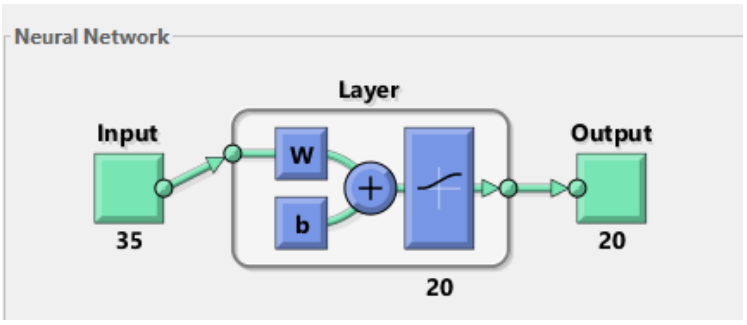
Fabiola Dąbroś , IO gr.1 rok III

Sprawozdanie z zagadnienia nr 4.

Temat ćwiczenia:
Uczenie sieci regułą Hebba.
Cel ćwiczenia:
Celem ćwiczenia jest poznanie działania reguły Hebba dla sieci jednowarstwowej na przykładzie grupowania liter alfabetu.

1. **Opis budowy użytej sieci oraz wykorzystanego algorytmu uczenia.**

Wszystkie sieci neuronowe składają się z neuronów połączonych synapsami powiązanymi z wagami, których interpretacja zależy od modelu. Do wykonania tego ćwiczenia korzystałam z sieci jednowarstwowej. Neurony w tej sieci ułożone są w jednej warstwie, zasilanej jedynie z węzłów wejściowych. W sieciach jednokierunkowych przepływ sygnału przebiega zawsze w ściśle określonym kierunku, czyli od warstwy wejściowej do warstwy wyjściowej. W węzłach wejściowych nie zachodzi proces obliczeniowy dlatego nie tworzą one warstwy neuronowej. Sieć jednowarstwową stanowią między innymi perceptrony jednowarstwowe.



Do wykonania danego zadania użyłam funkcji newff(). Służy ona do tworzenia jednowarstwowej lub wielowarstwowej sieci neuronowej złożonej z neuronów o nieliniowej funkcji aktywacji. Na wejściu ustalamy liczbę wejść sieci oraz liczbę neuronów sieci w warstwie. Ponadto mamy funkcję aktywacji neuronów oraz funkcję służącą do trenowania sieci wraz z algorytmem. Natomiast na wyjściu tworzy się struktura zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry sieci. Opisana funkcja wygląda następująco:

```
net=newff( minmax(dane_we), 20, {'logsig', 'learnh', 'trainr'} );
```

gdzie:

- minmax(dane_we) - spodziewane zakresy dla elementów wejściowych
- 20 - liczba neuronów w warstwie
- ‘logsig’ – nazwa funkcji aktywacji neuronów
- ‘learnh’ – nazwa funkcji trenowania sieci
- ‘trainr’ – nazwa algorytmu trenowania sieci

Neurony mogą wykorzystywać dowolną zróżnicowaną funkcję transferu *f,* aby wygenerować swoją wydajność. Newff() w tym przypadku korzysta z funkcji aktywacji ‘logsig’. Jest to funkcja logarytmiczno sigmoidalna unipolarna (niesymetryczna). Generuje ona wyjścia od 0 do 1, ponieważ wejście sieci neuronowej przechodzi od ujemnej do dodatniej nieskończoności.

W celu wykonania zadania został stworzony zestaw danych uczących, który składa się z 20 dużych liter alfabetu. Przedstawione są one jako matryca 5x7 o wartościach 0 lub 1 co daje nam 35 pól.

```
dane_we=[  dA(:) dB(:) dC(:) dD(:) dE(:) dF(:) dG(:) dH(:) dI(:) dJ(:)
           dK(:) dL(:) dM(:) dN(:) dO(:) dP(:) dR(:) dS(:) dT(:) dW(:)];
                                                                       ]
```

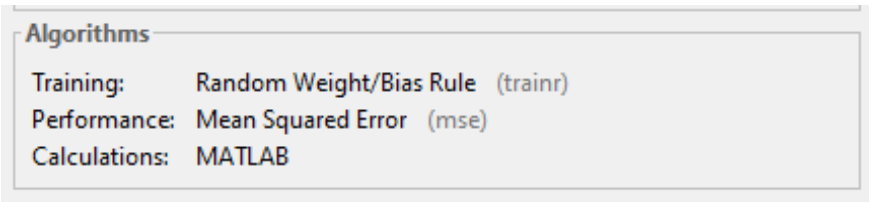
Jako dane wyjściowe mamy wartości 0 oraz 1, gdzie 1 oznacza daną literę w kolumnie tablicy. W rezultacie tworzy nam się tablica cyfr, na której przekątnej występują same 1.

```
dane_wy= [1 0 0 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 0 0 0 0 0 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 0 0 ;
          0 0 0 1 0 0 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 0 0 0 0 0 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 0 0 0 0 0 1 0 0 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 0 0 0 0 0 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 0 0 0 0 0 1 0 0 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 0 0 0 ;
          0 0 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 0 0 0 0 0 1 0 0 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 ;
          0 0 0 0 0 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 0 0 0 0 0 1 0 0 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 0 0 0 0 0 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 0 0 0 0 0 1 0 0 ;
          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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 ;
          ];
```

Ponadto dla funkcji newff() wywołane zostały następujące procedury:

```
net.trainParam.epochs = 100;
net.trainParam.goal = 0.001;
lp.lr = 0.01;
dw=learnh([],dane_we,[],[],dane_wy,[],[],[],[],[],lp,[]);
net.inputWeights{:,:}.learnFcn = 'learnh';
net.layerWeights {:,:} .learnFcn = 'learnh';
net=train(net, dane_we,dane_wy);
```

Rozpoczynamy od określenia długość treningu ustalając liczbę epok. Poprzez 'goal' określamy kryterium stopu - sumę kwadratów błędów wyjść sieci oraz poprzez 'lr' określamy wartość współczynnika uczenia sieci. Natomiast 'dw' wykorzystujemy do obliczenia zmiany wag poprzez wywołanie funkcji learnh. Kolejno ustalamy również każdą wagę z funkcji uczenia, czyli metodą Hebba. Na końcu wywołujemy funkcję 'train' dokonującą treningu sieci. Jej działanie dla sieci jednokierunkowych polega na obliczaniu metodą iteracyjną wartości współczynników wagowych.



Funkcja newff() opiera swoje działanie na algorytmie trainr. Szkoli on sieć zgodnie z regułami ważenia i obciążania z przyrostowymi aktualizacjami po każdej prezentacji danych wejściowych. Wejścia są wyświetlane w losowej kolejności. Działa on w oparciu o funkcję 'learnh', która polega na uczeniu wagami Hebba.

- Reguła Hebba

Jest to jedna z najpopularniejszych metod samouczenia sieci neuronowych. Polega ona na tym, że sieci pokazuje się kolejne przykłady sygnałów wejściowych, nie podając żadnych informacji o tym, co z tymi sygnałami należy zrobić. Sieć obserwuje otoczenie i odbiera różne sygnały, nikt nie określa jednak, jakie znaczenie mają pokazujące się obiekty i jakie są pomiędzy nimi zależności. Sieć na podstawie obserwacji występujących sygnałów stopniowo sama odkrywa, jakie jest ich znaczenie i również sama ustala zachodzące między sygnałami zależności. Ponad to zakłada ona umacnianie połączeń pomiędzy źródłami silnych sygnałów i osłabianie połączeń pomiędzy źródłami słabych sygnałów.
Adaptacja wag przebiega iteracyjnie. Hebb zaproponował algorytm, zgodnie z którym modyfikację wag przeprowadza się następująco:

$$w_i(t+1) = w_i(t) + \eta y x_i$$

gdzie:
- i- numer wagi neuronu,
- t -numer iteracji w epoce,
- y- sygnał wyjściowy neuronu,
- x- wartość wejściowa neuronu,
- η - współczynnik uczenia (0,1)

Po treningu sieci utworzony został zestaw testujący w celu sprawdzenia wyniku uczenia dla każdej litery. W zestawie tym znajdują się również zmodyfikowane litery, które sieć musi przyporządkować do poszczególnych grup liter.

```
rezultat=sim(net, testA3);

max=1;
for i=1:1:20;
   if(rezultat(max)<rezultat(i))
      max=i;
   end;
end
```

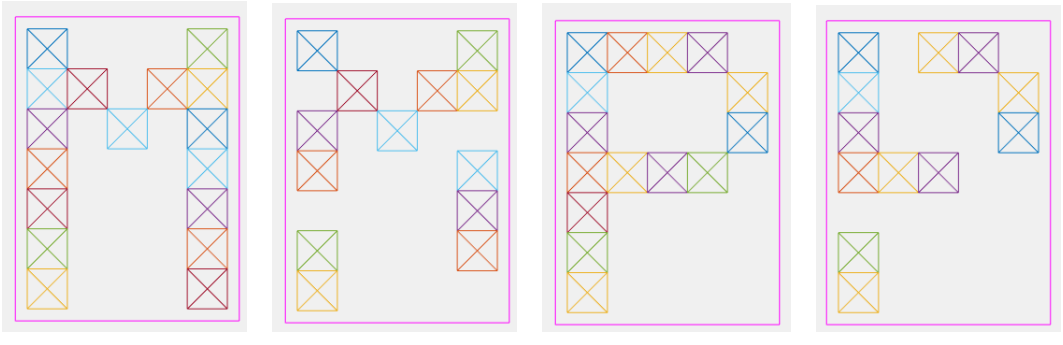
Następnie wykonujemy symulację sieci dla konkretnego testu oraz poszukujemy w pętli największej wartości dzięki czemu możliwe jest grupowanie.

2. Zestawienie otrzymanych wyników.

Tabela wyników uczenia dla testowania różnych wartości współczynnika uczecenia prezentuje się następująco:

Współczynnik uczenia	Liczba epok	Czas realizacji
0,1	13	0:00:01
2	11	0:00:01
0,0001	18	0:00:02
2,5	6	0:00:00

Tabela przedstawiająca podobieństwo konkretnej litery do poszczególnych liter prezentuje się następująco:



	M	Zmodyfikowane M	P	Zmodyfikowane P
A	0,00055901	0,00011466	0,0091139	0,008768
B	0,00000011339	0,00095343	0,0062345	0,00024968
C	0,0000013506	0,0018702	0,0012599	0,00019054
D	0,0016817	0,001267	0,017912	0,0036269
E	0,000000017168	0,00000024382	0,0000049699	0,00000017646
F	0,000050107	0,20584	0,003767	0,00039004
G	0,00017849	0,00061259	0,000031183	0,0000000084343
H	0,0017113	0,046398	0,00024483	0,00066562
I	0,00000030616	0,0017376	0,000053451	0,0000078281
J	0,0017058	0,22169	0,00029151	0,00000080687
K	0,0010675	0,01131	0,000012568	0,00000052009
L	0,00050302	0,00074653	0,0000561	0,011717
M	**0,99711**	**0,9471**	0,0002664	0,00022811
N	0,33702	0,3583	0,000074934	0,015584
O	0,0091664	0,00012005	0,00017618	0,00011119
P	0,00040105	0,0015197	**0,99933**	**0,99661**
R	0,0000061584	0,0000025796	0,000043428	0,00012979
S	0,000000096521	0,011028	0,0037224	0,00016043
T	0,00042299	0,0037111	0,000647	0,021243
W	0,00048361	0,00034636	0,00022182	0,0038488

3. Analiza i dyskusja błędów uczenia i testowania opracowanej sieci.

Analizując przedstawione wyniki w tabeli pierwszej można zauważyć, że wartości współczynnika uczenia mają wpływ na szybkość nauki. Im większa wartość współczynnika tym mniejsza jest liczba epok, a co za tym idzie – czas realizacji. Niezależnie jednak od jego wartości sieć jest w stanie osiągnąć spodziewane rezultaty.
Natomiast w tabeli drugiej gdzie przedstawione są wartości konkretnych liter zauważamy, że sieć bez problemu potrafiła rozpoznać daną literę oraz przyporządkować ją do konkretnej grupy. Możliwe to jest dzięki wybraniu największej wartości ze wszystkich. Analizie poddałam literę M oraz P. Wyraźnie widać, że litery te po zmodyfikowaniu w zestawie testującym wciąż mają wartości prawie takie same jak oryginały. Ponad to z tabeli możemy również wyczytać, która litera jest najbardziej podobna do tej poddanej analizie, np. w przypadku litery ‘P’ jest to ‘A’. Co więcej widzimy też, że wartość litery zmodyfikowanej jest mniejsza od tej oryginalnej.

4. Wnioski.

W celu poznania działania reguły Hebba korzystałam z funkcji dostępnych w pakiecie Matlab. Wybrałam funkcję newff() oraz podczas testowania sieci zauważyłam, że idealnie się nadaje do tego zadania. Podczas analizy wyników zauważamy, że parametry procesu uczenia mają ogromny wpływ na uzyskiwane wyniki. Zmiana współczynnika uczenia wpływa na szybkość procesu uczenia. Jednak jego wartość nie ma wpływu na wynik, który zawsze jest poprawny. W celu przyporządkowania litery do danej grupy, sieć wykorzystuje wartości obliczone dla każdej litery oraz wybiera największą wartość. Największa wartość świadczy o największym prawdopodobieństwie przydziału do danej grupy.

5. Listing całego kodu

```
close all; clear all; clc;

% Zestaw danych uczących składający się z 20 dużych liter.
Przedstawione są jako matryca 5x7 o wartościach 0 lub 1.

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 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 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 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];
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];
W=[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];
A2=[0 1 1 1 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];
A3=[0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1];

% Transpozycja każdej litery

  dA=transpose(A);
  dB=transpose(B);
  dC=transpose(C);
  dD=transpose(D);
  dE=transpose(E);
  dF=transpose(F);
  dG=transpose(G);
  dH=transpose(H);
  dI=transpose(I);
  dJ=transpose(J);
  dK=transpose(K);
  dL=transpose(L);
  dM=transpose(M);
  dN=transpose(N);
  dO=transpose(O);
  dP=transpose(P);
  dR=transpose(R);
  dS=transpose(S);
  dT=transpose(T);
  dW=transpose(W);

 % Wektor wejściowy złożony z dużych liter po transpozycji

dane_we=[  dA(:) dB(:) dC(:) dD(:) dE(:) dF(:) dG(:) dH(:) dI(:) dJ(:) dK(:)
dL(:) dM(:) dN(:) dO(:) dP(:) dR(:) dS(:) dT(:) dW(:)];

 % Dane wyjściowe w postaci tablicy zero-jedynkowej – 1 identyfikuje literę
```

```
%  A B C D E F G H I  J  K L M N O P R S T W
dane_wy= [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; % A
          0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; % B
          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; % C
          0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; % D
          0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; % E
          0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; % F
          0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ; % G
          0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ; % H
          0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 ; % I
          0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 ; % J
          0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 ; % K
          0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 ; % L
          0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 ; % M
          0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ; % N
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 ; % O
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 ; % P
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 ; % R
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ; % S
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 ; % T
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 ; % W
          ];
```

```
% Zestaw testujący

testA=[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;];

 testB=[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; 0;];

 testC=[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;];

 testD=[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;];
```

```
testE=[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;];

testF=[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;];

testG=[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;];

testH=[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;];

testI=[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;];

testJ=[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;];

testK=[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;];

testL=[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;];

testM=[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;];

testN=[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;];

testO=[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;];

testP=[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;];

testR=[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;];

testS=[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;];

testT=[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;];

testW=[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;];
```

%Zmodyfikowany zestaw testujący

```
testP2=[1; 0; 1; 1; 0;
        1; 0; 0; 0; 1;
        1; 0; 0; 0; 1;
        1; 1; 1; 0; 0;
        0; 0; 0; 0; 0;
        1; 0; 0; 0; 0;
        1; 0; 0; 0; 0;];

testM2=[1; 0; 0; 0; 1;
        0; 1; 0; 1; 1;
        1; 0; 1; 0; 0;
        1; 0; 0; 0; 1;
        0; 0; 0; 0; 1;
        1; 0; 0; 0; 1;
        1; 0; 0; 0; 0;];
```

Tworzenie sieci za pomocą funkcji newff()

```
net=newff( minmax(dane_we),  20, {'logsig',  'learnh', 'trainr'});
      % - minmax(dane_we) – spodziewane zakresy dla elementów wejściowych
      % - 20 – liczba neuronów w warstwie
      % - logsig – funkcja aktywacji neuronów
      % - learnh – funkcja trenowania sieci
      % - trainr – algorytm trenowania sieci

% Określenie długości treningu
net.trainParam.epochs = 100;
% Określenie wartości stopu
net.trainParam.goal = 0.001;
% Określenie współczynnika uczenia
lp.lr = 0.1;
% Obliczenie zmiany wagi
dw=learnh([],dane_we,[],[],dane_wy,[],[],[],[],[],lp,[]);
% Ustalenie wagi dla wejścia
net.inputWeights{:,:}.learnFcn = 'learnh';
% Ustalenie wagi dla warstwy
net.layerWeights {:,:} .learnFcn = 'learnh';
% Trening sieci
net=train(net, dane_we,dane_wy);

% Symulacja sieci dla konkretnego testu
rezultat=sim(net, testP2);

% Wyszukiwanie największej wartości
max=1;
for i=1:1:20;
    if(rezultat(max)<rezultat(i))
        max=i;
    end;
end
```

```
% Wypisanie wartości dla każdej litery
disp('Wartosci wyjsciowe dla wszytskich liter:')
disp('A='),disp(rezultat(1));
disp('B='),disp(rezultat(2));
disp('C='),disp(rezultat(3));
disp('D='),disp(rezultat(4));
disp('E='),disp(rezultat(5));
disp('F='),disp(rezultat(6));
disp('G='),disp(rezultat(7));
disp('H='),disp(rezultat(8));
disp('I='),disp(rezultat(9));
disp('J='),disp(rezultat(10));
disp('K='),disp(rezultat(11));
disp('L='),disp(rezultat(12));
disp('M='),disp(rezultat(13));
disp('N='),disp(rezultat(14));
disp('O='),disp(rezultat(15));
disp('P='),disp(rezultat(16));
disp('R='),disp(rezultat(17));
disp('S='),disp(rezultat(18));
disp('T='),disp(rezultat(19));
disp('W='),disp(rezultat(20));


switch max
    case 1
       test=testA;    % Przypisanie zmiennej test konkretnego wektora testującego
       disp('Wpisana litera to A') % Wypisanie komunikatu
    case 2
       test=testB;
        disp('Wpisana litera to B')
    case 3
       test=testC;
        disp('Wpisana litera to C')
    case 4
       test=testD;
        disp('Wpisana litera to D')
    case 5
       test=testE;
       disp('Wpisana litera to E')
    case 6
       test=testF;
       disp('Wpisana litera to F')
     case 7
       test=testG;
       disp('Wpisana litera to G')
    case 8
       test=testH;
       disp('Wpisana litera to H')
    case 9
       test=testI;
       disp('Wpisana litera to I')
    case 10
       test=testJ;
       disp('Wpisana litera to D')
    case 11
       test=testK;
       disp('Wpisana litera to K')
    case 12
       test=testL;
       disp('Wpisana litera to L')
    case 13
       test=testM;
```

```
        disp('Wpisana litera to M')
    case 14
        test=testN;
        disp('Wpisana litera to N')
    case 15
        test=testO;
        disp('Wpisana litera to O')
    case 16
        test=testP;
        disp('Wpisana litera to P')
    case 17
        test=testR;
        disp('Wpisana litera to R')
    case 18
        test=testS;
        disp('Wpisana litera to S')
    case 19
        test=testT;
        disp('Wpisana litera to T')
    case 20
        test=testW;
        disp('Wpisana litera to W')
    otherwise
         disp('nie ma takiej litery')
end


 % Wyświetlenie
      plotchar(testP2)
```