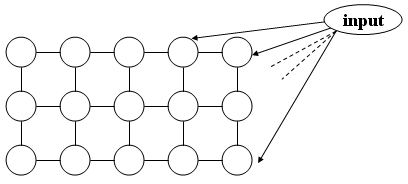
|  |  |  |
| --- | --- | --- |
| Grupa ćwicz. **2** | Data wykonania 22.12.2017 | Nr. Scenariusza  **6** |
| **Temat ćwiczenia:** Budowa i działanie sieci Kohonena dla WTM | | |
| Imię i nazwisko  **Andrzej Pawlikowski** | | Ocena i Uwagi |

**Cel ćwiczenia:**

Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTM do odwzorowywania istotnych cech liter alfabetu.

**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 węzłów (im więcej, tym wyższą rozdzielczość będzie miała mapa):



**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.

**WTM** (**Winner Takes Most**)

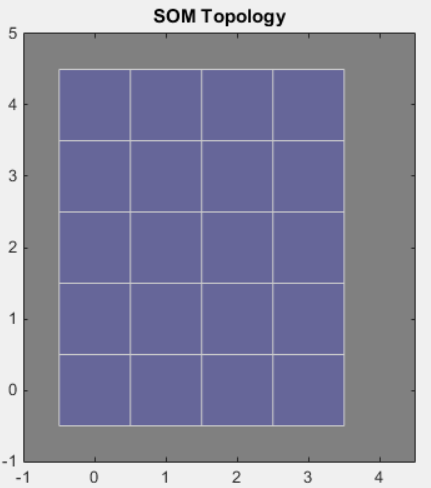
W metodzie tej zwycięski neuron ma prawo adaptacji swoich wag jak w WTA. Oprócz wag zwycięskiego neuronu zamianie podlegają również wagi sąsiadów według reguły:



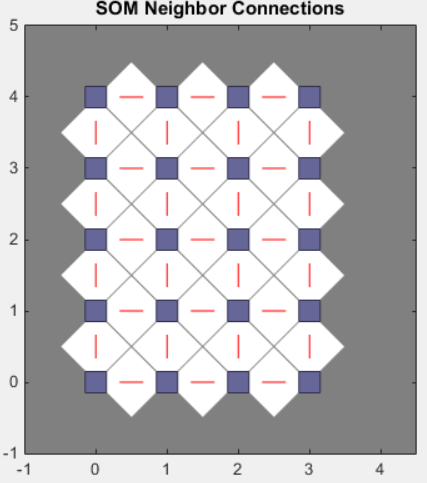
Gdzie G(x,i) jest funkcją sąsiedztwa.

**Wyniki:**

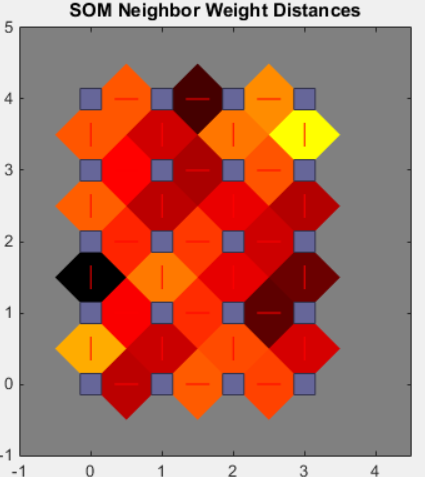
**Rys 1 - Topologia sieci**

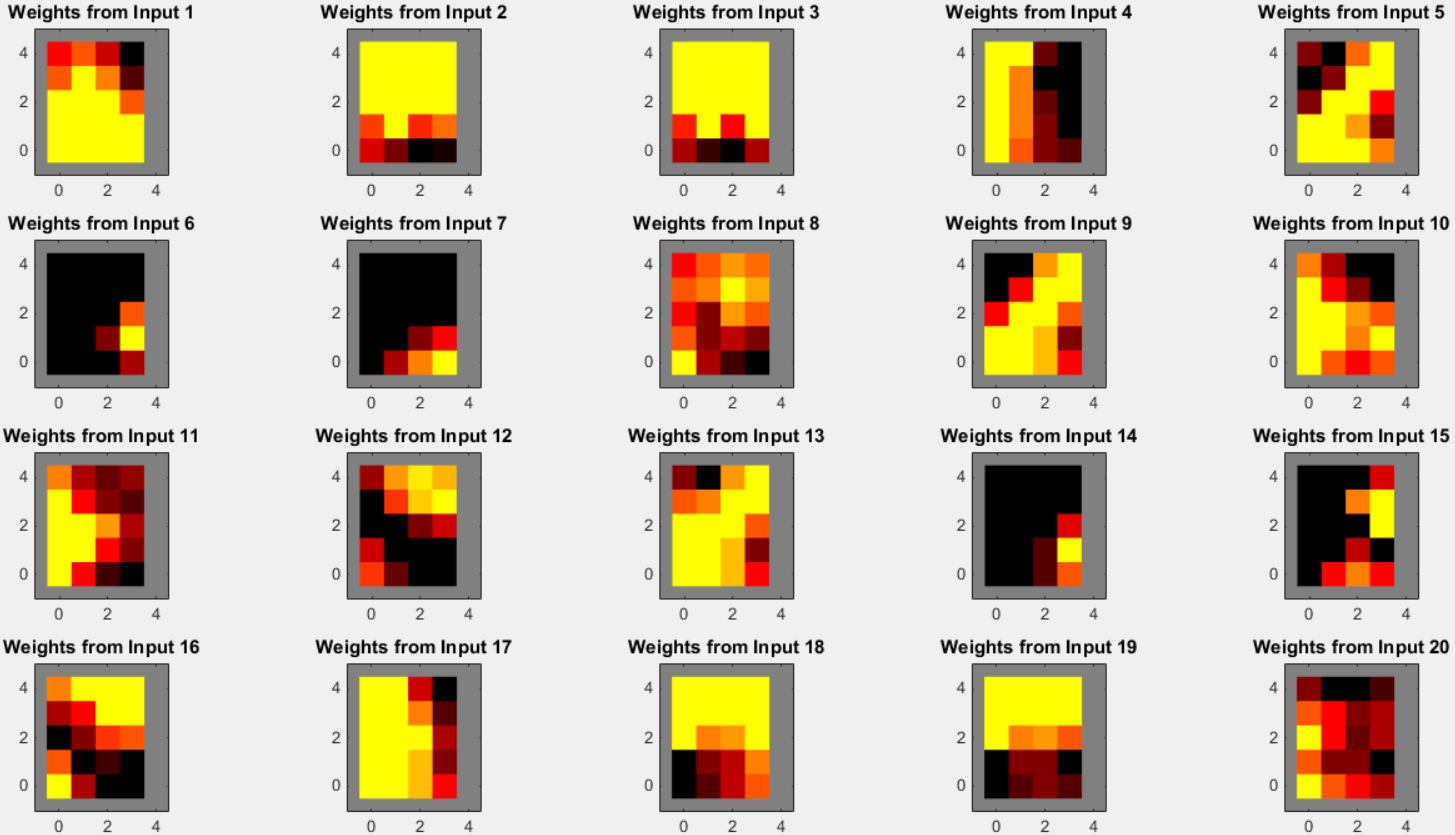
****

**Rys 2 - Połączenia pomiędzy sąsiadującymi neuronami**

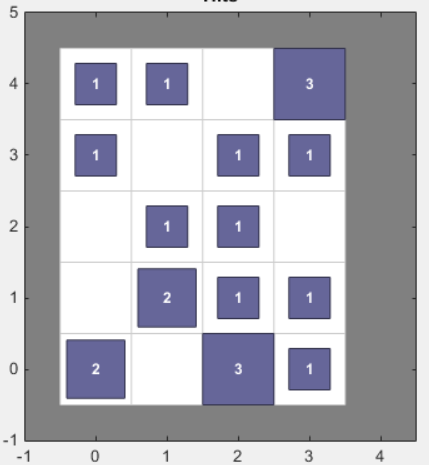
****

**Rys 3 - Dystans pomiędzy neuronami przedstawiony za pomocą kolorów**

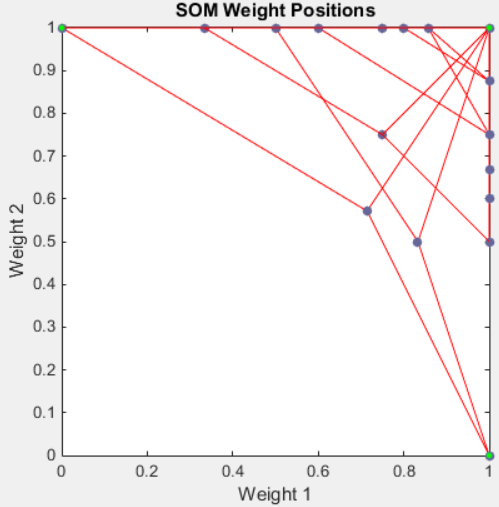
****

**Rys 4 -Rozkład wag dla każdego z wejść **

**Rys 5 -Wynik działania zasady WTM**

****

**Rys 6 - Efekt końcowy**



**WNIOSKI:**

Poprawność działania programu jest spowodowana tym, że zwycieżcy nie znajdują się w jednym miejscu tylko są „rozproszeni”. Skutkiem tego jest fakt, że sieć nie skupiała się na jednym fragmencie, ale na całej sieci.

Ilość neuronów na istotne znaczenie dla efektywności dzialania sieci. Mniejsza ilość

neuronów skutkowała, tym że wagi neuronów mogą być do siebie bardzo zbliżone.

Tym razem w porównaniu do metody WTA widać, że rozkład danych na grupy jest dużo bardziej uśredniony, grupy są znacznie mniejsze i więcej neuronów było w stanie zacząć je grupować. Dzięki temu nie ma żadnego neuronu który zagarnia wszystkie sygnały dla siebie.

Wielkość mapy nie wpływa na końcowy wynik programu.

WTM nie dopuszcza do zagłodzenia neuronu tak jak to mogło występować używając algorytmu WTA.

Kod programu:

close all; clear all; clc;

%Wczytywanie danych z pliku

filename = 'alfabet.txt';

delimiterIn = ' ';

WE = importdata(filename,delimiterIn);

dimensions = [4 5];

coverSteps = 1000;

initNeighbor = 1;

topologyFcn = 'gridtop';

distanceFcn = 'dist';

% SOM

net = selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn);

% Trenowanie

net = train(net,WE);

y = net(WE);

classes = vec2ind(y);

alfabet.txt

1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1

1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1 1 0 1

1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1 1 1 1

1 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 1 0 0 1

1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 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 1 0 0 0 0 0 0 0 1 0

1 1 1 1 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1

1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0

1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1 1

1 1 0 0 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 1

1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0

1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 1

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0

0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0

1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 0 0

1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 0 1

0 1 1 1 1 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1

0 1 1 1 1 0 1 0 0 1 0 1 1 0 0 1 1 0 0 1

1 0 0 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1