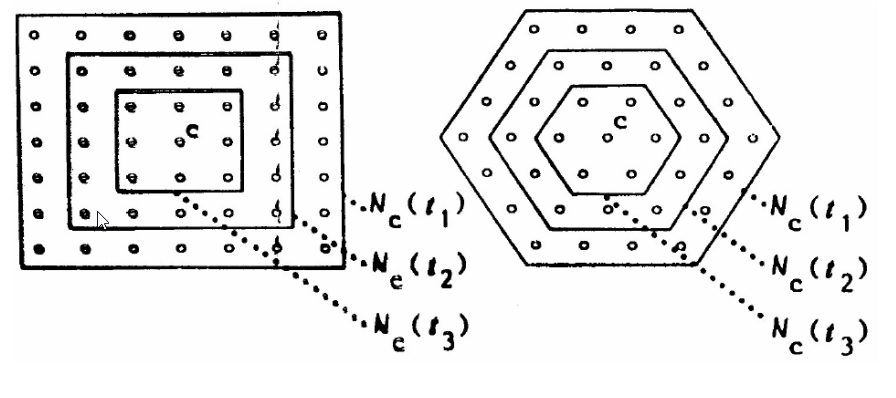
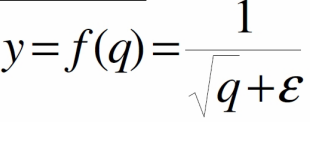
Sprawozdanie

Laboratorium numer 5 dotyczyło sieci Kohennena czyli sieci samoorganizującej się. W zasadzie poprawnie zbudowana sieć Kohennena jest to mapa, która sama sobie organizuje miejsce do przechowywania danych. Oczywiście głównym składnikiem takiej mapy jest odpowiednia ilość perceptronów z właściwą sobie liczbą wejść i funkcją aktywacyjną.

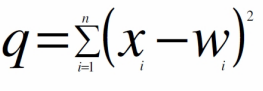
Zasada takiej sieci polega na wytypowaniu najmocniejszych perceptronów, za pomocą dwóch najczestszych metod czyli WTA (Winner Takes All) lub WTM (Winner Takes Most), zasadnicza różnica pomiędzy obydwiema metodami jest niewielka i polega na tym, iż w WTA zmieniamy wagi tylko najsilniej reagującemu perceptronowi, natomiast w WTM zmieniamy wagi również dla sąsiedztwa perceptronu zwycięskiego. Budowa samego perceptronu jest klasyczna. Składa się z wektora wag, który odpowiada różnorodności organizowanego zbioru (w naszym przypadku mieliśmy 4 wagi, ponieważ były 4 cechy kwiatów do sklasyfikowania, warto również pamiętać o biasie) oraz z wektora wag równego ilości wejść do perceptronu.



Funkcja aktywacyjna dla tej siatki jest dość specyficzna, określa się wzorem:



gdzie:



Ilość perceptronów również jest wyrażona wzorem 5 ,gdzie N to ilość wektorów danych.

Algorytm tworzenia takiej sieci można podzielić na kilka zasadniczych kroków:

- Przypisanie wag początkowych dla wszystkich perceptronów,

- Dołączenie nowego wektora uczącego,

- Wyliczenie funkcji aktywacji, wybór najsilniejszego perceptronu,

- Wyznaczenie nowych wartości wag ze wzoru:

2018-01-21 19_00_39-Slajd 1.png

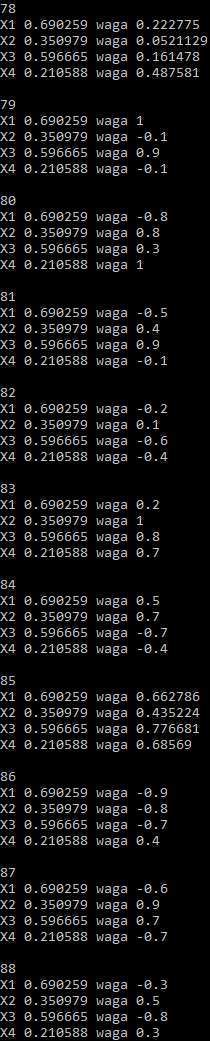
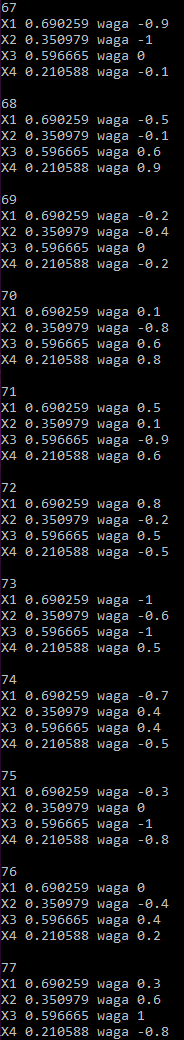
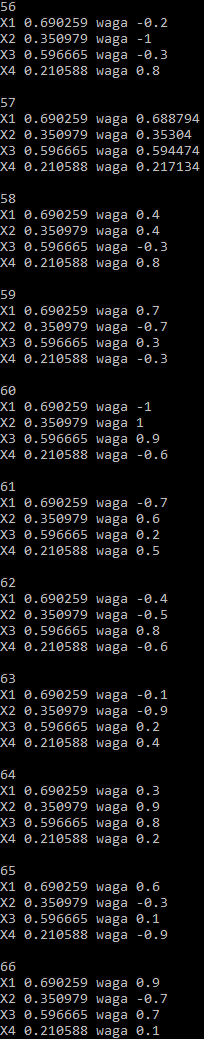
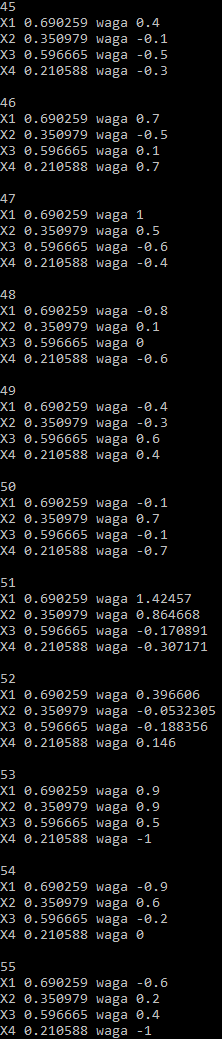
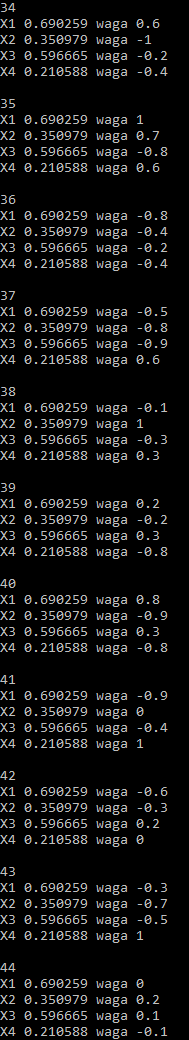
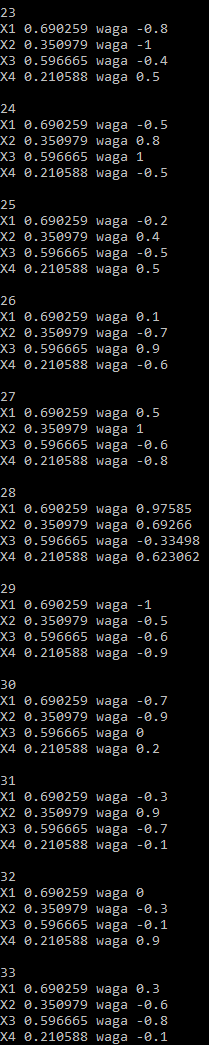
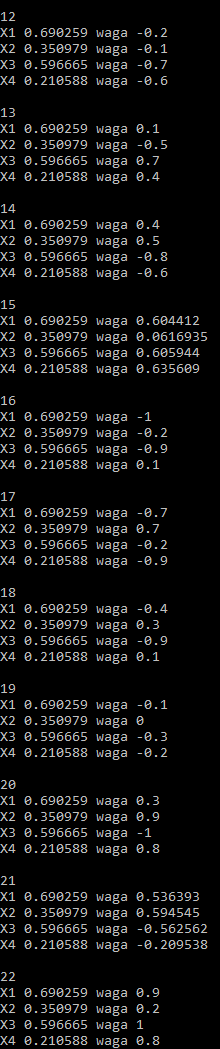
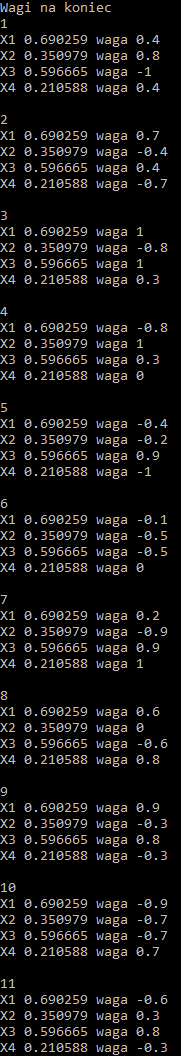
gdzie: w - waga, k - krok iteracji, η – współczynnik uczenia, x – wejscie

- Zmiana współczynnika uczenia,

- Powtarzanie kroków 2 – 4, aż do uzyskania rezultatu.

Niestety pomimo zastosowania się do wszystkich wymienionych kroków, nie udało mi się uzyskać poprawnej sieci Kohennena. Za prawie każdym razem wychodzi, że jeden perceptron wyucza się wszystkich przypadków i reaguje na każdy wektor najsilniej. Czasami zdarzało się, że dwa perceptrony dzieliły się wynikami pół na pół, jednakże nie jest to zadowalający wynik.

Nie pomogła wektoryzacja, każdego z wektorów wchodzących, zmiana ilości perceptronów, próba kombinowania z funkcjami aktywacji, próba uczenia do skutku czyli powtarzanie tych samych wektorów po parę tysięcy razy, zmiana wartości współczynnika uczenia (zwiększanie, zmniejszanie samej wartości), próba zmniejszania zgodnie z algorytmem po każdej iteracji współczynnika uczenia w sposób liniowy (zmniejszanie iteracji o 0.01 ponieważ wartość początkowa wynosiła 2.5),zmiana współczynnika dla samego perceptronu, który wygrał jak i dla wszystkich niezależnie, który wygrał. W żaden sposób nie udało mi się uzyskać zadawalającego wyniku. Poniżej kilka screenów pokazujących, że dla 100 perceptronów tylko kilka było w stanie się jakkolwiek przebić (reszta pozostała od początku do końca taka sama).



Większość wektorów wag pozostała nietknięta natomiast każdy ze 150 wektorów wejścia zostało rozpoznane przez perceptron nr 31.

Poniżej zamieszczam Listing Kodu:

Neuron.h

#pragma once

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <string>

class Neuron {

private:

double wspolczynnik;

double w[4];

double x[4];

std::string nazwa;

public:

Neuron();

Neuron(std::string);

void wypisz();

void ustaw\_wektor(double\*);

double aktywacja();

void uaktualnij\_wagi();

void zmniejsz\_wspolczynnik();

};

Neuron.cpp

#include "Neuron.h"

Neuron::Neuron() {

wspolczynnik = 0.5;

srand(time(NULL));

for (int i = 0; i < 5; i++) {

w[i] = ((rand() % 21) - 10) / 10.;

x[i] = 0;

}

x[4] = 1;

}

Neuron::Neuron(std::string tmp) {

wspolczynnik = 2.3;

srand(time(NULL));

for (int i = 0; i < 4; i++) {

w[i] = ((rand() % 21) - 10) / 10.;

x[i] = 0;

}

nazwa = tmp;

}

void Neuron::wypisz(){

std::cout << nazwa << std::endl;

for (int i = 0; i < 4; i++)

std::cout << "X" << i + 1 << " " << x[i] << " waga " << w[i] << std::endl;

std::cout<<std::endl;

}

void Neuron::ustaw\_wektor(double\* tab) {

x[0] = tab[0];

x[1] = tab[1];

x[2] = tab[2];

x[3] = tab[3];

}

double Neuron::aktywacja() {

double tmp = 0;

for (int i = 0; i < 4; i++)

tmp += (x[i] - w[i])\*(x[i] - w[i]);

return 1/sqrt(tmp);

}

void Neuron::uaktualnij\_wagi() {

for (int i = 0; i < 4; i++)

w[i] += wspolczynnik\*(x[i] - w[i]);

}

void Neuron::zmniejsz\_wspolczynnik() {

wspolczynnik -= 0.01;

}

Dane.h

#pragma once

#include <fstream>

#include <iostream>

#include <conio.h>

#include <string>

#include <math.h>

class Dane {

private:

double dane[150][5];

public:

Dane();

void wypisz();

double\* zwroc\_wektor(int);

void wektoryzuj();

};

Dane.cpp

#include "Dane.h"

Dane::Dane() {

std::fstream plik;

plik.open("plik.txt", std::ios::in);

std::string tmp;

for (int i = 0; i < 150; i++) {

std::getline(plik, tmp, ',');

dane[i][0] = stod(tmp);

std::getline(plik, tmp, ',');

dane[i][1] = stod(tmp);

std::getline(plik, tmp, ',');

dane[i][2] = stod(tmp);

std::getline(plik, tmp, ',');

dane[i][3] = stod(tmp);

std::getline(plik, tmp);

if (tmp == "Iris-setosa")

dane[i][4] = 1;

else if(tmp == "Iris-versicolor")

dane[i][4] = 2;

else if(tmp == "Iris-virginica")

dane[i][4] = 3;

}

plik.close();

}

void Dane::wypisz() {

for (int i = 0; i < 150; i++)

std::cout << dane[i][0] << " " << dane[i][1] << " " << dane[i][2] << " " << dane[i][3] << " " << dane[i][4] << std::endl;

}

double\* Dane::zwroc\_wektor(int tmp) {

double\* tab = new double[4];

tab[0] = dane[tmp][0];

tab[1] = dane[tmp][1];

tab[2] = dane[tmp][2];

tab[3] = dane[tmp][3];

return tab;

}

void Dane::wektoryzuj() {

double suma = 0;

for (int i = 0; i < 150; i++) {

suma = pow(dane[i][0], 2) + pow(dane[i][1], 2) + pow(dane[i][2], 2) + pow(dane[i][3], 2);

suma = sqrt(suma);

dane[i][0] /= suma;

dane[i][1] /= suma;

dane[i][2] /= suma;

dane[i][3] /= suma;

suma = 0;

}

}

Źródło.cpp

#include <iostream>

#include "Neuron.h"

#include "Dane.h"

#include <Windows.h>

using namespace std;

int main()

{

Neuron tab[10];

for (int i = 0; i < 10; i++) {

tab[i] = Neuron(to\_string(i+1));

Sleep(1000);

}

Dane d = Dane();

d.wektoryzuj();

cout << "Poczatkowe wagi" << endl;

for (int i = 0; i < 10; i++)

tab[i].wypisz();

int zapamietana;

for (int i = 0; i < 150; i++) {

zapamietana = 0;

for (int j = 0; j < 10; j++)

tab[j].ustaw\_wektor(d.zwroc\_wektor(i));

for (int j = 1; j < 10; j++)

if (tab[zapamietana].aktywacja() < tab[j].aktywacja())

zapamietana = j;

tab[zapamietana].uaktualnij\_wagi();

for(int z = 0 ; z < 10 ; z++)

tab[z].zmniejsz\_wspolczynnik();

}

cout << "Wagi na koniec" << endl;

for (int i = 0; i < 10; i++)

tab[i].wypisz();

double\* tab1;

for (int i = 0; i < 150; i++) {

tab1 = d.zwroc\_wektor(i);

zapamietana = 0;

for (int j = 0; j < 10; j++)

tab[j].ustaw\_wektor(tab1);

zapamietana = 0;

for (int j = 1; j < 10; j++)

if (tab[zapamietana].aktywacja() < tab[j].aktywacja())

zapamietana = j;

cout << endl << "Neutron " << zapamietana + 1 << endl;

}

system("Pause");

}

Bibliografia:

<http://www.uci.agh.edu.pl/uczelnia/tad/inteligencja_obliczeniowa/09%20-%20Sieci%20Kohonena.pdf>

<http://zsi.tech.us.edu.pl/~nowak/wi/som.pdf>