|  |  |  |
| --- | --- | --- |
| Grupa ćwicz. **2** | Data wykonania 11.12.2017 | Nr. Scenariusza  **6** |
| **Temat ćwiczenia:** Budowa i działanie sieci Kohonena dla WTM. | | |
| Imię i nazwisko  **Kamil Szczurkowski** | | Ocena i Uwagi |

**Cel ćwiczenia:**

Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły

WTM do odwzorowywania istotnych cech liter alfabetu.

**Wykonane zadania:**

1. Wygenerowano dane uczące i testujące, zawierające 20 dużych liter alfabetu łacińskiego w postaci dwuwymiarowej tablicy 7x5, reprezentowanej w kodzie jako jednowymiarowa tablica 35 elementowa.
2. Wykorzystano narzędzie Matlab do stworzenia sieci (selforgmap) oraz reguły WTM
3. Uczono sieć dla różnych współczynników uczenia.
4. Testowano poprawność działania sieci.

**Specyfikacja wykorzystanych funkcji:**

**train** - Trening sieci neuronowej.

TRAIN Funkcja realizuje trening sieci neuronowej, wykorzystując funkcję treningu, której

nazwa została podana w polu NET.trainFcn, zaś wartości niezbędnych parametrów – w polu NET.trainParam. Funkcja treningu jest uniwersalna, wywoływana w jednolity sposób dla wszystkich typów sieci neuronowych, stąd też niektóre argumenty wejściowe nie mają znaczenia dla sieci jednokierunkowych (bez sprzężeń zwrotnych i opóźnień w torach sygnałów wejściowych)

*Wywołanie funkcji:* **[NET, TR, Y, E, Pf, Af] = TRAIN(NET, P, T, Pi, Ai)**

*lub:* **[NET, TR, Y, E, Pf, Af] = TRAIN(NET, P, T, Pi, Ai, VV, TV)**

net.trainFcn = 'trainbu' - szkolenie bez nauczyciela w zakresie waga / obciążenie. trainbu trenuje sieć z regułami ważenia i stronniczości z aktualizacjami wsadowymi. Aktualizacje wag i błędów pojawiają się na końcu całego przebiegu danych wejściowych.

**Selforgmap** ( dimensions, coverSteps, initNeighbor, topologyFcn, distanceFcn ) - Mapa samoorganizująca się.

Samoorganizujące się mapy uczą się klastra danych na podstawie podobieństwa, topologii, z preferencją (ale bez gwarancji) przypisywania tej samej liczby wystąpień do każdej klasy.

Samoorganizujące się mapy są używane zarówno do grupowania danych, jak i do zmniejszania wymiarów danych. Inspirują się sensorycznymi i motorycznymi mapowaniami w mózgu ssaka, które również wydają się automatycznie organizować informacje topologicznie.

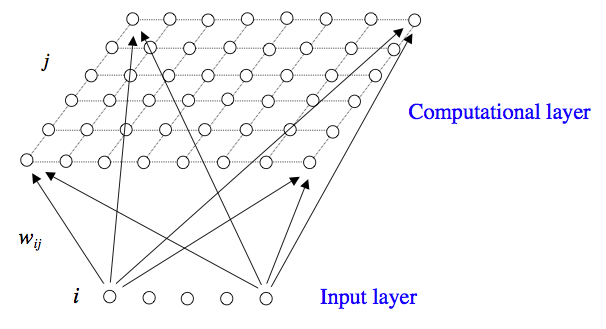
**dimensions –** Wektor rzędów wymiarów (domyślnie = [8 8])

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

**initNeighbor-** Początkowy rozmiar sąsiedztwa (domyślnie = 3)

**topologyFcn-** Funkcja topologii warstw (domyślnie = 'hextop')

**distanceFcn -** Funkcja odległości neuronowej (domyślnie = "linkdist")



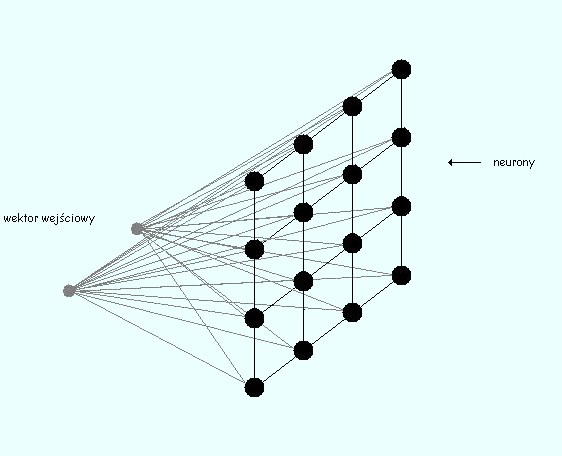
Rysunek 1 - sieć Kohonena

**Czym są sieci Kohonena:**

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.

**Architektura sieci samoorganizujących się:**

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.



Rysunek - Dwuwymiarowa mapa neuronów

 Funkcjonowanie samoorganizujących się sieci neuronowych odbywa się w trzech etapach:

* konstrukcja
* uczenie
* rozpoznawanie

System, który miałby realizować 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 i jego sąsiadów w zależności od siły, z jaką odpowiedział on na dane wejście. Topologię sieci można w łatwy sposób określić poprzez zdefiniowanie sąsiadów dla każdego neuronu. 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.

**WTM – Winner Takes Most** -neuron zwycięski oraz neurony sąsiadujące z neuronem zwycięskim, czyli należące do sąsiedztwa *Nc(k)*, aktualizują swoje wagi według zasady



Rysunek - Wzór WTM

**Wykonanie zadania:**

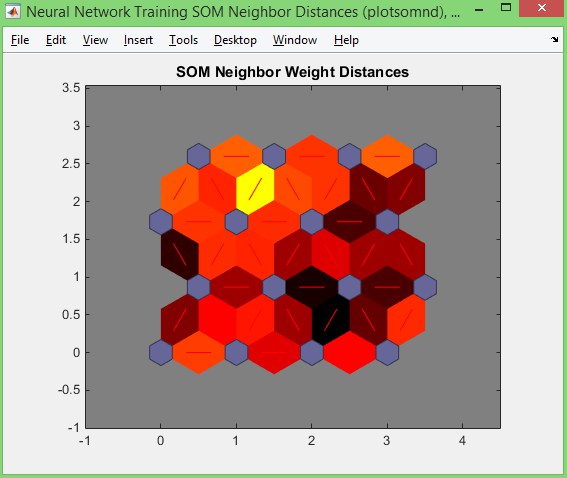
Do wykonania zadania użyłem zastawu danych z 20 literami. W sieci do odwzorowania istotnych cech liter użyłem:

a).

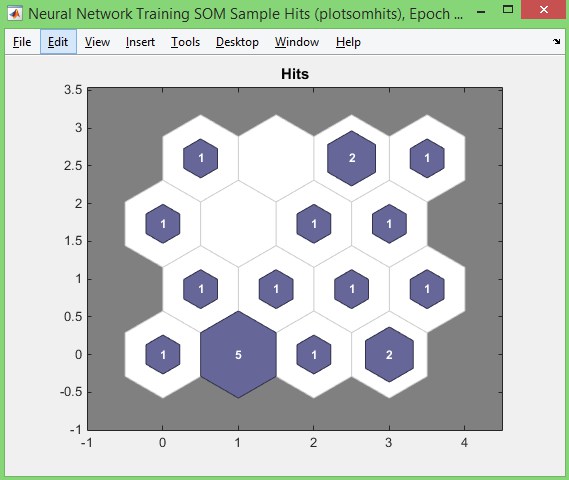
dimensions = [4 4];

coverSteps = 100;  
initNeighbor = 3;  
topologyFcn = hextop;  
distanceFcn = linkdist;

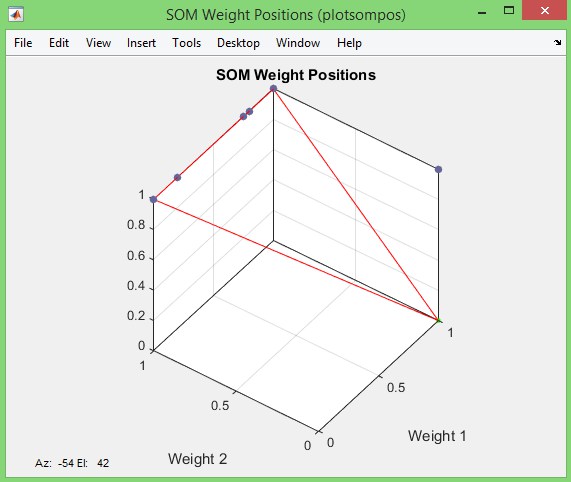
**Wyniki:**



Rysunek 4- Dystans miedzy neuronami



Rysunek 5 - zwycięskie neurony



Rysunek 6 - wynik działania programu

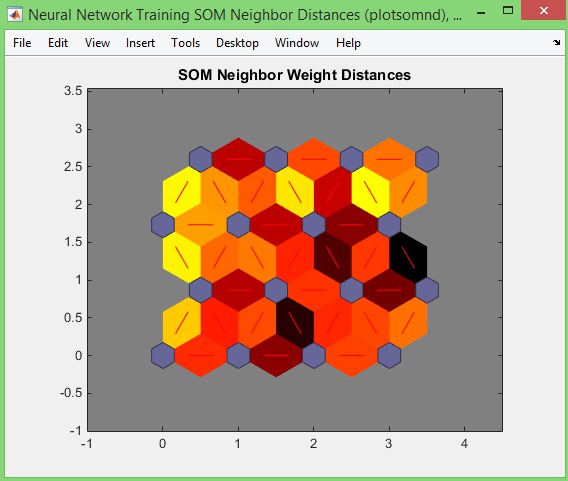
Przy tak zbudowanej sieci można zauważyć ze jedne z neuronów wygrywał bardzo często i przy takim doborze parametrów z sieci WTM zrobiła nam się WTA. Ustawienie większego sąsiedztwa nic nie pomogło. Odległości pomiędzy innymi neuronami tez są dość duże. Jeśli chodzi o wynik działania programu można zauważyć ze wyniki nam się pokryły i ciężko stwierdzić co gdzie jest.

b).

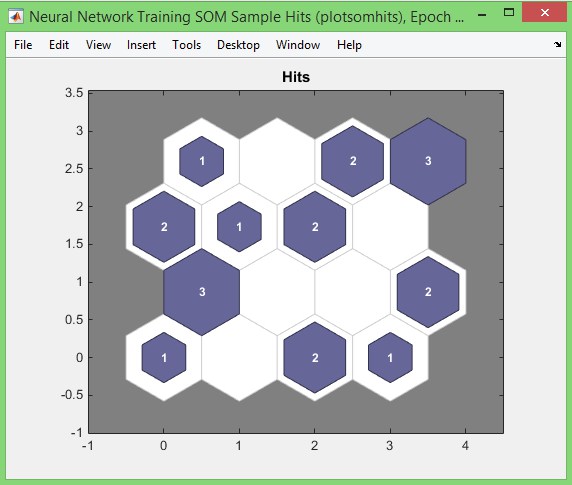
dimensions = [4 4];

coverSteps = 200;  
initNeighbor = 1;  
topologyFcn = hextop;  
distanceFcn = linkdist;

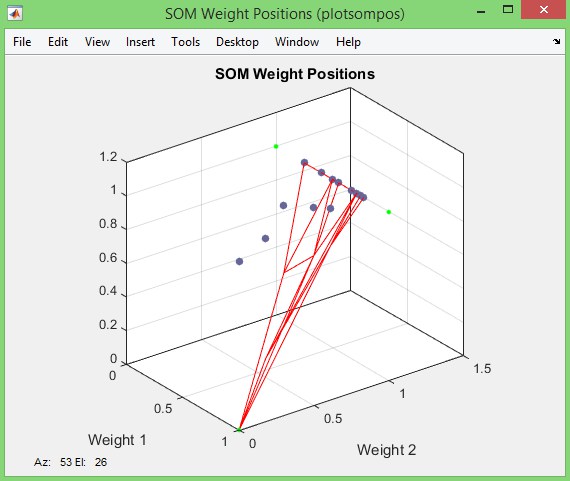
**Wyniki:**



Rysunek 7- Dystans miedzy neuronami

****

Rysunek 8- zwycięskie neurony



Rysunek 9 - wynik działania programu

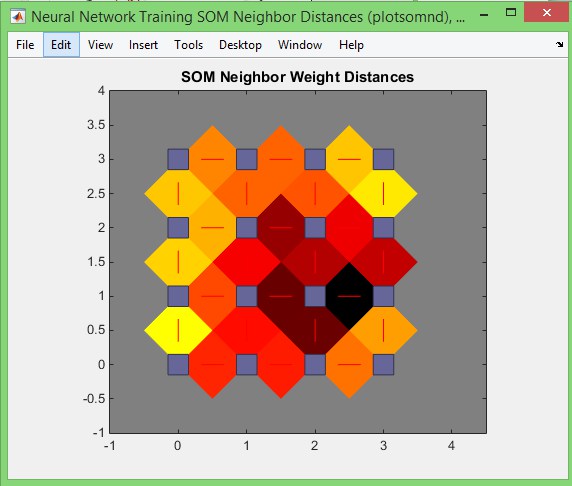
Przy tak zbudowanej sieci można zauważyć ze nie występuje tu wyższości jednego neuronu, są 2 które wygrały 3 razy oraz 4 które wygrały 2. Widać lepszy rozkład sąsiadów pomiędzy neuronami które wygrywały. Odległości pomiędzy neuronami są dużo mniejsze jak w przykładzie a). kilka połączeń jest bardzo dobrych można to zaobserwować na wykresie nr 5. Natomiast wynik programu obrazuje już nam troszkę lepiej dane cechy.

c).

dimensions = [4 4];

coverSteps = 400;  
initNeighbor = 1;  
topologyFcn = gridtop;  
distanceFcn = linkdist;

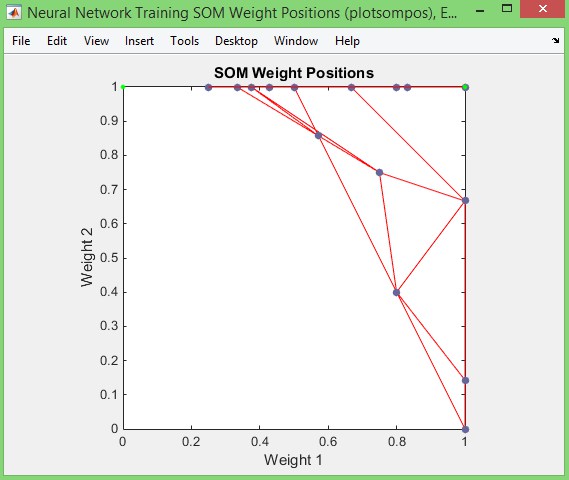
**Wyniki:**



Rysunek 10- Dystans miedzy neuronami



Rysunek 11- zwycięskie neurony



Rysunek 12- wynik działania programu

Tutaj natomiast użyliśmy reprezentacji gridtop która jest przedstawiana za pomocą kwadratów. Przy tej sieci otrzymaliśmy bardzo dobry rozkład sieci, można powiedzieć ze zrobił się kwadrat na około z neuronami które wygrywały taka samą ilość razy oraz z bardzo małymi odległościami pomiędzy nimi. Rozkład punktów wynikowych pokazuje zróżnicowanie naszych punktów co można uznać za zadowalający wynik.

d).

dimensions = [1 3];

coverSteps = 400;  
initNeighbor = 1;  
topologyFcn = gridtop;  
distanceFcn = linkdist;

**Wyniki:**



Rysunek 13- Dystans miedzy neuronami



Rysunek 14- wynik działania programu

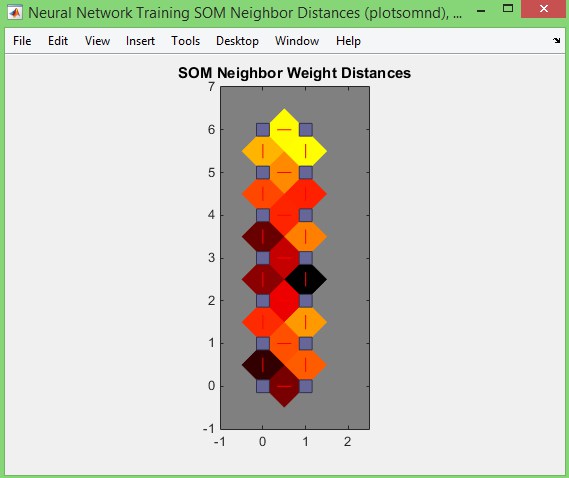
Przy zastosowaniu małej liczby neuronów podczas budowania sieci można zauważyć nie otrzymamy wyników które by miały jakiś duży sens co dziwne odległości pomiędzy nimi są ogromne .

d).

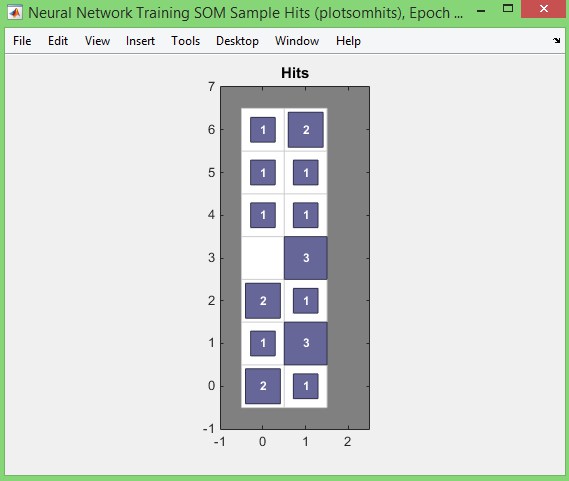
dimensions = [2 7];

coverSteps = 400;  
initNeighbor = 1;  
topologyFcn = gridtop;  
distanceFcn = linkdist;

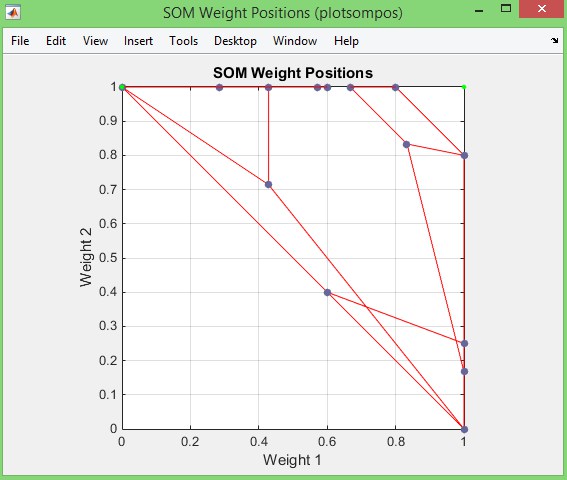
**Wyniki:**



Rysunek 15- Dystans miedzy neuronami



Rysunek 16- zwycięskie neurony



Rysunek 17- wynik działania programu

Podczas zastosowania sieci 2 wymiarowej można już zauważyć bardzo ciekawy rozkład wyników na wykresie, otrzymaliśmy je dzięki zastosowaniu większej liczby neuronów. Tylko jeden z neuronów nie wygrał w żadnej z iteracji, jest to bardzo dobry wynik patrząc na przeprowadzone wcześniejsze pomiary. Natomiast jeśli chodzi o odległości pomiędzy nimi nie są jakoś bardzo złe ale można już zauważyć patrząc od góry ze ich odległość się zwiększa.

**Wnioski:**

Na podstawie przeprowadzonych badan można było zauważyć ze dobranie liczby kroków szkoleniowych oraz początkowy rozmiar sąsiedztwa przykładało się na rozkłady wygranych neuronów.

Przy zastosowaniu zbyt małej liczby neuronów w sieci nie otrzymamy zbyt dokładnych danych lub co gorsza nasze dane nie będą miały żadnego sensu.

Moim zdaniem najlepiej sprawdziła się tu topologia kwadratów dla sieci 4x4 otrzymaliśmy bardzo dobry rozkład sąsiadów oraz odległości pomiędzy nimi w krótkim czasie

Wysoki wymiar sieci oraz duża liczba neuronów przekłada się na wysokie nakłady czasowe.

**Bibliografia:**

Wykład

<https://www.mathworks.com/help/nnet/ref/selforgmap.html>

<https://pl.wikipedia.org/wiki/Sie%C4%87_Kohonena>

<https://www.mathworks.com/help/nnet/ref/trainbu.html>

**Listing:**

close all; clear all; clc;

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 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 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 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];

AA=A'; BB=B'; CC=C'; DD=D'; EE=E'; FF=F'; GG=G'; HH=H'; II=I'; JJ=J';

KK=K'; LL=L'; MM=M'; NN=N'; OO=O'; PP=P'; QQ=Q'; RR=R'; SS=S'; TT=T';

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

dimensions = [6 7];

coverSteps = 400;

initNeighbor = 1;

topologyFcn = 'gridtop';

distanceFcn = 'linkdist';

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

net.trainFcn = 'trainbu';

net = train(net,wejscie);

y = net(wejscie);

classes = vec2ind(y);

plotsompos(net,wejscie);