

## Scenariusz 4

**Temat ćwiczenia:** Uczenie sieci regułą Hebba

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

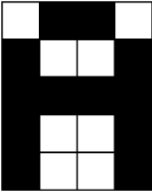
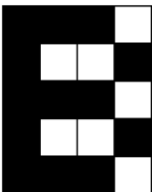
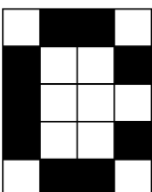
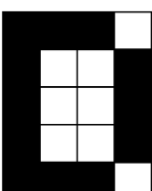
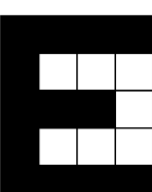
### 1. Cel ćwiczenia

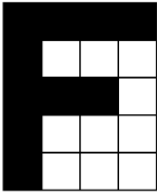
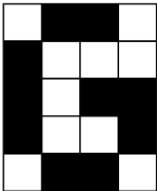
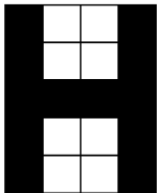
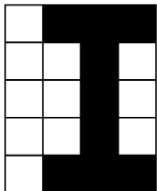
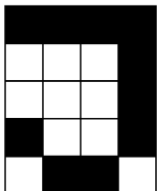
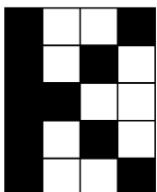
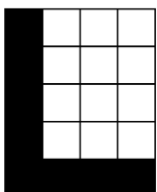
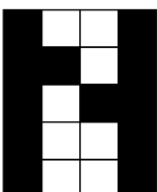
Celem ćwiczenia jest poznanie reguły Hebba dla sieci jednowarstwowej na przykładzie grupowania liter alfabetu.

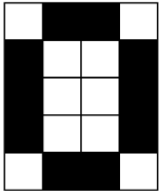
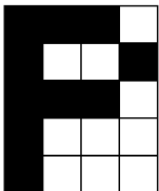
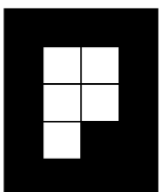
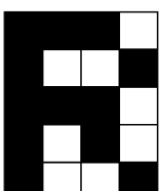
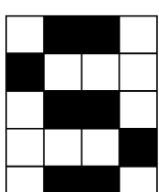
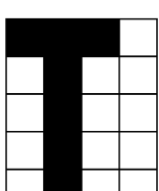
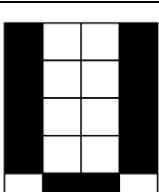
### 2. Opis budowy sieci i algorytmów uczenia.

Celem budowanej sieci jest podział wzorców uczących na klasy obrazów zbliżonych do siebie i przyporządkowanie każdej klasie osobnego elementu wyjściowego. Podział na grupy odbywa się w ten sposób, by elementy w tej samej grupie były do siebie podobne a jednocześnie jak najbardziej odmienne od elementów z pozostałych grup.

Tab. 1. Wykorzystane litery.

	Wektor wejściowy: 0 1 1 0 1 0 0 1 1 1 1 1 1 0 0 1 1 0 0 1
	Wektor wejściowy: 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0
	Wektor wejściowy: 0 1 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 0
	Wektor wejściowy: 1 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 0
	Wektor wejściowy: 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 0 1 1 1 1

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

	Wektor wejściowy: 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0
	Wektor wejściowy: 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0 0 1 0 0 0
	Wektor wejściowy: 1 1 1 1 1 0 0 1 1 0 0 1 1 0 1 1 1 1 1 1
	Wektor wejściowy: 1 1 1 0 1 0 0 1 1 1 1 0 1 0 1 0 1 0 0 1
	Wektor wejściowy: 0 1 1 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 0
	Wektor wejściowy: 1 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
	Wektor wejściowy: 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0

Aby możliwe była klasyfikacja liter na poszczególne grupy zostały odliczone odległości euklidesowe między poszczególnymi literami według wzoru:

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Tab. 2. Otrzymane wartości odległości euklidesowej między poszczególnymi literami.

	A	B	C	D	E	F	G	H	I	J	K	L	N	O	P	Q	R	S	T	U
A	x	2,24	2,65	2,45	2,45	2,45	2,45	2,00	3,61	3,16	3,16	3,16	2,45	2,45	2,00	2,65	2,00	2,83	3,61	3,16
B	2,24	x	2,00	1,73	2,00	2,24	2,24	2,65	3,46	2,65	3,32	2,65	3,00	2,24	1,73	2,45	2,00	2,24	3,16	2,65
C	2,65	2,00	x	1,73	2,83	3,00	1,73	3,32	3,16	2,24	3,61	2,65	3,32	1,00	2,65	2,45	3,00	2,24	3,16	2,24
D	2,45	1,73	1,73	x	2,65	2,83	2,00	2,83	3,61	2,00	3,46	2,45	2,83	1,41	2,45	1,73	2,83	2,83	3,32	2,00
E	2,45	2,00	2,83	2,65	x	1,73	2,65	2,65	2,83	3,00	2,65	2,24	3,00	3,00	2,24	2,45	2,24	2,65	3,16	3,00
F	2,45	2,24	3,00	2,83	1,73	x	2,83	2,45	3,32	3,16	2,45	2,83	2,83	3,16	1,41	3,00	2,00	2,83	3,00	3,16
G	2,45	2,24	1,73	2,00	2,65	2,83	x	3,16	3,00	2,45	3,74	2,83	3,16	1,41	2,83	2,65	3,16	2,00	3,32	2,45
H	2,00	2,65	3,32	2,83	2,65	2,45	3,16	x	3,87	3,16	2,45	2,83	1,41	3,16	2,45	2,65	2,65	3,46	3,87	2,45
I	3,61	3,46	3,16	3,61	2,83	3,32	3,00	3,87	x	3,00	3,32	3,32	3,87	3,32	3,61	3,16	3,32	2,65	3,16	3,61
J	3,16	2,65	2,24	2,00	3,00	3,16	2,45	3,16	3,00	x	3,74	3,16	3,16	2,00	3,16	2,24	3,46	2,83	3,00	2,00
K	3,16	3,32	3,61	3,46	2,65	2,45	3,74	2,45	3,32	3,74	x	2,45	2,83	3,74	2,83	3,00	2,45	3,74	3,61	3,16
L	3,16	2,65	2,65	2,45	2,24	2,83	2,83	2,83	3,32	3,16	2,45	x	2,83	2,83	2,83	2,65	2,83	3,16	3,32	2,45
N	2,45	3,00	3,32	2,83	3,00	2,83	3,16	1,41	3,87	3,16	2,83	2,83	x	3,16	2,83	2,65	2,83	3,74	3,87	2,45
O	2,45	2,24	1,00	1,41	3,00	3,16	1,41	3,16	3,32	2,00	3,74	2,83	3,16	x	2,83	2,24	3,16	2,45	3,32	2,00
P	2,00	1,73	2,65	2,45	2,24	1,41	2,83	2,45	3,61	3,16	2,83	2,83	2,83	2,83	x	3,00	1,41	2,83	3,00	3,16
Q	2,65	2,45	2,45	1,73	2,45	3,00	2,65	2,65	3,16	2,24	3,00	2,65	2,65	2,24	3	x	2,65	3,32	3,87	2,24
R	2,00	2,00	3,00	2,83	2,24	2,00	3,16	2,65	3,32	3,46	2,45	2,83	2,83	3,16	1,41	2,65	x	3,16	3,32	3,46
S	2,83	2,24	2,24	2,83	2,65	2,83	2,00	3,46	2,65	2,83	3,74	3,16	3,74	2,45	2,83	3,32	3,16	x	2,65	3,16
T	3,61	3,16	3,16	3,32	3,16	3,00	3,32	3,87	3,16	3,00	3,61	3,32	3,87	3,32	3,00	3,87	3,32	2,65	x	3,61
U	3,16	2,65	2,24	2,00	3,00	3,16	2,45	2,45	3,61	2,00	3,16	2,45	2,45	2,00	3,16	2,24	3,46	3,16	3,61	x

Metoda uczenia Hebba może odbywać się według reguły z nauczycielem i bez nauczyciela. Wykorzystano tą drugą opcję, więc sygnałem uczącym jest po prostu sygnał wyjściowy neuronu. Litery zostały podzielone po analizie powyższej tabeli na grupy pokazane poniżej. Im odległość euklidesowa między poszczególnymi literami mniejsza tym bardziej są one podobne do siebie.

Tab. 3 Podział na grupy na podstawie odległości euklidesowej

Lp. grupy	Litery należące do grupy
1	A
2	B C D O
3	E F P R
4	G S
5	H N
6	I
7	J U
8	K
9	Q
10	T

Stworzona sieć to sieć jednowarstwowa składająca się z 10 neuronów, tak aby podział na grupy był jak najbardziej zbliżony i możliwy do porównania z podziałem wyznaczonym samodzielnie.

Proces uczenia przebiega według następującego schematu w danej epoce uczenia:

1. Wybór  $\eta$  za zakresu 0 do 1
2. Wybór początkowych wartości wag, jako niewielkich liczb losowych z zakresu -1 do 1
3. Dla danego wektora uczącego obliczymy odpowiedź sieci. A więc dla pojedynczego neuronu obliczana jest suma ilorazów sygnałów wejściowych i wag -  $u_i^k$ . Następnie obliczamy wartość wyjściową neuronu  $a_i^k$  wykorzystując tą wartość za pomocą unipolarnej, binarnej funkcji aktywacji.
4. Modyfikacja wag odbywa się według następujących zależności:

$$w_{i,j}^k(t+1) = (w_{ij}(t)) + \eta \cdot x_i \cdot a_i$$

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

5. W przypadku wykorzystywania współczynnika zapominania modyfikacja wag odbywa się według wzoru:

$$w_{i,j}^k(t+1) = (w_{ij}(t)) \cdot (1 - \gamma) + \eta \cdot x_i \cdot a_i$$

$\gamma$  to współczynnik zapominania wybierany z zakresu 0 do 1 (zwykle 0,1)

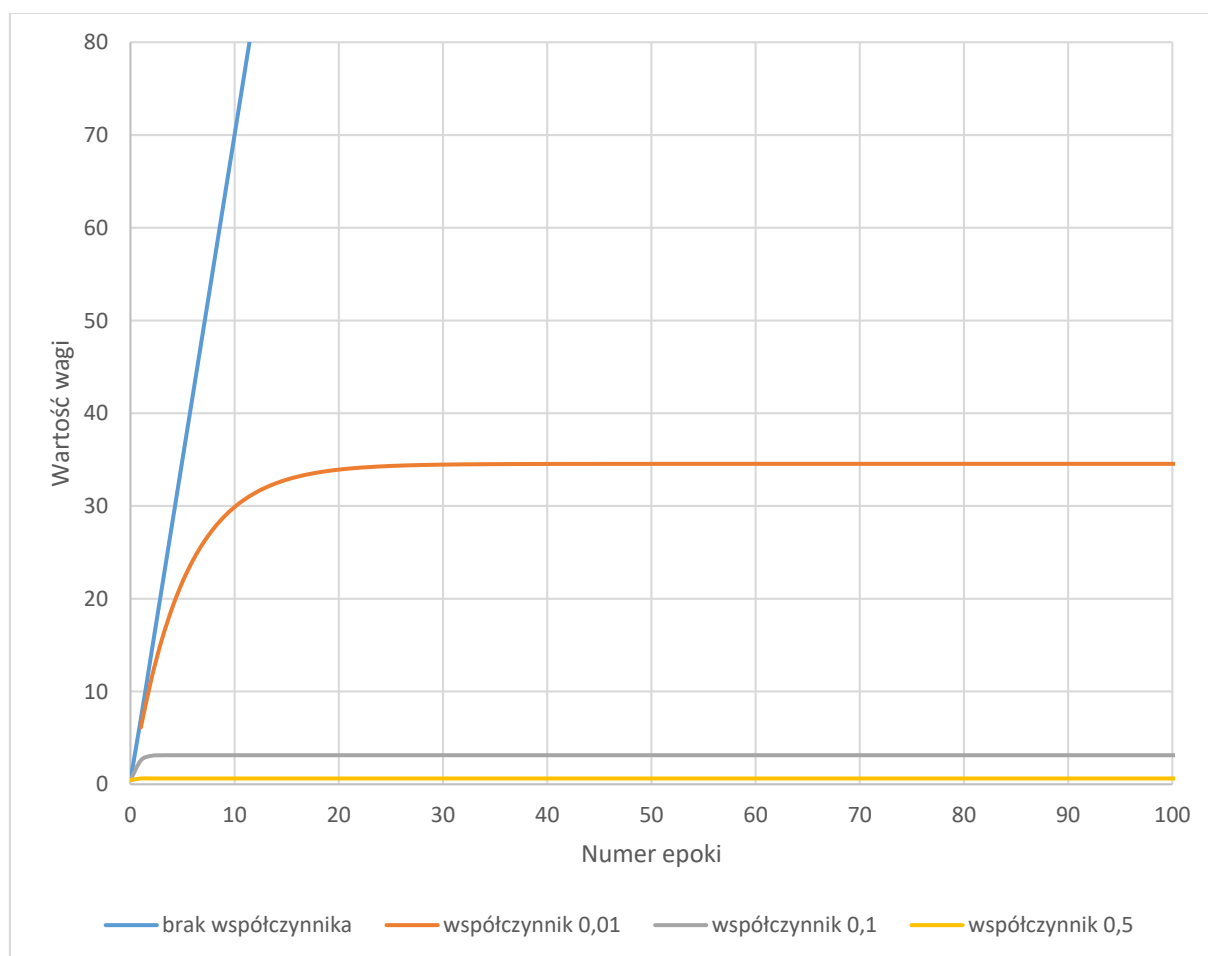
6. Obliczenie łącznego błędu epoki:

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

7. Uczenie odbywa się do momentu stabilizacji wag.

### 3. Otrzymane wyniki

Zestaw liter podzielono na 15 liter do uczenia sieci oraz 5 do testowania. Już w trakcie nauki zauważalne były różnice między poszczególnymi wariantami uczenia. Poniżej został pokazany wykres jak zachowywała się pojedyncza waga w procesie uczenia w każdej epoce w zależności od współczynnika zapominania.



Wykres 1. Zależność wartości wagi w poszczególnych epokach zależnie od współczynnika zapominania

Tab. 4 Stabilizacja procesu uczenia w zależności od współczynnika zapominania

Współczynnik zapominania	Liczba epok potrzebnych do stabilizacji
0	1700
0,001	480
0,005	149
0,01	65
0,05	15
0,1	8
0,5	1

Tab. 5 Zależność liczby epoki potrzebnych w procesie uczenie do współczynnika uczenia

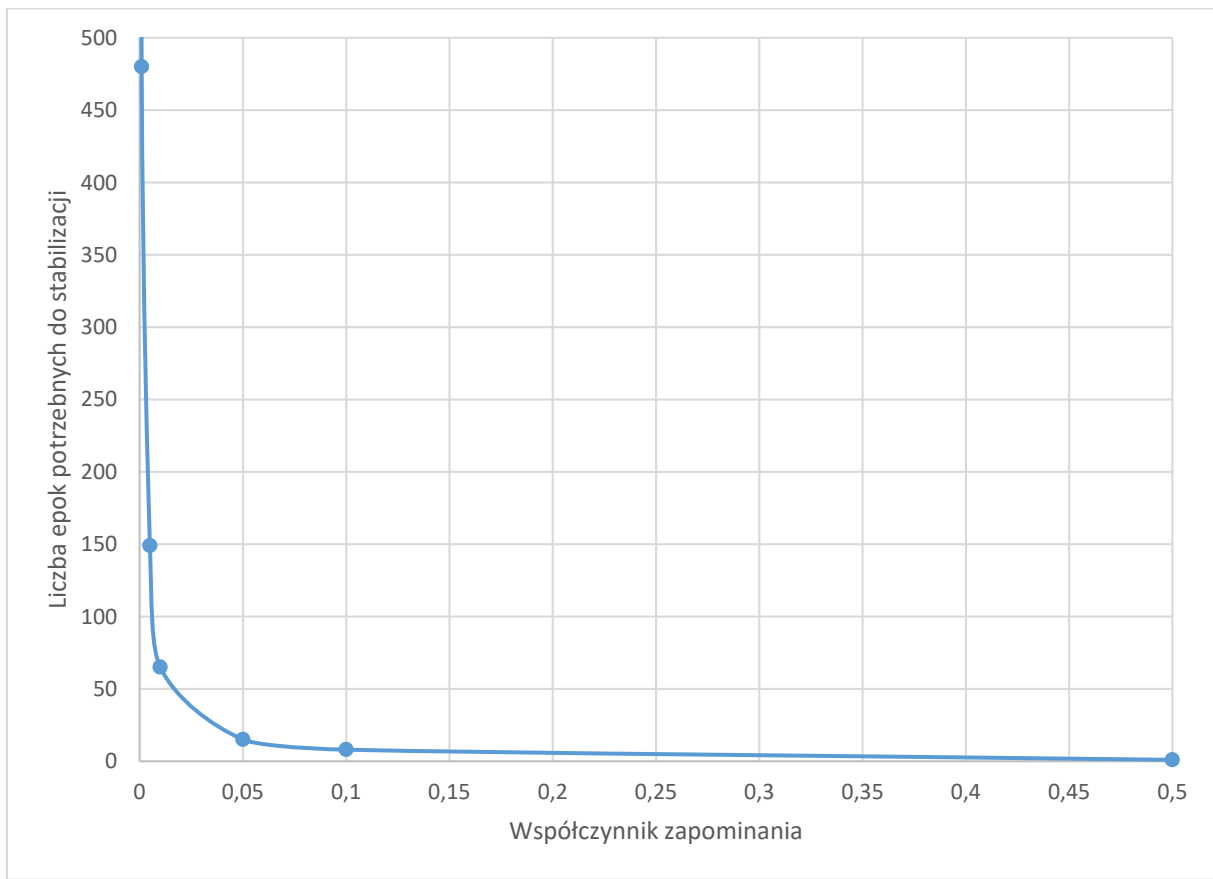
Współczynnik uczenia	liczba epok potrzebnych do nauczania
0,001	1675
0,005	850
0,01	581
0,05	426
0,1	350
0,5	195
1	121

W fazie testowania wykorzystano litery wcześniej nie używane w czasie uczenia sieci. Odpowiedzi sieci były w miarę zgodne z wcześniej założonym na podstawie odległości euklidesowych podziałem na grupy. Otrzymano następujące wyniki:

Tab. 6 Zestawienie wyników z fazy testowania

Testowana litera	Odpowiedź sieci klasyfikująca do tych samych grup co następujące litery
B	C D E P R
O	C D G J U
P	A B F R
S	G
N	H

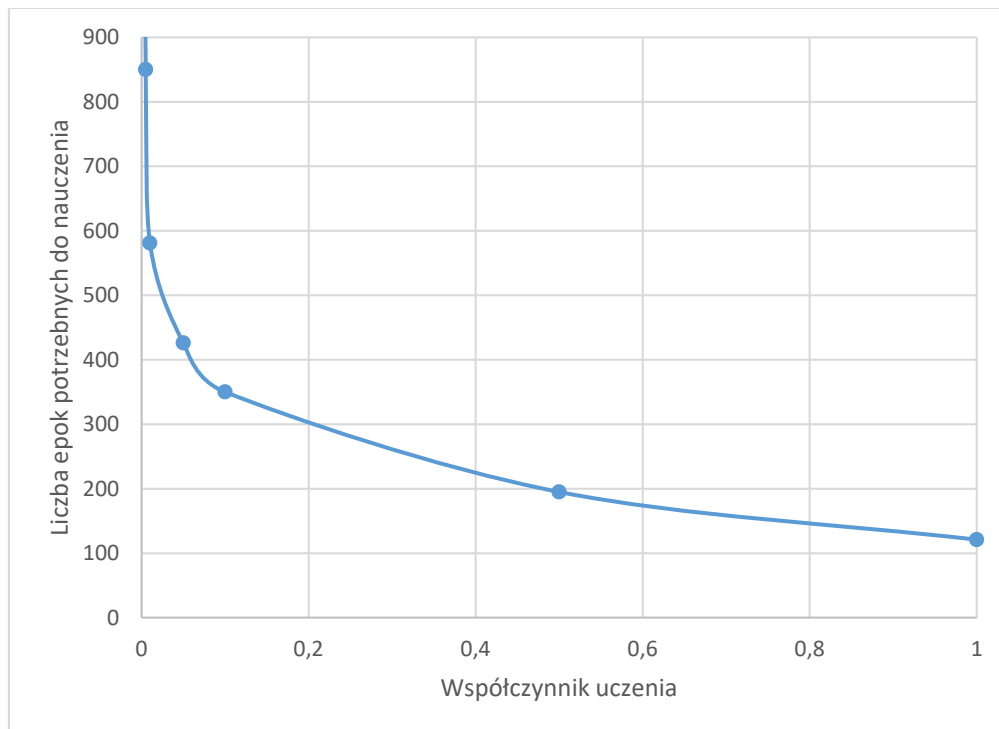
#### 4. Analiza wyników



Wykres 2. Zależność liczby epok potrzebnej do stabilizacji od współczynnika zapominania

Analizując wykres nr 1 oraz 2 widzimy, że przy braku współczynnika uczenia wagi ciągle rosną i nie następuje w ogóle moment stabilizacji. I tak po 100 epokach współczynnik osiąga wartość równą 700,006. Natomiast w pozostałych przypadkach następuje stabilizacja. Im współczynnik jest większy tym stabilizacja następuje szybciej. Oznacza to że należy rozważyć dobieranie jego wartości, bo przy zbyt dużym możemy zbyt szybko zakończyć proces uczenia. Bowiem w tej metodzie nauczanie trwa aż do momentu stabilizacji wag. I tak dla współczynnika na poziomie 0,5 widzimy że już w drugiej epoce następuje stabilizacja a niemożliwym jest aby sieć nauczyła się poprawnego grupowania tak szybko.





Wykres 3. Zależność ilości epok potrzebnych do nauczenia sieci w zależności od współczynnika uczenia.

Na powyższym wykresie widzimy, że przy bardzo małych współczynnikach uczenia liczba epok jest co najmniej 10 razy większa. Dokładnie wartość potrzebnych epok do całkowitego nauczenia dla współczynnika uczenia równego 0,001 wynosi 1675. Zgodnie z regułą delty wagi były zwiększane o bardzo małe współczynniki, a więc i sieć uczyła się bardzo wolno.

## 5. Wnioski

- Zmiany wag w dużym stopniu zależą od współczynnika zapominania. Gdy go nie mamy nie ma stabilizacji i wagi mogą rosnąć w nieskończoność. Lepiej radzi sobie sieć w której występuję współczynnik zapominania, który zdecydowanie poprawia stabilność procesu uczenia.
- Współczynnik zapominania powinien być niewielką wartością. Przyjęcie dużej jego wartości powoduje, że neuron zapomina większość tego, co zdołał nauczyć się w przeszłości. Jego wartości nie powinna przekraczać 0,1, dzięki temu neuron zachowuje większość informacji zgromadzonej w procesie uczenia, a jednocześnie możliwe jest ustabilizowanie wag na określonym poziomie.
- 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, a więc i proces uczenia przebiega szybciej.
- Stworzona sieć według metodyki Hebba jest w stanie poprawnie grupować litery. Nie wymaga ona nauczyciela oraz możliwe jest stosowanie różnorodnych funkcji aktywacji co dodatkowo daje wiele swobody. Bez problemu była w stanie nauczyć

się w jakim stopniu poszczególne litery są do siebie podobne i wykorzystać tę wiedzę w późniejszym testowaniu na literach wcześniej nie uczonych. Pozwala to sądzić, że przy odpowiedniej konstrukcji sieci jest ona w stanie w akceptowalnym stopniu radzić sobie z problemem grupowania liter.

## 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<double> > inputData, int numberOfOutputs, int
numberOfEntrances, int inputDataRow);
void ucz(Warstwa& layer, vector< vector<double> > inputData);
void testowanie(Warstwa& layer, vector< vector<double> > inputData);
void wczytajDaneUczace(vector< vector<double> > &learningInputData, int numberOfEntrances, int
amountOfOutputs);
void wczytajDaneTestujace(vector< vector<double> > &testingInputData, int numberOfEntrances, int
amountOfOutputs);

fstream PLIK_WYJSCIOWY_UCZENIE;
fstream PLIK_WYJSCIOWY_TESTOWANIE;
fstream DANE_UCZACE;
fstream DANE_TESTUJACE;

int main()
{
    srand(time(NULL));
    vector< vector<double> > learningInputData;
    vector< vector<double> > testingInputData;

    int liczbaNuronow = 10;
    int liczbaWejsc = 11;
    int numberWyjsc = 1;
    double wspolczynnikUczenia = 1.0;
    double wspolczynnikZapominania = 0.002;
    Warstwa hebbianNetwork(liczbaNuronow, liczbaWejsc, numberWyjsc, wspolczynnikUczenia,
wspolczynnikZapominania);

    wczytajDaneUczace(learningInputData, liczbaWejsc, numberWyjsc);
    wczytajDaneTestujace(testingInputData, liczbaWejsc, numberWyjsc);

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

```

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

            for (int liczbaEpok = 1, i = 0; i < 500 ;i++)
            {
                ucz(hebbianNetwork, learningInputData);
                PLIK_WYJSCIOWY_UCZENIE << hebbianNetwork.neurony[2].wezSynapse(2)

                PLIK_WYJSCIOWY_UCZENIE << "EPOCH NUMBER: " << liczbaEpok << endl;
                liczbaEpok++;
            }

            break;

        case 2:
            PLIK_WYJSCIOWY_TESTOWANIE.open("output_testing_data.txt", ios::out);
            testowanie(hebbianNetwork, testingInputData);
            break;

        case 3:
            PLIK_WYJSCIOWY_UCZENIE.close();
            PLIK_WYJSCIOWY_TESTOWANIE.close();
            return 0;
        }

    } while (true);

    return 0;
}

void ustawWartosciWejscowe(Neuron& neuron, vector< vector<double> > inputData, int numberOfOutputs, int
numberOfEntrances, int row)
{
    for (int i = 0; i < numberOfEntrances; i++)
        neuron.ustawWejscia(i, inputData[row][i]);
}

void ucz(Warstwa& layer, vector< vector<double> > inputData)
{
    for (int inputDataRow = 0; inputDataRow < inputData.size(); inputDataRow++)
    {
        for (int i = 0; i < layer.wezLiczbeNuronow(); i++)
        {
            ustawWartosciWejscowe(layer.neurony[i], inputData, layer.wezLiczbeNuronow(),
layer.neurony[i].wezLiczbeDendrytow(), inputDataRow);
            layer.neurony[i].sumujWejscia();
            layer.neurony[i].zapisz(false);
            layer.neurony[i].liczNoweWagi();

```

```

    }

    //OUTPUT_LEARNING_FILE << "\nNEW WEIGHTS" << endl;
    /*for (int i = 0; i < layer.getNumberOfNeurons(); i++)
    {
        //OUTPUT_LEARNING_FILE << "\nNEW NEURON" << i << endl;

        for (int j = 0; j < layer.neurons[i].getDendritesAmount(); j++)
        {
            if ( i == 2 && j == 2)
                OUTPUT_LEARNING_FILE << layer.neurons[i].getSynapse(j) << endl;
        }
    }*/
}
}

```

```

void testowanie(Warstwa& layer, vector< vector<double> > inputData)
{
    for (int inputDataRow = 0; inputDataRow < inputData.size(); inputDataRow++)
    {
        for (int i = 0; i < layer.wezLiczbeNuronow(); i++)
        {
            ustawWartosciWejscowe(layer.neurony[i], inputData, layer.wezLiczbeNuronow(),
layer.neurony[i].wezLiczbeDendrytow(), inputDataRow);
            layer.neurony[i].sumujWejscia();
            layer.neurony[i].zapisz(true);
        }

        layer.wlasciwaOdpowiedz();

        for (int i = 0; i < layer.wezLiczbeNuronow(); i++)
            PLIK_WYJSCIOWY_TESTOWANIE << layer.neurony[i].wezWartoscWyjsciowa() << " ";

        PLIK_WYJSCIOWY_TESTOWANIE << endl;
    }
}

```

```

void wczytajDaneUczace(vector< vector<double> > &learningInputData, int numberOfEntrances, int
amountOfOutputs)
{
    DANE_UCZACE.open("learning_data.txt", ios::in);
    vector<double> row;

    do
    {
        row.clear();

        for (int i = 0, inputTmp = 0; i < numberOfEntrances; i++)
        {
            DANE_UCZACE >> inputTmp;
            row.push_back(inputTmp);
        }
    }
}

```

```

        /*for (int i = 0, answerTmp; i < (numberOfEntrances - 1); i++)
        {
            LEARNING_DATA >> answerTmp;
            row.push_back(answerTmp);
        }*/

        learningInputData.push_back(row);
    } while (!DANE_UCZACE.eof());

    DANE_UCZACE.close();
}

void wczytajDaneTestujace(vector< vector<double> > &testingInputData, int numberOfEntrances, int
amountOfOutputs)
{
    DANE_TESTUJACE.open("testing_data.txt", ios::in);
    vector<double> row;

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

        for (int i = 0, inputTmp; i < numberOfEntrances; i++)
        {
            DANE_TESTUJACE >> inputTmp;
            row.push_back(inputTmp);
        }

        /*for (int i = 0, answerTmp; i < (numberOfEntrances - 1); i++)
        {
            TESTING_DATA >> answerTmp;
            row.push_back(answerTmp);
        }*/

        testingInputData.push_back(row);
    }
    DANE_TESTUJACE.close();
}

```

## Neuron.h

```

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

class Neuron
{
public:
    void stworzWejscia(int amountOfDendrites, int amountOfSynapses);
    void stworzDendryty() { _dendryty.push_back(0); }
    void stworzSynapsy(int index) { _synapsy.push_back(0); }

```

```

int wezLiczbeDendrytow() { return _dendryty.size(); }
int wezLiczbeSynaps() { return _synapsy.size(); }
double wezWejscia(int index) { return _dendryty[index]; }
void ustawWejscia(int index, double value) { _dendryty[index] = value; }
double wezSynapse(int index) { return _synapsy[index]; }
void ustawSynapse(int index, double value) { _synapsy[index] = value; }
double wezSumeWejsc() { return _sumaWejsc; }
double wezWartoscWyjsciowa() { return _wartosciWyjsciowe; }
double wczytaj(int index) { return _dendryty[index] * _synapsy[index]; }
void sumujWejscia();
void zapisz(bool testing);
void liczNoweWagi();
void setErrorSignal(double errorSignal) { _error = errorSignal; }
double getErrorSignal() { return _error; }

Neuron();
Neuron(int amountOfDendrites, int amountOfOutputs, double learningCoefficient, double
forgettingCoefficient);

private:
    double _liczWagi();
    vector<double> _dendryty;
    vector<double> _synapsy;
    double _sumaWejsc;
    double _wartosciWyjsciowe;
    double _WspolczynnikUczenia;
    double _wspolczynnikZapominania;
    double _error;
};

```

## Neuron.cpp

```

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

Neuron::Neuron()
{
    _dendryty.resize(0);
    _synapsy.resize(0);
    _sumaWejsc = 0.0;
    _wartosciWyjsciowe = 0.0;
    _WspolczynnikUczenia = 0.0;
    _wspolczynnikZapominania = 0.0;
}

Neuron::Neuron(int amountOfDendrites, int amountOfOutputs, double learningCoefficient, double
forgettingCoefficient)
{
    stworzWejscia(amountOfDendrites, amountOfOutputs);
    _WspolczynnikUczenia = learningCoefficient;
    _wspolczynnikZapominania = forgettingCoefficient;
}

```

```

        _sumaWejsc = 0.0;
        _wartosciWyjscowe = 0.0;
    }

void Neuron::stworzWejscia(int amountOfDendrites, int amountOfOutputs)
{
    for (int j = 0; j < amountOfDendrites; j++)
    {
        _dendryty.push_back(0);
        _synapsy.push_back(_liczWagi());
    }
}

void Neuron::sumujWejscia()
{
    _sumaWejsc = 0.0;
    for (int i = 0; i < wezLiczbeDendrytow(); i++)
        _sumaWejsc += wczytaj(i);
}

void Neuron::zapisz(bool testing)
{
    if (_sumaWejsc > 0)
        _wartosciWyjscowe = 1.0;
    else
        _wartosciWyjscowe = 0.0;
}

void Neuron::liczNoweWagi()
{
    for (int i = 0; i < wezLiczbeDendrytow(); i++)
        _synapsy[i] = _synapsy[i] * (1.0 - _wspolczynnikZapominania) + _WspolczynnikUczenia *
        _dendryty[i] * _wartosciWyjscowe;
}

double Neuron::_liczWagi()
{
    double max = 1.0;
    double min = -1.0;
    double weight = ((double(rand()) / double(RAND_MAX)) * (max - min)) + min;
    return weight;
}

```

## Warstwa.h

```

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

class Warstwa
{

```

```

public:
    Warstwa(int numberOfNeurons, int amountOfDendrites, int amountOfOutputs, double
learningCoefficient, double forgettingCoefficient);
    vector<Neuron> neurony;
    int wezLiczbeNuronow() { return _liczbaNuronow; }
    void wlasciwaOdpowiedz();
    vector<double> wezOdpowiedz() { return _odpowiedz; }
    double getAnswer(int index) { return _odpowiedz[index]; }

private:
    int _liczbaNuronow;
    vector<double> _odpowiedz;
};

```

## Warstwa.cpp

```

#include "Warstwa.h"

Warstwa::Warstwa(int numberOfNeurons, int amountOfDendrites, int amountOfOutputs, double
learningCoefficient, double forgettingCoefficient)
{
    _liczbaNuronow = numberOfNeurons;
    neurony.resize(numberOfNeurons);
    for (int i = 0; i < numberOfNeurons; i++)
        neurony[i].Neuron::Neuron(amountOfDendrites, amountOfOutputs, learningCoefficient,
forgettingCoefficient);
}

void Warstwa::wlasciwaOdpowiedz()
{
    _odpowiedz.clear();
    for (int i = 0; i < _liczbaNuronow; i++)
        _odpowiedz.push_back(neurony[i].wezWartoscWyjsciowa());
}

```