18.12.2017r.

Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Podstawy sztucznej inteligencji

Sprawozdanie numer 5

Budowa i działanie sieci Kohonena dla WTM

Inżynieria Obliczeniowa
Urszula Ślusarz

nr. indeksu: 286132

1. Cel ćwiczenia

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

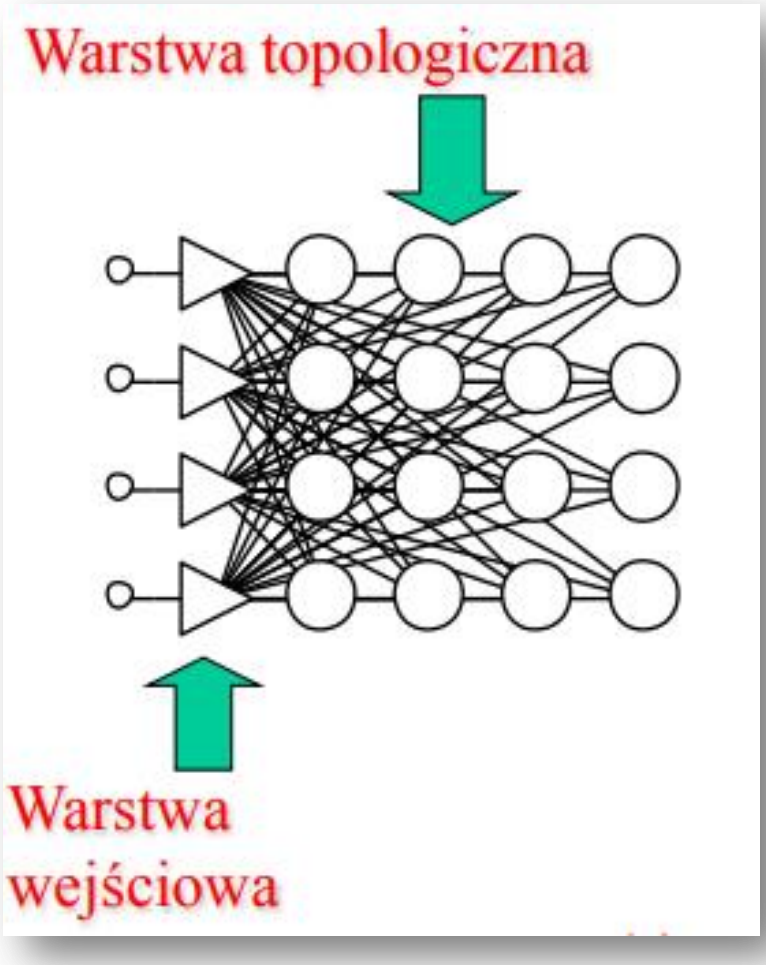
2. Samoorganizująca się sieć Kohenena

Sieć Kohonena uczy się całkiem sama, wyłącznie obserwując przesyłane do niej dane, których wewnętrzna struktura i ukryta w tej strukturze nieznana logika będą decydować o końcowym obrazie klasyfikacji i o ostatecznym działaniu sieci. Można założyć, że w sieci Kohonena poszczególne neurony będą identyfikowały i rozpoznawały poszczególne skupienia danych. Przez skupienie danych rozumieć tu będziemy grupę danych: dane należące do skupienia są między sobą w jakimś stopniu podobne, natomiast dane należące do różnych skupień różnią się pomiędzy sobą.

Sieci Kohonena realizują pewne odwzorowania, stąd ich powszechnie używany symbol: SOM (Self Organizing Maps)

Sieci uczące się bez nauczyciela w trakcie uczenia opierają się wyłącznie na obserwacji danych wejściowych, nikt im natomiast nie mówi, co z tych danych wejściowych powinno wynikać to sieci muszą wykryć i ustalić same.

Struktura sieci Kohonena



Cechy charakterystyczne:

- sieć uczy się bez nauczyciela,
- uporządkowane neurony wyjściowe ,
- jest konkurencja i wyłaniany jest neuron „zwycięski” ,
- ważną rolę odgrywa „sąsiedztwo” ,
- w wyniku uczenia powstaje mapa topologiczna ,
- aprioryczna interpretacja wartości wyjściowych jest niemożliwa,
- po uczeniu można ustalić, jakie znaczenie mają poszczególne rejony mapy topologicznej - ale wyłącznie na podstawie analizy konkretnych Warstwa przykładów danych wejściowych.

Sieci samouczące się:

Samouczenie tym się głównie różni od uczenia, że podczas samouczenia nie ma żadnego zewnętrznego źródła wiedzy (bazy danych uczących, nauczyciela itp.), dostarczającego gotowe widomości, które wystarczy tylko sobie przyswoić. Umysł samouczącego się człowieka, a także samoucząca się sieć neuronowa, musi najpierw sama odkryć wiedzę, którą następnie zapamięta.

WTM - Winner Takes Most

WTM - 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.

W metodzie tej oprócz wag zwycięskiego neuronu zmianie podlegają również wagi jego sąsiadów według reguły:

$$w_i(k+1) = w_i(k) + \eta_i(k)G(i,x)[x - w_i(k)]$$

Gdzie G(x,i) jest funkcją sąsiedztwa. W klasycznym algorytmie Kohenena funkcja G(x,i) zdefiniowana jest jako:

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

Gdzie d(i,w) jest odległością pomiędzy neuronami w przestrzeni sieci tzn. na siatce (a nie w przestrzeni danych wejściowych!), a lambda jest promieniem sąsiedztwa malejącym do zera w trakcie nauki. Jest to tzw. sąsiedztwo prostokątne. Innym rodzajem sąsiedztwa jest sąsiedztwo gaussowskie:

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

W tym przypadku stopień aktywacji neuronów jest zróżnicowany w zależności od wartości funkcji Gaussa. Sąsiedzi neuronu zwycięskiego są modyfikowani w różnym stopniu.

Często stosowanym podejściem jest wyróżnienie dwóch faz uczenia sieci Kohenena:

- wstępna, podczas której modyfikacji ulegają wagi neuronu zwycięzcy oraz jego sąsiedztwa malejącego z czasem
- końcowa, podczas której modyfikowane są wagi jedynie neuronu zwycięzcy

Mają one na celu najpierw wstępne nauczenie sieci, a następnie jej dostrojenie.

3. Zrealizowane kroki scenariusza

✓ W pierwszej kolejności wygenerowałam dane uczące i testujące, które zawierają 20 dużych liter w postaci dwuwymiarowej tablicy np. 5x7 pikseli dla jednej litery.

Litery wybrane przeze mnie: **A C D E F G H K M N O P R S T U W X Y Z ,** litery są utworzone na matrycy **5 x 7** czyli na **35 polach**.
Dane uczące, poniższa tabele przedstawiają matryce 20 liter po konwersji zero-jedynkowej:

```
letterA=   [0 0 1 0 0 ...
            0 1 0 1 0 ...
            0 1 0 1 0 ...
            1 0 0 0 1 ...
            1 1 1 1 1 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ];

letterC=   [0 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 0 0 0 1 ...
            0 1 1 1 0];

letterD  = [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 0 0 0 1 ...
            1 1 1 1 0 ];

letterE  = [1 1 1 1 1 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 1 1 1 0 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 1 1 1 1 ];

letterF =  [1 1 1 1 1 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 1 1 1 0 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ];

letterG =  [0 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 0 ...
            1 0 1 1 1 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            0 1 1 1 0];

letterH =  [1 0 0 0 1 ...
            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 ...
            1 0 0 0 1];

letterK =  [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 ...
            1 0 0 0 1];

letterM=   [1 0 0 0 1 ...
            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 0 1];

letterN=   [1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 1 0 0 1 ...
            1 0 1 0 1 ...
            1 0 0 1 1 ...
            1 0 0 0 1 ...
            1 0 0 0 1];

letterO=   [0 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 0 0 0 1 ...
            0 1 1 1 0]';

letterP=   [1 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 1 1 1 0 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 0 0 0 0]';

letterR=   [1 1 1 1 0 ...
            1 0 0 0 1  ...
            1 0 0 0 1 ...
            1 1 1 1 0 ...
            1 0 1 0 0 ...
            1 0 0 1 0 ...
            1 0 0 0 1];

letterS=   [0 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 0 ...
            0 1 1 1 0 ...
            0 0 0 0 1 ...
            1 0 0 0 1 ...
            0 1 1 1 0];

letterT=   [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 ...
            0 0 1 0 0 ...
            0 0 1 0 0];
```

```
letterU=  [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 0 0 0 1 ...
           0 1 1 1 0]';

letterW=  [1 0 0 0 1 ...
           1 0 0 0 1 ...
           0 1 0 1 0 ...
           0 0 1 0 0 ...
           0 1 0 1 0 ...
           1 0 0 0 1 ...
           1 0 0 0 1]';

letterZ=  [1 1 1 1 1 ...
           0 0 0 0 1 ...
           0 0 0 1 0 ...
           0 0 1 0 0 ...
           0 1 0 0 0 ...
           1 0 0 0 0 ...
           1 1 1 1 1]';

letterX=  [1 0 0 0 1 ...
           1 0 0 0 1 ...
           1 0 0 0 1 ...
           1 0 0 0 1 ...
           1 0 1 0 1 ...
           1 0 1 0 1 ...
           0 1 0 1 0]';

letterY=  [1 0 0 0 1 ...
           1 0 0 0 1 ...
           0 1 0 1 0 ...
           0 0 1 0 0 ...
           0 0 1 0 0 ...
           0 0 1 0 0 ...
           0 0 1 0 0]';
```

Liniowym rozwinięciem tych matryc będą wektory wejściowe ciągu uczącego:

```
189 -    in=[0 0 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 1
190          1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1
191          1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1
192          1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1
193          0 0 0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1
194          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0
195          0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
196          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
197          0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
198          1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1
199          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0
200          0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0
201          0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0
202          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
203          1 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 0 0 0
204          1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 0
205          1 0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0
206          1 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 0 0 1 1 1
207          1 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0
208          1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 1 1 0 0 0 0
209          1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 0
210          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
211          0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0
212          0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
213          1 0 1 0 0 1 1 0 1 1 1 0 0 1 0 1 1 0 0 0
214          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1
215          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
216          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
217          0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
218          1 1 1 0 0 1 1 0 1 1 1 0 0 1 0 1 1 1 0 0
219          1 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 0 1
220          0 1 1 1 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1
221          0 1 1 1 0 1 0 0 0 0 1 0 0 1 1 1 0 0 1 1
222          0 1 1 1 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1
223          1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 0 0 1 0 1];
224        %A C D E F G H K M N O P R S T U W X Y Z
```

✓ Przygotowanie (implementacja lub wykorzystanie gotowych narzędzi) sieci Kohonena i algorytmu uczenia opartego o regułę Winner Takes Most (WTM).

Aby używam sieci samoorganizującej się stosujemy funkcji selforgmap, której postać jest następująca:

Składnia:

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

dimensions- wektor rzędów wymiarów (domyślnie = [8 8])

coverSteps- liczba kroków szkoleniowych dla początkowego pokrycia przestrzeni wejściowej (domyślnie = 100)

initNeighbor – początkowy rozmiar sąsiedztwa (domyślnie = 3)

Sąsiedztwo może być sztywne (0 - nie należy, 1 - należy) lub też wyznaczane funkcją przynależności przyjmującej wartości rzeczywiste z przedziału [0,1].

topologyFCN – funkcja topologii warstw (domyślnie= 'hextop')

distanceFCN – funkcja odległości neuronowej (domyślnie = 'linkdist')

Zaimplementowana przeze mnie w MatLabie funkcja wygląda następująco:

Topologie (gridtop, hextop, randtop):

gridtop-topologia ta przedstawiona jest w prostokątnej siatce.

dist- funkcja wagi odległości euklidesowej.

Liczbę maksymalną epok ustawiłam na 800.

```
27 -    net = selforgmap([5 7],100,1,'gridtop','dist');
28 -    net.trainParam.epochs = 800;
```

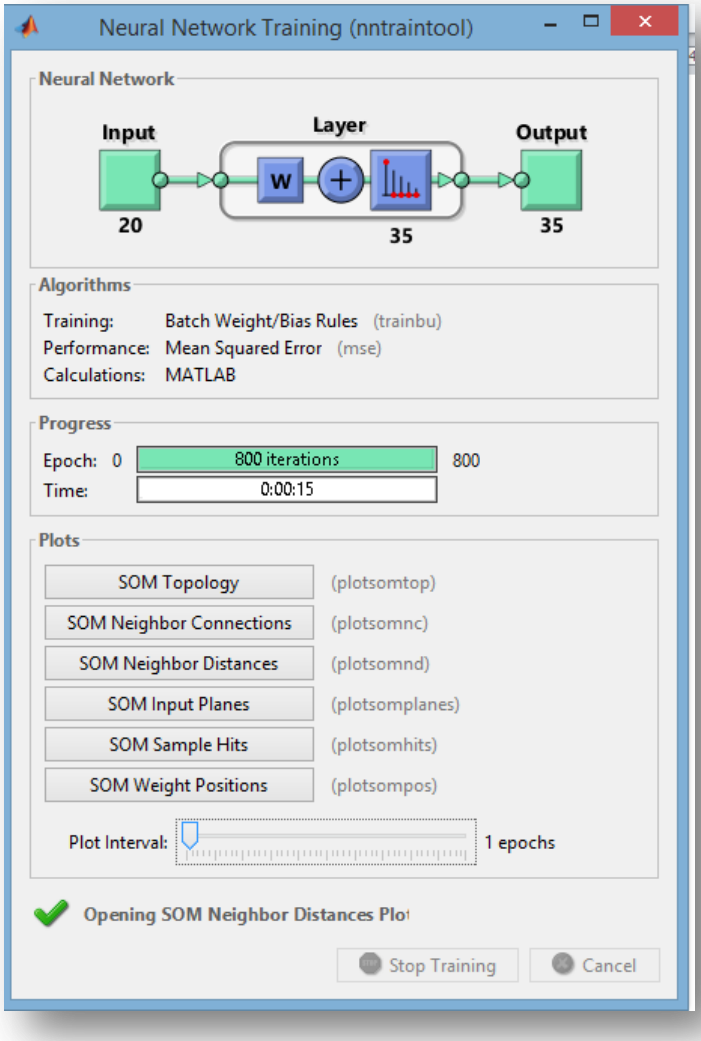
Następnie dokonuję nauki sieci.

```
30 -    net = train(net,in);
31 -    siec = net(in);
32 -    classes = vec2ind(siec);
```

W wierszu 32, wykorzystuję metodę „**vec2ind**”, która konwertuje wektory na indeksy.

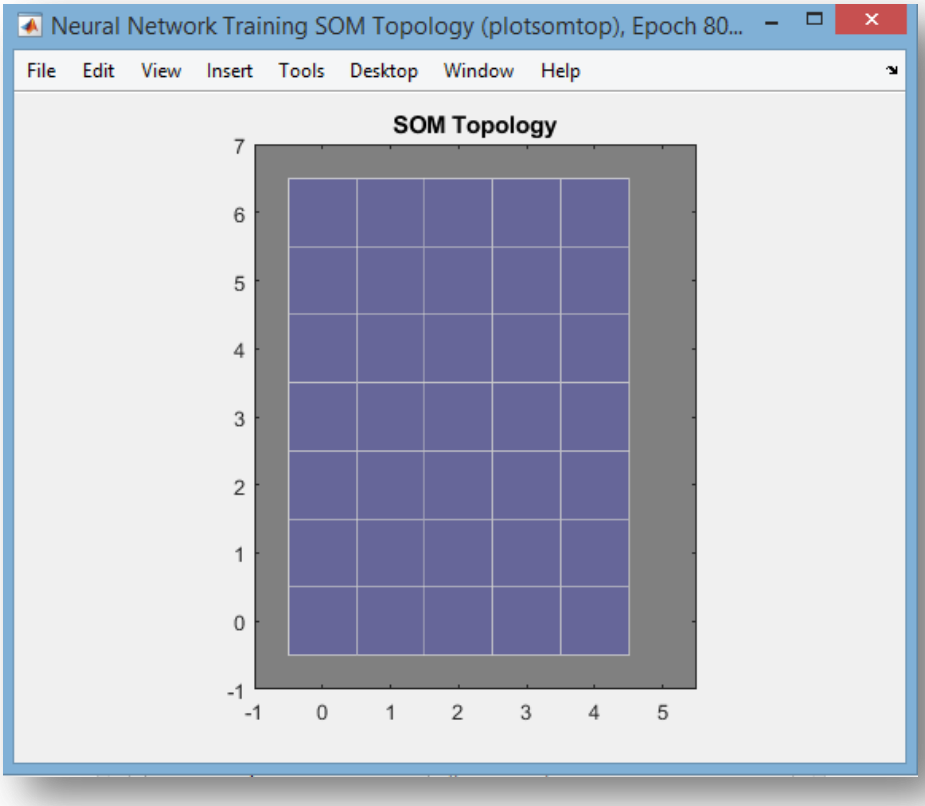
✓Wyniki

Trenowanie obejmuje maksymalną liczbę epok, która wynosi 800, czas wykonania zadania to 0:00:15.



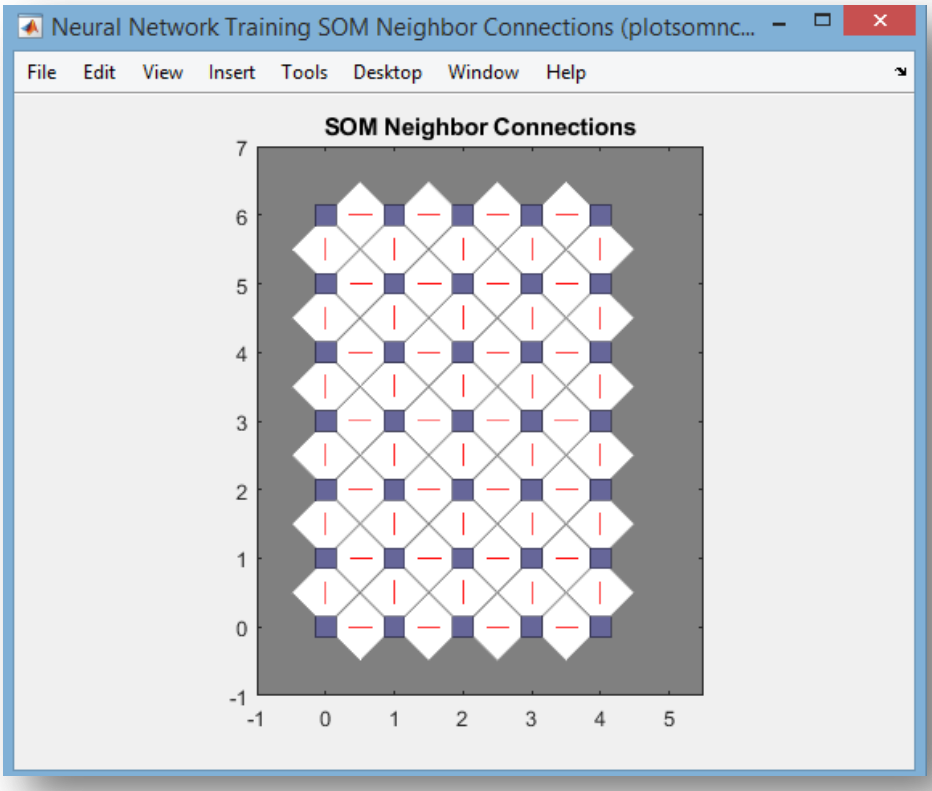
SOM Topology:

Na tej figurze każdy z sześciokątów reprezentuje neuron. Siatka to 5 na 7, więc w tej sieci jest łącznie 35 neuronów.



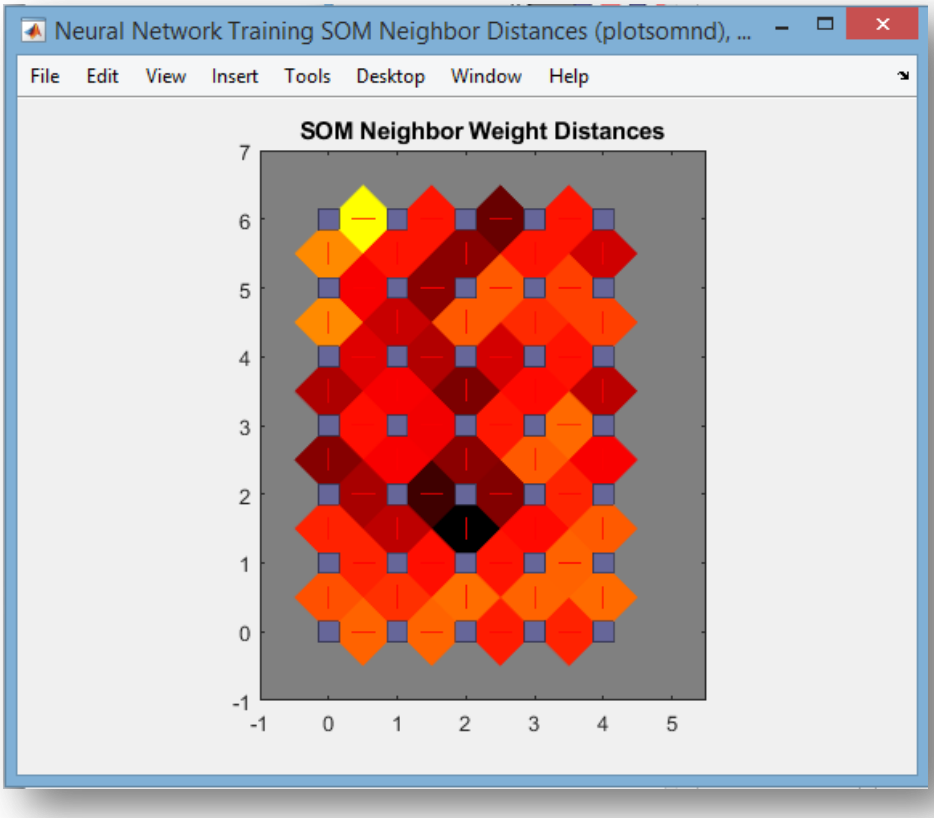
SOM Neighbor Connections:

Niżej umieszczona mapa przedstawia powiązanie sąsiedzkie.

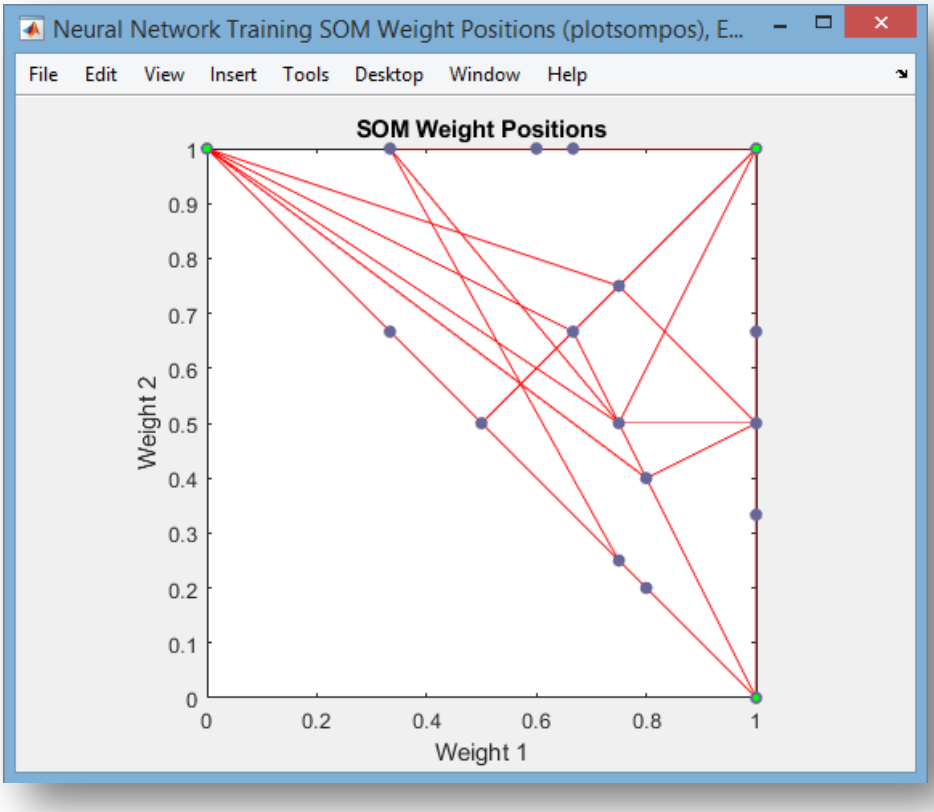


SOM Neighbor Distance:

- na tej figurze niebieskie sześciokąty reprezentują neurony.
- czerwone linie łączą sąsiednie neurony
- kolory w obszarach zawierających czerwone linie wskazują odległości między neuronami
- ciemniejsze kolory reprezentują większe odległości
- jaśniejsze kolory oznaczają mniejsze odległości
- ciemny segment znajduje się wyłącznie w środkowo dolnym regionie

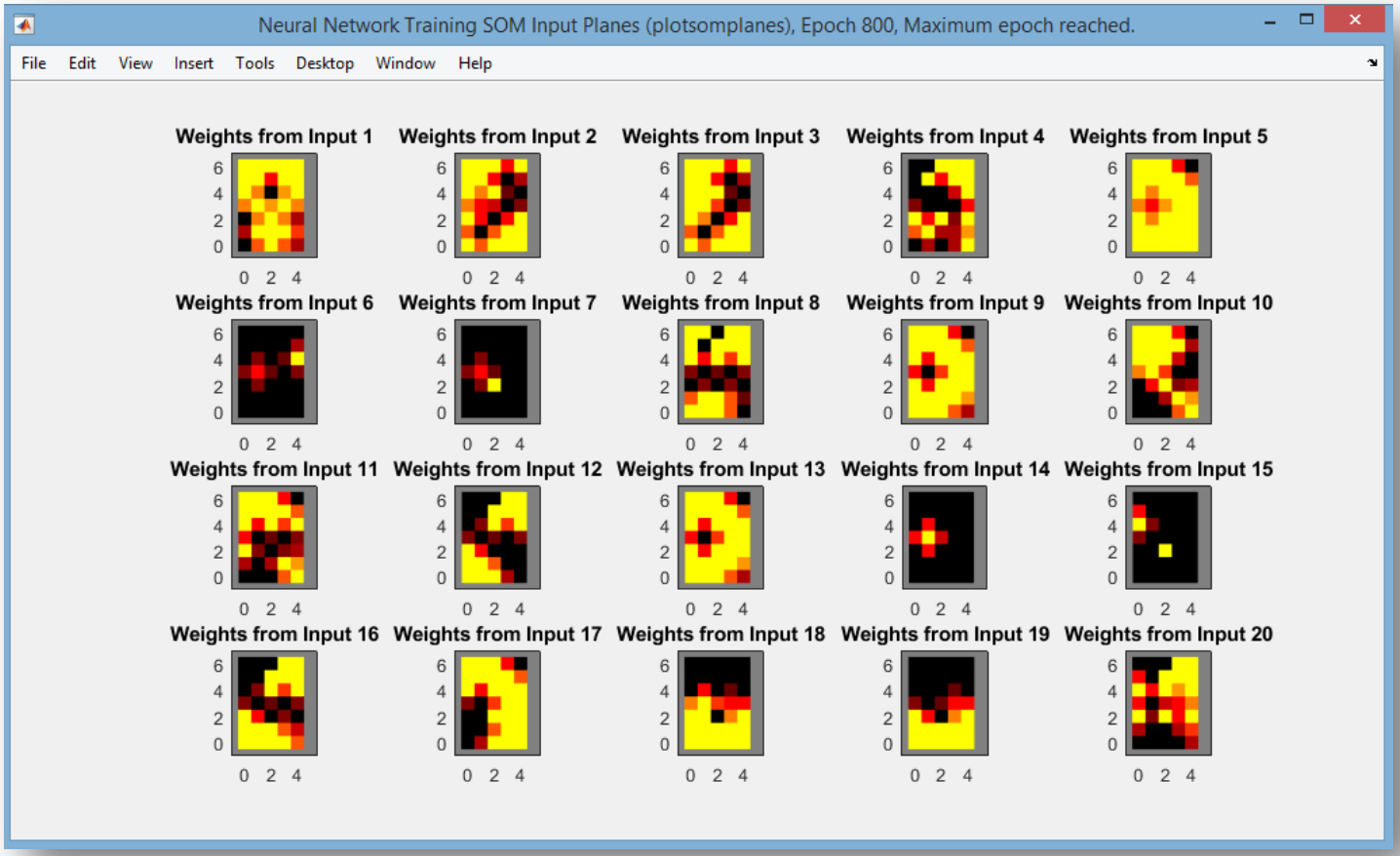


SOM Weight Positions:



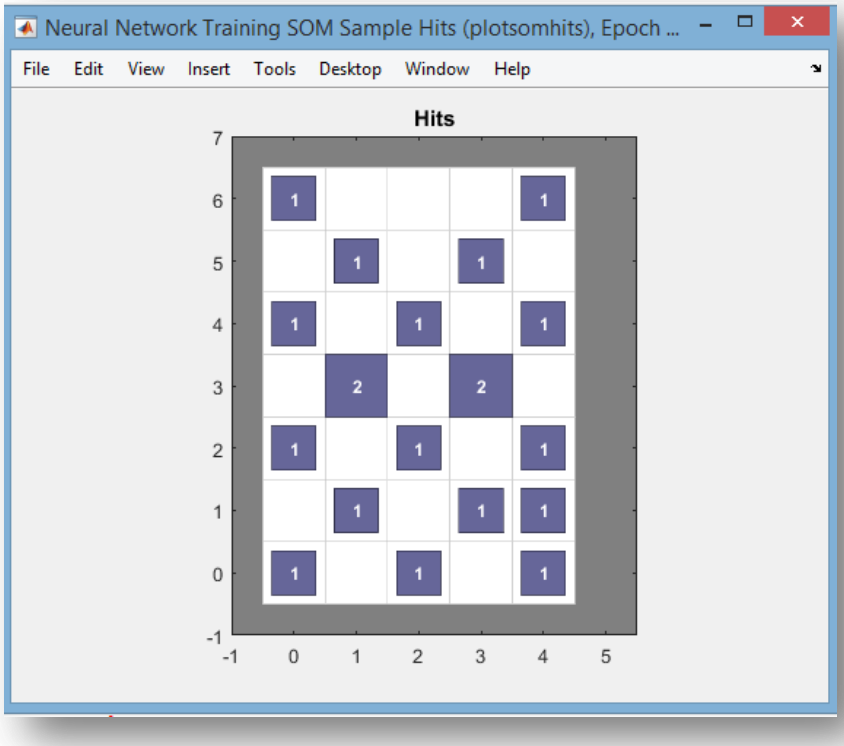
Dodatkowo samoorganizacja prowadzi do tego, że neurony „obstawiające” podobne obiekty wejściowe ulokowane są w warstwie topologicznej sieci jako neurony sąsiednie

SOM Input Planes:



Ta figura przedstawia płaszczyznę ciężkości dla każdego elementu wektora wejściowego (w tym przypadku 20). Są to wizualizacje wag, które łączą każde wejście z każdym z neuronów. Ciemniejsze kolory oznaczają większe wagi. Jeśli schematy połączeń dwóch wejść były bardzo podobne, można założyć, że dane wejściowe są wysoce skorelowane.

SOM Sample Hits:



Domyślna topologia SOM jest sześciokątna. Ta figura pokazuje lokalizacje neuronów w topologii i wskazuje, ile danych treningowych jest powiązanych z każdym z neuronów (centrami klastra). Topologia to 5 na 7, więc istnieje 35 neuronów.

✓ **Wnioski**:

Głównymi cechami odróżniającymi algorytm WTA(którego używałam w sprawozdaniu numer 5) od algorytmu WTM, jest to, że w WTA tylko jeden neuron może podlegać adaptacji w każdej iteracji, są algorytmami słabo zbieżnymi, szczególnie przy dużej liczbie neuronów.
Natomiast algorytm WTM, który zastąpił w praktyce wcześniej omawiany algorytm, charakteryzuje się tym, że oprócz zwycięzców, również neurony z jego sąsiedztwa uaktualniają swoje wagi.

W strukturze sieci Kohonena istotne jest to, że każdy neuron warstwy wejściowej komunikuje się z każdym neuronem warstwy topologicznej – natomiast neurony w warstwach nie komunikują się pomiędzy sobą. Przy uczeniu się mapy bardzo ważną rolę odgrywa pojęcie sąsiedztwa neuronów. Sąsiedztwo to wyznacza się według położenia wektorów referencyjnych na mapie.
W warunkach laboratoryjnych łatwo można doprowadzić do tego, że sieć Kohonena będzie dobrze różnicowała poszczególne sytuacje występujące w wielowymiarowej przestrzeni danych.