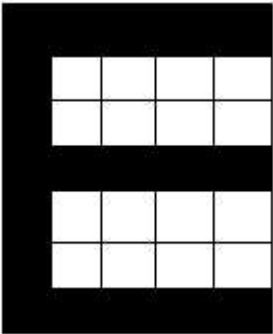
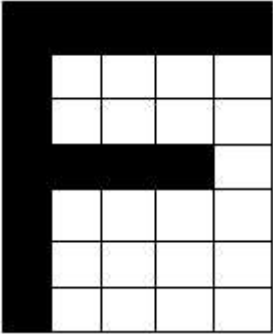
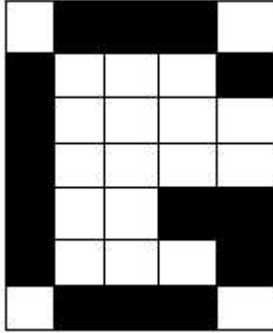
Rys. 1¹ – Jednowarstwowa sieć neuronowa

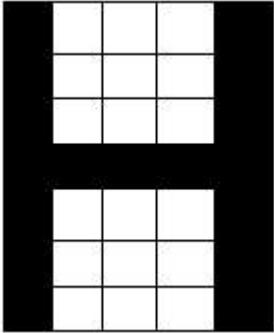
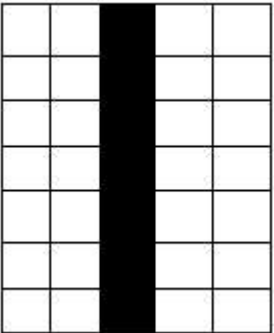
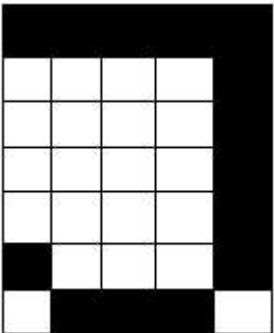
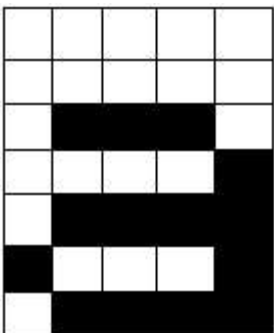
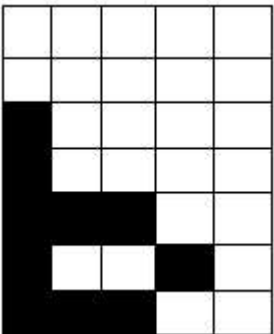
W tym ćwiczeniu należało nauczyć jednowarstwową sieć rozpoznawania wielkości liter. W tym celu zostały wykorzystano litery (10 wielkich oraz 10 małych) przedstawione graficznie na matrycach 5x7, które następnie zamieniono na 0 i 1, z czego finalnie powstały wektory. Każda wartość liczbową odpowiada jednemu pikselowi – jeżeli pole w matrycy jest puste przypisujemy mu wartość 0, jeżeli jest 'pełne' przypisujemy mu wartość 1.

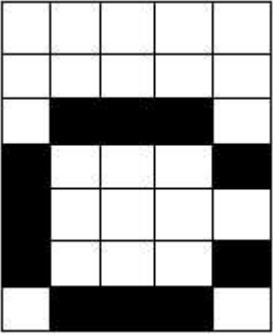
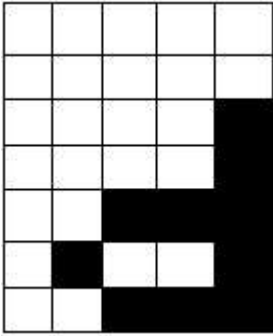
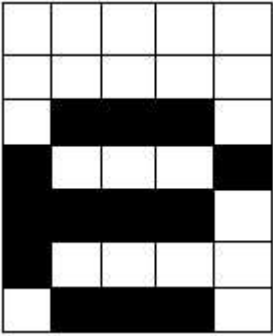
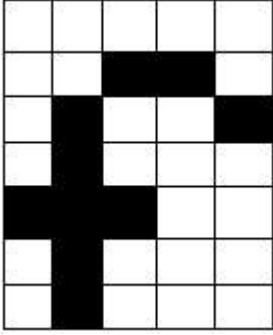
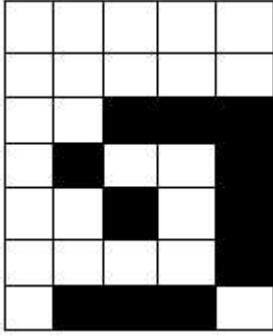
Każdy wektor jest unikalny, tzn. żaden się nie powtarza.

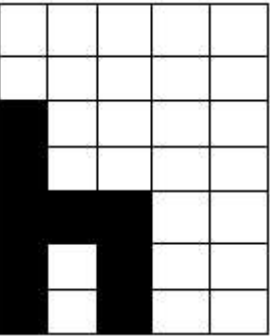
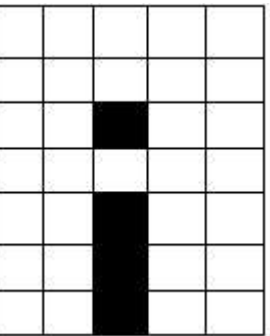
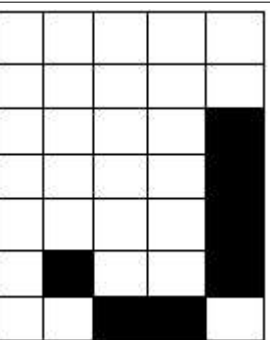
1 Źródło obrazu: http://galaxy.agh.edu.pl/~vlsi/AI/koho_t/siec.jpg

Litera przedstawiona graficznie	Litera po konwersji do 0 i 1	Wektor
	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 1 1 1 0 1 0 0 0 1 1 0 0 0 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	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	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

	<pre>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 0 0 0 1 1 1 1 1 0</pre>	<pre>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</pre>
	<pre>1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1</pre>	<pre>1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1</pre>
	<pre>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 0 0 0 0</pre>	<pre>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</pre>
	<pre>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 1 1 1 0 0 0 1 0 1 1 1 0</pre>	<pre>0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 0</pre>

	<div>10001 10001 10001 11111 10001 10001 10001</div>	<div>10001100011000111111100011000110001</div>
	<div>00100 00100 00100 00100 00100 00100 00100 00100</div>	<div>0010000100001000010000100001000010000100</div>
	<div>11111 00001 00001 00001 00001 00001 10001 01110</div>	<div>1111100001000010000100001000011000101110</div>
	<div>00000 00000 01110 00001 01111 10001 01111</div>	<div>000000000000111000001011111000101111</div>
	<div>00000 00000 10000 10000 11100 10010 11100</div>	<div>000000000001000010000111001001011100</div>

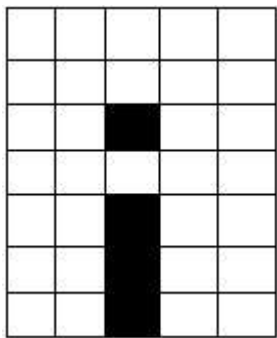
	<div>00000</div> <div>00000</div> <div>01110</div> <div>10001</div> <div>10000</div> <div>10001</div> <div>01110</div>	<div>0000000000000111010001100001000101110</div>
	<div>00000</div> <div>00000</div> <div>00001</div> <div>00001</div> <div>00111</div> <div>01001</div> <div>00111</div>	<div>0000000000000000100001001110100100111</div>
	<div>00000</div> <div>00000</div> <div>01110</div> <div>10001</div> <div>11110</div> <div>10000</div> <div>01110</div>	<div>0000000000000111010001111101000001110</div>
	<div>00000</div> <div>00110</div> <div>01001</div> <div>01000</div> <div>11100</div> <div>01000</div> <div>01000</div>	<div>00000001100100101000111000100001000</div>
	<div>00000</div> <div>00000</div> <div>00111</div> <div>01001</div> <div>00101</div> <div>00001</div> <div>01110</div>	<div>000000000000011101001001010000101110</div>

	<pre> 00000 00000 00100 00000 00100 00100 00100 00100 </pre>	<pre> 000000000001000010000111001010010100 </pre>
	<pre> 00000 00000 00100 00000 00100 00100 00100 00100 </pre>	<pre> 000000000000010000000001000010000100 </pre>
	<pre> 00000 00000 00001 00001 00001 00001 01001 00110 </pre>	<pre> 000000000000000100001000010100100110 </pre>

```

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 0 0

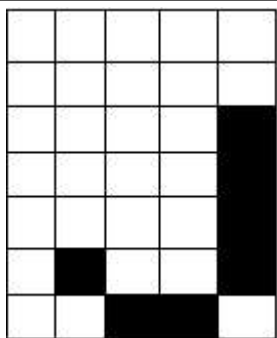
```



```

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 0 0

```



```

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

```

Tab. 1 – Litery przedstawione graficznie, przy użyciu 0 i 1 oraz jako wektor

Algorytmy uczenia sieci

Delta Rule Learning

Metoda uczenia polegająca na aktualizowaniu wag, które na początku dobierane są losowo. Sam proces aktualizowania/dobierania wag opisany jest wzorem:

$$\Delta w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i,$$

where

α is a small constant called *learning rate*

$g(x)$ is the neuron's activation function

t_j is the target output

h_j is the weighted sum of the neuron's inputs

y_j is the actual output

x_i is the i th input.

Rys. 2² – Wzór

Adaline

Tak samo jak dla Delta Rule Learning, metoda uczenia polegająca na aktualizowaniu wag, które na początku dobierane są losowo. Jednak wzór według którego dobierane są wagi jest zupełnie inny:

- η is the learning rate (some positive constant)
- y is the output of the model
- o is the target (desired) output

then the weights are updated as follows $\mathbf{w} \leftarrow \mathbf{w} + \eta(o - y)\mathbf{x}$. The ADALINE converges to the least squares error which is $E = (o - y)^2$.

Rys. 3³ – Wzór

2 Źródło: https://en.wikipedia.org/wiki/Delta_rule

3 Źródło: <https://en.wikipedia.org/wiki/ADALINE>

Zestawienie otrzymanych wyników

Delta Rule Learning

```
DELTA RULE LEARNING
Wspolczynnik uczenia sieci: 0.1
Numer epoki: 66
Blad Sredniokwadratowy: 0.0987045

Czy litera jest wielka?

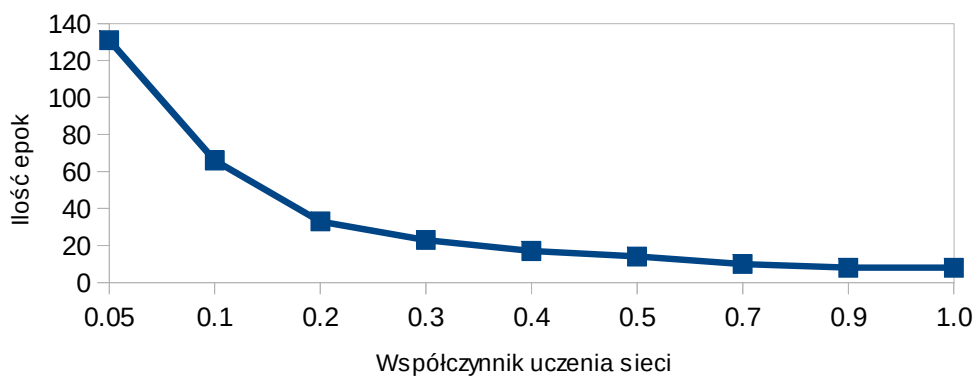
A - Tak
B - Tak
C - Tak
D - Tak
E - Tak
F - Tak
G - Tak
H - Tak
I - Tak
J - Tak
a - Nie
b - Nie
c - Nie
d - Nie
e - Nie
f - Nie
g - Nie
h - Nie
i - Nie
j - Nie
Press any key to continue . . .
```

Rys. 2 – Przykładowe działanie programu

Współczynnik uczenia sieci	Ilość epok potrzebna do nauczania	Błąd średniokwadratowy
0.05	131	0.0991234
0.1	66	0.0987045
0.2	33	0.0996345
0.3	23	0.0952909
0.4	17	0.097683
0.5	14	0.095646
0.7	10	0.0953098
0.9	8	0.0967219
1.0	8	0.0855974

Tab. 2 – Zestawienie wyników dla Delta Rule Learning

Wykres zależności Ilości epok od współczynnika uczenia sieci



Wykres 1 –
Zależność
liczby epok
od

współczynnika uczenia sieci dla Delta Rule Learning

Adaline

```
ADALINE
Ilosc epok: 13
Blad Sredniokwadratowy: 0.0988026

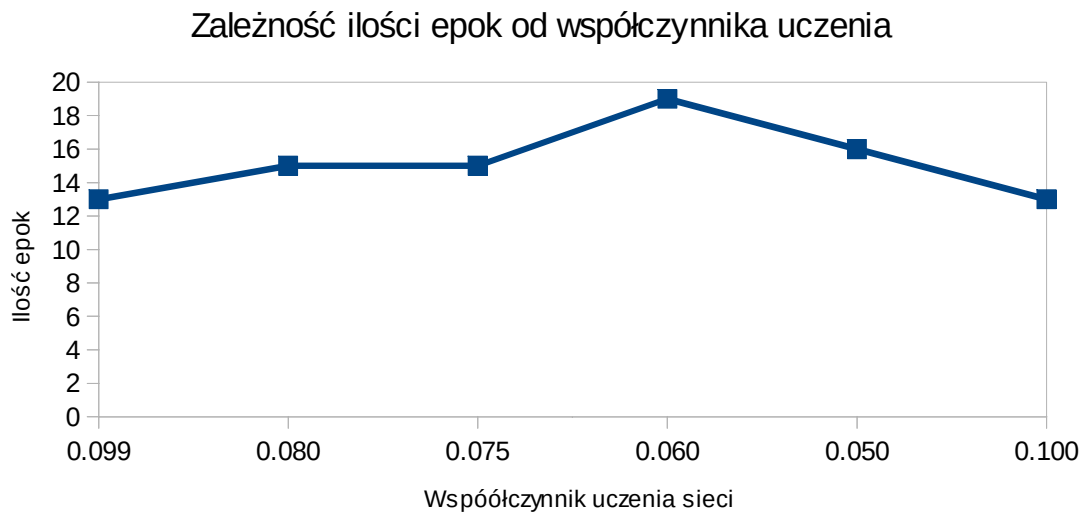
Czy litera jest wielka?

A - Tak
B - Tak
C - Tak
D - Tak
E - Tak
F - Tak
G - Tak
H - Tak
I - Tak
J - Tak
a - Nie
b - Nie
c - Nie
d - Nie
e - Nie
f - Nie
g - Nie
h - Nie
i - Nie
j - Nie
Press any key to continue . . .
```

Rys. 3 – Przykładowe działanie programu

Współczynnik uczenia sieci	Ilość epok potrzebna do nauczania	Błąd średniokwadratowy
0.099	13	0.0989084
0.080	15	0.0926449
0.075	15	0.0992469
0.060	19	0.0946001
0.050	16	0.0996547
0.100	13	0.0988026

Tab. 3 – Zestawienie wyników dla Adaline



Wykres 2 – Zależność liczby epok od współczynnika uczenia sieci dla Adaline

Wnioski

Czynnikiem mającym największy wpływ na proces uczenia się jednowarstwowej sieci neuronowej jest współczynnik uczenia. W tym ćwiczeniu dobierany jest on w sposób eksperymentalny. Im większa wartość współczynnika uczenia się sieci, tym cały proces jest bardziej efektywny (mniejsza wartość błędu średniokwadratowego), a dodatkowo sieć uczy się coraz szybciej (mniejsza ilość epok potrzebnych do poprawnego nauczania sieci).

Uczenie zostało przeprowadzone dla dwóch algorytmów: Delta Rule Learning oraz Adaline. Wykorzystano również różne współczynniki uczenia sieci. Dla Delta Rule Learning było to 0.05 – 1.0, natomiast dla sieci Adaline 0.99 – 0.100. Patrząc na wykresy 1 i 2 możemy zaobserwować, że dobór zarówno algorytmu jak i współczynnika ma duży wpływ na cały proces uczenia sieci neuronowej.

Listing kodu wraz z komentarzami

main.cpp

```
//      Program realizujący uczenie neuronowej sieci jednowarstwowej rozpoznawania wielkości
//      liter.
//
//      Data utworzenia pliku: 19.12.2017          Autor: Izabela Musztyfaga
//      Autor modyfikacji          Co zostalo zmienione          Data
//
//*****

#include "DeltaRuleLearning.h"
#include "AdalineLearning.h"

int main()
{
    double WspolczynnikUczenia = 0.1;          //wartość współczynnika uczenia

    DeltaRuleLearning delta(WspolczynnikUczenia);          //Stworzenie obiektu
    delta.Uczenie();          //Uczenie
    delta.Testowanie();          //Testowanie

    //AdalineLearning adaline(WspolczynnikUczenia);          //Stworzenie obiektu
    //adaline.Uczenie();          //Uczenie
    //adaline.Testowanie();          //Testowanie

    return 0;
}
```

AdalineLearning.h

```
//*****
//      Klasa odpowiedzialna za uczenie sieci Adaline
//
//
//      Data utworzenia pliku: 19.12.2017          Autor: Izabela Musztyfaga
//      Autor modyfikacji          Co zostalo zmienione          Data
//
//*****
#include <ctime>
#include <fstream>
#include <iostream>
#include <cstdlib>

class AdalineLearning {
```

```
public:
    int IloscLiter;
    int IloscWagLiter;
    int DaneWejscowe[20][35];
    int OczekiwanyRezultat[20];
    // w pliku 1 to duza litera, 0 to duza
    double * WagiLiter;
    double WspolczynnikUczenia;
    double BladSredniokwadratowy;
    double delta;
    //jest to wyliczona roznica (dodawana do wag)

    AdalineLearning();
    //konstruktor bezparametrowy
    AdalineLearning(double);
    //konstruktor
    ~AdalineLearning();
    //destruktor
    void WczytanieDanychTestowych();
    //funkcja wczytuje dane do nauki z pliku
    double WartoscWyjscia(int [], double *);
    //funkcja zwraca sume wejscia danej litery
    double LosoweWagiPocztkowe();
    //ustawienie wag poczatkowych
    bool FunkcjaAktywacji(double);
    //funkcja aktywacji
    void Uczenie();
    //funkcja uczaca
    void Testowanie();
    //funkcja testujaca

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

    //tablica danych testowych
    int DaneTestowe[20][35] = {
        { 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 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, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
1, 0, 0, 0, 1, 0, 1, 1, 1, 0 },
        { 1, 0, 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, 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, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 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, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 1, 1, 0, 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, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 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, 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, 0, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 1, 1, 0 },
    { 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 }
};

};

```

AdalineLearning.cpp

```

#include "AdalineLearning.h"

AdalineLearning::AdalineLearning()
{

}

AdalineLearning::~AdalineLearning()
{

}

AdalineLearning::AdalineLearning(double WspolczynnikUczenia_)
{
    IloscWagLiter = 35;
    IloscLiter = 20;
    WspolczynnikUczenia = WspolczynnikUczenia_;
    WagiLiter = new double[IloscWagLiter];           //tablica przechowujaca wagi

    for (int i = 0; i < 35; i++)                     //przypisanie losowych wag poczatkowych
    {
        this->WagiLiter[i] = LosoweWagiPoczatkowe();
    }

    WczytanieDanychTestowych();                     //wczytanie danych z pliku
}

void AdalineLearning::WczytanieDanychTestowych()    //wczytanie danych z pliku - metod
{
    std::fstream file;
    file.open("Dane.txt");

    while (!file.eof())
        for (int i = 0; i < 20; i++)
        {
            for (int j = 0; j < 35; j++)
            {
                file >> this->DaneWejscowe[i][j];
            }

            file >> this->OczekiwanyRezultat[i];
        }
}

```

```

        //0 lub 1 w pliku mowi, czy litera jest duza czy mala
    }

    file.close();
}

double AdalineLearning::LosoweWagiPocztkowe()
    //ustawienie wag pocztkowych
{
    double Waga = ((double)rand() / (double)RAND_MAX);
    return Waga;
}

bool AdalineLearning::FunkcjaAktywacji(double sum)
    //funkcja aktywacji
{
    if (sum > 0.5)
    {
        return true;
    }
    else
    {
        return false;
    }
}

//zwraca sume danego wejscia
double AdalineLearning::WartoscWyjscia(int Litery[], double * Wagi)
{
    double sum = 0.0;
    for (int i = 0; i < IloscWagLiter; i++)
    {
        sum += Litery[i] * Wagi[i];
    }

    return sum;
}

//funkcja uczaca
void AdalineLearning::Uczenie()
{
    std::cout << "ADALINE" << std::endl;

    bool AkceptowalnaWartoscBledu = true; //zmienna, stwierdzajaca czy blad jest mozliwy
do zaakceptowania

    int epoka = 0;

    do {
        epoka++;
        BladSredniokwadratowy = 0.0;
        for (int i = 0; i<IloscLiter; i++)
        {
            //obliczanie roznicy pomiedzy wynikiem oczekiwany a wynikiem
otrzymany
            delta = OczekiwanyRezultat[i] - WartoscWyjscia(DaneWejscowe[i],
WagiLiter);

            //aktualizowanie wag
            for (int j = 0; j < IloscWagLiter; j++)
            {
                WagiLiter[j] += WspolczynnikUczenia*delta*DaneWejscowe[i][j];
            }

            //aktualizowanie bledu glownego

```

```

        BładSredniokwadratowy += delta*delta;
    }

    BładSredniokwadratowy /= 2;

    //porownywanie błedu z progiem
    if (BładSredniokwadratowy > 0.1)
    {
        AkceptowalnaWartoscBledu = false;
    }
    else
    {
        AkceptowalnaWartoscBledu = true;
    }
} while (!AkceptowalnaWartoscBledu);

std::cout << "Ilość epok: " << epoka << std::endl;
std::cout << "Bład Sredniokwadratowy: " << BładSredniokwadratowy << std::endl <<
std::endl;
}

void AdalineLearning::Testowanie()
{
    std::cout << "Czy litera jest wielka?" << std::endl << std::endl;
    for (int i = 0; i<IlośćLiter; i++)
    {
        std::cout << Litery[i] << " - ";
        if (FunkcjaAktywacji(WartoscWyjscia(DaneTestowe[i], WagiLiter)))
        {
            std::cout << "Tak";
        }
        else
        {
            std::cout << "Nie";
        }
        std::cout << std::endl;
    }
}

```

DeltaRuleLearning.h

```

//*****
//    Klasa odpowiedzialna za uczenie sieci metodą Delta Rule
//
//
//    Data utworzenia pliku: 19.12.2017          Autor: Izabela Musztyfaga
//
//    Autor modyfikacji          Co zostało zmienione          Data
//
//*****
#include <ctime>
#include <fstream>
#include <iostream>
#include <cstdlib>

//using namespace std;

class DeltaRuleLearning {
public:
    int IlośćLiter;
    int IlośćWagLiter;
    int DaneWejscowe[20][35];

```

```

int OczekiwanyRezultat[20];
double * WagiLiter;
double WspolczynnikUczenia;
double BladSredniokwadratowy;
double delta; //jest to wyliczona roznicza(dodawana do wag)
double WartoscWyjscia; //przechowuje aktualna wartosc (porownywana z wartoscia
                        oczekiwana)

```

```

DeltaRuleLearning(); //konstruktor bezparametrowy
DeltaRuleLearning(double); //konstruktor
~DeltaRuleLearning(); //destruktor
void WczytanieDanychTestowych(); //wczytanie danych z pliku
double LosoweWagiPoczkowe(); //generator liczb losowych
double FunkcjaAkttywacji(double); //funkcja aktywacji
double Pochodna(double); //pochodna funkcji aktywacji
double SumaWejscia(int [], double * ); //zwraca sume wejscia danej litery
void Uczenie(); //uczenie sieci
void Testowanie(); //testowanie sieci

```

```

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

int DaneDoPorownania[20][35] = { //tablica danych testowych
{ 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 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, 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, 0,
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, 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, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1,
1, 0, 0, 0, 1, 0, 1, 1, 1, 1 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
1, 0, 0, 1, 0, 1, 1, 0, 0 },
{ 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, 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, 0, 1, 0, 0, 0, 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, 1,
1, 0, 0, 0, 0, 0, 1, 1, 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,
0, 0, 1, 0, 0, 0, 0, 1, 1, 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, 0,
0, 0, 0, 0, 1, 1, 1, 1, 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,
0, 0, 1, 0, 0, 0, 0, 1, 1, 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, 0,
0, 0, 1, 0, 0, 1, 0, 0, 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, 0,
0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0 }
};

```



```
};
```

DeltaRuleLearning.cpp

```
#include "DeltaRuleLearning.h"

DeltaRuleLearning::DeltaRuleLearning()           //konstruktor bezparametrowy
{
}

DeltaRuleLearning::DeltaRuleLearning(double _learningRate) //konstruktor
{
    IloscWagLiter = 35;
    IloscLiter = 20;
    WspolczynnikUczenia = _learningRate;
    WagiLiter = new double[IloscWagLiter];

    WczytanieDanychTestowych();           //wczytanie z pliku danych uczacych

    for (int i = 0; i < 35; i++)
    {
        this->WagiLiter[i] = LosoweWagiPocztakowe();
    }
}

DeltaRuleLearning::~DeltaRuleLearning()         //destruktor
{
}

double DeltaRuleLearning::LosoweWagiPocztakowe() //pocztakowe, losowe wagi wejsc
{
    double Waga = ((double)rand() / (double)RAND_MAX);
    return Waga;
}

void DeltaRuleLearning::WczytanieDanychTestowych() //wczytywanie danych z pliku
{
    std::fstream file;
    file.open("Dane.txt");

    while (!file.eof())
    {
        for (int i = 0; i < 20; i++)           //20 liter
        {
            for (int j = 0; j < 35; j++)       //35 pikseli/wag
            {
                file >> this->DaneWejscowe[i][j];
            }
            file >> this->OczekiwanyRezultat[i]
            //reszta okresla czy litera jest duza czy mala
        }

        file.close();           //zamknięcie pliku
    }
}
```

```

double DeltaRuleLearning::FunkcjaAktywacji(double sum)           //funkcja aktywacji
{
    return (1 / (1 + exp(-1.0 * sum)));
}

double DeltaRuleLearning::Pochodna(double sum)                 //pochodna funkcji
{
    return (1.0*exp(-1.0*sum)) / (pow(exp(-1.0*sum) + 1, 2));
}

//zwraca sume danego wejścia
double DeltaRuleLearning::SumaWejscia(int letter[], double * weights)
{
    double sum = 0.0;
    for (int i = 0; i < IloscWagLiter; i++)
    {
        sum += letter[i] * weights[i];
    }

    return sum;
}

//funkcja ucząca
void DeltaRuleLearning::Uczenie() {
    std::cout << std::endl << "DELTA RULE LEARNING" << std::endl;

    bool AkceptowalnaWartoscBledu = false; //zmienna, stwierdzająca czy błąd jest możliwy
do zaakceptowania
    int epoka = 0;

    do {
        epoka++;
        BladSredniokwadratowy = 0.0;
        for (int i = 0; i < IloscLiter; i++)
        {
            //wynik otrzymany
            WartoscWyjscia = FunkcjaAktywacji(SumaWejscia(DaneWejscia[i], WagiLiter));

            //obliczanie różnicy pomiędzy wynikiem oczekiwanym a wynikiem otrzymanym
            delta = OczekiwanyRezultat[i] - WartoscWyjscia;

            //aktualizowanie wag według wzoru
            for (int j = 0; j < IloscWagLiter; j++)
            {
                WagiLiter[j] += WspolczynnikUczenia*delta*DaneWejscia[i][j] *
                Pochodna(SumaWejscia(DaneWejscia[i], WagiLiter));
            }

            BladSredniokwadratowy += delta*delta;
        }
        BladSredniokwadratowy /= 2;

        //porównywanie błędu z progiem
        if (BladSredniokwadratowy > 0.1)
        {
            AkceptowalnaWartoscBledu = false;
        }
        else
        {
            AkceptowalnaWartoscBledu = true;
        }
    }
}

```

```

} while (!AkceptowalnaWartoscBledu);

std::cout << "Wspolczynnik uczenia sieci: " << WspolczynnikUczenia << std::endl;
std::cout << "Numer epoki: " << epoka << std::endl;
std::cout << "Blad Sredniokwadratowy: " << BladSredniokwadratowy << std::endl << std::endl;
}

void DeltaRuleLearning::Testowanie()
    //testowanie + drukowanie wyniku
{
    std::cout << "Czy litera jest wielka?" << std::endl << std::endl;
    for (int i = 0; i < IloscLiter; i++)
    {
        std::cout << setTestLetters[i] << " - ";
        if (FunkcjaAktywacji(SumaWejscia(DaneDoPorownania[i], WagiLiter)) > 0.5)
        {
            std::cout << "Tak";
        }
        else {
            std::cout << "Nie";
        }
        std::cout << std::endl;
    }
}

```