**Jakub Bąk  
Scenariusz 6**  
**Temat :** Budowa i działanie sieci Kohonena dla WTM.

**Cel ćwiczenia** Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTM do odwzorowywania istotnych cech liter polskiego alfabetu.  
  
**1.Syntetyczny opis budowy i wykorzystania sieci i algorytmu uczenia.**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 0 0 0 1  1 0 0 0 1  1 1 0 0 1  1 0 1 0 1  1 0 0 1 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 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 0 0 0 1  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 0 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 0  1 1 1 1 1 |
|  | 1 0 0 0 1  1 0 0 1 0  1 0 1 0 0  1 1 0 0 0  1 0 1 0 0  1 0 0 1 0  1 0 0 0 1 |  | 1 0 0 0 1  1 1 0 1 1  1 0 1 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 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 |  | 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 |
|  | 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 |  | 0 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  0 1 1 1 0 |
|  | 1 1 1 1 1  1 0 0 0 1  1 0 0 0 0  1 0 1 1 1  1 0 0 0 1  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 1 1 1 0  1 0 0 0 0  1 0 0 0 0  1 0 0 0 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 1 1 1 0  1 0 0 0 1  1 0 0 0 1  1 0 0 0 1  1 0 1 0 1  1 0 0 1 0  0 1 1 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 0  1 0 0 0 1  1 0 0 0 1  1 1 1 1 0  1 0 1 0 0  1 0 0 1 0  1 0 0 0 1 |
|  | 1 1 1 1 1  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 0 1 0 0 |  | 0 1 1 1 0  1 0 0 0 1  1 0 0 0 0  0 1 1 1 0  0 0 0 0 1  1 0 0 0 1  0 1 1 1 0 |

Nauka sieci polega na podziale danych na grupy i przyporządkowanie każdej danego elementu wyjścia. Dane należące do jednej grupy są do siebie podobne, zaś występuję różnice między danymi należącymi do różnych grup.  
  
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. Reguła ta jest podobna do reguły WTA jednak występują pewne różnice. W przypadku reguły WTM występuje promień, który pozwala na aktualizację wag. Aktualizacja dotyczy neuronów których wyjście wynosi 0, czyli te które nie zwyciężyły. Wartość promienia maleje przy każdej iteracji konsekwencją czego coraz mniej neuronów ma możliwość zmiany wagi. Na koniec zmiana wagi dotyczy tylko jednego neuronu, neuronu zwycięskiego.   
  
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 : Dla każdego neuronu liczona jest suma ilorazów wag oraz sygnałów wejściowych

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

gdzie:

– funkcja sąsiedztwa (wg Gaussa), obliczana ze wzoru:

gdzie:  
R – promień sąsiedztwa

d – jest to odległość pomiędzy zwycięskim neuronem oraz każdym dowolnym innym neuronem

gdzie:

i – wektor wejściowy w – waga neuronu t – obecna iteracja λ – stała czasowa

gdzie:

x – liczba iteracji

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 :**  
  
  
  
  
Powyższe dwa wykresy zależności liczby błędów od długości promienia informują nas, że najmniejsza liczba błędów przypada na współczynnik 0.5, natomiast dla współczynnika 0.1 liczba błędów jest największa. Dla współczynnika wynoszącego 0.3 występuje niewielka liczba błędów. Warto także zauważyć, że wraz ze wzrostem promienia zmniejsza się liczba błędów.  
  
TEST DLA 12 LITER :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Współczynnik uczenia = 0.5 (promień 2) | | Współczynnik uczenia = 0.3 (promień 2) | | Współczynnik uczenia = 0.1 (promień 2) | |
| Grupa 1 | A | Grupa 1 | A, H | Grupa 1 | A, B, D, H |
| Grupa 2 | B | Grupa 2 | B | Grupa 2 | C, E , F |
| Grupa 3 | C | Grupa 3 | C, D, L | Grupa 3 | I |
| Grupa 4 | D, G | Grupa 4 | E, F | Grupa 4 | G, J,K |
| Grupa 5 | E, F | Grupa 5 | G | Grupa 5 | L |
| Grupa 6 | I | Grupa 6 | I,J |  |  |
| Grupa 7 | J | Grupa 7 | K |  |  |
| Grupa 8 | K |  |  |  |  |
| Grupa 9 | L, H |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Współczynnik uczenia = 0.5 (promień 5) | | Współczynnik uczenia = 0.3 (promień 5) | | Współczynnik uczenia = 0.1 (promień 5) | |
| Grupa 1 | A | Grupa 1 | A | Grupa 1 | A |
| Grupa 2 | B | Grupa 2 | B | Grupa 2 | B, L |
| Grupa 3 | C | Grupa 3 | C, D | Grupa 3 | C, D, E |
| Grupa 4 | D | Grupa 4 | E, F | Grupa 4 | F |
| Grupa 5 | E, F | Grupa 5 | G | Grupa 5 | G |
| Grupa 6 | G | Grupa 6 | H | Grupa 6 | H |
| Grupa 7 | H | Grupa 7 | I,J | Grupa 7 | I, J |
| Grupa 8 | I,J | Grupa 8 | K | Grupa 8 | K |
| Grupa 9 | K | Grupa 9 | L |  |  |
| Grupa 10 | L |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Współczynnik uczenia = 0.5 (promień 10) | | Współczynnik uczenia = 0.3 (promień 10) | | Współczynnik uczenia = 0.1 (promień 10) | |
| Grupa 1 | A | Grupa 1 | A | Grupa 1 | A |
| Grupa 2 | B | Grupa 2 | B | Grupa 2 | B |
| Grupa 3 | C | Grupa 3 | C | Grupa 3 | C |
| Grupa 4 | D | Grupa 4 | D | Grupa 4 | D |
| Grupa 5 | E | Grupa 5 | E | Grupa 5 | E |
| Grupa 6 | F | Grupa 6 | F | Grupa 6 | F |
| Grupa 7 | G | Grupa 7 | G | Grupa 7 | G |
| Grupa 8 | H | Grupa 8 | H | Grupa 8 | H |
| Grupa 9 | I | Grupa 9 | I | Grupa 9 | I |
| Grupa 10 | J | Grupa 10 | J | Grupa 10 | J |
| Grupa 11 | K | Grupa 11 | K | Grupa 11 | K |
| Grupa 12 | L | Grupa 12 | L | Grupa 12 | L |

Powyższe tabele przedstawiają grupowanie wektorów uczących. Dla promienia wynoszącego 10 siec jest w stanie nie zależnie od współczynnika uczenia pogrupować dane prawidłowo. Natomiast im mniejsza jest wartość promienia, tym większą rolę zaczyna odgrywać współczynnik uczenia. Dla promienia wynoszącego 5, wektory wraz ze wzrostem współczynnika uczenia, dane są lepiej grupowane, do większej ilości grup. W przypadku promienia wynoszącego 2 i przy współczynniku 0.1 sieć nie uczy się prawidłowo i grupuje duże ilości liter to jednej 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. Kiedy porównamy reguły WTA z WTM zauważymy, że zaletą na korzyść WTM jest fakt, iż jest lepiej uporządkowana ze względu na większą zbieżność algorytmu. Natomiast zaletą WTA w odróżnieniu od WTM jest to, że zajmuje mniej miejsca w pamięci komputera oraz krótszy czas działania. Spowodowane to jest tym, że w przypadku WTM aktualizacja wag dotyczy nie tylko zwycięskiego neuronu ale i pozostałych tylko w otoczeniu zwycięskiego neuronu. **4.Listing programu:**

**Layer.h**

#include "Neuron.h"

#include <vector>

using namespace std;

class Layer {

public:

vector<Neuron> neuron;

vector<double> scalarProducts; //odleglosci euklidesowych

int liczba\_neuronow;

double promien; //promien wyznaczajacy obszar od zwycieskiego neuronu

double czas;

int zwycieski\_neuron; //indeks

void zmiana\_wag(double obecnaIteracja, bool testing); //przy aktualnej iteracji

void minimum\_odleglosc\_euklidesowa();

void getOdleglosc\_euklidesowa(); //zwraca odleglosci euklidesowe

Layer(int liczba\_neuronow, int numberOfInputs, double wspolczynnik\_uczenia, double iterationsNumber);

};

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

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

this->liczba\_neuronow = liczba\_neuronow;

neuron.resize(liczba\_neuronow);

this->czas = iterationsNumber / this->promien;

this->promien = 5;

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

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

}

void Layer::minimum\_odleglosc\_euklidesowa() { //szuka najmniejszej odleglosci euklidesowej

double tmp = scalarProducts[0];

this->zwycieski\_neuron = 0;

for (int i = 1; i < scalarProducts.size(); i++) {

if (tmp < scalarProducts[i]) {

this->zwycieski\_neuron = i;

tmp = scalarProducts[i];

}

}

}

void Layer::zmiana\_wag(double obecna\_iteracja, bool uczing) {

minimum\_odleglosc\_euklidesowa();

getOdleglosc\_euklidesowa();

neuron[zwycieski\_neuron].funkcja\_aktywacji();

if (uczing) {

neuron[zwycieski\_neuron].oblicz\_odleglosc\_od\_zwycieskiego(promien, obecna\_iteracja, czas);// szukanie neuronow w otoczeniu wygraneno neuronu

int promien = neuron[zwycieski\_neuron].odleglosc\_od\_zwycieskiego;

int leftBorderNeuronIndex = 0;

int rightBorderNeuronIndex = 0;

if (zwycieski\_neuron - promien < 0) //sprawdzenie czy dany neuron miesci sie w siatce

leftBorderNeuronIndex = 0;

else

leftBorderNeuronIndex = zwycieski\_neuron - promien;

if (zwycieski\_neuron + promien >= liczba\_neuronow)

rightBorderNeuronIndex = liczba\_neuronow - 1;

else

rightBorderNeuronIndex = zwycieski\_neuron + promien;

promien = (promien <= 0) ? 0 : --promien;

for (int i = leftBorderNeuronIndex; i < rightBorderNeuronIndex; i++) {

neuron[i].odl\_euklides = (i < zwycieski\_neuron) ? (zwycieski\_neuron - i) : (i - zwycieski\_neuron); //zmiana wag dla neuronow z otoczenia

neuron[i].odleglosc\_od\_zwycieskiego = neuron[zwycieski\_neuron].odleglosc\_od\_zwycieskiego;

neuron[i].nowa\_waga();

}

}

}

void Layer::getOdleglosc\_euklidesowa() {

scalarProducts.clear();

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

scalarProducts.push\_back(neuron[i].oblicz\_odleglosc\_skalar());

}  
 **Neuron.h**class Neuron {

public:

vector<double> wejscia;

vector <double> wagi;

double wartosc\_wyjscie;

double odl\_euklides;

double odleglosc\_od\_zwycieskiego;

double wspolczynnik\_uczenia;

double wartosc\_sasiedztwa; //wartosc funkcji sasiedztwa (Gaussian neighborhood function)

double sumowanie\_wejsc;

void normalizacja\_wag\_zaktualizowanych();

double losowanie\_Wag();

void oblicz\_odleglosc\_od\_zwycieskiego(); //oblicza wartosc funkcji sasiedztwa (Gaussian neighborhood function)

void nowa\_waga();

void stworz\_wejscie(int liczba\_wejsc); //ustawienie wejsc na 0 i skorzystanie z metody : losowanie\_Wag())

void funkcja\_aktywacji(); //funkcja sigmoidalna

double oblicz\_odleglosc\_skalar();

void oblicz\_odleglosc\_od\_zwycieskiego(double promien, double obecna\_iteracja, double czas);

Neuron(); // konstruktory

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

int getRozmiar\_wejsc() { return wejscia.size(); } //zwraca rozmiar wejsca

int getRozmiar\_wag() { return wagi.size(); } // podaje wage

};  
  
 **Neuron.cpp**Neuron::Neuron() {

this->wejscia.resize(0);

this->wagi.resize(0);

this->sumowanie\_wejsc = 0.0;

this->wartosc\_wyjscie = 0.0;

this->wspolczynnik\_uczenia = 0.0;

}

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

stworz\_wejscie(liczba\_wejsc);

normalizacja\_wag\_zaktualizowanych();

this->wspolczynnik\_uczenia = wspolczynnik\_uczenia;

this->sumowanie\_wejsc = 0.0;

this->wartosc\_wyjscie = 0.0;

}

void Neuron::stworz\_wejscie(int liczba\_wejsc) { //stworzenie poczatkowych wejsc(ustawienie wejsc na 0, wykorzystanie metody losowanie\_Wag())

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

wejscia.push\_back(0);

wagi.push\_back(losowanie\_Wag());

}

}

double Neuron::oblicz\_odleglosc\_skalar() {

sumowanie\_wejsc = 0.0;

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

sumowanie\_wejsc += pow(wejscia[i] - wagi[i], 2);

sumowanie\_wejsc = sqrt(sumowanie\_wejsc);

return sumowanie\_wejsc;

}

void Neuron::funkcja\_aktywacji() {

double beta = 1.0;

this->wartosc\_wyjscie = (1.0 / (1.0 + (exp(-beta \* sumowanie\_wejsc))));

}

void Neuron::nowa\_waga() {

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

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

normalizacja\_wag\_zaktualizowanych();

}

void Neuron::normalizacja\_wag\_zaktualizowanych() {

double vectorodl\_euklides = 0.0;

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

vectorodl\_euklides += pow(wagi[i], 2);

vectorodl\_euklides = sqrt(vectorodl\_euklides);

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

wagi[i] /= vectorodl\_euklides;

}

void Neuron::oblicz\_odleglosc\_od\_zwycieskiego(double promien, double currentIteraton, double czasConstant) {

this->odleglosc\_od\_zwycieskiego = promien \* exp(-currentIteraton / czasConstant);

}

void Neuron::oblicz\_odleglosc\_od\_zwycieskiego() {

this->wartosc\_sasiedztwa = exp(-pow(this->odl\_euklides, 2) / (2 \* pow(this->odleglosc\_od\_zwycieskiego, 2)));

}

double Neuron::losowanie\_Wag() {

double max = 1.0;

double min = 0.0;

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

return weight;

}  
**Source.cpp**int main() {

srand(time(NULL));

vector<vector<double>> dane\_testujace;

vector<vector<double>> dane\_uczace;

int liczba\_neuronow = 20;

int liczba\_wejsc = 35;

double wspolczynnik\_uczenia = 0.05;

int epoka = 50;

Layer siec\_Kohonena(liczba\_neuronow, liczba\_wejsc, wspolczynnik\_uczenia, epoka);

wczytaj\_dane\_uczace(dane\_uczace, liczba\_wejsc);

wczytaj\_dane\_testujace(dane\_testujace, liczba\_wejsc);

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

for (int epokaNumber = 1, i = 0; i < epoka; i++, epokaNumber++) {

ucz(siec\_Kohonena, dane\_uczace); // ropoczynanie procesu uczenia

OUTPUT\_FILE\_uczING << "epoka: " << epokaNumber << endl;

cout << "epoka: " << epokaNumber << endl;

}

OUTPUT\_FILE\_uczING.close();

OUTPUT\_FILE\_strumien\_danych\_testujacych.open("output\_strumien\_danych\_testujacych.txt", ios::out);

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

test(siec\_Kohonena, dane\_testujace);

OUTPUT\_FILE\_uczING.close();

OUTPUT\_FILE\_strumien\_danych\_testujacych.close();

system("pause");

return 0;

}

void dane\_wejsciowe(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 ucz(Layer& layer, vector<vector<double>> inputData)

{

static int obecna\_iteracja = 0;

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

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

dane\_wejsciowe(layer.neuron[i], inputData, layer.neuron[i].getRozmiar\_wejsc(), rowOfData);

layer.neuron[i].oblicz\_odleglosc\_skalar();

}

layer.zmiana\_wag(obecna\_iteracja, true);

OUTPUT\_FILE\_uczING << layer.zwycieski\_neuron << endl;

cout << "Winner: " << layer.zwycieski\_neuron << endl;

obecna\_iteracja++;

}

}

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

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

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

dane\_wejsciowe(layer.neuron[i], inputData, layer.neuron[i].getRozmiar\_wejsc(), rowOfData);

layer.neuron[i].oblicz\_odleglosc\_skalar();

}

char letter = 'A';

layer.zmiana\_wag(0, false);

OUTPUT\_FILE\_strumien\_danych\_testujacych << layer.neuron[layer.zwycieski\_neuron].getRozmiar\_wejsc() << endl;

OUTPUT\_FILE\_TESTING\_NEURON << (char)(letter + rowOfData) << " " << layer.zwycieski\_neuron << endl;

cout << (char)(letter + rowOfData) << " " << layer.zwycieski\_neuron << endl;

}

}

void wczytaj\_dane\_testujace(vector<vector<double>> &dane\_testujace, int liczba\_wejsc) {

strumien\_danych\_testujacych.open("datatest.txt", ios::in);

vector<double> row;

double odl\_euklides = 0;

while (!strumien\_danych\_testujacych.eof()) {

row.clear();

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

double inputTmp = 0.0;

strumien\_danych\_testujacych >> inputTmp;

row.push\_back(inputTmp);

}

for (int i = 0; i < liczba\_wejsc; i++) //znormalizowanie danych uczacych

odl\_euklides += pow(row[i], 2);

odl\_euklides = sqrt(odl\_euklides);

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

row[i] /= odl\_euklides;

dane\_testujace.push\_back(row);

}

strumien\_danych\_testujacych.close();

}

void wczytaj\_dane\_uczace(vector<vector<double>> &inputData, int liczba\_wejsc) {

strumie\_danych\_uczacych.open("data.txt", ios::in);

double odl\_euklides = 0;

vector<double> row;

do {

row.clear();

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

double inputTmp = 0.0;

strumie\_danych\_uczacych >> inputTmp;

row.push\_back(inputTmp);

}

for (int i = 0; i < liczba\_wejsc; i++)//znormalizowanie danych uczacych

odl\_euklides += pow(row[i], 2);

odl\_euklides = sqrt(odl\_euklides);

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

row[i] /= odl\_euklides;

inputData.push\_back(row);

} while (!strumie\_danych\_uczacych.eof());

strumie\_danych\_uczacych.close();

}