Fabiola Dąbroś , IO gr.1 rok III

Sprawozdanie z zagadnienia nr 5.

Temat ćwiczenia:
Budowa i działanie sieci Kohonena dla WTA.
Cel ćwiczenia:
Celem ćwiczenia jest poznanie budowy i działanie sieci Kohonena przy wykorzystaniu reguły WTA do odwzorowania istotnych cech kwiatów.

1. Opis budowy użytej sieci oraz wykorzystanego algorytmu uczenia.

Do wykonania tego ćwiczenia korzystałam z sieci Kohonena – sieci samoorganizujących się. Należą one go grupy sieci uczących się w sposób rywalizacyjny bez nauczyciela. Oznacza to, że nie ma żadnego zewnętrznego źródła wiedzy, dostarczającego gotowe widomości, które wystarczy tylko sobie przyswoić. Samoucząca się sieć neuronowa, musi najpierw sama odkryć wiedzę, którą następnie zapamięta. Sieć Kohonena wyróżnia się tym od innych sieci, że zachowuje odwzorowanie sąsiedztwa przestrzeni wejściowej. Wynikiem działania sieci jest klasyfikacja przestrzeni w sposób grupujący zarówno przypadki ze zbioru uczącego, jak i wszystkie inne wprowadzenia po procesie uczenia. Polega ona na podawaniu na wejścia sieci sygnałów, a następnie wybraniu w drodze konkurencji zwycięskiego neuronu, który najlepiej odpowiada wektorowi wejściowemu. W sieci tej neurony wyjściowe są uporządkowane. Istotne jest również właściwe zdefiniowanie sąsiedztwa neuronów.
SOMy dokonują redukcji wymiarów danych przez tworzenie map, zazwyczaj jedno lub dwuwymiarowych, które przedstawiają graficzne podobieństwo analizowanych danych, grupując podobne dane razem. Jest to sieć złożona z jednej warstwy, w której neurony ułożone są w pewnym porządku topologicznym. Mapa jednowymiarowa to jeden wiersz lub jedna kolumna neuronów. 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.

Etapy tworzenia algorytmu:
1. Inicjacja-przyjęcie losowych wartości wag wszystkich połączeń.
2. Współzawodnictwo-dla każdego z sygnałów wejściowych, wyliczana jest wartość sygnału wyjściowego. W efekcie wyznaczany jest neuron zwycięski(najbliższy sygnałowi wejściowemu).
3. Współpraca – neuron zwycięski wyznacza swoje topologiczne sąsiedztwo neuronów, które będą aktywowane w ramach współpracy.
4. Adaptacja-neuron zwycięski modyfikuje wagi w zależności od sygnału wejściowego. W efekcie kolejna prezentacja podobnego sygnału wejściowego spowoduje silniejszą reakcję tego neuronu.

W przypadku tego ćwiczenia działanie sieci oparte jest o zasadę WTA – Winner Takes All. Algorytm polega na obliczaniu aktywacji każdego neuronu, a następnie wyborze zwycięzcy o największym sygnale wyjściowym. Zwycięski neuron, a więc ten dla którego odległość między wektorem wag w a wektorem wejściowym x jest najmniejsza podlega uczeniu, polegającym na adaptacji swoich wag w kierunku wektora x(k).

Algorytm przebiega więc następująco:

1. Przyjęcie losowych, znormalizowanych wartości wag poszczególnych neuronów.
2. Po podaniu pierwszego wektora wejściowego x wyłaniany jest zwycięzca o numerze k

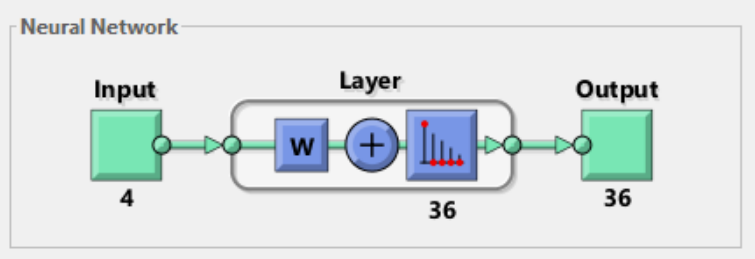
$$\mathbf{w}_k^T \mathbf{x} = \max_{i=1,2,\ldots,K} (\mathbf{w}_i^T \mathbf{x})$$

3. Aktualizacja wag neuronu zwycięzcy wg reguły Kohonena:

$$w_C(k+1) = w_C(k) + \eta(k)[x(k) - w_C(k)]$$

$\eta(k)$ - współczynnik uczenia jest na ogół malejącą funkcją wraz z iteracjami

Neurony przegrywające mają na wyjściu stan zero, co blokuje proces aktualizacji ich wag.
Wektor wag neuronu zwycięzcy jest zwiększany o ułamek różnicy x-w, w wyniku czego w następnych krokach lepiej odtwarza rozpatrywany wektor wejściowy.



Do wykonania danego zadania użyłam funkcji z pakietu Matlab - selforgmap(). Tworzy samoorganizującą się mapę do grupowania zestawu danych. Samoorganizujące się mapy używane są zarówno do grupowania danych jak i do zmniejszania wymiarów danych, ilustrują również graficzne podobieństwo między nimi. Dzięki niej możliwe jest stworzenie mapy neuronów z odpowiednimi parametrami.

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

gdzie:

- dimensions – wektor rzędów wymiarów sieci
- coverSteps – liczba kroków szkoleniowych
- initNeighbor – początkowy rozmiar sąsiedztwa
- topologyFcn – funkcja topologii warstw
- distanceFcn – funkcja odległości neuronowej

Parametry te zostały ustawione następująco:

```
dimensions   = [6 6];
coverSteps   = 100;
initNeighbor = 0;
topologyFcn  = 'hextop';
distanceFcn  = 'dist';
```

gdzie:

- ‘hextop’ – funkcja topologii sześciokątnej ; oblicza położenia neuronów dla warstw, których neurony są ułożone w n-wymiarowy sześciokątny wzór
- ‘dist’ – funkcja odległości euklidesowej

Jako funkcje topologii warstw możliwe jest użycie również opcji ‘gridtop’, która ukazuje topologię przy użyciu prostokątów (oblicza położenia neuronów dla warstw, których neurony są ułożone w n-wymiarową siatkę) , oraz ‘randtop’ , która prezentuje losowo ułożone neurony (oblicza położenia neuronów dla warstw, których neurony są ułożone w n-wymiarowy losowy wzór).

W przypadku funkcji odległości neuronowej mamy do dyspozycji również opcję ‘linkdist’ , która jest używana do znalezienia odległości między neuronami warstwy biorąc pod uwagę ich położenie.

Jak widać powyżej początkowy rozmiar sąsiedztwa ustawiony jest na 0, ponieważ korzystamy tutaj z reguły WTA, która nie bierze pod uwagę neuronów sąsiednich przy wyznaczaniu zwycięzcy.

W celu wykonania zadania został wczytany zestaw danych uczących składający się z 4 parametrów opisujących 150 różnych kwiatów (macierz 4x150). W zestawie iris_dataset znajdują się trzy gatunki kwiatów: setosa, versicolor oraz virginica.

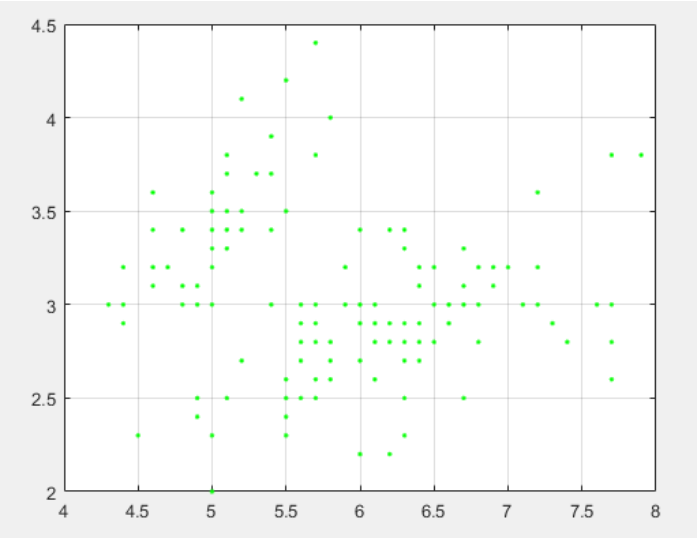
Przykładowe parametry opisujące 6 kwiatów:

5.1000	4.9000	4.7000	4.6000	5.0000	5.4000
3.5000	3.0000	3.2000	3.1000	3.6000	3.9000
1.4000	1.4000	1.3000	1.5000	1.4000	1.7000
0.2000	0.2000	0.2000	0.2000	0.2000	0.4000

gdzie kolejno wierszami mamy:

- długość działki kielicha
- szerokość działki kielicha
- długość płatka
- szerokość płatka

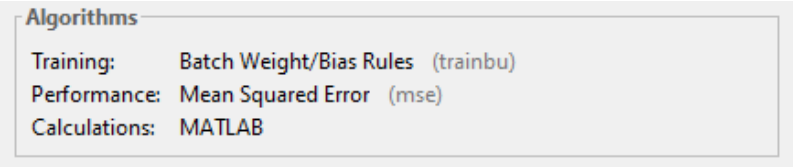
Po wczytaniu dane prezentują się następująco:



Ponadto dla funkcji selforgmap() wywołane zostały następujące procedury:

```
net.trainParam.epochs = 400;
net.trainFcn = 'trainbu';
[net,tr] = train(net,dane);
```

Rozpoczynamy od określenia długość treningu ustalając liczbę epok. Na końcu wywołujemy funkcję ‘train’ dokonującą treningu sieci. Pozwala nam uczyć daną sieć na podstawie zadanych parametrów w net.trainParam za pomocą odpowiedniej funkcji uczącej net.trainFcn. Jej działanie dla sieci jednokierunkowych polega na obliczaniu metodą iteracyjną wartości współczynników wagowych.

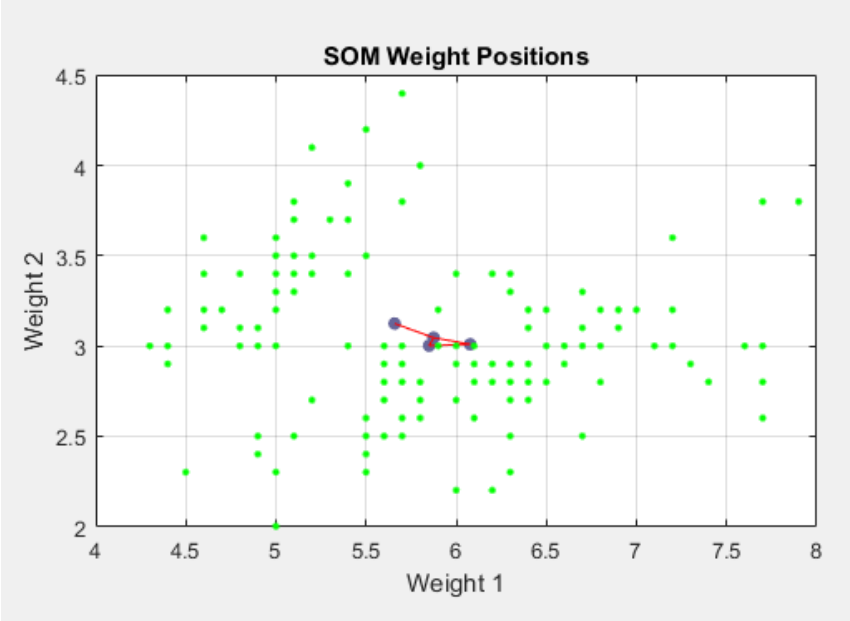


Algorytm ‘trainbu’ szkoli sieć z regułami ważenia i obciążania z aktualizacjami wsadowymi. Aktualizacje wag i błędów pojawiają się na końcu całego przebiegu danych wejściowych. Jest to funkcja szkolenia dla samoorganizujących się map.

Po treningu została wywołana procedura:

```
plotsompos(net,dane);
```

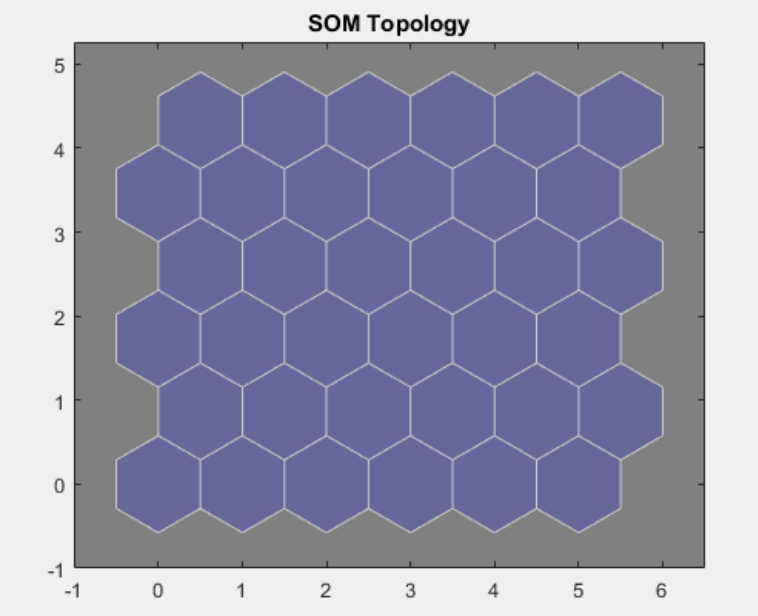
Dzięki niej wyświetlone zostały pozycje wag mapy samoorganizującej się.



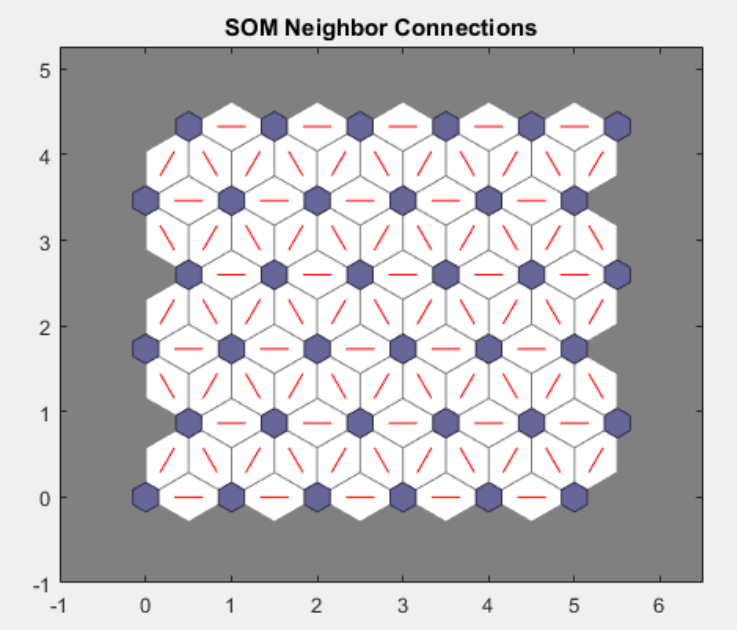
Wektory wejściowe ukazane są jako zielone kropki. Funkcja ta pokazuje w jaki sposób SOM klasyfikuje przestrzeń wejściową. W tym celu wyświetla niebiesko-szare kropki dla każdej wagi neuronu oraz łączy sąsiednie neurony czerwonymi liniami.

2. Zestawienie otrzymanych wyników.

Samoorganizująca się mapa w topologii sześciokątnej:

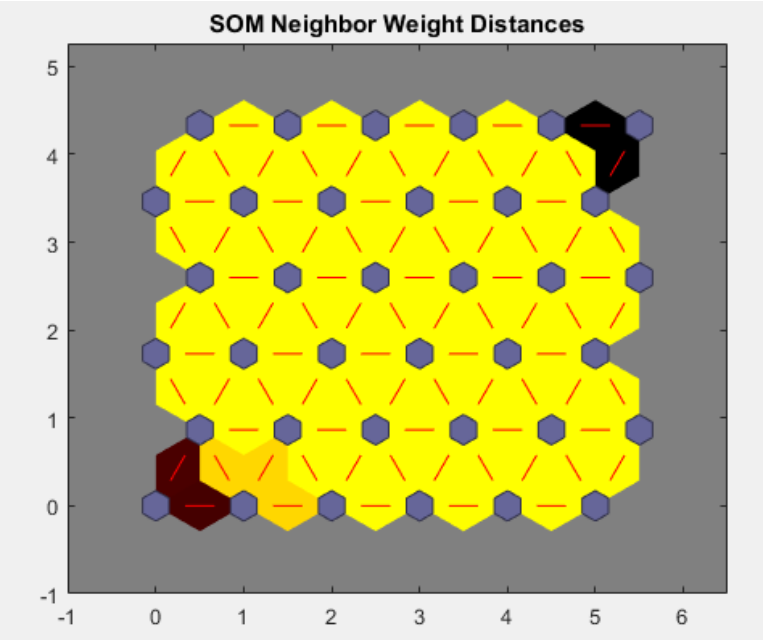


Samoorganizująca się mapa ukazująca połączenia sąsiedzkie:



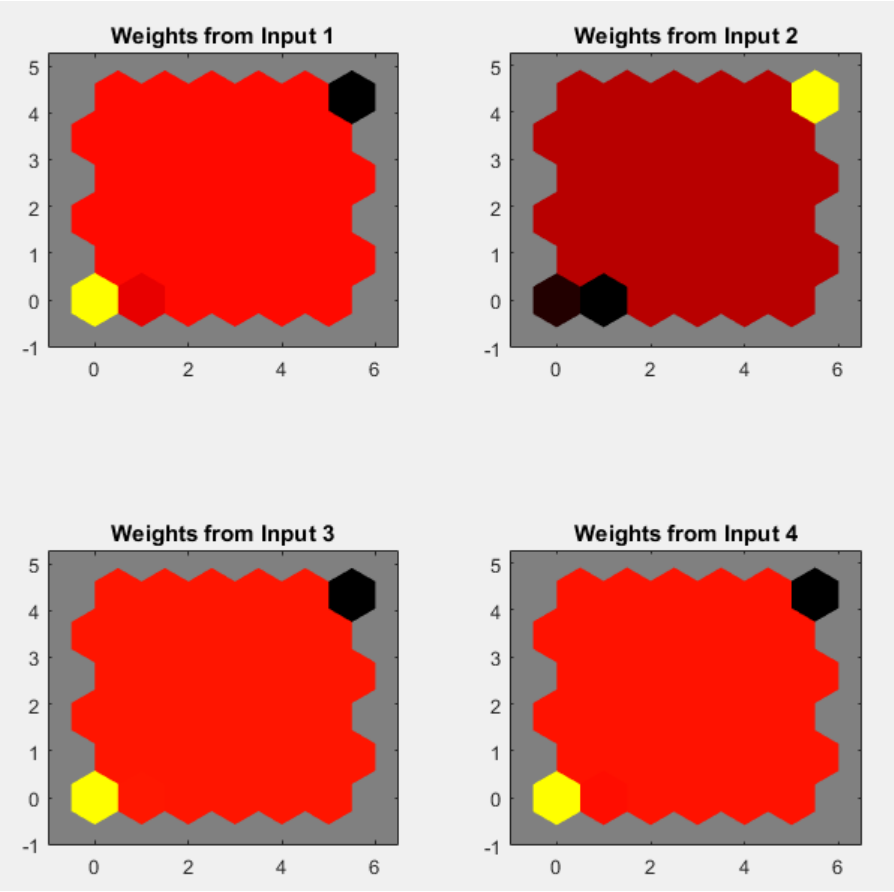
Na mapie widoczne są neurony przedstawione jako szaro-niebieskie łaty oraz ich bezpośrednie relacje z sąsiadującymi neuronami ukazane poprzez czerwone linie.

Samoorganizująca się mapa ukazująca dystans między neuronami:



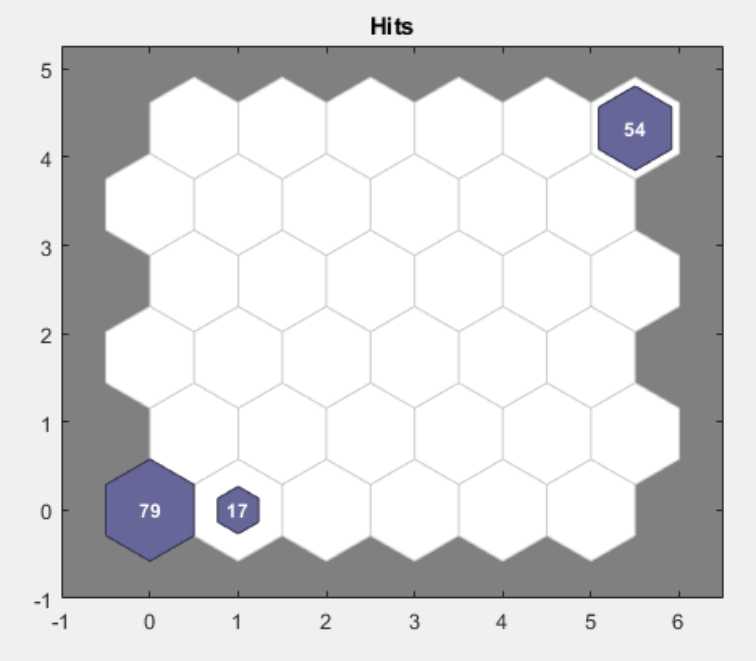
Sąsiadujące łaty są zabarwione od koloru czarnego do żółtego, aby pokazać, jak blisko każdy neuron jest swojego sąsiada. Im ciemniejszy kolor tym dystans jest większy.

Ukazanie wag dla każdego z czterech wejść:



Najbardziej ujemne połączenia są oznaczone jako niebieskie, zero połączeń jako czarne, a najsilniejsze czyli najbardziej pozytywne połączenia jako czerwone.

Samoorganizująca się mapa prezentująca trafienia neuronu:



Każdy neuron pokazuje liczbę wektorów wejściowych, które klasyfikuje. Przedstawia ile razy był zwycięzcą i jakie grupy stworzył.

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

Z przedstawionych powyżej wykresów możemy zauważyć, że ważne jest dobranie odpowiedniej liczby neuronów w celu grupowania sieci. Na ostatniej mapie prezentującej trafienia słusznie zostały ukazane 3 grupy, jednak są one bardzo nierównomierne. Świadczy to o tym, że wagi były modyfikowane zgodnie z regułą WTA, gdzie uaktualnianie następowało tylko dla zwycięzcy. W wyniku współzawodnictwa, neurony które nic nie trafiły nie miały możliwości wygrać z neuronami zwycięskimi.

Podczas testowania różnych funkcji topologii sieci okazało się, że ‘randtop’ nie spełnia naszych oczekiwań. Nie można rozwiązać tego zadania przy jej użyciu. Jednak ‘gridtop’ oraz ‘hextop’ dają podobne rezultaty.

Analiza wyników opracowanej sieci w zależności od wartości współczynnika uczenia nie jest możliwa do zrealizowania w tej wersji programu Matlab. Pojawia się on jednak w regule Kohonena. Na ogół jest on malejącą funkcją wraz z iteracjami. Dzięki temu kontroluje rozmiar wektora wag, a więc jest kluczowy do osiągnięcia prawidłowego rozwiązania.

4. Wnioski.

W celu poznania działania sieci Kohonena przy wykorzystaniu reguły WTA korzystałam z funkcji dostępnych w pakiecie Matlab. Wybrałam funkcję selforgmap() oraz podczas testowania sieci zauważyłam, że idealnie się nadaje do odwzorowania istotnych cech kwiatów . Podczas analizy wyników zauważamy, że zasada WTA powoduje rywalizację między neuronami. Z racji tego, że wagi są uaktualniane dla zwycięskiego neuronu może się okazać, że zgromadzi on większość lub nawet całość sygnałów wejściowych przez co inne neurony nie otrzymają nic. Ponad to istotne jest dobranie odpowiedniej liczby neuronów w celu grupowania sieci. Biorąc pod uwagę współczynnik uczenia wnioskujemy, że kontroluje on rozmiar wektora wag, przez co musi być odpowiednio dokładny aby wynik był prawidłowy.

5. Listing całego kodu.

```
close all; clear all; clc;

% wczytanie zestawu danych kwiatów składającego się z matrycy 4x150
dane = iris_dataset;
size(dane);
% wyświetlenie danych w postaci punktów
plot(dane(1,:),dane(2,:),'g.');
hold on; grid on; %utrzymanie wykresu

Określenie parametrów SOM
% wektor rzędów wymiarów sieci
dimensions = [6 6];
% liczba kroków szkoleniowych
coverSteps = 100;
% początkowy rozmiar sąsiedztwa
initNeighbor = 0;
% funkcja topologii warstw - sześciokątna
topologyFcn = 'hextop';
% - funkcja odległości neuronowej - odległość Euklidesowa
distanceFcn = 'dist';

Tworzenie sieci Kohonena przy pomocji funkcji selforgmap() o rozmiarze 6x6
net = selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn);
% Określenie długości treningu
net.trainParam.epochs = 400;
% Określenie algorytmu treningu
net.trainFcn = 'trainbu';
% Trening sieci
[net,tr] = train(net,dane);
wyjscie = net(dane);
% Skonwertowanie wektorów na indeksy
classes = vec2ind(wyjscie);
% Wyświetlenie pozycji wag
plotsompos(net,dane)
```