Marcin Damek
Inżynieria obliczeniowa, gr 1
Nr. albumu 285952

Sprawozdanie numer 6

Budowa i działanie sieci Kohonena dla WTM

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

Zadania do wykonania:

a) Wygenerowanie danych uczących i testujących, zawierających 20 dużych liter dowolnie wybranego alfabetu w postaci dwuwymiarowej tablicy np. 4x5 pikseli dla jednej litery. przykładowe znaki: http://www.ai.c-labtech.net/sn/litery.html
b) Przygotowanie (implementacja lub wykorzystanie gotowych narzędzi) sieci Kohonena i algorytmu uczenia opartego o regułę Winner Takes Most (WTM).
c) Uczenie sieci dla różnych współczynników uczenia.
d) Testowanie sieci.

Część teoretyczna

Sieci Kohonena są jednym z podstawowych typów sieci samoorganizujących się. Właśnie dzięki zdolności samoorganizacji otwierają się zupełnie nowe możliwości - adaptacja do wcześniej nieznanych danych wejściowych, o których bardzo niewiele wiadomo. Wydaje się to naturalnym sposobem uczenia, który jest używany chociażby w naszych mózgach, którym nikt nie definiuje żadnych wzorców, tylko muszą się one krystalizować w trakcie procesu uczenia, połączonego z normalnym funkcjonowaniem. Sieci Kohonena stanowią synonim całej grupy sieci, w których uczenie odbywa się metodą samoorganizującą typu konkurencyjnego. Polega ona na podawaniu na wejścia sieci sygnałów, a następnie wybraniu w drodze konkurencji zwycięskiego neuronu, który najlepiej odpowiada wektorowi wejściowemu. Dokładny schemat konkurencji i późniejszej modyfikacji wag synaptycznych może mieć różną postać. Wyróżnia się wiele podtypów sieci opartych na konkurencji, które różnią się dokładnym algorytmem samoorganizacji.

Struktura sieci neuronowej

Bardzo istotną kwestią jest struktura sieci neuronowej. Pojedynczy neuron jest mechanizmem bardzo prostym i przez to niewiele potrafiącym. Dopiero połączenie wielu neuronów ze sobą umożliwia prowadzenie dowolnie skomplikowanych operacji. Ze względu na raczej niewielką wiedzę o faktycznych zasadach funkcjonowania ludzkiego mózgu, powstało wiele różnych architektur, które starają się naśladować budowę i zachowanie poszczególnych fragmentów układu nerwowego. Najczęściej stosuje się w tego typu sieciach architekturę jednokierunkową jednowarstwową. Jest to podyktowane faktem, że wszystkie neurony muszą uczestniczyć w konkurencji na równych prawach. Dlatego każdy z nich musi mieć tyle wejść ile jest wejść całego systemu.

Algorytm uczenia

Algorytmem, od którego nazwę wzięła cała klasa sieci są samoorganizujące się mapy Kohonena. Zostały one opisane przez ich twórcę w publikacji "The Self Organising Map". Kohonen zaproponował dwa rodzaje sąsiedztwa: prostokątne i gaussowskie.
Pierwsze ma postać:

$$G(i,x) = \begin{cases} 1 & dla \quad d(i,w) \le \lambda \\ 0 & dla \quad d(i,w) > \lambda \end{cases}$$

A drugie:

$$G(i,x) = \exp\left(-\frac{d^2(i,w)}{2\lambda^2}\right)$$

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)

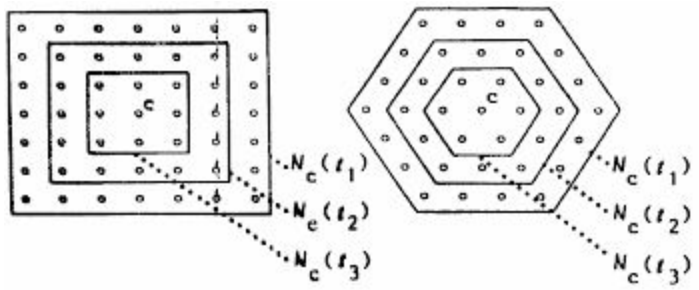
- Każdy węzeł może być stowarzyszony z pewnymi innymi, sąsiednimi węzłami - wówczas te węzły również zostają zmodyfikowane, jednak w mniejszym stopniu.

WTM - opis

Winner Takes Most Zwycięzca bierze najwięcej. W tej strategii nie tylko neuron najbardziej podobny, ale także jego otoczenie zostają zmodyfikowane. Najczęściej ta modyfikacja jest zależna od odległości sąsiada od zwycięzcy.

Promień sisiedztwa jest duży, co powoduje, że oprócz neuronu-zwycięzcy również jego sąsiedzi (z mapy) zmieniają swoje wektory kodowe. Współczynnik uczenia η jest w tej fazie stosunkowo duży. Zmiany wag następuje według wzoru :

$$\mathbf{w}_m(k+1) = \mathbf{w}_m(k) + \eta(k)\, h_{cm}(k)\, [\mathbf{x}(k) - \mathbf{w}_m(k)], \quad m = 1, \ldots, M.$$



Część praktyczna

Do wykonania zadania użyłem narzędzia Neural Network Training Tool z pakietu Matlab. Dzięki niemu możemy w stosunkowo łatwy sposób tworzyć i uczyć sieci neuronowe.
Do tego celu utworzony został zestaw liter (20 dużych), które są reprezentowane w postaci dwuwymiarowej tablicy 5x5 pikseli dla jednej litery.
Duże litery których użyłem w ćwiczeniu: ABCDEFGHIJKLMNOPRSTU
Do nauki użyłem 20 tablic po 25 znaków każda, tzn w każdym znaku były dwie możliwości tj. 1 lub 0 w następujacy sposób:

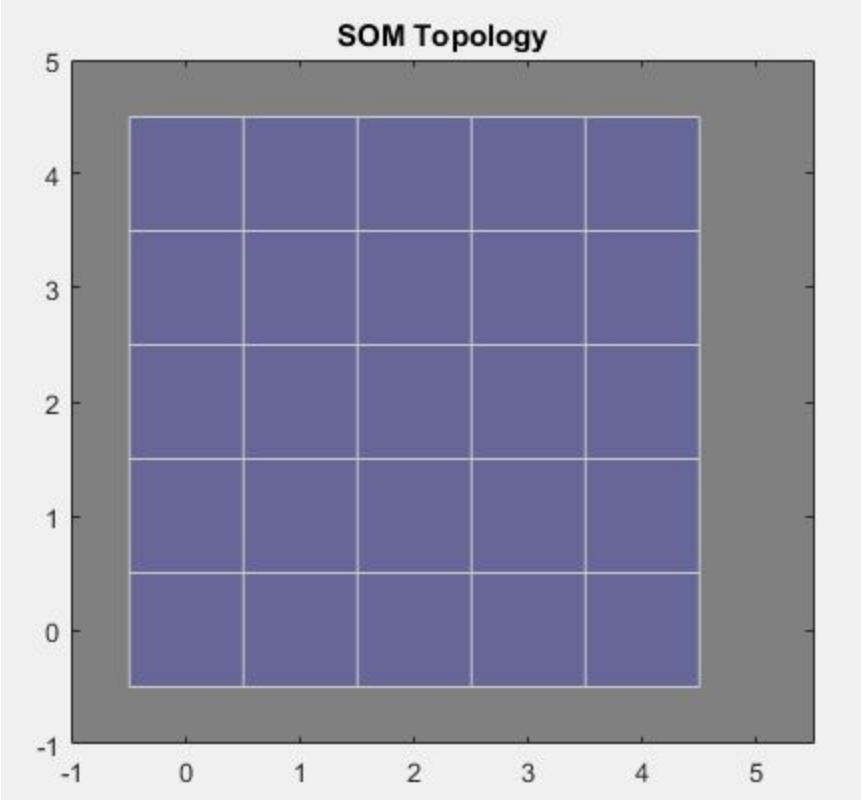

Litera “A” której kod wynosi: 01110 10001 10001 11111 10001

W zadaniu najważniejsza okazała się funkcja:

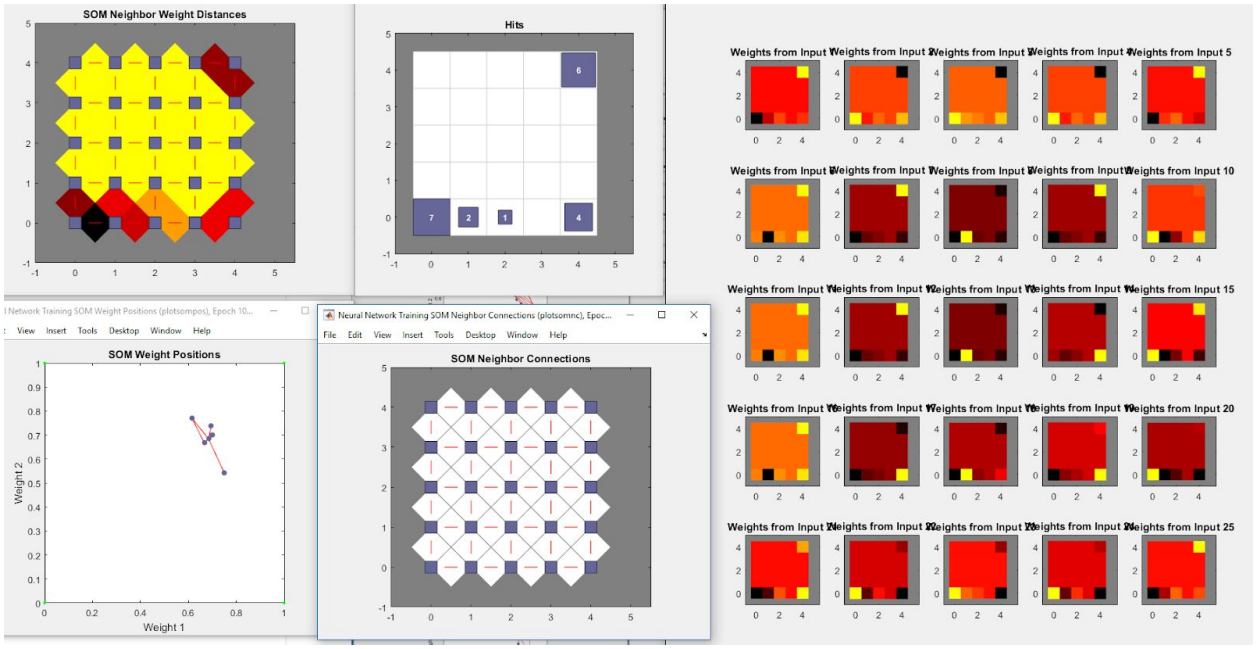
selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn)

Dimensions - rozmiar wektora
coverSteps - liczba kroków szkoleniowych
initNeighbor - początkowy rozmiar sąsiedztwa
topologyFcn - funkcja topologii warstw
distanceFcn - odległość neuronów

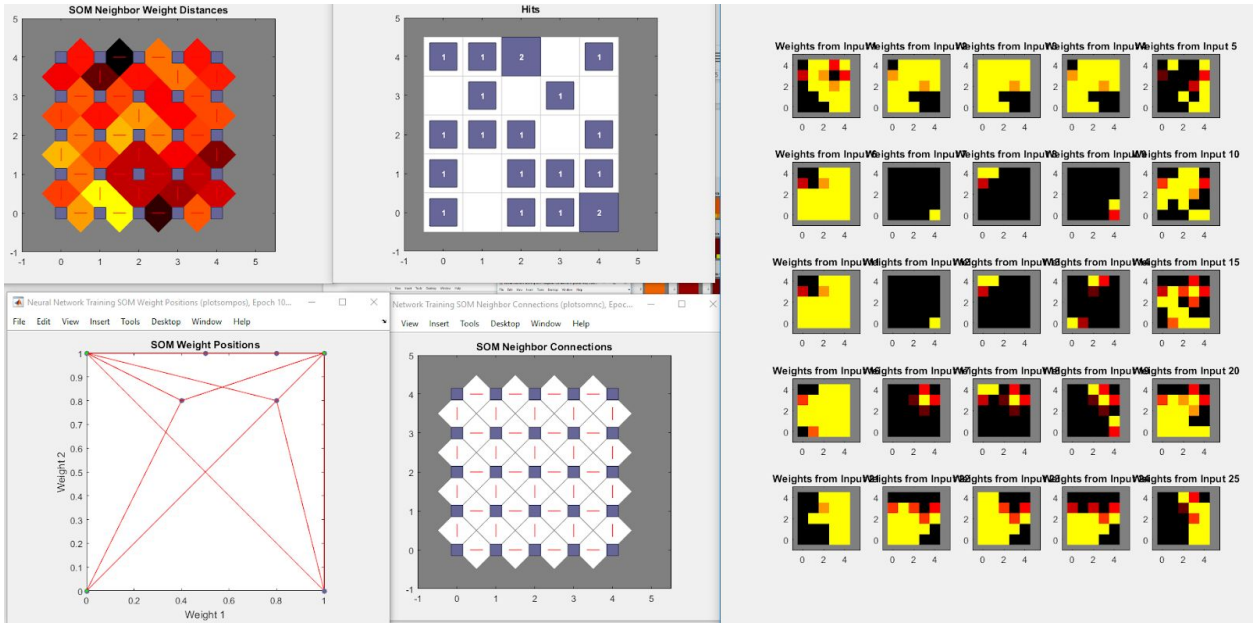
Wszystkie próby wykonywałem dla topologii warstwy prostokątnej, określonej w parametrze topologyFcn równej gridtop. Poniżej wygląd zastosowanej topologii.



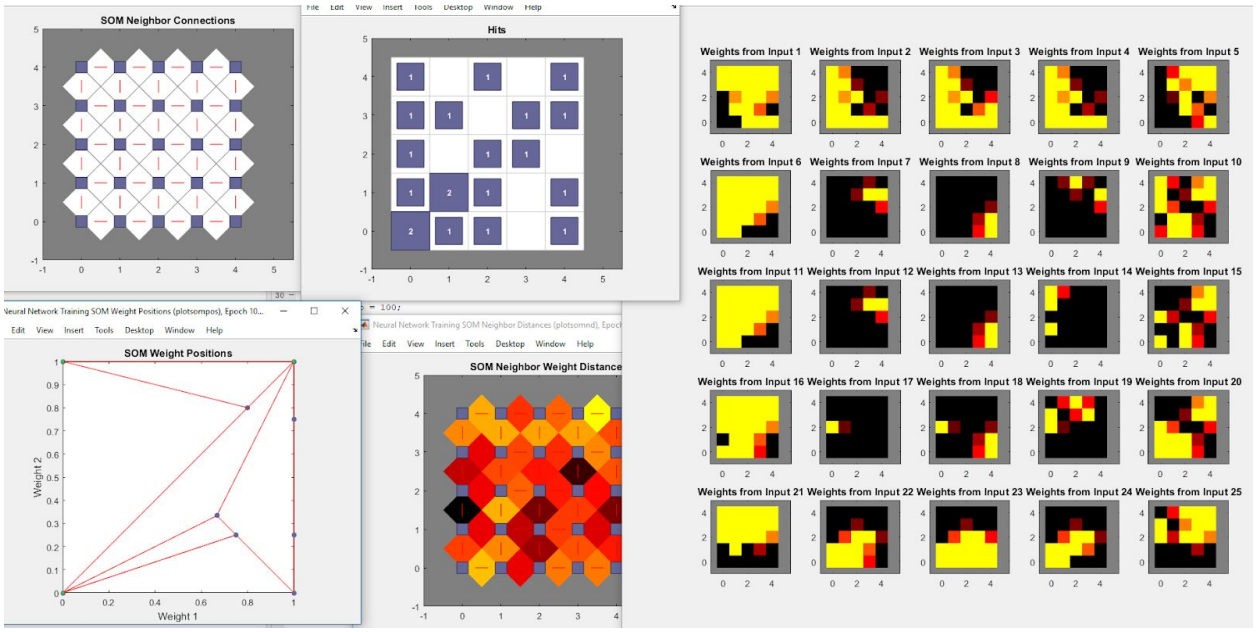
Analizy dokonałem dla: współczynnika uczenia równego 0.5 przy liczbie sąsiadujących neuronów równych 0,3,5 oraz dla liczby sąsiadujących neuronów równej 0 przy współczynnikach nauki równcyh 0.1, 0.5, 0.9.
Wymiar wektora ustaliłem na 5x5, oraz etapy szkolenia w celu pokrycia przestrzeni wejściowej ustawiłem na 100. Maksymalna liczna epok treningowych to 1000.



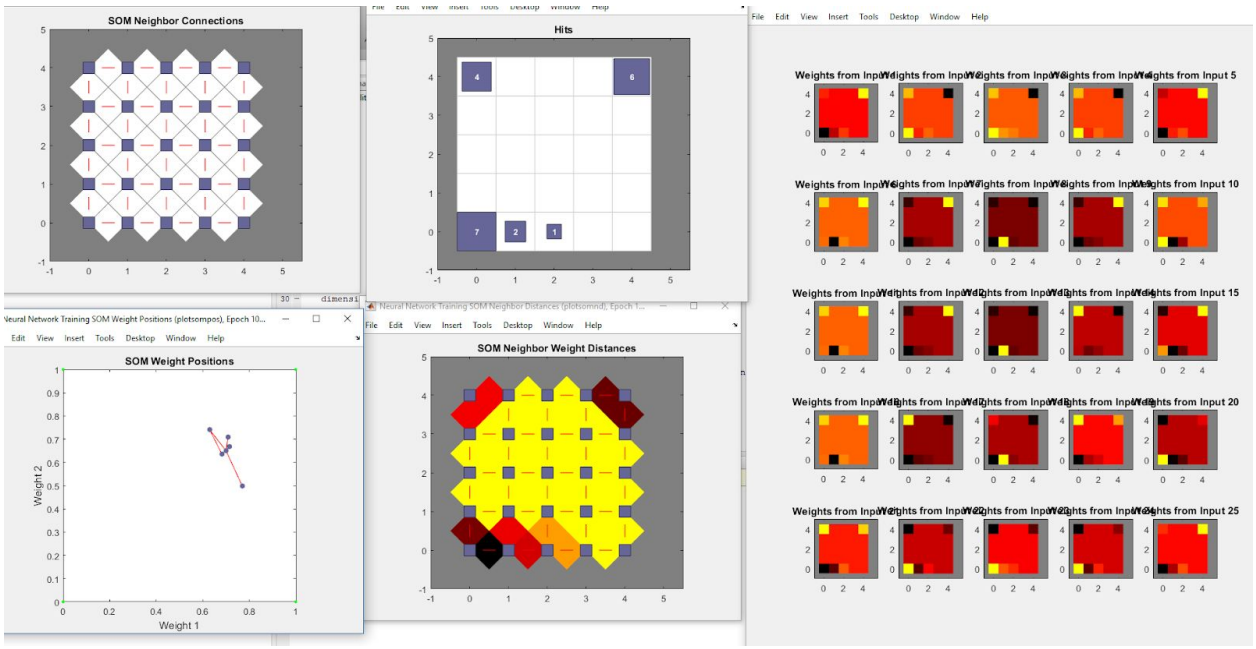
Wsp. nauki = 0.5 Sąsiedzi: 0



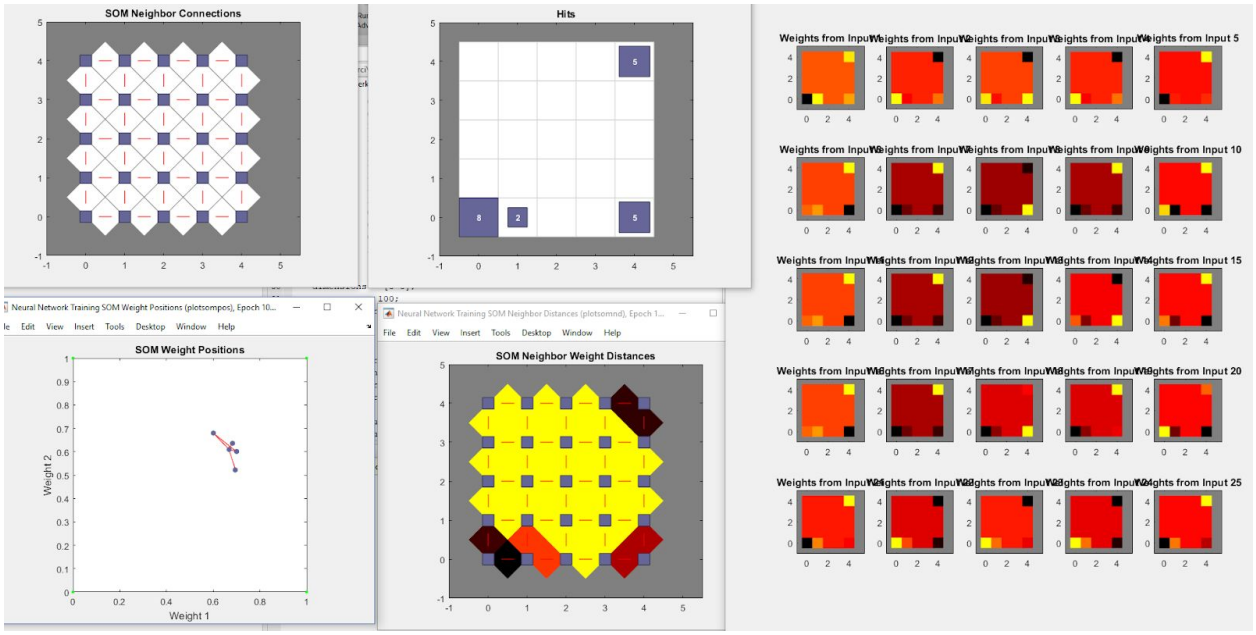
Wsp. nauki = 0.5 Sąsiedzi: 3



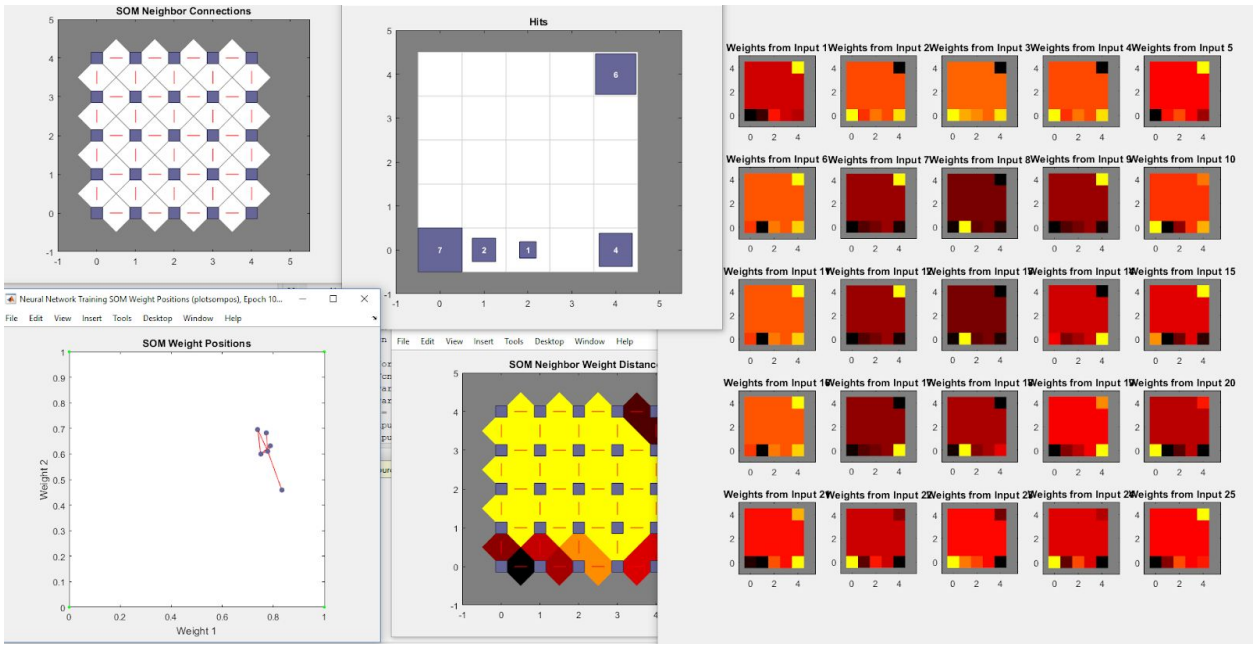
Wsp. nauki = 0.5 Sąsiedzi: 5



Wsp. nauki = 0.1 Sąsiedzi: 0



Wsp. nauki = 0.5 Sąsiedzi: 0



Wsp. nauki = 0.9 Sąsiedzi: 0

Wnioski

Możemy zauważyć, że rozkład sił skupia się wzdłuż brzegów siatki topologii i zależy od współczynnika uczenia, im wyższy tym bardziej możemy zaobserwować to zjawisko. Ilość neuronów ma bardzo duże znaczenie dla efektywności działania sieci. Rozkłady sił stają się równomierne, im sąsiedztwo jest wyższe. W metodologii WTM obiekty są bardziej ze sobą związane, mimo zmian w sąsiedztwie. Jedyne co się zmienia to ilość wag i kształt tych powiązań. Algorytm WTM równomiernie rozłożył zwycięzców, inaczej niż algorytm WTA. Poprawność działania programu jest spowodowana tym, że zwycięzcy 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. Zwiększenie ilości epok w których sieć musiała się nauczyć nie dawało lepszych wyników. Zapewne jest to spowodowane, tym że sieć uczy się bez nauczyciela, czyli sieć dłużej się uczy niż z nauczycielem. Aby zaobserwować znaczące wyniki trzeba by dać sieci bardzo dużo czasu na naukę. Uczenie odbywało się bardzo sprawnie dzięki ograniczeniu ilości epok. Każde uczenie trwało około 2 sekundy.

Kod programu

```
close all; clear all; clc;

input = [ 0 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1;
        1 1 1 1 1 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0;
        1 1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0;
        1 1 1 1 1 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0;
        0 0 0 0 1 1 0 1 0 0 1 0 1 1 0 0 0 0 1 1;
        1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1;
        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 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0;
        0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0;
        1 1 1 1 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 1;
        1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1;
        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 0 0 0 1 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 1 1 1 0 0;
        1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 0 0 0 0 1;
        1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 1;
        1 0 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 1 0 0 0 0 0 0 0 0 0 1 0;
        1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0;
        1 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 0 1 0 1;
        1 1 0 1 1 1 0 1 0 0 1 1 1 1 0 1 1 0 0 0;
        0 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 0 1 0 1;
        0 1 1 1 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 1;
        0 1 1 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1;
        1 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 0];

dimensions = [5 5];
coverstep = 100;
initNeighbor = 0;
topologyFcn = 'gridtop';
distanceFcn = 'dist';
net = selforgmap(dimensions, coverstep, initNeighbor, topologyFcn, distanceFcn);
net.trainFcn = 'trainbu';
net.trainParam.epochs = 1000;
net.trainParam.lr = 0.9;
[net, tr] = train(net, input);
y = net(input);
indexOfOutput = vec2ind(y);
```