

**Podstawy Sztucznej Inteligencji**

Scenariusz 5: Budowa i działanie sieci Kohonena dla WTA

Wykonała:

**Magdalena Migas**

**Wydział Inżynierii Metali i Informatyki Przemysłowej**

**Informatyka Stosowana**

1. **Cel ćwiczenia**

Wykonane ćwiczenie miało na celu poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły Winner Takes All do odwzorowywania istotnych cech kwiatów na przykładzie irysów.

1. **Reguła Winner Takes All**

***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).

Uczenie rywalizacyjne jest metodą uczenia sieci samoorganizujących. Neurony na początku mają przypadkowe wartości wag. Następnie, na podstawie prezentowanych danych, „uczą się” rozpoznawać te dane i zbliżają się odpowiednio do obszarów zajmowanych przez te dane. Po każdej prezentacji wzorca x(k) zwycięzcą zostaje tylko jeden neuron, najbliższy(w sensie ustalonej metryki) prezentowanemu wzorcowi. Neurony rywalizują między sobą, a „neuron wygrywający” jest jedynym pobudzanym przy konkretnej obserwacji wejściowej.

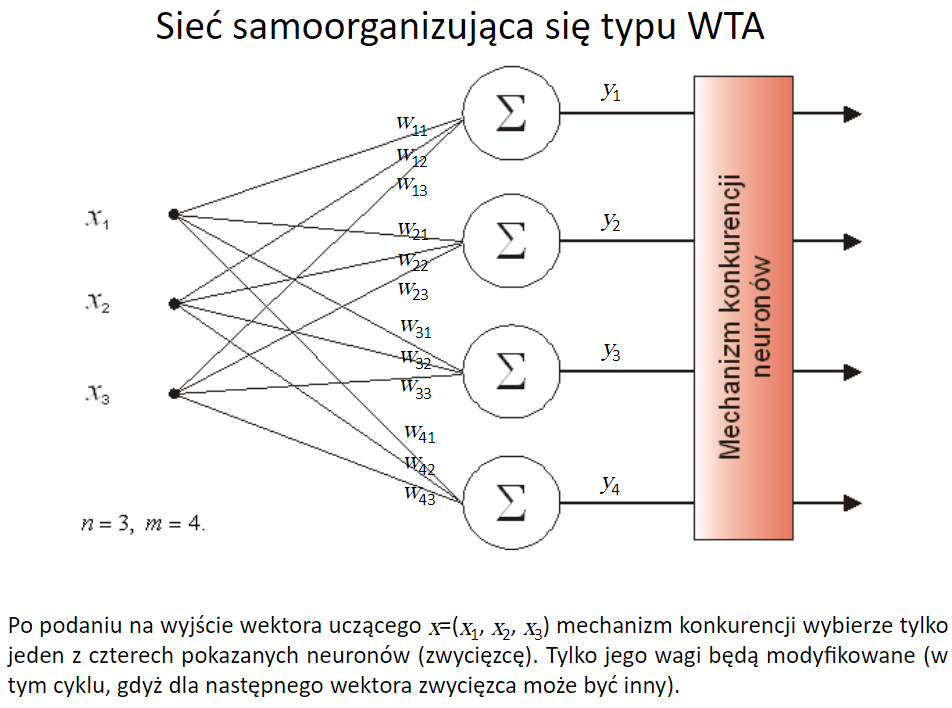
Neuron zwycięski ma prawo uaktualniać swoje wagi wg jednej z dwóch zasad:

* WTA – Winner Takes All
* WTM – Winner Takes Most

Sieci WTA klasyfikują wektory wejściowe do jednej z zadanych ***m*** klas. Neurony dopasowują swoje wagi w ten sposób, że przy prezentacji grup wektorów wejściowych zbliżonych do siebie zwycięża zawsze ten sam neuron. Neuron poprzez zwycięstwo we współzawodnictwie rozpoznaje swoją klasę.

**ALGORYTM UCZENIA WTA**

1. Przyjęcie losowych, znormalizowanych wartości wag poszczególnych neuronów.
2. Po podaniu pierwszego wektora wejściowego wyłaniany jest zwycięzca o numerze *v*:
3. Aktualizacja wag neuronu zwycięzcy wg tzw. Reguły Kohonena:



1. **Zestaw danych uczących**

**Iris flower data set** – jest to wielowymiarowy zestaw danych wprowadzony przez brytyjskiego statystyka i biologa Ronalda Fishera. Zestaw ten składa się z 50 próbek dla każdego z 3 gatunków Irysa (SETOSA, VIRGINICA, VERSICOLOR).

Dla każdej z próbek zmierzono wartości 4 cech(w centymetrach):

- długość płatków typu SEPAL;

- szerokość płatków typu SEPAL;

- długość płatków typu PETAL;

- szerokość płatków typu PETAL.

W oparciu o kombinację tych 4 cech Fisher opracował liniowy model dyskryminacyjny, aby odróżnić gatunki od siebie nawzajem.

**Iris versicolor Iris virginica Iris setosa**

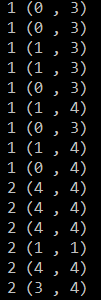
1. **Uzyskane wyniki**

Stworzona sieć Kohonena uczy się aż osiągnie określoną przez użytkownika wartość błędu. Ponadto grupuje ona kwiaty na matrycy o rozmiarze również zdefiniowanym przez użytkownika. W związku z powyższym w trakcie testów wykonano kilka wariantów, które różnią się od siebie przyjętą tolerancją błędu oraz wielkością matrycy wyjściowej.

**WARIANT I**

* Założona wartość błędu: 0.0000001;
* Rozmiar matrycy: 5x5

Rysunek 1: Pogrupowane kwiaty dla wariantu I

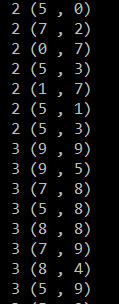


Rysunek 2: Fragment wydruku programu dla wariantu I

**WARIANT II**

* Założona wartość błędu: 0.0000001;
* Rozmiar matrycy: 10x10

Rysunek 3: Pogrupowane kwiaty dla wariantu II

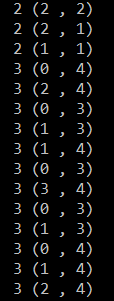


Rysunek 4: Fragment wydruku programu dla wariantu II

**WARIANT III**

* Założona wartość błędu: 0. 0001;
* Rozmiar matrycy: 5x5

Rysunek 5: Pogrupowane kwiaty dla wariantu III

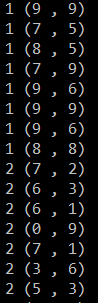


Rysunek 6: Fragment wydruku programu dla wariantu III

**WARIANT IV**

* Założona wartość błędu: 0. 0001;
* Rozmiar matrycy: 10x10

Rysunek 7: Pogrupowane kwiaty dla wariantu IV



Rysunek 8: Fragment wydruku programu dla wariantu IV

1. **Wnioski**

* Opracowane zagadnienie dotyczyło grupowania kwiatów irysa na 3 grupy na podstawie ich cech charakterystycznych przy wykorzystaniu sieci Kohonena i algorytmu Winner Takes All.
* Stworzoną sieć poddano testom. W ich trakcie przeprowadzono 4 warianty, które różniły się między sobą założonym błędem oraz rozmiarem wyjściowej matrycy. W rezultacie wykonano 2 testy dla matrycy 5x5(dla różnej wartości tolerowanego błędu) i 2 testy dla matrycy 10x10(również dla różnej wartości tolerowanego błędu).
* Ogólnie podsumowując przeprowadzone testy należy zaznaczyć, że w każdym z rozpatrywanych przypadków sieć poprawnie grupuje kwiaty gdyż na każdej z otrzymanych map wyjściowych możemy jednoznacznie wyróżnić 3 obszary – każdy należący do jednego z badanych kwiatów irysa.
* Pierwszym z wniosków, który się nasuwa po obserwacji uzyskanych wyników jest fakt, że zastosowanie większej matrycy powoduje, że rozróżnienie kwiatów jest bardziej wyraźne. Wynika to z faktu, że zbiór uczący składa się z 150 kwiatów i matryca 5x5 nie jest wystarczająca do jego przedstawienia.
* Kolejny wniosek dotyczy przyjętej tolerancji błędu. Widać w przypadku oczywistą zależność, że im mniejsza jest ta tolerancja tym wyniki są dokładniejsze.
* Ogólnie rzecz ujmując należy stwierdzić, że sieć Kohonena i algorytm WTA dobrze się sprawdza do zadań polegających na grupowaniu podanych mu wzorców.

1. **Literartura**

* <http://www.uci.agh.edu.pl/uczelnia/tad/inteligencja_obliczeniowa/09%20-%20Sieci%20Kohonena.pdf>
* <http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad4/w4.htm>
* <https://pl.wikipedia.org/wiki/Sie%C4%87_Kohonena>
* <http://galaxy.agh.edu.pl/~vlsi/AI/koho_t/>
* <https://en.wikipedia.org/wiki/Iris_flower_data_set>

1. **Listing kodu programu**

#include<iostream>

#include<cstdlib>

#include<fstream>

#include<list>

#include<ctime>

#include<math.h>

using namespace std;

class Neuron {

public:

int patternSize;

int inputSize;

double \*\*inputs;

double output;

double \*weights;

void calculateValue(int index);

void adjustWeights(int index,int it);

void drawWeights();

void normalizeData(double \*tab, int size);

void printWeights();

double learningCoefficient(int it);

Neuron();

};

Neuron::Neuron() {

fstream plik;

plik.open("zbior\_uczacy2.txt");

if (plik.good())

{

inputSize = 4;

patternSize = 30;

inputs = new double\*[patternSize];

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

inputs[i] = new double[inputSize];

for (int k = 0; k < patternSize; k++)

{

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

{

inputs[k][i] = 0;

}

}

weights = new double[4];

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

weights[i] = 0;

for (int k = 0; k < patternSize; k++)

{

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

{

plik >> inputs[k][i];

}

}

}

else

{

cout << "blad otwarcia pliku!";

}

drawWeights();

}

void Neuron::calculateValue(int index) {

output = 0;

for (int i = 0; i < inputSize - 1; i++)

{

output +=pow( inputs[index][i]-weights[i],2);

}

output = 1/sqrt(output);

}

void Neuron::drawWeights() {

double sum = 0;

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

{

weights[i] = 5 + (double)rand() / RAND\_MAX\* (10);

sum += pow(weights[i], 2);

}

}

void Neuron::normalizeData(double \*tab, int size)

{

double sum = 0;

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

{

sum += pow(tab[i], 2);

}

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

{

tab[i] =( tab[i] / sqrt(sum));

}

}

void Neuron::printWeights() {

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

{

cout << weights[i] << " ";

}

cout << endl;

}

void Neuron::adjustWeights(int index, int it) {

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

{

weights[i] = weights[i] + learningCoefficient(it)\*(inputs[index][i] - weights[i]);

}

}

double Neuron::learningCoefficient(int it) {

return exp(-it / 1000)\*0.1;

}

class KohonenNetwork {

public:

Neuron\* network;

double N;

void learnNetwork();

void testNetwork(double \*tab, int s);

KohonenNetwork(double N, int size);

int size;

};

KohonenNetwork::KohonenNetwork(double N, int size):N(N),size(size){

network = new Neuron[size];

}

void KohonenNetwork::learnNetwork() {

int index\_max = 0;

double max = 0;

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

{

for (int j = 0; j < 30; j++)

{

for (int k = 0; k < size; k++)

{

network[k].calculateValue(j);

if (network[k].output > max)

{

max = network[k].output;

index\_max = k;

}

}

network[index\_max].adjustWeights(j,i);

index\_max = 0;

max = 0;

}

}

}

void KohonenNetwork::testNetwork(double \*tab, int s) {

int index\_max = 0;

double max = 0;

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

{

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

{

network[i].output += (network[i].weights[j] \* tab[j]);

}

if (network[i].output > max)

{

max = network[i].output;

index\_max = i;

}

}

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

{

cout << i+1 << " " << network[i].output << endl;

}

}