**Sprawozdanie z projektu nr 6**

Monika Pawlik, gr 3

**Temat ćwiczenia:**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 alfabetu.

1. **Syntetyczny opis budowy sieci Kohonena**

**Sieć Kohonena** – [sieć neuronowa](https://pl.wikipedia.org/wiki/Sie%C4%87_neuronowa) uczona w trybie bez nauczyciela w celu wytworzenia niskowymiarowej (przeważnie dwuwymiarowej) [zdyskretyzowanej](https://pl.wikipedia.org/wiki/Matematyka_dyskretna) reprezentacji przestrzeni wejściowej. Sieć Kohonena wyróżnia się tym od innych sieci, że zachowuje odwzorowanie sąsiedztwa przestrzeni wejściowej.

**Zasada uczenia sieci Kohonena :**

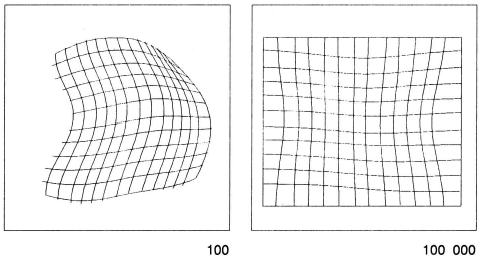
- wektor wejściowy X jest przed procesem uczenia znormalizowany tak, aby jego długość ||X|| = 1

- numer poddawanego treningowi neuronu m nie jest przypadkowy, tylko jest to ten neuron, którego   
 sygnał wyjściowy jest największy. Oznacza to, że przy każdorazowym podaniu sygnału wejściowego   
 neurony rywalizują ze sobą i wygrywa ten, który uzyskał największy sygnał wyjściowy.

- tylko zwycięski neuron podlega uczeniu, którego efektem jest lepsze dopasowanie wektora wag do   
 rozpoznawania obiektów podobnych do X(j) .

- reguła uczenia Kohonena jest wzbogacana o dodatkowy element związany z topologią uczącej się sieci. Neurony w sieci są uporządkowane, można wprowadzić pojęcie sąsiedztwa – uczeniu podlega neuron wygrywający w konkurencji z innymi neuronami sieci, ale także neurony, które z nim sąsiadują.

Dla sieci neuronowych dwuwymiarowych można odwzorować strukturę obiektu – utworzyć siatkę równomiernie pokrywającą określony kształt analizowanego obszaru. Poniżej znajdują się rysunki dla jednego z przykładowych zadań.



1. **Opis algorytmu uczenia opartego o regułę Winner Takes Most (WTM)**

**WTA – Winner Takes All** – Zwycięzca bierze wszystko. Po przedstawieniu sieci wektora wejściowego, neuron najbardziej podobny do elementu prezentowanego (którego wagi są najbardziej podobne składowym wektora wejściowego) zostaje zmodyfikowany zgodnie z funkcją f tak aby jego wagi były jak najbardziej zbliżone do wektora wejściowego. Neurony niewygrywające pozostają martwe. Numer v wygranego neuronu ustala się na podstawie kryterium:

Wagi zwycięskiego neuronu mogą zostać uaktualnione zgodnie z regułą Kohonena:



**Punkty z krokami algorytmu WTA:**

1. inicjalizacja wag sieci

2. wzór na obliczenie odległości wektora wejściowego do wag każdego z neuronów:   
 d = sqrt((w1 – x1)2 + (w2 – x2)2)

3. wybranie neuronu zwycięzcy (wygrywającego) dla którego odległość wag od wektora wejściowego   
 jest najmniejsza (na podstawie Euklidesowej miary odległości).

4. zmiana wartości poszczególnych wag tego neuronu przyjmując, że szybkość nauki wynosi n = 0,1

5. powtórzenie kroków 2-5 dla wszystkich przykładów uczących.

1. **Wygenerowanie danych uczących i testujących**

Dane uczące i testujące zawierają 20 liter dużych liter alfabetu w postaci dwuwymiarowej tablicy dla jednej litery.

Przykładowe dane:

0 1 1 1 0 1 1 1 1 0

1 0 0 0 1 1 0 0 0 1

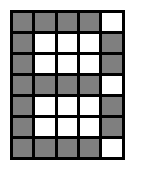
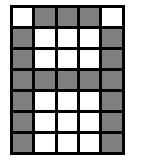
1 0 0 0 1 1 0 0 0 1

1 1 1 1 1 **A** 1 1 1 1 0 **B**

1 0 0 0 1 1 0 0 0 1

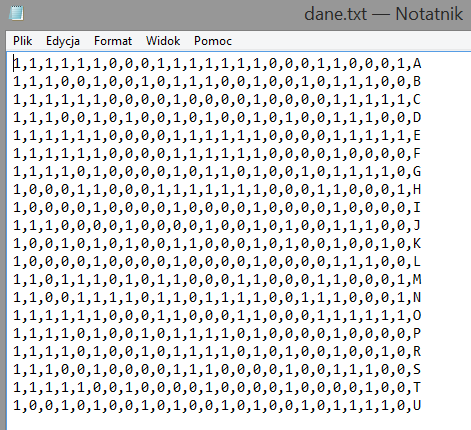
1 0 0 0 1 1 0 0 0 1

1 0 0 0 1 1 1 1 1 0



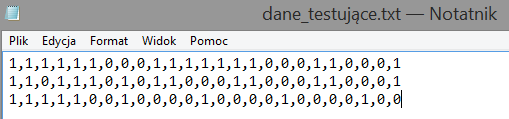
Białym pikselom jest przyporządkowana liczba - 0, szarym - 1.

Poniżej został przedstawiony screen pliku dane.txt zawierającego dane uczące, dla potrzeb implementacji programu litery zostały zaprezentowane jako wektory ciągu uczącego.



Dane testujące :

(posiadają podobny układ jak w pliku z danymi uczącymi, lecz są bez litery)



1. **Implementacja sieci Kohonena i algorytmu uczenia**

Krótkie omówienie implementacji programu napisanego w JavaScript:

- ustanowienie liczby warstw: wejściowej, ukrytej i wyjściowej  
 var inputLayer = new Layer(25);   
 var hiddenLayer = new Layer(5);  
 var outputLayer = new Layer(20);

- stworzenie sieci

var network = new Network({

input: inputLayer,

hidden: [hiddenLayer],

output: outputLayer

});

- czytanie danych z pliku uczącego  
 fs.readFile('dane.txt','utf8',function(err, data){

- ustawienie wartości współczynnika uczenia i trenowanie sieci  
 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);

}

}

- czytanie danych testowych z pliku: dane\_testujące.txt  
 fs.readFile('dane\_testujące.txt','utf8',function(err, data){

- funkcja do obliczania indexu danej litery  
 function getLargestIndex(arr){

var result = 0;

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

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

result = i;

return result;

}

- funkcja odczytująca indexy liter  
 function getFlowerName(arr){

var index = getLargestIndex(arr);

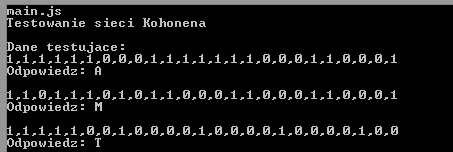
if(index == 0) return "A";

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

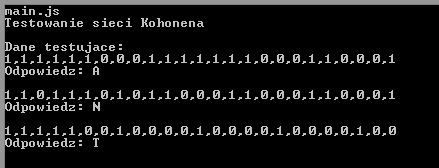
if(index == 2) return "C";

1. **Zestawienie otrzymanych wyników**

Screen konsoli przedstawiającej testowanie sieci dla współczynnika uczenia od 0,2 do 1



Dla współczynnika uczenia = 0,1 jedna odpowiedź sieci była błędna.



Zamiast literki N powinna być literka M. Reprezentacja tych liter jest do siebie bardzo podobna, oraz współczynnik uczenia był bardzo niski i sieć podała błędną odpowiedź.

Wykres przedstawia odległość od sąsiada dla współczynnika uczenia

Odległość od sąsiada – promień sąsiedztwa maleje w trakcie uczenia sieci, zwykle liniowo. Określenie sąsiedztwa jest rozumiane w sensie geometrycznym jako położenie neuronu względem zwycięzcy. Zasięg sąsiedztwa określany jest jako tzw. promień sąsiedztwa.

1. **Analiza i dyskusja błędów uczenia i testowania oraz wyłonionych cech dla wyników opracowanej sieci w zależności od wartości współczynnika uczenia**

Sieci neuronowe mają zdolność do uczenia się, czyli zdolność do samodzielnego dostosowywania współczynników wagowych. Uczenie sieci to wymuszenie na niej określonego zareagowania na sygnały wejściowe. Sieci neuronowe potrafią samodzielnie dostosować się do zmieniających warunków. Im większa jest liczba danych uczących, tym sieć zachowuje się lepiej.

Przy konstrukcji skutecznego algorytmu uczenia sieci Kohonena istotny jest dobór wielkości odległości od sąsiada oraz współczynnik uczenia.

Uczenie zostało przeprowadzone dla różnych współczynników uczenia. Współczynnik uczenia może przyjmować wartości z przedziału (0,1).

Im mniejszy współczynnik uczenia, tym mniejsza liczba iteracji, co ma wpływ na działanie programu, ponieważ zdarzały się błędne wyniki. Zmiany wag neuronów nie były bardzo skokowe. Wybranie małego współczynnika uczenia doprowadziło do bardzo powolnego procesu uczenia (wagi były poprawiane w każdym kroku bardzo słabo, żeby więc osiągnęły pożądane wartości trzeba było wykonać bardzo dużo takich kroków).

Im większa liczba danych uczących tym sieć szybciej się uczy. Podczas testów okazało się że dla 20 danych reakcja sieci była poprawna. Ale należy uważać aby sieć nie przeuczyć. Dla bardzo dużej liczby iteracji odpowiedzi sieci były różne, częściej dobre, ale nie zawsze.

1. **Wnioski**

W zależności od wartości współczynnika uczenia sieć uczy się inaczej, dla bardzo niskiej wartości – zdarzają się błędy.

Dla sieci samoorganizującej Kohonena ilość danych uczących jest ważna – przy zbyt małej można otrzymać błędne wyniki. W programie zbiór 20 wersów danych okazał się wystarczający.

Cechy charakterystyczne sieci Kohonena:  
- sieć uczy się bez nadzoru  
- uczony jest głównie neuron zwycięski – neuron zwycięzca oraz neurony z nim sąsiadujące   
 uaktualniają swoje wagi w takim kierunku, że podążają one za wektorem wejściowym  
- ważną rolę odgrywa sąsiedztwo  
- w wyniku uczenia powstaje mapa topologiczna

W sieciach samoorganizujących się stosowany jest algorytm uczenia, nazywany „uczeniem konkurencyjnym”. W testowanym programie jest to WTA. 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.

1. **Listing kodu**

var synaptic = require('synaptic');

var fs = require('fs');

var Layer = synaptic.Layer,

Network = synaptic.Network;

// set up neural 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 = [];

// read training data file

fs.readFile('iris.txt','utf8',function(err, data){

if(err) throw err;

// load data into training data array

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

});

}

// train the network

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);

}

// if(i%1000 == 0)

// console.log("Training... "+i/100+"% complete. ");

//console.log(network.log);

}

// use the network to classify flowers based on testing data

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

if(err) throw err;

console.log("\n\nResults\n===============================\n");

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);

}

});

});

// helper functions

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";

}