

### Scenariusz 3

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

**Wykonała:** Burnat Magdalena, IS, gr. 3, 270652

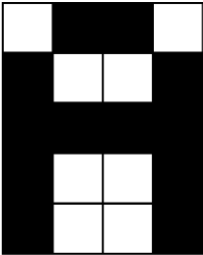
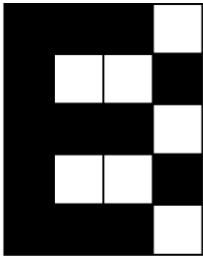
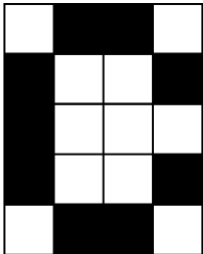
#### 1. Cel ćwiczenia

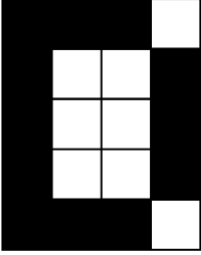
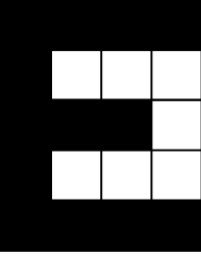
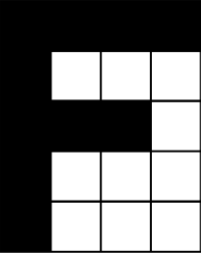
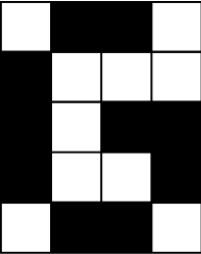
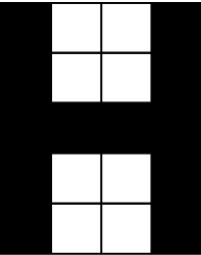
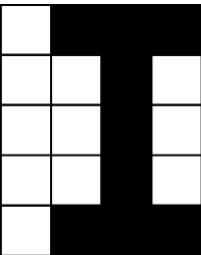
Celem ćwiczenia jest poznanie budowy i działanie wielowarstwowych sieci neuronowych poprzez uczenie z użyciem algorytmu wstecznej propagacji błędu rozpoznawania konkretnych liter alfabetu.

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

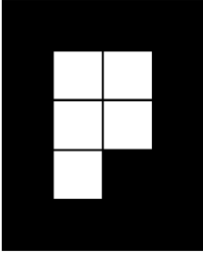
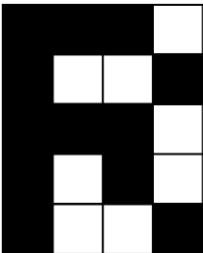
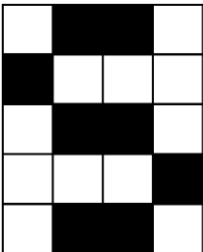
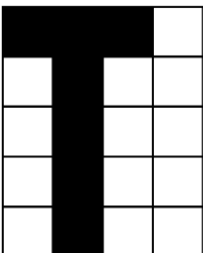
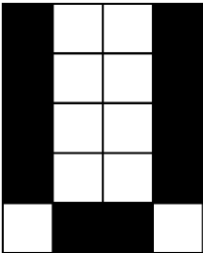
Celem budowanej sieci jest rozpoznawanie liter w tym celu stworzony został zestaw liter( 20 dużych liter), które są reprezentowane w postaci dwuwymiarowej tablicy 4x5 pikseli dla jednej litery.

Tab. 1. Wykorzystane litery.

	<p>Wektor wejściowy: 0 1 1 0 1 0 0 1 1 1 1 1 1 0 0 1 1 0 0 1</p> <p>Wektor wyjściowy: 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p>
	<p>Wektor wejściowy: 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0</p> <p>Wektor wyjściowy: 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p>
	<p>Wektor wejściowy: 0 1 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 0</p> <p>Wektor wyjściowy: 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p>

	<p>Wektor wejściowy: 1 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 0</p> <p>Wektor wyjściowy: 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p>
	<p>Wektor wejściowy: 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 0 1 1 1 1</p> <p>Wektor wyjściowy: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p>
	<p>Wektor wejściowy: 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0</p> <p>Wektor wyjściowy: 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p>
	<p>Wektor wejściowy: 0 1 1 0 1 0 0 0 1 0 1 1 1 0 0 1 0 1 1 0</p> <p>Wektor wyjściowy: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p>
	<p>Wektor wejściowy: 1 0 0 1 1 0 0 1 1 1 1 1 1 0 0 1 1 0 0 1</p> <p>Wektor wyjściowy: 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0</p>
	<p>Wektor wejściowy: 0 1 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 1</p> <p>Wektor wyjściowy: 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0</p>

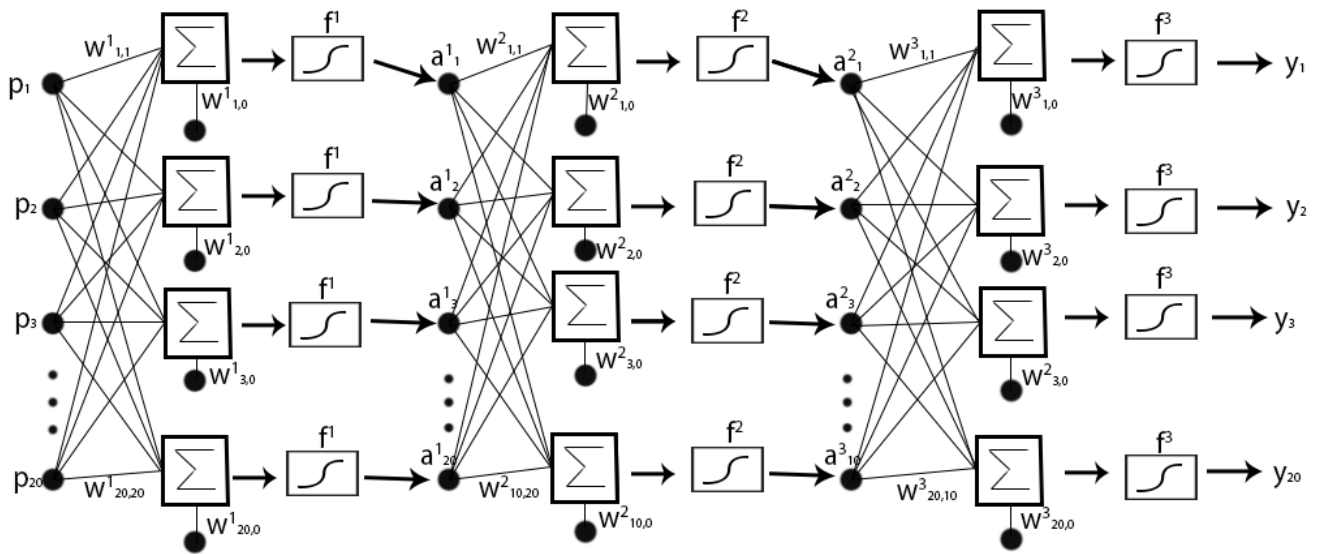


	<p>Wektor wejściowy: 1 1 1 1 1 0 0 1 1 0 0 1 1 0 1 1 1 1 1 1</p> <p>Wektor wyjściowy: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0</p>
	<p>Wektor wejściowy: 1 1 1 0 1 0 0 1 1 1 1 0 1 0 1 0 1 0 0 1</p> <p>Wektor wyjściowy: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0</p>
	<p>Wektor wejściowy: 0 1 1 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 0</p> <p>Wektor wyjściowy: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0</p>
	<p>Wektor wejściowy: 1 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0</p> <p>Wektor wyjściowy: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</p>
	<p>Wektor wejściowy: 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0</p> <p>Wektor wyjściowy: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</p>

Tak więc wektor wejściowy składa się z 20 wartości tak samo wektor wyjściowy. Postanowiono stworzyć sieć trójwarstwową (warstwa wejściowa – warstwa ukryta – warstwa wyjściowa). Poszczególne warstwy składają się z następującej liczby neuronów:

20 - 10 - 20

Schemat stworzonej sieci został zamieszczony na poniższym rysunku.



Rys. 1 Trójwarstwowa sieć neuronowa

W celu przeprowadzenia uczenia dla sieci wielowarstwowej konieczne jest wykorzystanie algorytmu wstecznej propagacji błędów. Sprowadza się to do obliczenia błędów dla każdego neuronu korzystając z wszystkich błędów neuronów, do których wysłał on swój sygnał. Uwzględniane są także wagi tych połączeń. Propagacja wsteczna odbywa się w przeciwnym kierunku niż przepływ sygnałów w sieci.

Także aby uchronić się od gwałtownych zmian wartości wagowych, niekorzystnych dla uczenia, należy nadać procesowi zmian pewną bezwładność. W jej wyniku zmiana wag w bieżącym cyklu zależy również od zmiany jaka nastąpiła w cyklu poprzednim i pośrednio w cyklach jeszcze wcześniejszych. Bezwładność tę nadaje współczynnik *momentum*. Jak łatwo zauważyć, reguluje on wpływ zmiany wag na proces uczenia. Kiedy stosowane jest momentum, korekta wag neuronu zależy nie tylko od sygnału wejściowego i błędów, jaki neuron popełnił, ale również od tego, jak duża była korekta wag w poprzednim kroku uczenia. W ten sposób szybkość uczenia (wielkość korekty wag) automatycznie maleje w miarę zbliżania się do właściwego rozwiązania, a proces uczenia staje się płynniejszy.

Proces uczenia przebiega według następującego schematu w danej epoce uczenia:

1. Wybór  $\eta > 0$  oraz  $E_{\max} > 0$
2. Wybór początkowych wartości wag, jako niewielkich liczb losowych z zakresu -1 do 1
3. Ustawienie sumy błędów kwadratowego na zero ( $E = 0$ )
4. Dla danego wektora uczącego obliczmy odpowiedź sieci (warstwa po warstwie). A więc dla pojedynczego neuronu obliczana jest suma ilorazów sygnałów wejściowych i wag -  $u^k_i$ . Następnie obliczamy wartość wyjściową neuronu  $a^k_i$  wykorzystując tę wartość za pomocą unipolarnej, sigmoidalnej funkcji aktywacji według wzoru:

$$a_i^k = f(u_i^k) = \frac{1}{1 + \exp(-\beta \cdot u_i^k)}$$

gdzie dla  $\beta$  wybrano wartość równą 1

A więc elementy wektora wyjściowego w naszym przypadku możemy zapisać jako:

$$y_i = f^3 \left( \sum w^3 \cdot f^2 \left( \sum w^2 \cdot f^1 \left( \sum w^1 \cdot p \right) \right) \right)$$

5. Dla każdego neuronu wyliczany jest jego błąd, który można określić jako:

$$\delta_i = \sum w_z \cdot \delta_z$$

gdzie  $w_z, \delta_z$  to kolejno wagi oraz błędy neuronów, do których analizowany neuron wysyła swój sygnał.

6. Każdy neuron modyfikuje wagi na podstawie wartości błędu z wykorzystaniem następujących zależności:

$$w_{i,j}^k(t+1) += \alpha \cdot (w_{ij}(t) - w_{ij}(t-1)) + \eta \cdot \delta_i^k \cdot \frac{df(u_i^k)}{df u_i^k} \cdot a_i^{k-1}$$

gdzie pochodną funkcji wyliczamy ze wzoru:

$$\frac{df(u_i^k)}{df u_i^k} = f(u_i^k)(1 - f(u_i^k))$$

$\alpha$  to współczynnik bezwładności, którego wartość optymalna jest testowana

$\eta$  to współczynnik uczenia wybierany z zakresu od 0 do 1

7. Obliczenie łącznego błędu epoki:

$$E = E + \frac{1}{2} \sum (d_i - y_i)^2$$

8. Uczenie uważamy za zakończone, jeżeli  $E < E_{\max}$ . W przeciwnym razie rozpoczynamy nową epokę uczenia i wracamy do punktu nr 3.

### 3. Otrzymane wyniki

Pierwszym aspektem pod którym było sprawdzane poprawność działania stworzonej sieci to wartość maksymalnego błędu średniokwadratowego dla pojedynczej epoki przy którym sieć gwarantuje dobre odpowiedzi podczas testowania na tych samych używanych podczas uczenia. Po wielokrotnym uruchomieniu procesu uczenia stworzonej sieci ustalono, że dla wartości  $E = 1,0$  sieć jest nauczona poprawnego działania.

Kolejną sprawą było zbadanie wpływu różnych wartości współczynnika uczenia oraz współczynnika bezwładności na szybkość uczenia się sieci. Testy przeprowadzano osobno dla każdej z kombinacji, a otrzymane wyniki zostały zamieszczone w poniższej tabeli.

Tab.2. Zależność liczby epok potrzebnych do poprawnego nauczania neuronu w różnych kombinacjach współczynnika uczenia i współczynnika bezwładności.

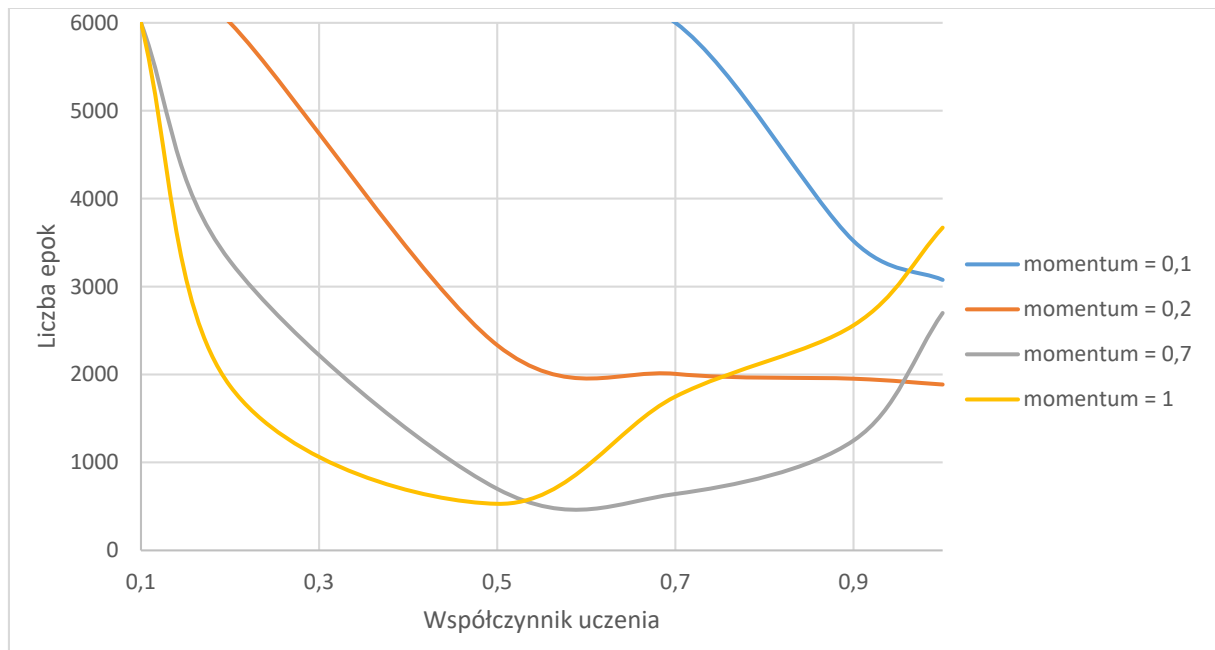
Liczba epok potrzebnych, aby błąd średniokwadratowy spadł poniżej wartości 1,0						
$\alpha \backslash \eta$	0,1	0,2	0,5	0,7	0,9	1
0,1	>5000 (E=7,85)	>5000 (E=5,95)	>5000 (E=1,36)	>5000 (E=3,57)	3517	3075
0,2	>5000 (E=5,95)	>5000 (E=1,33)	2333	2007	1952	1886
0,5	>5000 (E=3,57)	3290	700	642	1250	2700
1	>5000 (E=2,01)	1870	530	1750	2561	3671

W niektórych kombinacjach pomiar był niewspółmiernie długi, dlatego też nie przeprowadzano testu do końca (tj. do osiągnięcia  $E < 1,0$ ) tylko zakończono go po 5000 epokach i odczytano wartość błędu średniokwadratowego w momencie zakończenia.

Kolejnym z przeprowadzonych rodzajów testów był wpływ współczynnika bezwładności na otrzymywane wartości błędu średniokwadratowego i skoki tej wartości na przestrzeni wzrastającej liczby epok.

Jako ostatecznie sprawdzono skuteczność sieci podczas rozpoznawania liter, które nie znajdowały się w danych uczących. Sieć nie była w stanie w 100% dobrze zidentyfikować literę, ale poziom poprawnych odpowiedzi poszczególnych wartości w wektorze wyjściowym wynosił powyżej 90%.

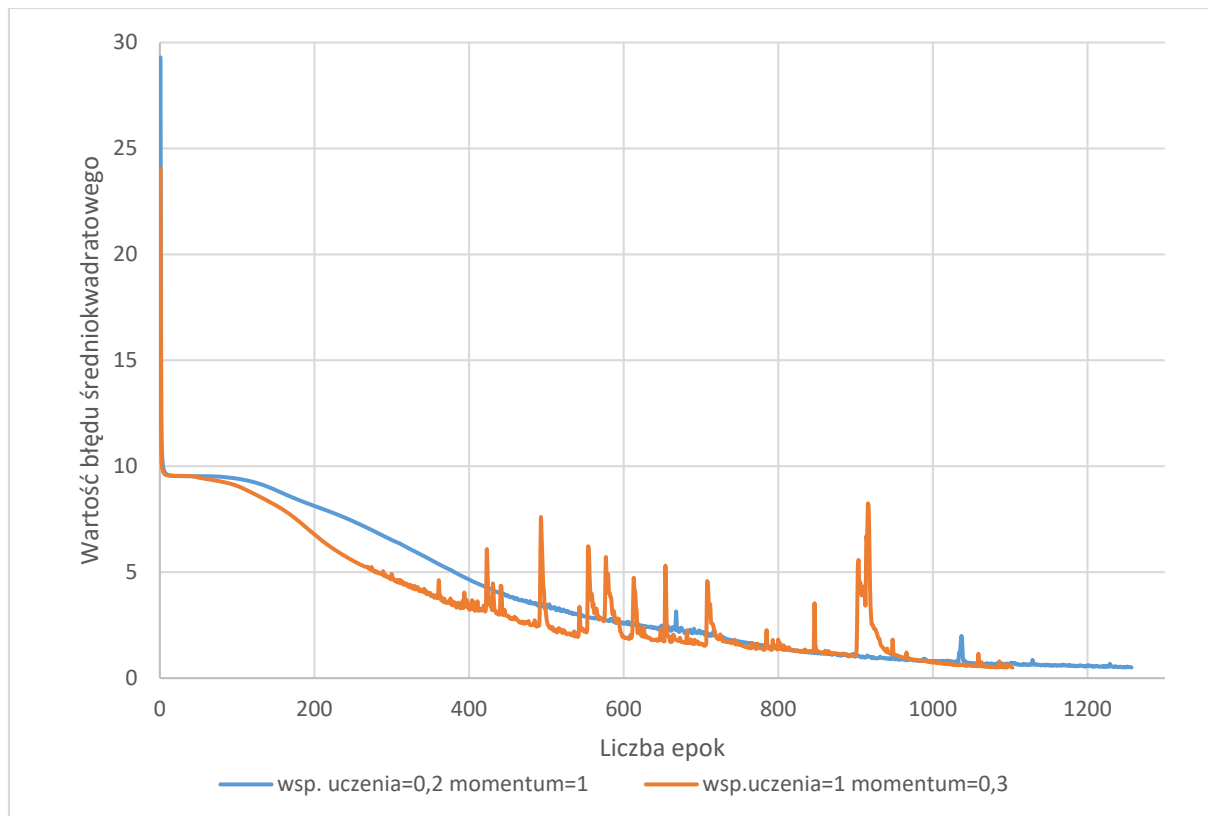
#### 4. Analiza wyników



Wykres 1. Zależność ilości epok potrzebnych do nauczenia sieci w zależności od współczynnika uczenia oraz wartości współczynnika bezwładności.

Analizując powyższy wykres możemy stwierdzić, że w procesie uczenia współczynnik uczenia ma znaczny wpływ na jego szybkość. Zauważalnym jest, że wraz ze wzrostem tegoż współczynnika obserwujemy spadek liczby epok potrzebnych sieci na naukę. Jednak nie tylko on wpływa na dany proces, kolejnym współczynnikiem jest momentum. Widzimy, że przy jego niskich wartościach nawet, gdy  $\eta$  ma wysokie wartości sieć nie uczy się efektywnie. Podobną sytuację możemy zauważyć w przypadku, gdy oba współczynniki osiągają wysokie wartości- sieć potrzebuje większej ilości epok do uczenia- na wykresie zauważalny jest nagły wzrost liczby epok potrzebnych do uczenia. Dodatkowo jeżeli wartości też zostaną znacznie zwiększone niski poziom błędu może być w ogóle nieosiągnięty. Najlepsze rezultaty osiągamy w środkowych wartościach dla obu współczynników.





Wykres 2. Spadek wartości błędu średniokwadratowego wraz z kolejnymi epokami przy różnych kombinacjach współczynników.

Na powyższym wykresie pokazane jest jak bardzo współczynnik bezwładności wpływa na wartości błędu średniokwadratowego. Widzimy, że przy niższym współczynniku bezwładności występują momentami bardzo duże i gwałtowne wartości błędu. Natomiast przy większym współczynniku linia przebiega dużo bardziej łagodnie, również występują wahania wartości błędu, ale nie są one tak duże i gwałtowne jak w poprzednim przypadku. Jest to kolejny dowód na to, że nieodpowiednio wybrane wartości współczynników mogą niekorzystnie wpływać na czas, wydajności i złożoność obliczeniową naszego algorytmu.

## 5. Wnioski

- Skuteczność procesu uczenia zależy od współczynnika uczenia oraz współczynnika bezwładności. Koniecznym jest takie dobieranie tych wartości, aby uniknąć bardzo powolnego przyrostu zmian, gdyż związane jest to z bardzo długim czasem przeprowadzania procesu uczenia, jednak nie narażamy się na sytuację, w której sieć w ogóle nie jest w stanie się nauczyć poprawnego działania. Dzieje się tak, gdy wartości współczynników są za duże i przyrost wag jest bardzo szybki, ale w momencie zbliżania się do zamierzonej granicy błędu wartość ta gwałtownie skacze do góry i nie jest możliwe uzyskanie poprawnie działającej sieci. Najlepiej uczyły się sieci przy średnich wartościach z przedziału  $(0,1>$  dla obu współczynników
- Kolejnym aspektem do przetestowania jest architektura sieci, o ile ilość neuronów w warstwie wejściowej i wyjściowej jest ustalana na podstawie ilości liter do nauki oraz to jak duża ma być tablica je reprezentująca to ilość neuronów w warstwie ukrytej należy odpowiednio dobrać. Przy zbyt małej czy dużej liczbie sieć nie uczy

się efektywnie lub wcale. Zalecane jest, aby ilość warstw ukrytych była jak najmniejsza.

- Stworzona sieć doskonale radzi sobie po odpowiednim nauczaniu z rozpoznawaniem liter, które się uczyła 100% poprawnych odpowiedzi, natomiast w przypadku liter spoza zakresu nauczania poziom poprawnych odpowiedzi wynosi 90% co pozwala sądzić, że przy odpowiedniej konstrukcji sieci wielowarstwowej jest ona w stanie w akceptowalnym stopniu radzić sobie z problemem.

## 6. Kod programu

main.cpp

```
#include <iostream>
#include <vector>
#include "Layer.h"
#include <time.h>
#include <fstream>
using namespace std;

void ustawWartosciWejscowe(Neuron neuron, vector< vector<int> > daneWejscowe, vector<double>
odpowiedziNeuronow, int liczbaWyjsc, int liczbaWejsc, int numerWarstwy, int rzad);
void ucz(vector<Layer> warstwy, vector< vector<int> > daneWejscowe);
void funkcjaWstecznejPropagacji(vector<Layer> warstwy);
void test(vector<Layer>& warstwy, vector< vector<int> > daneWejscowe);
void wczytajDaneUczace(vector< vector<int> > &wejscoweDaneUczace, int liczbaWejsc, int
liczbaWyjsc);
void wczytajDaneTestujace(vector< vector<int> > &testujaceDaneWejscowe, int liczbaWejsc, int
liczbaWyjsc);

fstream WYNIK_UCZENIA;
fstream WYNIK_TESTOWANIA;
fstream DANE_UCZACE;
fstream DANE_TESTUJACE;
double E;
double E_MAX = 0.5;
double E_TESTOWANIE;
int LICZBA_ZLYCH_ODPOWIEDZI;
int LICZBA_DOBRYCH_ODPOWIEDZI;
int LICZBA_PRAWIDLOWYCH_ODPOWIEDZI;
int *PRAWIDLOWE_ODPOWIEDZI;

int main()
{
    srand(time(NULL));
    vector<Layer> warstwy;
    vector< vector<int> > wejscoweDaneUczace;
    vector< vector<int> > testujaceDaneWejscowe;

    int liczbaNeuronowWPierwszejWarstwie = 20;
    int liczbaNeuronowWDrugiejWarstwie = 10;
    int liczbaNeuronowWTrzeciejWarstwie = 20;
```

```

int liczbaWejscWPierwszejWarstwie = 21;
int liczbaWejscWDrugiejWarstwie = 21;
int liczbaWejscWTrzeciejWarstwie = 11;
int liczbaWyjsc = 1;
double wspUczenia = 1.0;
double wspBezwladnosci = 1.0;

warstwy.push_back(Layer::Layer(liczbaNeuronowWPierwszejWarstwie,
liczbaWejscWPierwszejWarstwie, liczbaWyjsc, wspUczenia, wspBezwladnosci));
warstwy.push_back(Layer::Layer(liczbaNeuronowWDrugiejWarstwie,
liczbaWejscWDrugiejWarstwie, liczbaWyjsc, wspUczenia, wspBezwladnosci));
warstwy.push_back(Layer::Layer(liczbaNeuronowWTrzeciejWarstwie,
liczbaWejscWTrzeciejWarstwie, liczbaWyjsc, wspUczenia, wspBezwladnosci));
wczytajDaneUczace(wejscoweDaneUczace, liczbaWejscWPierwszejWarstwie, liczbaWyjsc);
wczytajDaneTestujace(testujaceDaneWejscowe, liczbaWejscWPierwszejWarstwie,
liczbaWyjsc);
PRAWIDLOWE_ODPOWIEDZI = new int[LICZBA_PRAWIDLOWYCH_ODPOWIEDZI];

do
{
    cout << "1. Ucz" << endl;
    cout << "2. Testuj" << endl;
    cout << "3. Wyjd" << endl;

    int wybor;
    cin >> wybor;
    switch (wybor)
    {
        case 1:
            WYNIK_UCZENIA.open("output_learning_data.txt", ios::out);

            for (int numerEpoki = 1; ;)
            {
                ucz(warstwy, wejscoweDaneUczace);
                WYNIK_UCZENIA << "NUMER EPOKI: " << numerEpoki << " E: " << E
<< endl;

                if (E < E_MAX)
                    break;
                numerEpoki++;

                E = 0;
            }

            break;

        case 2:
            WYNIK_TESTOWANIA.open("output_testing_data.txt", ios::out);
            LICZBA_ZLYCH_ODPOWIEDZI = 0;
            LICZBA_DOBRYCH_ODPOWIEDZI = 0;
            E_TESTOWANIE = 0;
            test(warstwy, testujaceDaneWejscowe);
            WYNIK_TESTOWANIA << "Liczba zlych odpowiedzi: " <<
LICZBA_ZLYCH_ODPOWIEDZI << endl;

```

```

        WYNIK_TESTOWANIA << "Liczba dobrych odpowiedzi: " <<
LICZBA_DOBRYCH_ODPOWIEDZI << endl;
        WYNIK_TESTOWANIA << "E: " << E_TESTOWANIE << endl;
        break;

        case 3:
            WYNIK_UCZENIA.close();
            WYNIK_TESTOWANIA.close();
            return 0;
        }
    } while (true);
    return 0;
}

void ustawWartosciWejscowe(Neuron& neuron, vector< vector<int> > daneWejscowe,
vector<double> odpowiedziNeuronow, int liczbaWyjsc, int liczbaWejsc, int numerWarstwy, int rzad)
{
    for (int i = 0; i < liczbaWejsc; i++)
    {
        if (numerWarstwy == 0)
            neuron.setWejscie(i, daneWejscowe[i][rzad]);
        else
        {
            if (i == 0)
                neuron.setWejscie(i, 1);
            else
                neuron.setWejscie(i, odpowiedziNeuronow[i - 1]);
        }
    }
    if (numerWarstwy == 2)
    {
        for (int i = 0, j = (daneWejscowe[0].size() - liczbaWyjsc); i <
LICZBA_PRAWIDLOWYCH_ODPOWIEDZI; i++, j++)
            PRAWIDLOWE_ODPOWIEDZI[i] = daneWejscowe[j][rzad];
    }
}

void funkcjaWstecznejPropagacji(vector<Layer>& warstwy)
{
    for (int numerWarstwy = (warstwy.size() - 1); numerWarstwy >= 0; numerWarstwy--)
    {
        for (int i = 0; i < warstwy[numerWarstwy].getLiczbaNeuronow(); i++)
        {
            if (numerWarstwy == (warstwy.size() - 1))

                warstwy[numerWarstwy].neurony[i].setBlad(PRAWIDLOWE_ODPOWIEDZI[i] -
warstwy[numerWarstwy].neurony[i].getWartoscWyjscowa());

            else
            {
                double blad = 0.0;

```

```

        for (int j = 0; j < warstwy[numerWarstwy + 1].getLiczbaNeuronow();
j++)
            blad += warstwy[numerWarstwy + 1].neurony[j].getBlad()
* warstwy[numerWarstwy + 1].neurony[j].getWaga(i);

        warstwy[numerWarstwy].neurony[i].setBlad(blad);
    }
}
}
}

```

```

void ucz(vector<Layer>& warstwy, vector< vector<int> > daneWejscowe)
{
    for (int rzadWejscia = 0; rzadWejscia < daneWejscowe.size(); rzadWejscia++)
    {
        for (int numerWarstwy = 0; numerWarstwy < warstwy.size(); numerWarstwy++)
        {
            for (int i = 0; i < warstwy[numerWarstwy].getLiczbaNeuronow(); i++)
            {
                ustawWartosciWejscowe(warstwy[numerWarstwy].neurony[i],
daneWejscowe, warstwy[(numerWarstwy > 0) ? (numerWarstwy - 1) : 0].getAnswers(),
warstwy[numerWarstwy].getLiczbaNeuronow(),
warstwy[numerWarstwy].neurony[i].getLiczbaWejsc(), numerWarstwy, rzadWejscia);
                warstwy[numerWarstwy].neurony[i].sumujWejscia();
                warstwy[numerWarstwy].neurony[i].obliczWyjscie(false);
            }
            warstwy[numerWarstwy].zbierzOdpowiedzi();

            if (numerWarstwy == (warstwy.size() - 1))
                for (int i = 0; i < warstwy[warstwy.size() - 1].getLiczbaNeuronow();
i++)
                    E += 0.5 * pow((((double)PRAWIDLOWE_ODPOWIEDZI[i] -
warstwy[warstwy.size() - 1].getAnswer(i)), 2);
        }
    }
}

```

funkcjaWstecznejPropagacji(warstwy);

```

    for (int numerWarstwy = 0; numerWarstwy < warstwy.size(); numerWarstwy++)
        for (int i = 0; i < warstwy[numerWarstwy].getLiczbaNeuronow(); i++)
            warstwy[numerWarstwy].neurony[i].obliczNoweWagi();
    }
}

```

```

void test(vector<Layer>& warstwy, vector< vector<int> > daneWejscowe)
{
    for (int rzad = 0; rzad < daneWejscowe.size(); rzad++)
    {
        for (int numerWarstwy = 0; numerWarstwy < warstwy.size(); numerWarstwy++)
        {
            for (int i = 0; i < warstwy[numerWarstwy].getLiczbaNeuronow(); i++)
            {

```

```

        ustawWartosciWejscowe(warstwy[numerWarstwy].neurony[i],
daneWejscowe, warstwy[(numerWarstwy > 0) ? (numerWarstwy - 1) : 0].getAnswers(),
warstwy[numerWarstwy].getLiczbaNeuronow(),
warstwy[numerWarstwy].neurony[i].getLiczbaWejsc(), numerWarstwy, rzad);
        warstwy[numerWarstwy].neurony[i].sumujWejscia();
        if (numerWarstwy != (warstwy.size()))
            warstwy[numerWarstwy].neurony[i].obliczWyjscie(false);

        else
            warstwy[numerWarstwy].neurony[i].obliczWyjscie(true);
    }

    warstwy[numerWarstwy].zbierzOdpowiedzi();
}

for (int i = 0; i < warstwy[warstwy.size()].getLiczbaNeuronow(); i++)
{
    if (warstwy[warstwy.size()].getAnswer(i) == PRAWIDLOWE_ODPOWIEDZI[i])
    {
        LICZBA_DOBRYCH_ODPOWIEDZI++;
        E_TESTOWANIE += 0.5 *
pow((((double)PRAWIDLOWE_ODPOWIEDZI[i] - warstwy[warstwy.size() - 1].getAnswer(i)), 2);
        WYNIK_TESTOWANIA << warstwy[warstwy.size() -
1].neurony[i].getWartoscWyjscowa() << " + ";
    }

    else
    {
        LICZBA_ZLYCH_ODPOWIEDZI++;
        E_TESTOWANIE += 0.5 *
pow((((double)PRAWIDLOWE_ODPOWIEDZI[i] - warstwy[warstwy.size() - 1].getAnswer(i)), 2);
        WYNIK_TESTOWANIA << warstwy[warstwy.size() -
1].neurony[i].getWartoscWyjscowa() << " - ";
    }
}

WYNIK_TESTOWANIA << endl;
}
}

```

```

void wczytajDaneUczace(vector< vector<int> > &wejscoweDaneUczace, int liczbaWejsc, int
liczbaWyjsc)
{
    DANE_UCZACE.open("learning_data.txt", ios::in);
    vector<int> rzad;

    do
    {
        rzad.clear();

        for (int i = 0, wejTmp = 0; i < liczbaWejsc; i++)
        {
            DANE_UCZACE >> wejTmp;

```

```

        rzad.push_back(wejTmp);
    }

    for (int i = 0, odpTmp; i < (liczbaWejsc - 1); i++)
    {
        DANE_UCZACE >> odpTmp;
        rzad.push_back(odpTmp);
    }

    wejscoweDaneUczace.push_back(rzad);
}while (!DANE_UCZACE.eof());

LICZBA_PRAWIDLOWYCH_ODPOWIEDZI = rzad.size();

DANE_UCZACE.close();
}

void wczytajDaneTestujace(vector< vector<int> > &testujaceDaneWejscowe, int liczbaWejsc, int
liczbaWyjsc)
{
    DANE_TESTUJACE.open("testing_data.txt", ios::in);
    vector<int> rzad;

    while (!DANE_TESTUJACE.eof())
    {
        rzad.clear();

        for (int i = 0, wejTmp; i < liczbaWejsc; i++)
        {
            DANE_TESTUJACE >> wejTmp;
            rzad.push_back(wejTmp);
        }

        for (int i = 0, odpTmp; i < (liczbaWejsc - 1); i++)
        {
            DANE_TESTUJACE >> odpTmp;
            rzad.push_back(odpTmp);
        }
        testujaceDaneWejscowe.push_back(rzad);
    }
    DANE_TESTUJACE.close();
}

```

## Neuron.h

```

#pragma once
#include <iostream>
#include <vector>
using namespace std;

```

```

class Neuron

```

```

{

```

```

public:

```

```

    void stworzWejscia(int liczbaWejsc, int liczbaWag);
    void stworzWejscie() { _wejscia.push_back(0); }

```

```

void stworzWage(int index) { _wagi.push_back(0); }
int getLiczbaWejsc() { return _wejscia.size(); }
int getLiczbaWag() { return _wagi.size(); }
double getWejscie(int index) { return _wejscia[index]; }
void setWejscie(int index, double value) { _wejscia[index] = value; }
double getWaga(int index) { return _wagi[index]; }
void setWaga(int index, double value) { _wagi[index] = value; }
double getSumaWszystkichWejsc() { return _sumaWszystkichWejsc; }
double getWartoscWyjsciowa() { return _wartoscWyjsciowa; }
double obliczPojedynczeWejscie(int index) { return _wejscia[index] * _wagi[index]; }
void sumujWejscia();
void obliczWyjscie(bool testowanie);
void obliczNoweWagi();
void setBlad(double blad) { _blad = blad; }
double getBlad() { return _blad; }
Neuron();
Neuron(int liczbaWejsc, int liczbaWyjsc, double wspUczenia, double wspBezwladnosci);

```

private:

```

double obliczPierwszeWagi();
vector<double> _wejscia; //inputs
vector<double> _wagi; //weights
vector<double> _poprzednieWagi;
double _sumaWszystkichWejsc;
double _wartoscWyjsciowa;
double _wspUczenia;
double _wspBezwladnosci;
double _blad;
};

```

## Neuron.cpp

```

#include "Neuron.h"
#include <time.h>
#include <math.h>

```

Neuron::Neuron()

```

{
    _wejscia.resize(0);
    _wagi.resize(0);
    _poprzednieWagi.resize(0);
    _sumaWszystkichWejsc = 0;
    _wartoscWyjsciowa = 0;
    _wspUczenia = 0;
    _wspBezwladnosci = 0;
}

```

Neuron::Neuron(int liczbaWejsc, int liczbaWyjsc, double wspUczenia, double wspBezwladnosci)

```

{
    stworzWejscia(liczbaWejsc, liczbaWyjsc);
    _wspUczenia = wspUczenia;
    _wspBezwladnosci = wspBezwladnosci;
    _sumaWszystkichWejsc = 0;
    _wartoscWyjsciowa = 0;
}

```



```

}

void Neuron::stworzWejscia(int liczbaWejsc, int liczbaWyjsc)
{
    double weight = obliczPierwszeWagi();
    for (int j = 0; j < liczbaWejsc; j++)
    {
        _wejscia.push_back(0);
        _poprzednieWagi.push_back(0);
        _wagi.push_back(obliczPierwszeWagi());
    }
}

void Neuron::sumujWejscia()
{
    _sumaWszystkichWejsc = 0.0;
    for (int i = 0; i < getLiczbaWejsc(); i++)
        _sumaWszystkichWejsc += obliczPojedynczeWejscie(i);
}

void Neuron::obliczWyjscie(bool testowanie)
{
    if (testowanie == true)
    {
        if (_sumaWszystkichWejsc > 0)
            _wartoscWyjsciowa = 1;
        else
            _wartoscWyjsciowa = 0;
    }
    else
    {
        double beta = 1.0;
        _wartoscWyjsciowa = (1.0 / (1.0 + (exp(-beta * _sumaWszystkichWejsc))));
    }
}

void Neuron::obliczNoweWagi()
{
    for (int i = 0; i < getLiczbaWejsc(); i++)
        _wagi[i] += _wspBezwladnosci * _wspUczenia * _blad * (_wartoscWyjsciowa * (1 -
        _wartoscWyjsciowa) * _wejscia[i]);
}

double Neuron::obliczPierwszeWagi()
{
    double max = 0.5;
    double min = -0.5;
    double weight = ((double(rand()) / double(RAND_MAX)) * (max - min)) + min;
    return weight;
}

```

Layer.h

#pragma once

```

#include<vector>
#include"Neuron.h"
using namespace std;

class Layer
{
public:
    Layer(int liczbaNeuronow, int liczbaWejsc, int liczbaWyjsc, double wspUczenia, double
wspBezwladnosci);
    vector<Neuron> neurony;
    int getLiczbaNeuronow(){ return _liczbaNeuronow; }
    void zbierzOdpowiedzi();
    vector<double> getAnswers() { return _odpowiedzi; }
    double getAnswer(int index) { return _odpowiedzi[index]; }

private:
    int _liczbaNeuronow;
    double _wspUczenia;
    vector<double> _odpowiedzi;
};

```

### Layer.cpp

```

#include"Layer.h"

Layer::Layer(int liczbaNeuronow, int liczbaWejsc, int liczbaWyjsc, double wspUczenia, double
wspBezwladnosci)
{
    _liczbaNeuronow = liczbaNeuronow;
    _wspUczenia = wspUczenia;
    neurony.resize(liczbaNeuronow);
    for (int i = 0; i < liczbaNeuronow; i++)
        neurony[i].Neuron::Neuron(liczbaWejsc, liczbaWyjsc, _wspUczenia, wspBezwladnosci);
}

void Layer::zbierzOdpowiedzi()
{
    _odpowiedzi.clear();
    for (int i = 0; i < _liczbaNeuronow; i++)
        _odpowiedzi.push_back(neurony[i].getWartoscWyjsciowa());
}

```