Marcelina Bałamut, IO, gr. 1

Temat ćwiczenia:

Budowa i działanie sieci Kohonena dla WTM.

Cel ćwiczenia

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

Opis budowy oraz wykorzystanych sieci i algorytmów uczenia:

System realizujący funkcjonowanie sieci samoorganizującej powinien składać się z kilku podstawowych elementów. Pierwszym z nich jest macierz neuronów pobudzanych przez sygnały wejściowe. Sygnały te powinny opisywać pewne charakterystyczne cechy zjawisk zachodzących w otoczeniu, tak, aby na ich podstawie sieć była w stanie je pogrupować. Informacja o zdarzeniach jest przekładana na bodźce pobudzające neurony. Zbiór sygnałów przekazywanych do każdego neuronu nie musi być identyczny, nawet ich ilość może być różna. Muszą one jednak spełniać pewien warunek, a mianowicie jednoznacznie określać dane zdarzenia. Kolejną częścią składową sieci jest mechanizm, który dla każdego neuronu określa stopień podobieństwa jego wag do danego sygnału wejściowego oraz wyznacza jednostkę z największym dopasowaniem - zwycięzcę. Obliczenia zaczynamy dla wag równych małym liczbom losowym, przy czym ważne jest, aby nie zachodziła żadna symetria. W trakcie uczenia wagi te są modyfikowane w taki sposób, aby najlepiej odzwierciedlać wewnętrzną strukturę danych wejściowych. Istnieje jednak niebezpieczeństwo, że zwiążą się one z pewnymi wartościami zanim jeszcze grupy zostaną prawidłowo rozpoznane i wtedy trzeba ponawiać uczenie z innymi wagami. Wreszcie konieczne do przeprowadzenia samoorganizacji jest, aby sieć była wyposażona w zdolność do adaptacji wartości wag neuronu zwycięzcy w zależności od siły, z jaką odpowiedział on na dane wejście. Załóżmy, że jednostkę, której odpowiedź na dane pobudzenie jest maksymalna, będziemy nazywali "obrazem" tego pobudzenia. Wtedy możemy przyjąć, że sieć jest uporządkowana, jeśli topologiczne relacje między sygnałami wejściowymi i ich obrazami są takie same.

Zatem sieć SOM jest to sieć jednokierunkowa składająca się w dwóch warstw: wejściowej i wyjściowej, która zawiera mapę topologiczną. Natomiast każdy neuron połączony jest ze wszystkimi składowymi n-wymiarowego wektora wejściowego, a wagi połączeń neuronów tworzą wektor wi=[wi1, wi2…win]^T. W strukturze sieci Kohonena istotne jest to, że każdy neuron warstwy wejściowej komunikuje się z każdym neuronem warstwy wyjściowej, a neurony w warstwach nie komunikują się między 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) biorąc pod uwagę promień sąsiedztwa.
3. Adaptacja-neuron zwycięski modyfikuje wagi w zależności od sygnału wejściowego. W efekcie kolejna prezentacja podobnego sygnału wejściowego spowoduje silniejszą reakcję tego neuronu.

Reguła WTM:

Reguły WTA w których tylko jeden neutron może podlegać adaptacji każdej iteracji, sa algorytmami słabo zbieżnymi, szczególnie przy dużej liczbie neuronów. W praktyce zostały one zastąpione algorytmami WTM (ang. Winner Takes Most), w których oprócz zwycięzcy uaktualniają swoje wagi również neurony z jego sąsiedztwa.

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

Odmianą uczenia konkurencyjnego jest uczenie typu WTM w którym neuron wygrywający konkurencję uaktywnia się w sposób maksymalny przyjmując wartość sygnału wyjściowego i umożliwiając częściowe uaktywnienie innych neuronów z sąsiedztwa. Stopień uaktywnienia neuronów z sąsiedztwa zależy od odległości ich wektorów wagowych od wag neuronu wygrywającego.

Algorytm:

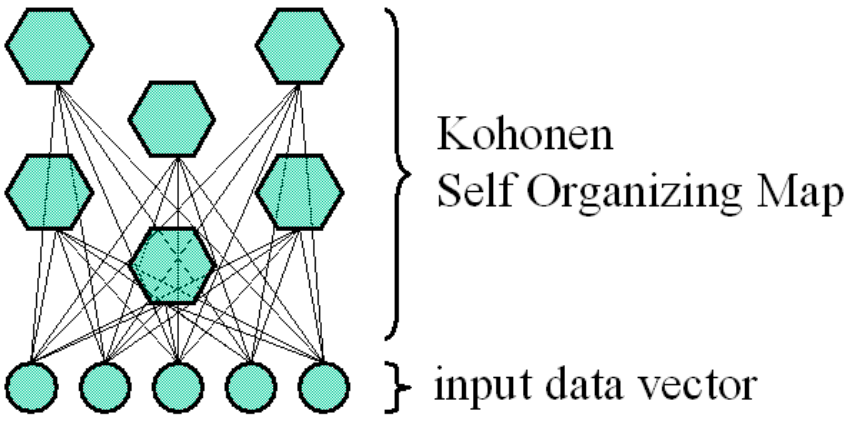
1. Przyjęcie losowych znormalizowanych wartości wag poszczególnych neuronów.
2. Po podaniu pierwszego wektora wejściowego *x* wyłaniany jest zwycięzca o mierze *k*.

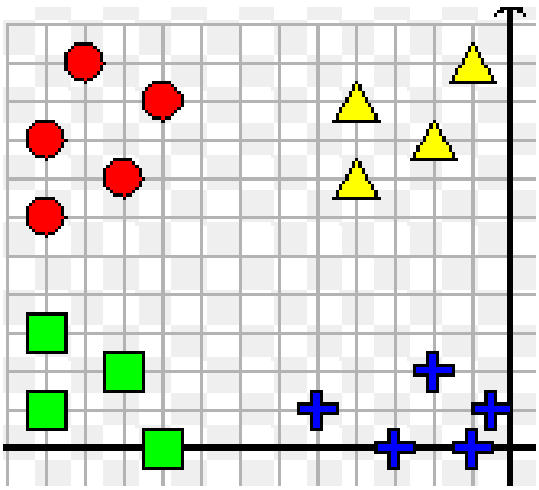
$$\mathbf{w}_k^T \mathbf{x} = \max_{i=1,2,\ldots,K} (\mathbf{w}_i^T \mathbf{x})$$

3. Aktualizacja wag neuronu zwycięzcy oraz neuronów z określonego sąsiedztwa, według wzoru:

$$w_i(k+1) = w_i(k) + \eta(k) G(i,c)[x(k) - w_i(k)], \quad i \in N_c(k)$$

Gdzie funkcja G oznacza wpływ sąsiedztwa na aktualizację wag.

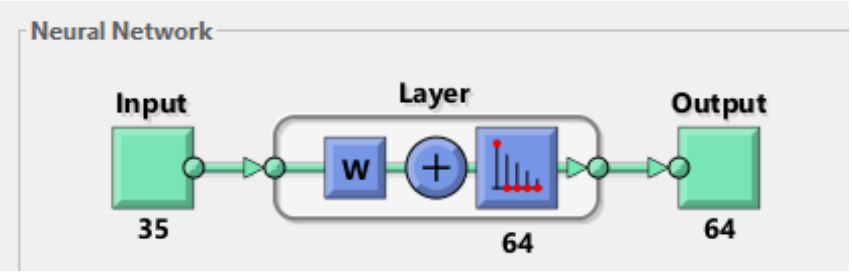




W wykonywanym zadaniu jako dane uczące został wykorzystany wektor składający się z 20 liter, zapisanych w formie binarnej za pomocą macierzy 5x7, co daje 35 pól dla każdej litery, które opisują właściwości liter. Każda litera została transponowana i zapisana do wektora x.

```
x=[  AA(:) BB(:) CC(:) DD(:) EE(:) FF(:) GG(:) HH(:) II(:) JJ(:) KK(:) LL(:) MM(:) NN(:) OO(:) PP(:) QQ(:) RR(:) SS(:) TT(:)];
```

Do implementacji danego problemu użyłam wbudowanej funkcji pakietu Matlab o nazwie selforgmap().



Funkcja selforgmap() tworzy samoorganizującą się mapę do grupowania zestawu danych. Samoorganizujące się mapy uczą się klastra danych na podstawie podobieństwa przypisując tę samą liczbę wystąpień do każdej klasy. Są używane zarówno do grupowania danych jak i do zmniejszania wymiarów danych. Po wywołaniu funkcji tworzony jest w przestrzeni obiekt net w którym zapisane są wszystkie informacje na temat utworzonej sieci.

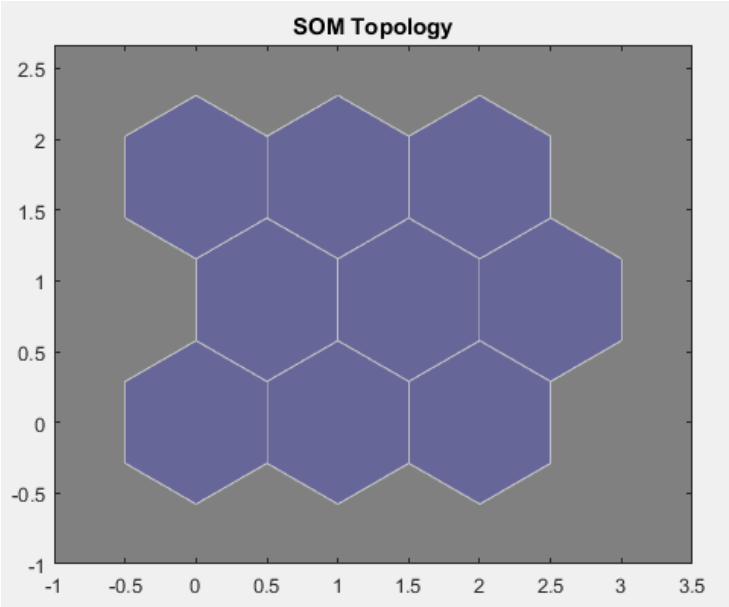
```
dimensions= [3 3];
coverSteps=100;
initNeighbor=2;
topologyFcn='hextop';
distanceFcn='dist';

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

gdzie:
demensions- wektor rzędów wymiarów, domyślnie [3 3]
coverSteps- liczba kroków szkoleniowych dla początkowego pokrycia przestrzeni wejściowej, domyślnie 100
initNeighbor-początkowy rozmiar sąsiedztwa
topologyFcn-finkcja topologii warstw, domyślnie 'hextop'
distanceFcn-funkcja odległości neuronowej, ustawiona na 'dist'

Jak widać powyżej promień sąsiedztwa ustawiony jest na 2, ponieważ korzystamy tutaj z reguły WTM, która bierze pod uwagę neuronów sąsiednich przy wyznaczaniu zwycięzcy.

Do analizowanego problemu została wykorzystana funkcja topologii sześciokątnej. Hextop oblicza położenia neuronów dla warstw, których neurony są ułożone w n-wymiarowy sześciokątny wzór.



Natomiast dist jest to funkcja odległości warstw używana do znalezienia odległości pomiędzy neuronami warstwy, biorąc pod uwagę ich położenie używając odległości euklidesowej. Funkcje wagowe stosują wagi do danych wejściowych aby uzyskać ważone dane wejściowe.

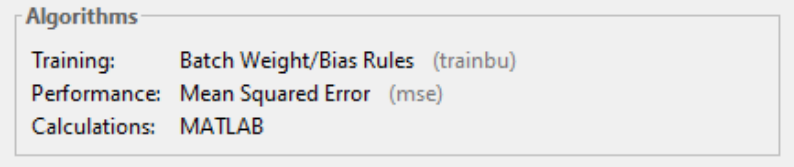
Wzór na miarę euklidesową:

$$d(x, W_i) = \|x - W_i\| = \sqrt{\sum_{j=1}^{N} \left(x_j - W_j^{(i)}\right)^2}$$

Dla funkcji selforgmap została wywołana procedura train, dokonująca treningu zbudowanej sieci. Jest ona uniwersalna, wywoływana w jednolity sposób dla wszystkich typów sieci neuronowych. Działa ona dla sieci jednokierunkowych obliczając wartości współczynników wagowych metodą iteracyjną. Jako parametry przyjmuje obiekt net i wektor wejścia. Przed wywołaniem ustalona została maksymalna liczba kroków uczących(epochs) oraz funkcja trenująca.

```
net.trainParam.epochs = 300;
net.trainFcn='trainbu';
[net,tr] = train(net,x);
```

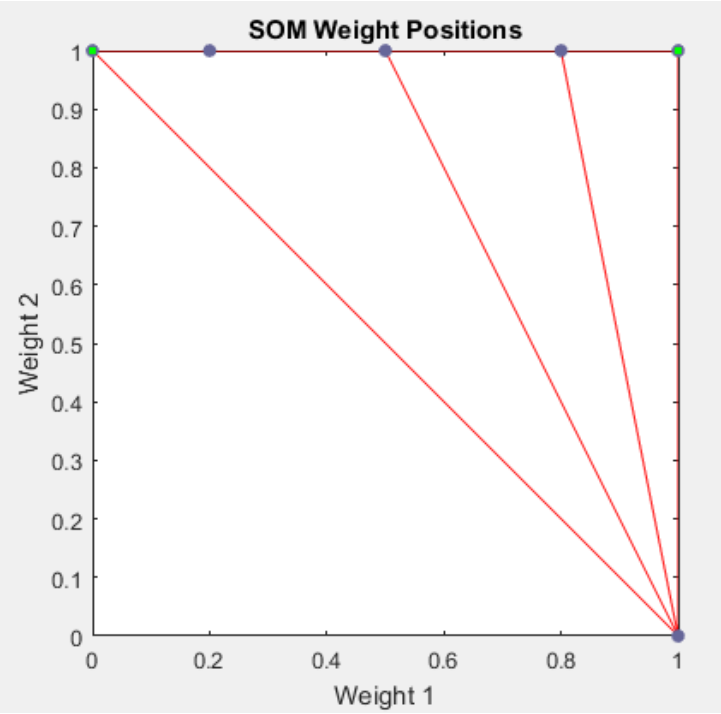
Do treningu sieci została wykorzystana funkcja ‘trainbu’, która realizuje szkolenie w zakresie wagowym i obciążeniowym z aktualizacjami wsadowymi. Aktualizacja wag i błędów następuje na końcu całego przebiegu danych wejściowych. Jest to funkcja szkolenia dla samoorganizujących się map. Sieć identyfikuje zwycięski neuron dla każdego wektora wejściowego. Każdy wektor wagowy przesuwa się do średniej pozycji wszystkich wektorów wejściowych, dla których jest zwycięzcą lub znajduje się w jego pobliżu.



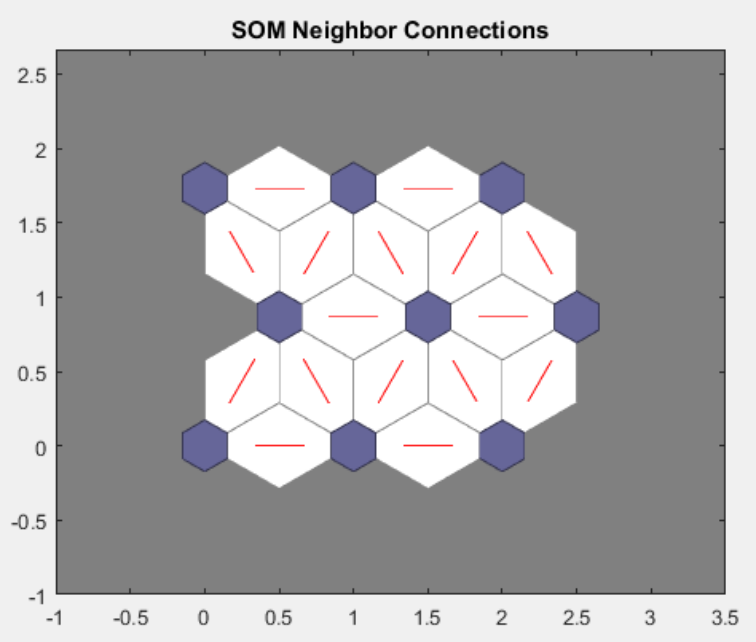
Zestawienie otrzymanych wyników

W analizowanym problemie wyniki przeprowadzonych testów najlepiej odzwierciedlają wykresy określające różne parametry procesu, generowane przez pakiet Matlab.

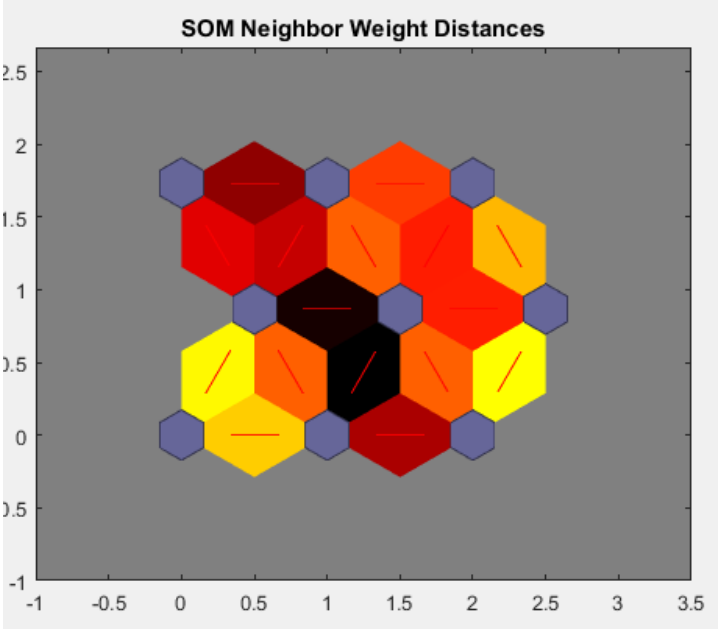
- Na poniższym wykresie kolorem zielonym zostały zaznaczone punkty które odzwierciedlają rozkład danych wejściowych za pomocą wag. Punkty o kolorze niebieskim identyfikują neurony, które są połączone czerwoną linią. Wykres ten pokazuje w jaki sposób SOM klasyfikuje przestrzeń wejściową.



- Poniższy wykres przedstawia neurony jako niebieskie sześciokąty oraz połączenia między nimi jako czerwone linie.

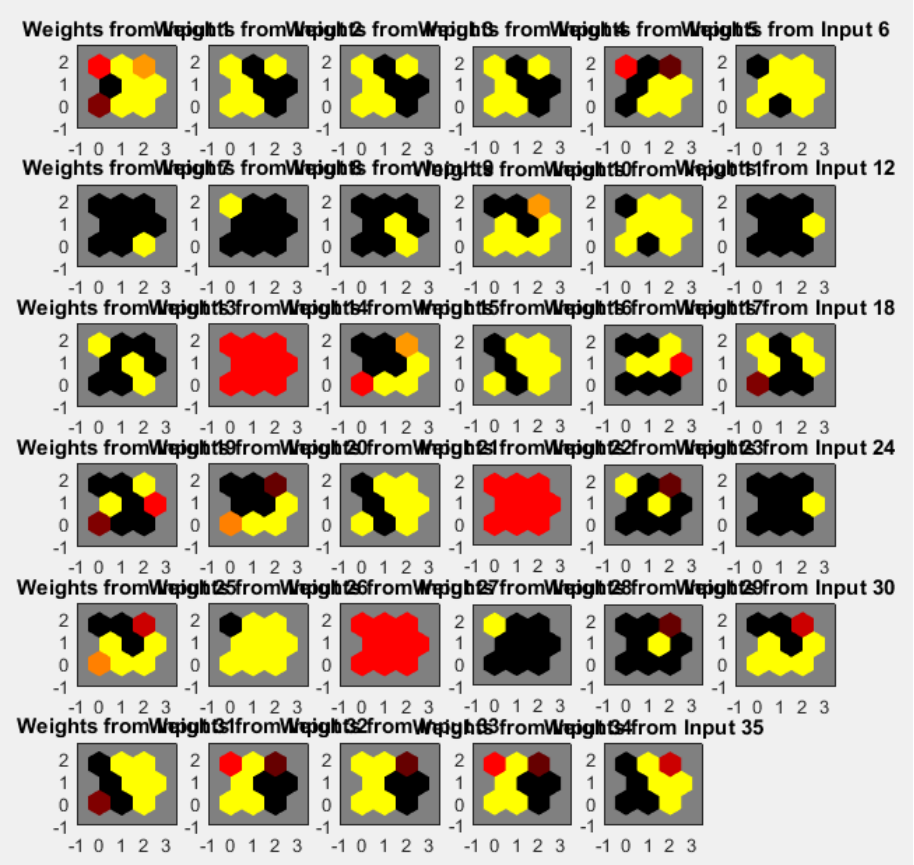


- Odległości między neuronami wyznaczone metodą euklidesową przedstawią się następująco:



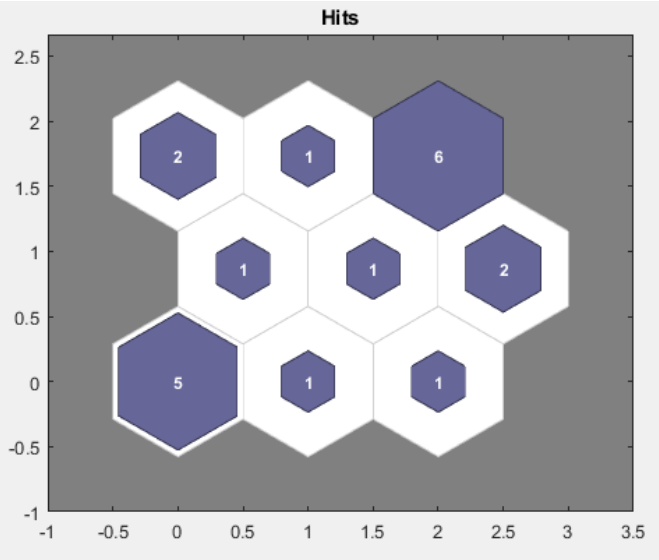
Sąsiadujące elementy przybierają barwy od czarnego do żółtego. Im kolor jest ciemniejszy, tym odległość między neuronami jest większa. Zróżnicowanie kolorów jest dużo większe niż można to było obserwować przy regule WTA, wynika to z faktu wpływu sąsiedztwa na neuron zwycięski.

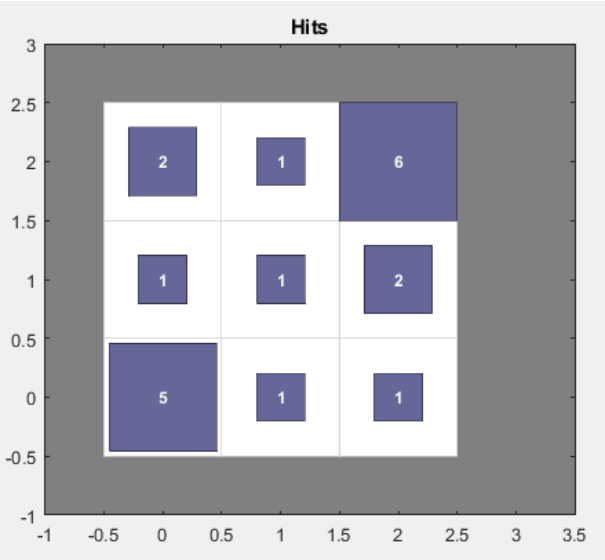
- Poniższy wykres przedstawia rozkład wag dla każdego z wejścia. Im kolor jest ciemniejszy tym waga jest większa.



Ujemne połączenia są oznaczone kolorem niebieskim, połączenia czarne oznaczają zero, natomiast czerwone są to połączenia najsilniejsze i najbardziej pozytywne.

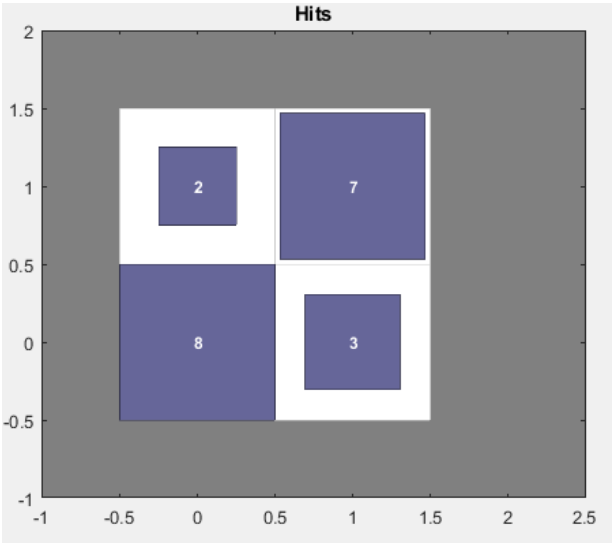
- Kolejne wykresy przedstawiają rozkład trafień neuronu, tzn. dla każdego neuronu przypisują pewną liczbę, która odpowiada za to ile razy dany neuron został zwycięzcą oraz jakie stworzył grupy.



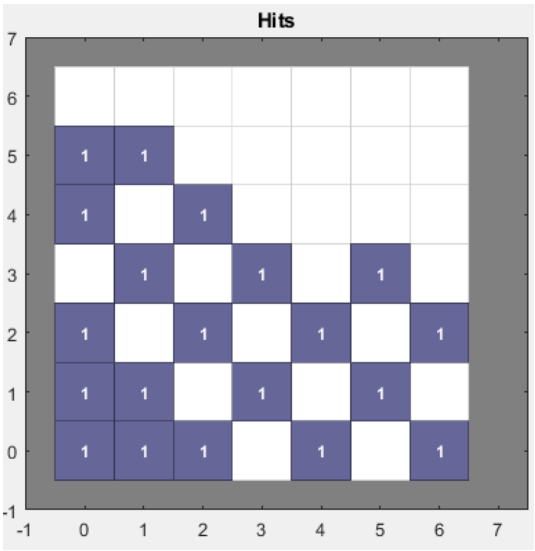


Analiza i dyskusja błędów uczenia i testowania opracowanej sieci.

Z analizy powyższych wykresów oraz testowania różnych wariantów wynika wiele zależności. Najważniejszym chyba aspektem jest fakt, iż sieć prawidłowo wyodrębniła wiele grup, jednak są one bardzo mało liczebne. Niemniej jednak reguła ta działa lepiej niż WTA, ponieważ biorąc pod uwagę sąsiedztwo nie grupuje danych w taki sposób, że grupy są bardzo nierównomierne, tzn. so jednej z grup zostaje przypisana większa część danych. Tutaj zaś przydział do grup został wykonany w sposób bardziej równomierny. Ważna uwagą jest to, że jak widać z powyższych wykresów Messynie jedna grupa zawiera dwie dane, natomiast resztę grup tylko po jednej. Nie jest to do końca dobre działanie grupowania, ponieważ nie jest ono do końca dokładne. MA na to wpływ liczba neuronów. Jeśli ustalimy zbyt duża ich wartość powstanie tyle grup ile jest danych, natomiast jeśli wu stlimy zbyt małą wartość powstanie jedna grupa ze zbyt dużą ilością danych. Najlepszą wartością jest 9 lub 16.



Zbyt mała liczba



Zbyt duża liczba

Dla sieci samoorganizującej się można zastosować funkcję ‘linkdist’, która liczy odległość w zależności od ilości danych wejściowych oraz wykorzystać różne topologie np. prostokątną - ‘gridtop’, lub losową - ‘randtop’. Analizując wyniki pod kątem topologii sieci widać, że niezależnie czy użyjemy topologii prostokątnej czy sześciokątnej zawsze tyle samo grup, tak samo jak ilość w grupach. Natomiast używając topologii losowej nie da się przeprowadzić stosownych testów. Niestety użycie współczynnika uczenia nie było możliwe w tej wersji pakietu Matlab, dlatego nie udało się przeprowadzić testów oraz ich analizy. Jednak w algorytmie Kohohena współczynnik uczenia jest bardzo ważny do uzyskania pożądanego

wyniku i na ogół jest malejącą funkcją wraz z iteracjami, kontrolując przy tym rozmiar wektora wag.

Wnioski

Sieć samoorganizująca się to jednokierunkowa której wyjściem jest mapa. W rozpatrywanym zadaniu sieć posiada wejście w postaci 35 pól dla 20 liter. Sygnały przesyłane są od warstwy wejściowej do warstwy wyjściowej. Do implementacji został użyty algorytm Kohonena z regułą WTM, wykorzystujący zasadę, że wyznaczany jest neuron zwycięski najbardziej podobny do danych wejściowych biorąc pod uwagę promień sąsiedztwa i aktualizowane są tylko jego wagi oraz wagi neuronów sąsiednich., Wykorzystując funkcje wbudowane do programu Matlab udało mi się przeprowadzić stosowne testy oraz na ich podstawie przeprowadzić analizę z której wynika wiele wniosków. Analizując wyniki działania funkcji można stwierdzić, iż ustawiane parametry mają znaczący wpływ na działanie sieci. W kodzie programu można wprowadzać różne topologie sieci oraz różne miary odległości między neuronami co nieznacznie wpływa na wynik sieci. Niezależnie czy wybierzemy topologię prostokątną, czy sześciokątną sieci widać, że powstaje taka sama liczba grup z inna wartością danych. Wynika stąd wniosek, iż wykorzystując regułę WTM nie udało się podzielić liter na równe grupy jednak nie są one bardzo zróżnicowane. Natomiast topologią losową nie da się rozwiązać zadanego problemu. Ważnym wnioskiem jest fakt, że ilość neuronów ma wpływ na wynik grupowania. Tak samo zbyt duża liczba jak zbyt mała nie da pożądanego wyniku. Niestety nie udało się uzyskać wyników w zależności od zmiany wartości współczynnika uczenia.

Listing kodu wykonanego programu

```
close all; clear all; clc;

% 20 dużych liter alfabetu, zapisanych w każda jako 35 pól
A=[0 1 1 1 0 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];
B=[1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0];
C=[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];
D=[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];
E=[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];
F=[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];
G=[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];
H=[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];
I=[0 1 1 1 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 0 1 1 1 0];
J=[1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0];
K=[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];
L=[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 0 0 1 1 1 1 1];
M=[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];
N=[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];
```

```
O=[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];
P=[1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0];
Q=[0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 1 0 1 1 0 0 1 1 0 1 1 1 1];
R=[1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 1];
S=[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];
T=[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];

% transpozycja każdej z litery
  AA=transpose(A);
  BB=transpose(B);
  CC=transpose(C);
  DD=transpose(D);
  EE=transpose(E);
  FF=transpose(F);
  GG=transpose(G);
  HH=transpose(H);
  II=transpose(I);
  JJ=transpose(J);
  KK=transpose(K);
  LL=transpose(L);
  MM=transpose(M);
  NN=transpose(N);
  OO=transpose(O);
  PP=transpose(P);
  QQ=transpose(Q);
  RR=transpose(R);
  SS=transpose(S);
  TT=transpose(T);

% wektor x składający się z wcześniej podanych liter
x=[  AA(:) BB(:) CC(:) DD(:) EE(:) FF(:) GG(:) HH(:) II(:) JJ(:) KK(:)
LL(:) MM(:) NN(:) OO(:) PP(:) QQ(:) RR(:) SS(:) TT(:)];


size(x);
% wyświetlanie danych w postaci punktów
plot( x(1, :), x(2, :), '.b');

 dimensions= [3 3];       % wektor rzędów wymiarów, domyślnie [3 3]
 coverSteps=100;          % liczba kroków szkoleniowych dla początkowego
                          pokrycia przestrzeni wejściowej, domyślnie 100
 initNeighbor=2;          % promień sąsiedztwa
 topologyFcn='hextop';    % sześciokątna topologia warstw
 distanceFcn='dist';      % euklidesowa funkcja odległości neuronowej
 % tworzenie samoorganizującej się mapy w parametrami opisanymi wyżej
 net=
selforgmap(dimensions,coverSteps,initNeighbor,topologyFcn,distanceFcn);

 net.trainParam.epochs = 300;  % maksymalna liczba epok
 net.trainFcn='trainbu';       % funkcja uczenia
% trening sieci
 [net,tr] = train(net,x);
 y = net(x);
 classes = vec2ind(y);
 % wyświetlanie
 plotsompos(net, x);
```