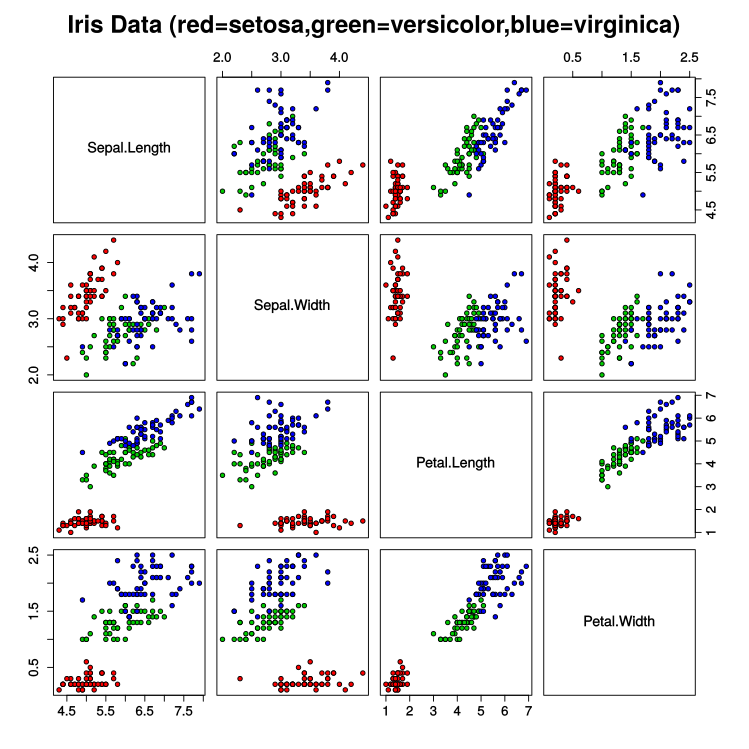
**Mateusz Bożek  
Scenariusz 5**  
**Temat :** Budowa i działanie sieci Kohonena dla WTA.

**Cel ćwiczenia** Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTA do odwzorowywania istotnych cech kwiatów.

**1.Syntetyczny opis budowy i wykorzystania sieci i algorytmu uczenia.**

| **Sepal length** | **Sepal width** | **Petal length** | **Petal width** | **Species** |
| --- | --- | --- | --- | --- |
| 5.2 | 3.5 | 1.4 | 0.2 | *I. setosa* |
| 4.9 | 3.0 | 1.4 | 0.2 | *I. setosa* |
| 4.7 | 3.2 | 1.3 | 0.2 | *I. setosa* |
| 4.6 | 3.1 | 1.5 | 0.2 | *I. setosa* |
| 5.0 | 3.6 | 1.4 | 0.3 | *I. setosa* |
| 5.4 | 3.9 | 1.7 | 0.4 | *I. setosa* |
| 4.6 | 3.4 | 1.4 | 0.3 | *I. setosa* |
| 5.0 | 3.4 | 1.5 | 0.2 | *I. setosa* |
| 4.4 | 2.9 | 1.4 | 0.2 | *I. setosa* |
| 4.9 | 3.1 | 1.5 | 0.1 | *I. setosa* |
| 5.4 | 3.7 | 1.5 | 0.2 | *I. setosa* |
| 4.8 | 3.4 | 1.6 | 0.2 | *I. setosa* |
| 4.8 | 3.0 | 1.4 | 0.1 | *I. setosa* |
| 4.3 | 3.0 | 1.1 | 0.1 | *I. setosa* |
| 5.8 | 4.0 | 1.2 | 0.2 | *I. setosa* |
| 5.7 | 4.4 | 1.5 | 0.4 | *I. setosa* |
| 5.4 | 3.9 | 1.3 | 0.4 | *I. setosa* |
| 5.1 | 3.5 | 1.4 | 0.3 | *I. setosa* |
| 5.7 | 3.8 | 1.7 | 0.3 | *I. setosa* |
| 5.1 | 3.8 | 1.5 | 0.3 | *I. setosa* |
| 5.4 | 3.4 | 1.7 | 0.2 | *I. setosa* |
| 5.1 | 3.7 | 1.5 | 0.4 | *I. setosa* |
| 4.6 | 3.6 | 1.0 | 0.2 | *I. setosa* |
| 5.1 | 3.3 | 1.7 | 0.5 | *I. setosa* |
| 4.8 | 3.4 | 1.9 | 0.2 | *I. setosa* |
| 5.0 | 3.0 | 1.6 | 0.2 | *I. setosa* |
| 5.0 | 3.4 | 1.6 | 0.4 | *I. setosa* |
| 5.2 | 3.5 | 1.5 | 0.2 | *I. setosa* |
| 5.2 | 3.4 | 1.4 | 0.2 | *I. setosa* |
| 4.7 | 3.2 | 1.6 | 0.2 | *I. setosa* |
| 4.8 | 3.1 | 1.6 | 0.2 | *I. setosa* |
| 5.4 | 3.4 | 1.5 | 0.4 | *I. setosa* |
| 5.2 | 4.1 | 1.5 | 0.1 | *I. setosa* |
| 5.5 | 4.2 | 1.4 | 0.2 | *I. setosa* |
| 4.9 | 3.1 | 1.5 | 0.2 | *I. setosa* |
| 5.0 | 3.2 | 1.2 | 0.2 | *I. setosa* |
| 5.5 | 3.5 | 1.3 | 0.2 | *I. setosa* |
| 4.9 | 3.6 | 1.4 | 0.1 | *I. setosa* |
| 4.4 | 3.0 | 1.3 | 0.2 | *I. setosa* |
| 5.1 | 3.4 | 1.5 | 0.2 | *I. setosa* |
| 5.0 | 3.5 | 1.3 | 0.3 | *I. setosa* |
| 4.5 | 2.3 | 1.3 | 0.3 | *I. setosa* |
| 4.4 | 3.2 | 1.3 | 0.2 | *I. setosa* |
| 5.0 | 3.5 | 1.6 | 0.6 | *I. setosa* |
| 5.1 | 3.8 | 1.9 | 0.4 | *I. setosa* |
| 4.8 | 3.0 | 1.4 | 0.3 | *I. setosa* |
| 5.1 | 3.8 | 1.6 | 0.2 | *I. setosa* |
| 4.6 | 3.2 | 1.4 | 0.2 | *I. setosa* |
| 5.3 | 3.7 | 1.5 | 0.2 | *I. setosa* |
| 5.0 | 3.3 | 1.4 | 0.2 | *I. setosa* |
| 7.0 | 3.2 | 4.7 | 1.4 | *I. versicolor* |
| 6.4 | 3.2 | 4.5 | 1.5 | *I. versicolor* |
| 6.9 | 3.1 | 4.9 | 1.5 | *I. versicolor* |
| 5.5 | 2.3 | 4.0 | 1.3 | *I. versicolor* |
| 6.5 | 2.8 | 4.6 | 1.5 | *I. versicolor* |
| 5.7 | 2.8 | 4.5 | 1.3 | *I. versicolor* |
| 6.3 | 3.3 | 4.7 | 1.6 | *I. versicolor* |
| 4.9 | 2.4 | 3.3 | 1.0 | *I. versicolor* |
| 6.6 | 2.9 | 4.6 | 1.3 | *I. versicolor* |
| 5.2 | 2.7 | 3.9 | 1.4 | *I. versicolor* |
| 5.0 | 2.0 | 3.5 | 1.0 | *I. versicolor* |
| 5.9 | 3.0 | 4.2 | 1.5 | *I. versicolor* |
| 6.0 | 2.2 | 4.0 | 1.0 | *I. versicolor* |
| 6.1 | 2.9 | 4.7 | 1.4 | *I. versicolor* |
| 5.6 | 2.9 | 3.6 | 1.3 | *I. versicolor* |
| 6.7 | 3.1 | 4.4 | 1.4 | *I. versicolor* |
| 5.6 | 3.0 | 4.5 | 1.5 | *I. versicolor* |
| 5.8 | 2.7 | 4.1 | 1.0 | *I. versicolor* |
| 6.2 | 2.2 | 4.5 | 1.5 | *I. versicolor* |
| 5.6 | 2.5 | 3.9 | 1.1 | *I. versicolor* |
| 5.9 | 3.2 | 4.8 | 1.8 | *I. versicolor* |
| 6.1 | 2.8 | 4.0 | 1.3 | *I. versicolor* |
| 6.3 | 2.5 | 4.9 | 1.5 | *I. versicolor* |
| 6.1 | 2.8 | 4.7 | 1.2 | *I. versicolor* |
| 6.4 | 2.9 | 4.3 | 1.3 | *I. versicolor* |
| 6.6 | 3.0 | 4.4 | 1.4 | *I. versicolor* |
| 6.8 | 2.8 | 4.8 | 1.4 | *I. versicolor* |
| 6.7 | 3.0 | 5.0 | 1.7 | *I. versicolor* |
| 6.0 | 2.9 | 4.5 | 1.5 | *I. versicolor* |
| 5.7 | 2.6 | 3.5 | 1.0 | *I. versicolor* |
| 5.5 | 2.4 | 3.8 | 1.1 | *I. versicolor* |
| 5.5 | 2.4 | 3.7 | 1.0 | *I. versicolor* |
| 5.8 | 2.7 | 3.9 | 1.2 | *I. versicolor* |
| 6.0 | 2.7 | 5.1 | 1.6 | *I. versicolor* |
| 5.4 | 3.0 | 4.5 | 1.5 | *I. versicolor* |
| 6.0 | 3.4 | 4.5 | 1.6 | *I. versicolor* |
| 6.7 | 3.1 | 4.7 | 1.5 | *I. versicolor* |
| 6.3 | 2.3 | 4.4 | 1.3 | *I. versicolor* |
| 5.6 | 3.0 | 4.1 | 1.3 | *I. versicolor* |
| 5.5 | 2.5 | 4.0 | 1.3 | *I. versicolor* |
| 5.5 | 2.6 | 4.4 | 1.2 | *I. versicolor* |
| 6.1 | 3.0 | 4.6 | 1.4 | *I. versicolor* |
| 5.8 | 2.6 | 4.0 | 1.2 | *I. versicolor* |
| 5.0 | 2.3 | 3.3 | 1.0 | *I. versicolor* |
| 5.6 | 2.7 | 4.2 | 1.3 | *I. versicolor* |
| 5.7 | 3.0 | 4.2 | 1.2 | *I. versicolor* |
| 5.7 | 2.9 | 4.2 | 1.3 | *I. versicolor* |
| 6.2 | 2.9 | 4.3 | 1.3 | *I. versicolor* |
| 5.1 | 2.5 | 3.0 | 1.1 | *I. versicolor* |
| 5.7 | 2.8 | 4.1 | 1.3 | *I. versicolor* |
| 6.3 | 3.3 | 6.0 | 2.5 | *I. virginica* |
| 5.8 | 2.7 | 5.1 | 1.9 | *I. virginica* |
| 7.1 | 3.0 | 5.9 | 2.1 | *I. virginica* |
| 6.3 | 2.9 | 5.6 | 1.8 | *I. virginica* |
| 6.5 | 3.0 | 5.8 | 2.2 | *I. virginica* |
| 7.6 | 3.0 | 6.6 | 2.1 | *I. virginica* |
| 4.9 | 2.5 | 4.5 | 1.7 | *I. virginica* |
| 7.3 | 2.9 | 6.3 | 1.8 | *I. virginica* |
| 6.7 | 2.5 | 5.8 | 1.8 | *I. virginica* |
| 7.2 | 3.6 | 6.1 | 2.5 | *I. virginica* |
| 6.5 | 3.2 | 5.1 | 2.0 | *I. virginica* |
| 6.4 | 2.7 | 5.3 | 1.9 | *I. virginica* |
| 6.8 | 3.0 | 5.5 | 2.1 | *I. virginica* |
| 5.7 | 2.5 | 5.0 | 2.0 | *I. virginica* |
| 5.8 | 2.8 | 5.1 | 2.4 | *I. virginica* |
| 6.4 | 3.2 | 5.3 | 2.3 | *I. virginica* |
| 6.5 | 3.0 | 5.5 | 1.8 | *I. virginica* |
| 7.7 | 3.8 | 6.7 | 2.2 | *I. virginica* |
| 7.7 | 2.6 | 6.9 | 2.3 | *I. virginica* |
| 6.0 | 2.2 | 5.0 | 1.5 | *I. virginica* |
| 6.9 | 3.2 | 5.7 | 2.3 | *I. virginica* |
| 5.6 | 2.8 | 4.9 | 2.0 | *I. virginica* |
| 7.7 | 2.8 | 6.7 | 2.0 | *I. virginica* |
| 6.3 | 2.7 | 4.9 | 1.8 | *I. virginica* |
| 6.7 | 3.3 | 5.7 | 2.1 | *I. virginica* |
| 7.2 | 3.2 | 6.0 | 1.8 | *I. virginica* |
| 6.2 | 2.8 | 4.8 | 1.8 | *I. virginica* |
| 6.1 | 3.0 | 4.9 | 1.8 | *I. virginica* |
| 6.4 | 2.8 | 5.6 | 2.1 | *I. virginica* |
| 7.2 | 3.0 | 5.8 | 1.6 | *I. virginica* |
| 7.4 | 2.8 | 6.1 | 1.9 | *I. virginica* |
| 7.9 | 3.8 | 6.4 | 2.0 | *I. virginica* |
| 6.4 | 2.8 | 5.6 | 2.2 | *I. virginica* |
| 6.3 | 2.8 | 5.1 | 1.5 | *I. virginica* |
| 6.1 | 2.6 | 5.6 | 1.4 | *I. virginica* |
| 6.3 | 3.4 | 5.6 | 2.4 | *I. virginica* |
| 6.4 | 3.1 | 5.5 | 1.8 | *I. virginica* |
| 6.0 | 3.0 | 4.8 | 1.8 | *I. virginica* |
| 6.9 | 3.1 | 5.4 | 2.1 | *I. virginica* |
| 6.7 | 3.1 | 5.6 | 2.4 | *I. virginica* |
| 6.9 | 3.1 | 5.1 | 2.3 | *I. virginica* |
| 5.8 | 2.7 | 5.1 | 1.9 | *I. virginica* |
| 6.8 | 3.2 | 5.9 | 2.3 | *I. virginica* |
| 6.7 | 3.3 | 5.7 | 2.5 | *I. virginica* |
| 6.7 | 3.0 | 5.2 | 2.3 | *I. virginica* |
| 6.3 | 2.5 | 5.0 | 1.9 | *I. virginica* |
| 6.5 | 3.0 | 5.2 | 2.0 | *I. virginica* |
| 6.2 | 3.4 | 5.4 | 2.3 | *I. virginica* |
| 5.9 | 3.0 | 5.1 | 1.8 | *I. virginica* |

  
  
Zbudowana sieć ma na celu podzielenie danych uczących na grupy, której to zostaje przyporządkowany dany element wyjściowy. Należy pamiętać, że elementy składowe danej grupy były do siebie podobne. W porównaniu do pozostałych grup, każda grupa ma jednak charakterystyczne dla siebie elementy, które różnią się w zależności od przynależności do grupy.

OPIS SIECI:  
W przypadku podanej sieci występuje metoda uczenia sieci samoorganizującej, którym jest uczenie rywalizujące. Neurony w danej sieci uczą się rozpoznawania danych, na których bazuję a następnie zbliża się do terenu, w którym dane te są najmocniej osadzone. Najważniejszą zasadą w podanej sieci jest fakt, że z pośród wszystkich neuronów wybierany jest ten, który znajduje się najbliżej centrum terenu. Neuron, którego wartość jest największa zostaje zwycięzcą, dzięki czemu na wyjściu jego wartość wynosi 1. Natomiast w przypadku pozostałych neuronów ich wartość na wyjściu wynosi 0.  
  
Schemat uczenia sieci :  
Krok 1: Na początku wszystkie dane podlegają normalizacji.

Krok 2 : Wybór współczynnika uczenia   
Krok 3 : Losowanie początkowych wartości wag z zakresu od 0 do 1

Krok 4 : Na podstawie zbioru danych obliczana jest odpowiedź

Krok 5 : Dla neuronu z najwyższym wynikiem aktualizacja wag za pomocą wzoru :

Krok 6 : Następnie normalizacja wartości nowego wektora wag

Krok 7: Ustawienie wartości wyjściowej dla zwycięskiego wektora na 1 ( reszta 0 ) i pobranie kolejnego wektora uczącego.  
 **2.Zestawienie wyników i analiza programu :**

Z powyższych 3 wykresów możemy wywnioskować, że im wyższy jest współczynnik uczenia tym liczba błędów wzrasta. Sieć uczy sie najlepiej dla małych współczynników uczenia ( najlepiej dla współczynnika 0.01 ). Przy współczynniku wynoszącym 1 liczba błędów jest spora. Na wynik może wpływać fakt, iż przy dużym współczynniku neuron zwycięski jest wzmacniany na tyle, że przy kolejnym teście zwycięży ten sam neuron.  
  
  
  
Z powyższego wykresu wynika, że wraz ze wzrostem współczynnika uczenia maleje liczba epok potrzebnych do nauki danego badania. Najmniejsza liczba epok potrzebna była dla sieci przy badaniu działki kielicha. Natomiast największą liczbę epok, niezależnie od współczynnika uczenia potrzebna była na badaniu płatka. Na kształt wykresów może wpływać fakt, że neurony po podzieleniu się na grupy i odebraniu nowych danych, mogły zmienić swoją przynależność do innej grupy.

**3.Wnioski:**Sieć Kohonena to sieć samoorganizująca się. Neurony grupują się i każda z grup ma różne wartości dla poszczególnych cech. Dzięki temu siec nie potrzebuje nauczyciela więc może uczyć się sama. Jak przedstawiają wykresy powyżej, sieć uczy się różnie w zależności od współczynnika uczenia. Wraz ze wzrostem wartości współczynnika sieć uczy się szybciej jednak im wyższy jest ten współczynnik tym skuteczność nauki jest słabsza. **4.Listing programu:**

**Layer.h**

#include <vector>

#include "Neuron.h"

using namespace std;

class Layer {

public:

vector<Neuron> neuron; //wektor neuronow

vector<double> suma; //wektor sum wejsc

int liczba\_neuronow; //liczba neuronow

int wygrany\_index; //indeks zwyciezcy

Layer(int liczba\_neuronow, int liczba\_wejsc, double wspolczynnik\_uczenia); //konstruktor

void wygrany\_neuron(bool learning); //szukanie zwycieskiego neuronu

void suma\_wszystkich\_wejsc(); //obliczenie sumy wszystkich wejsc

void zmiana\_wag(bool learning); //zmiana wag (learning = true dla procesu uczenia, = false dla procesu testowania)

};

**Layer.cpp**#include "Layer.h"

Layer::Layer(int liczba\_neuronow, int liczba\_wejsc, double wspolczynnik\_uczenia) {

this->liczba\_neuronow = liczba\_neuronow;

this->neuron.resize(liczba\_neuronow);

for (int i = 0; i < liczba\_neuronow; i++)

this->neuron[i].Neuron::Neuron(liczba\_wejsc, wspolczynnik\_uczenia);

}

void Layer::wygrany\_neuron(bool learning) { //poszukiwanie wejscia o najwiekszej sumie

double tmp = this->suma[0];

this->wygrany\_index = 0;

for (int i = 1; i < this->suma.size(); i++) {

if (tmp < this->suma[i]) {

this->wygrany\_index = i;

tmp = this->suma[i];

}

}

this->neuron[this->wygrany\_index].funkcja\_aktywacji();

if (learning)

this->neuron[this->wygrany\_index].calculateNewWeight(); //aktualizacja wag

}

void Layer::zmiana\_wag(bool learning) { //obliczenie sum wszystkich wejsc, poszukiwanie tego o najwiekszej sumie i aktualizacja jego wag

suma\_wszystkich\_wejsc();

wygrany\_neuron(learning);

}

void Layer::suma\_wszystkich\_wejsc() { //obliczenie sumy wszystkich wejsc

this->suma.clear();

for (int i = 0; i < this->liczba\_neuronow; i++)

this->suma.push\_back(neuron[i].calculateSumOfAllwejscia());

}

**Neuron.h**#pragma once

#include <iostream>

#include <vector>

using namespace std;

class Neuron {

public:

vector<double> wagi;

vector<double> wejscia;

double wartosc\_wyjsciowa; //wartosc wyjsciowa

double wspolczynnik\_uczenia; //wspolczynnik uczenia

double sumOfAllwejscia; //suma wszystkich wejsc

Neuron(); //konstruktor

Neuron(int liczba\_wejsc, double wspolczynnik\_uczenia); //konstruktor

double losowanie\_wag(); //wylosowanie poczatkowych wag z zakresu <0;1)

void normalizacja\_wag(); //znormalizowanie wag (podczas procesu uczenia)

void createwejscia(int liczba\_wejsc); //stworzenie poczatkowych wejsc (ustawienie wejsc na 0, wykorzystanie metody losowanie\_wag())

void calculateNewWeight(); //obliczenie nowej wagi dla zwycieskiego neuronu

double calculateSumOfAllwejscia(); //obliczenie sumy wszystkich wejsc

void funkcja\_aktywacji(); //funkcja sigmoidalna obliczajaca wyjscie

int getRozmiar\_wagi() { //zwraca rozmiar wag

return wagi.size();

}

int getliczba\_wejsc() { //zwraca rozmiar wejsc

return wejscia.size();

}

}; **Neuron.cpp**#include "Neuron.h"

#include <ctime>

#include <cmath>

Neuron::Neuron() {

this->wejscia.resize(0);

this->wagi.resize(0);

this->sumOfAllwejscia = 0.0;

this->wartosc\_wyjsciowa = 0.0;

this->wspolczynnik\_uczenia = 0.0;

}

Neuron::Neuron(int liczba\_wejsc, double wspolczynnik\_uczenia) {

createwejscia(liczba\_wejsc);

normalizacja\_wag();

this->wspolczynnik\_uczenia = wspolczynnik\_uczenia;

this->sumOfAllwejscia = 0.0;

this->wartosc\_wyjsciowa = 0.0;

}

//stworzenie poczatkowych wejsc (ustawienie wejsc na 0, wykorzystanie metody losowanie\_wag())

void Neuron::createwejscia(int liczba\_wejsc) {

for (int j = 0; j < liczba\_wejsc; j++) {

this->wejscia.push\_back(0);

this->wagi.push\_back(losowanie\_wag());

}

}

//obliczenie sumy wszystkich wejsc

double Neuron::calculateSumOfAllwejscia() {

this->sumOfAllwejscia = 0.0;

for (int i = 0; i < getliczba\_wejsc(); i++)

this->sumOfAllwejscia += wejscia[i] \* wagi[i];

return sumOfAllwejscia;

}

//funkcja sigmoidalna obliczajaca wyjscie

void Neuron::funkcja\_aktywacji() {

double beta = 1.0;

this->wartosc\_wyjsciowa = (1.0 / (1.0 + (exp(-beta \* this->sumOfAllwejscia))));

}

//obliczenie nowych wag

void Neuron::calculateNewWeight() {

for (int i = 0; i < getRozmiar\_wagi(); i++)

this->wagi[i] += this->wspolczynnik\_uczenia\*(this->wejscia[i] - this->wagi[i]);

normalizacja\_wag();

}

//ustalenie poczatkowych wag dla wszystkich wejsc - zakres <0;1)

double Neuron::losowanie\_wag() {

double max = 1.0;

double min = 0.0;

double weight = ((double(rand()) / double(RAND\_MAX))\*(max - min)) + min;

return weight;

}

//znormalizowanie nowo obliczonej wagi zwycieskiego neuronu

void Neuron::normalizacja\_wag() {

double vectordlugosc = 0.0;

for (int i = 0; i < getRozmiar\_wagi(); i++)

vectordlugosc += pow(this->wagi[i], 2);

vectordlugosc = sqrt(vectordlugosc);

for (int i = 0; i < getRozmiar\_wagi(); i++)

this->wagi[i] /= vectordlugosc;

}  
  
**Source.cpp**#include <fstream>

#include <vector>

#include <iostream>

#include <ctime>

#include "Layer.h"

using namespace std;

fstream OUTPUT\_FILE\_LEARNING, OUTPUT\_FILE\_TESTING\_SUM, OUTPUT\_FILE\_TESTING\_WINNER; //strumienie do plikow sluzace do wczytania danych uczacych oraz zapisu wynikow

fstream Dane\_do\_nauki, Dane\_do\_testu;

void uczenie(Layer& layer, vector<vector<double>> inputData); //uczenie sieci

void test(Layer& layer, vector<vector<double>> inputData); //testowanie sieci

void wczytanie\_danych\_uczących(vector<vector<double>>&learnData, int liczba\_wejsc); //wczytanie danych potrzebnych do nauki

void wczytanie\_danych\_testowych(vector<vector<double>>&testData, int numberOfInpus); //wczytaniee danych testowych

void wczytanie\_danych\_wejsciowych(Neuron& neuron, vector<vector<double>> inputData, int liczba\_wejsc, int inputDataRow); //wczytanie do tablic danych wejsciowych

int main() {

srand(time(NULL));

//wektory z danymi uczacymi oraz testujacymi

vector<vector<double>> trainData;

vector<vector<double>> testData;

int liczba\_neuronow = 10;

int liczba\_wejsc = 4;

double wspolczynnik\_uczenia = 0.5;

Layer siecKohonena(liczba\_neuronow, liczba\_wejsc, wspolczynnik\_uczenia); //stworzenie sieci Kohonena

wczytanie\_danych\_uczących(trainData, liczba\_wejsc); //wczytanie danych uczacych

wczytanie\_danych\_testowych(testData, liczba\_wejsc); //wczytaniee danych testowych

OUTPUT\_FILE\_LEARNING.open("output\_learning\_data.txt", ios::out);

for (int Epoka = 1, i = 0; i < 2; i++, Epoka++) {

//uczenie

OUTPUT\_FILE\_LEARNING << "Epoka: " << Epoka << endl;

cout << "Epoka: " << Epoka << endl;

uczenie(siecKohonena, trainData);

}

OUTPUT\_FILE\_LEARNING.close();

OUTPUT\_FILE\_TESTING\_SUM.open("output\_Dane\_do\_testu.txt", ios::out);

OUTPUT\_FILE\_TESTING\_WINNER.open("output\_testing\_neuron.txt", ios::out);

test(siecKohonena, testData);

OUTPUT\_FILE\_LEARNING.close();

OUTPUT\_FILE\_TESTING\_SUM.close();

OUTPUT\_FILE\_TESTING\_WINNER.close();

system("pasue");

return 0;

}

void uczenie(Layer& layer, vector<vector<double>> inputData)

{

int licznik = 0;

for (int rowOfData = 0; rowOfData < inputData.size(); rowOfData++) {

for (int i = 0; i < layer.liczba\_neuronow; i++) {

wczytanie\_danych\_wejsciowych(layer.neuron[i], inputData, layer.neuron[i].getliczba\_wejsc(), rowOfData); //wczytanie danych do tablic

layer.neuron[i].calculateSumOfAllwejscia(); //wyliczenie sumy wejscia

}

layer.zmiana\_wag(true); //zmiana wag

//przeskoczenie na kolejny rodzaj kwiatka (wyzerowanie licznika)

if (licznik == 50) {

licznik = 0;

OUTPUT\_FILE\_LEARNING << "Next flower" << endl;

cout << "Next flower" << endl;

}

OUTPUT\_FILE\_LEARNING << layer.wygrany\_index << endl;

cout << "Winner: " << layer.wygrany\_index << endl;

licznik++;

}

}

void wczytanie\_danych\_wejsciowych(Neuron& neuron, vector<vector<double>> inputData, int liczba\_wejsc, int row)

{

for (int i = 0; i < liczba\_wejsc; i++) {

neuron.wejscia[i] = inputData[row][i];

}

}

void test(Layer& layer, vector<vector<double>> inputData) {

int licznik = 0;

for (int wierszDanych = 0; wierszDanych < inputData.size(); wierszDanych++) {

for (int i = 0; i < layer.liczba\_neuronow; i++) {

wczytanie\_danych\_wejsciowych(layer.neuron[i], inputData, layer.neuron[i].getliczba\_wejsc(), wierszDanych);//wczytanie danych do tablic

layer.neuron[i].calculateSumOfAllwejscia(); //wyliczenia sumy wejscia

}

//przeskoczenie na kolejny rodzaj kwiatka (wyzerowanie licznika)

if (licznik == 15) {

licznik = 0;

OUTPUT\_FILE\_TESTING\_WINNER << "Next flower" << endl;

cout << "Next flower" << endl;

}

layer.zmiana\_wag(false); //wagi nie beda zaktualiowane dla zwyciezcy

OUTPUT\_FILE\_TESTING\_SUM << layer.neuron[layer.wygrany\_index].sumOfAllwejscia << endl;

OUTPUT\_FILE\_TESTING\_WINNER << layer.wygrany\_index << endl;

cout << "Which neuron: " << layer.wygrany\_index << endl;

licznik++;

}

}

void wczytanie\_danych\_testowych(vector<vector<double>> &testData, int liczba\_wejsc) {

Dane\_do\_testu.open("datatest.txt", ios::in);

vector<double> row;

while (!Dane\_do\_testu.eof()) {

row.clear();

for (int i = 0; i < liczba\_wejsc; i++) {

double inputTmp = 0.0;

Dane\_do\_testu >> inputTmp;

row.push\_back(inputTmp);

if (i == liczba\_wejsc - 1) {

Dane\_do\_nauki >> inputTmp;

}

}

double dlugosc = 0.0; //znormalizowanie danych uczacych

for (int i = 0; i < liczba\_wejsc; i++)

dlugosc += pow(row[i], 2);

dlugosc = sqrt(dlugosc);

for (int i = 0; i < liczba\_wejsc; i++)

row[i] /= dlugosc;

testData.push\_back(row);

}

Dane\_do\_testu.close();

}

void wczytanie\_danych\_uczących(vector<vector<double>> &trainData, int liczba\_wejsc) {

Dane\_do\_nauki.open("data.txt", ios::in);

vector<double> row;

do {

row.clear();

for (int i = 0; i < liczba\_wejsc; i++) {

double inputTmp = 0.0;

Dane\_do\_nauki >> inputTmp;

row.push\_back(inputTmp);

if (i == liczba\_wejsc - 1) {

Dane\_do\_nauki >> inputTmp;

}

}

//znormalizowanie danych uczacych

double dlugosc = 0.0;

for (int i = 0; i < liczba\_wejsc; i++)

dlugosc += pow(row[i], 2);

dlugosc = sqrt(dlugosc);

for (int i = 0; i < liczba\_wejsc; i++)

row[i] /= dlugosc;

trainData.push\_back(row);

} while (!Dane\_do\_nauki.eof());

Dane\_do\_nauki.close();

}