Weronika Muszyńska, gr. 03

**Sprawozdanie z ćwiczenia nr 6**

**Temat ćwiczenia:**

Budowa i działanie sieci Kohonena dla WTM.

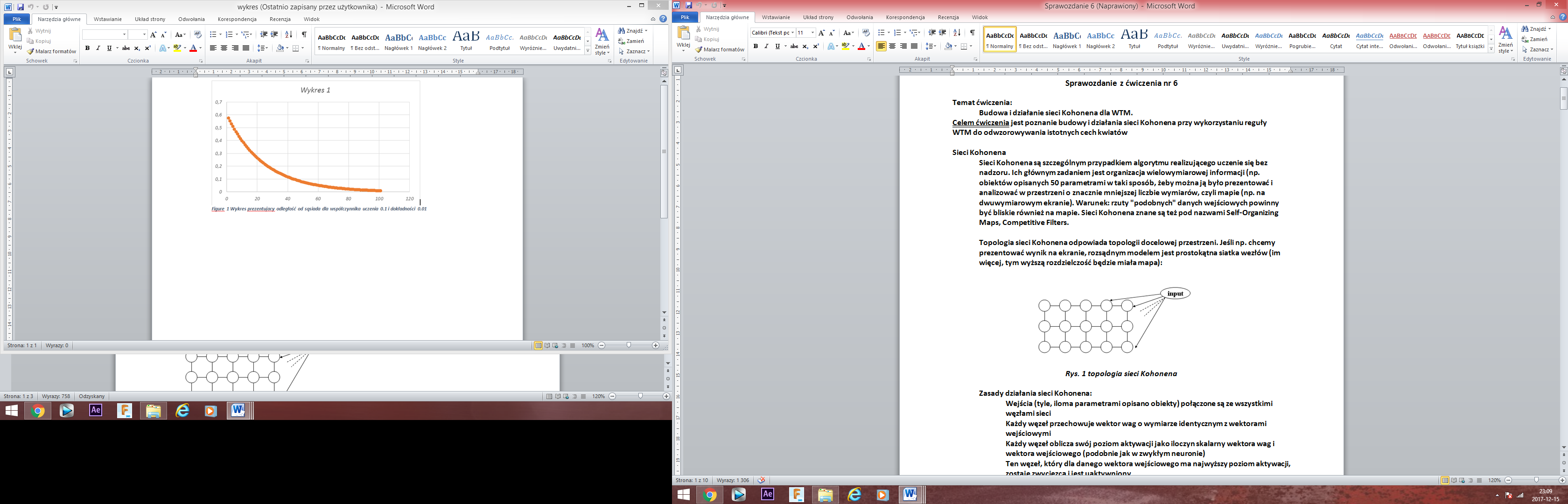
**Celem ćwiczenia** jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły

WTM do odwzorowywania istotnych cech kwiatów

**Sieci Kohonena**

Sieci Kohonena są szczególnym przypadkiem algorytmu realizującego uczenie się bez nadzoru. Ich głównym zadaniem jest organizacja wielowymiarowej informacji (np. obiektów opisanych 50 parametrami w taki sposób, żeby można ją było prezentować i analizować w przestrzeni o znacznie mniejszej liczbie wymiarów, czyli mapie (np. na dwuwymiarowym ekranie). Warunek: rzuty "podobnych" danych wejściowych powinny być bliskie również na mapie. Sieci Kohonena znane są też pod nazwami Self-Organizing Maps, Competitive Filters.

Topologia sieci Kohonena odpowiada topologii docelowej przestrzeni. Jeśli np. chcemy prezentować wynik na ekranie, rozsądnym modelem jest prostokątna siatka wezłów (im więcej, tym wyższą rozdzielczość będzie miała mapa):



***Rys. 1 topologia sieci Kohonena***

Zasady działania sieci Kohonena:

Wejścia (tyle, iloma parametrami opisano obiekty) połączone są ze wszystkimi węzłami sieci

Każdy węzeł przechowuje wektor wag o wymiarze identycznym z wektorami wejściowymi

Każdy węzeł oblicza swój poziom aktywacji jako iloczyn skalarny wektora wag i wektora wejściowego (podobnie jak w zwykłym neuronie)

Ten węzeł, który dla danego wektora wejściowego ma najwyższy poziom aktywacji, zostaje zwycięzcą i jest uaktywniony

Wzmacniamy podobieństwo węzła-zwycięzcy do aktualnych danych wejściowych poprzez dodanie do wektora wag wektora wejściowego (z pewnym współczynnikiem uczenia)

Każdy węzeł może być stowarzyszony z pewnymi innymi, sąsiednimi węzłami - wówczas te węzły również zostają zmodyfikowane, jednak w mniejszym stopniu.

Inicjalizacja wag sieci Kohonena jest losowa. Wektory wejściowe stanowią próbę uczącą, podobnie jak w przypadku zwykłych sieci rozpatrywaną w pętli podczas budowy mapy. Wykorzystanie utworzonej w ten sposób mapy polega na tym, że zbiór obiektów umieszczamy na wejściu sieci i obserwujemy, które węzły sieci się uaktywniają. Obiekty podobne powinny trafiać w podobne miejsca mapy.

Odmianą uczenia konkurencyjnego jest uczenie typu WTM (Winner Takes Most), w którym neuron wygrywający konkurencję uaktywnia się w sposób maksymalny przyjmując wartość sygnału wyjściowego ym=1 y1 ym yp zwycięski neuron w 11 w pn x1 xn xi i umożliwiając częściowe uaktywnienie innych neuronów z sąsiedztwa. Stopień uaktywnienia neuronów z sąsiedztwa zależy od odległości ich wektorów wagowych od wag neuronu wygrywającego.

Algorytmy WTA, w których tylko jeden neuron może podlegać adaptacji w każdej iteracji, są algorytmami słabo zbieżnymi, szczególnie przy dużej liczbie neuronów. W praktyce zostały one zastąpione algorytmami WTM (ang. Winner Takes Most), w których oprócz zwycięzcy uaktualniają swoje wagi również neurony z jego sąsiedztwa.

**Zestaw danych uczących**

Dane uczące składają się z tablicy 20 liczb reprezentujących 20 wielkich liter:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Każdy wers określa jedną daną uczącą. Jeden rekord składa się z 25 liczb oddzielonych przecinkami określającymi jedną literkę w postaci tablicy 5x5 oraz na końcu literka odpowiadająca temu zestawu. „1” oznacza że w danym punkcie istnieje zawartość, natomiast wartość „0” oznacza, że pole jest puste.

1,1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,A

1,1,1,0,0,1,0,0,1,0,1,1,1,0,0,1,0,0,1,0,1,1,1,0,0,B

1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1,C

1,1,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,1,1,0,0,D

1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,E

1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,F

1,1,1,1,0,1,0,0,0,0,1,0,1,1,0,1,0,0,1,0,1,1,1,1,0,G

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,H

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,I

1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,1,1,1,0,0,J

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,K

1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,L

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,M

1,1,0,0,1,1,1,1,0,1,1,0,1,1,1,1,0,0,1,1,1,0,0,0,1,N

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,O

1,1,1,1,0,1,0,0,1,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0,P

1,1,1,1,0,1,0,0,1,0,1,1,1,1,0,1,0,1,0,0,1,0,0,1,0,R

1,1,1,0,0,1,0,0,0,0,1,1,1,0,0,0,0,1,0,0,1,1,1,0,0,S

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,T

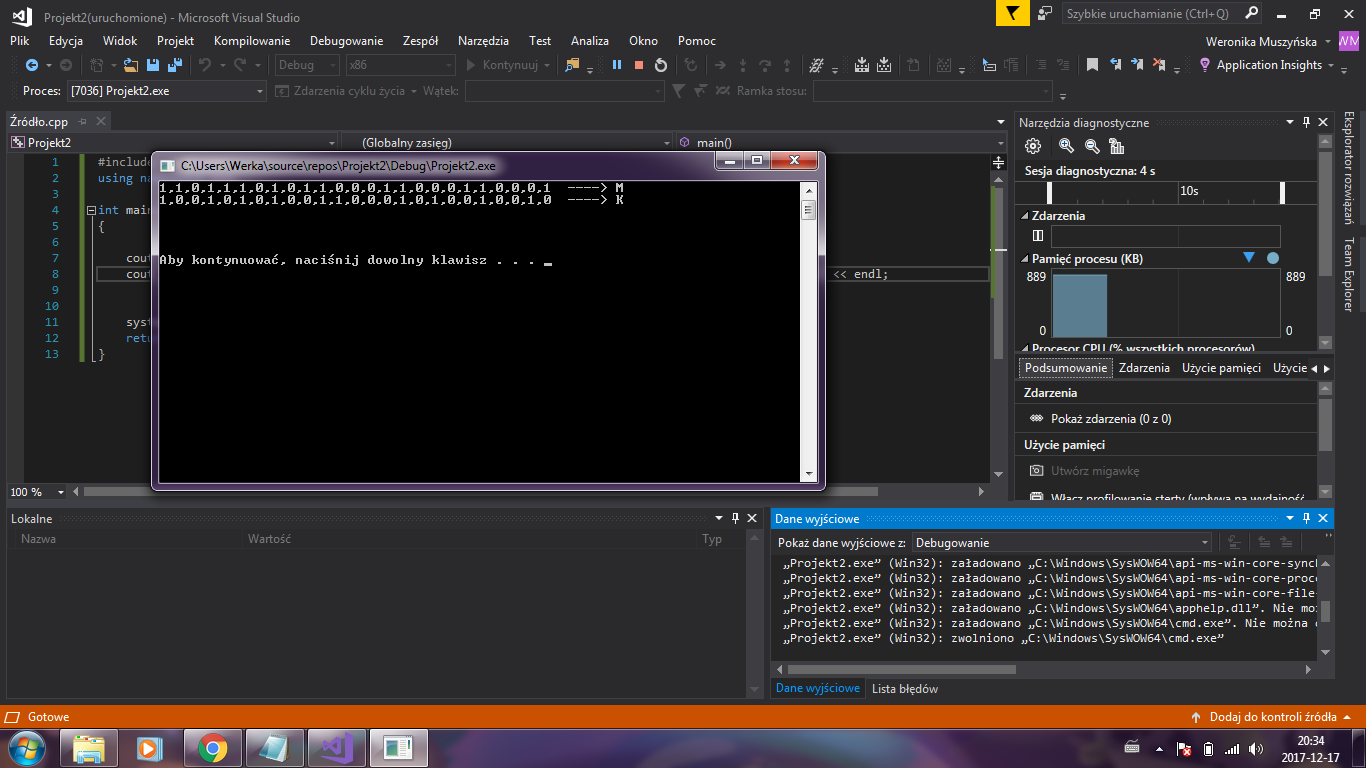
1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,1,1,1,0,U

**Zestaw danych testujących:**

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

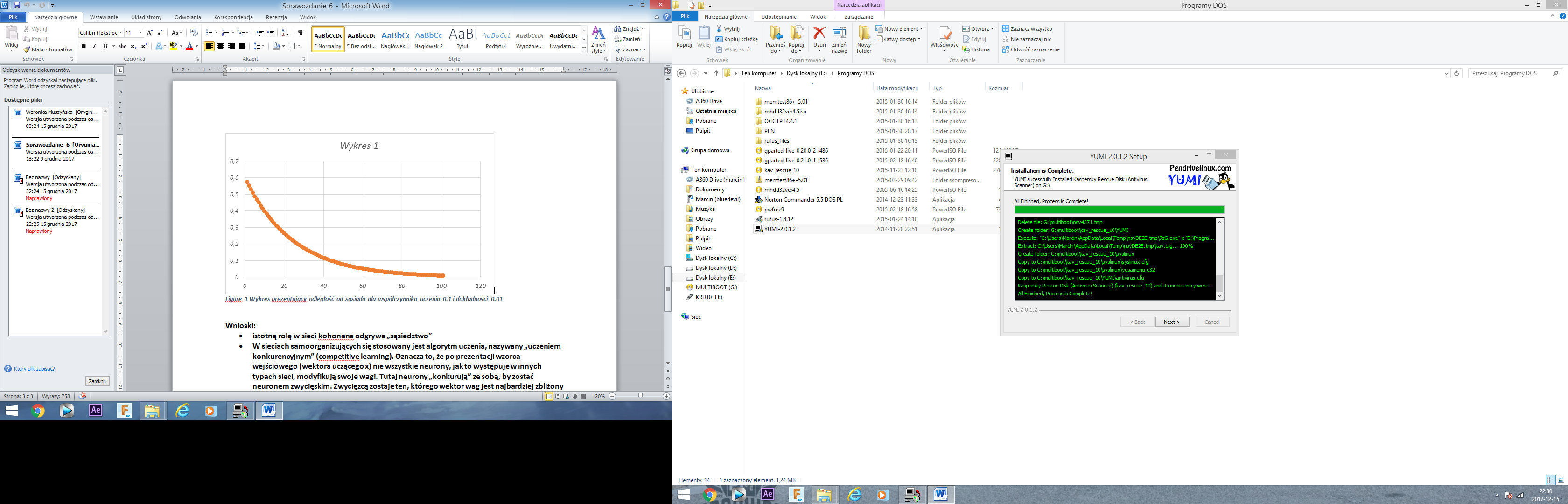
1,0,0,1,0,1,0,1,0,0,1,1,0,0,0,1,0,1,0,0,1,0,0,1,0

**Wyniki :**



Rys. 2 Wynik z konsoli dla współczynnika uczenia 0.1 i dokładności 0.01

**Zestawienie wyników :**



Rys. 3 Wykres przedstawiający odległość od sąsiada dla współczynnika uczenia 0.1 i dokładności 0.01

Wnioski:

Istotną rolę w sieci kohonena odgrywa „sąsiedztwo”

Ważnym elementem dla zasady WTM jest określenie tzw. sąsiedztwa. Jest ono rozumiane w sensie geometrycznym jako położenie neuronu względem zwycięzcy. Zasięg sąsiedztwa określany jest jako tzw. promień sąsiedztwa.

Wraz ze zwiększaniem wartości współczynnika uczenia zmniejsza się liczba iteracji, co ma wpływ na działanie programu i powoduje uzyskanie błędnych wyników. Aby zredukować liczbę błędnych wyników można zwiększyć liczbę iteracji.

Dobór wag ma bezpośredni wpływ na działanie perceptronów w sieci, w zależności od ich wartości zmienia się poprawność wyników i liczba iteracji. Wartości wag mają największy wpływ na efekt końcowy, często wagi są ustalane losowo.

Dane uczące wpływają na poprawność uczenia perceptronu, przy zbyt małej ilości otrzymujemy błędne wyniki. Aby uzyskać lepsze wyniki powinniśmy dostarczyć wystarczającą liczbę danych wejściowych.

W sieciach samoorganizujących się stosowany jest algorytm uczenia, nazywany „uczeniem konkurencyjnym” (competitive learning). Oznacza to, że po prezentacji wzorca wejściowego (wektora uczącego x) nie wszystkie neurony, jak to występuje w innych typach sieci, modyfikują swoje wagi. Tutaj neurony „konkurują” ze sobą, by zostać neuronem zwycięskim. Zwycięzcą zostaje ten, którego wektor wag jest najbardziej zbliżony (ma najmniejszą odległość) do prezentowanego wzorca wejściowego.

Tylko neuron wygrywający oraz ewentualnie neurony znajdujące się w jego sąsiedztwie mają możliwość uaktualnienia swoich wag, tak by jeszcze bardziej zbliżyć je do podanego właśnie wektora uczącego.

Gdy tylko neuron zwycięski zmienia swoje wagi, mamy do czynienia z najbardziej skrajną formą uczenia konkurencyjnego, zwaną „zwycięzca bierze wszystko”

**Źródła:**

1. S. Osowski. Sieci neuronowe w ujęciu algorytmicznym. WNT, Warszawa 1997.
2. J. Żurada, M. Barski, W. Jędruch. Sztuczne sieci neuronowe. PWN, Warszawa 1996.
3. R. Tadeusiewicz, Sieci neuronowe. Warszawa : Akademicka Oficyna Wydaw. RM, 1993.

**Listing kodu:**

Main.js:

var synaptic = require('synaptic');

var fs = require('fs');

var Layer = synaptic.Layer,

Network = synaptic.Network;

var inputLayer = new Layer(25);

var hiddenLayer = new Layer(5);

var outputLayer = new Layer(20);

inputLayer.project(hiddenLayer);

hiddenLayer.project(outputLayer);

var network = new Network({

input: inputLayer,

hidden: [hiddenLayer],

output: outputLayer

});

var som = require('node-som');

var somInstance = new som({

inputLength: 25,

maxClusters: 10,

loggingEnabled:false,

target: 0.1,

scale:{

min:0,

max:10

}

});

somInstance.train(trainingData);

var sample = [0,1,1,1,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,0,1];

var group = somInstance.classify(sample);

console.log(getFlowerName2(group));

var trainingData = [];

fs.readFile('learning\_data.txt','utf8',function(err, data){

if(err) throw err;

var lines = data.split("\n");

for(var i = 0; i < lines.length; i++){

var line = lines[i].trim();

var splitLine = line.split(",");

var input = splitLine.slice(0, 25);

var output = splitLine[25]=='A' ? [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'B' ? [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'C' ? [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'D' ? [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'E' ? [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'F' ? [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'G' ? [0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'H' ? [0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'I' ? [0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'J' ? [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'K' ? [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]

: splitLine[25] == 'L' ? [0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0]

: splitLine[25] == 'M' ? [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0]

: splitLine[25] == 'N' ? [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0]

: splitLine[25] == 'O' ? [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0]

: splitLine[25] == 'P' ? [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]

: splitLine[25] == 'R' ? [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]

: splitLine[25] == 'S' ? [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]

: splitLine[25] == 'T' ? [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]

: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1];

trainingData.push({

input: input,

output: output

});

}

var learningRate = .01;

for (var i = 1; i <= 10000; i++)

{

for(var j = 0; j < trainingData.length; j++){

network.activate(trainingData[j].input);

network.propagate(learningRate, trainingData[j].output);

}

}

fs.readFile('testing\_data.txt','utf8',function(err, data){

if(err) throw err;

var lines = data.split("\n");

for(var i = 0; i < lines.length; i++){

var input = lines[i].trim().split(",");

var result = getFlowerName(network.activate(input));

console.log(lines[i].trim()+" ---> "+result);

}

});

});

function getLargestIndex(arr){

var result = 0;

for(var i = 1; i < arr.length; i++)

if(arr[i] > arr[result])

result = i;

return result;

}

function getFlowerName(arr){

var index = getLargestIndex(arr);

if(index == 0) return "A";

if(index == 1) return "B";

if(index == 2) return "C";

if(index == 3) return "D";

if(index == 4) return "E";

if(index == 5) return "F";

if(index == 6) return "G";

if(index == 7) return "H";

if(index == 8) return "I";

if(index == 9) return "J";

if(index == 10) return "K";

if(index == 11) return "L";

if(index == 12) return "M";

if(index == 13) return "N";

if(index == 14) return "O";

if(index == 15) return "P";

if(index == 16) return "R";

if(index == 17) return "S";

if(index == 18) return "T";

return "U";

}

function getFlowerName2(index){

if(index == 0) return "A";

if(index == 1) return "B";

if(index == 2) return "C";

if(index == 3) return "D";

if(index == 4) return "E";

if(index == 5) return "F";

if(index == 6) return "G";

if(index == 7) return "H";

if(index == 8) return "I";

if(index == 9) return "J";

if(index == 10) return "K";

if(index == 11) return "L";

if(index == 12) return "M";

if(index == 13) return "N";

if(index == 14) return "O";

if(index == 15) return "P";

if(index == 16) return "R";

if(index == 17) return "S";

if(index == 18) return "T";

return "U";

}