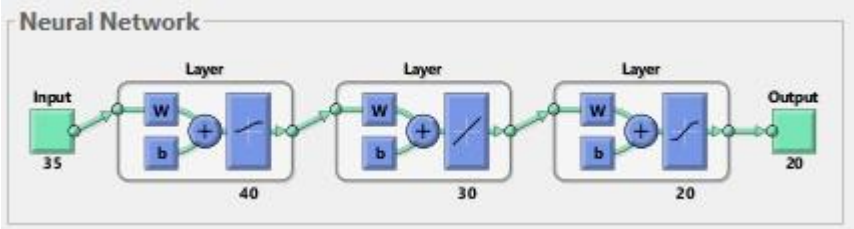
Fabiola Dąbroś , IO gr.1 rok III

Sprawozdanie z zagadnienia nr 3.

Temat ćwiczenia:
Budowa i działanie sieci wielowarstwowej.
Cel ćwiczenia:
Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie z użyciem algorytmu wstecznej propagacji błędu rozpoznawania konkretnych liter alfabetu.

1. Opis budowy użytej sieci oraz wykorzystanego algorytmu uczenia.

Do wykonania tego ćwiczenia korzystałam z sieci wielowarstwowej. Ten typ sieci cieszy się największym zainteresowaniem. Jej cechą charakterystyczną jest występowanie co najmniej jednej warstwy ukrytej neuronów, pośredniczącej w przekazywaniu sygnałów pomiędzy wejściami a wyjściami sieci. Możliwości neuronów gwałtownie wzrastają w wyniku jednoczesnej pracy wielu neuronów połączonych w sieci tworzące strukturę wielowarstwową. W sieciach jednokierunkowych przepływ sygnału przebiega zawsze w ściśle określonym kierunku, czyli od warstwy wejściowej do warstwy wyjściowej.
W klasycznej terminologii sieci neuronowych wyróżnia się warstwę wejściową, jedną lub dwie warstwy ukryte oraz warstwę wyjściową. Warstwa wejściowa pobiera dane z otoczenia i przesyła je do pierwszej warstwy ukrytej. Jedyną funkcją warstwy wejściowej jest przesyłanie i rozprowadzanie sygnałów do pierwszej warstwy ukrytej. Jej neurony nie podlegają procesowi uczenia, nie posiadają wag, w których mogłaby być gromadzona wiedza o przybliżanych zależnościach i nie biorą udziału w procesie generalizacji. Następnie sygnał przesyłany jest na wejścia pierwszej warstwy ukrytej, która przetwarza dane i generuje sygnał wyjściowy podawany na wejścia warstwy kolejnej. Powyższy schemat powtarza się dla wszystkich kolejnych warstw ukrytych i kończy na warstwie wyjściowej, która zgodnie ze wzorcową architekturą oblicza wartości wyjść całej sieci i przekazuje je na zewnątrz.
Warstwy ukryte oraz warstwa wyjściowa składają się z tego samego rodzaju neuronów, podlegających procesowi uczenia i na wzór komórki nerwowej, gromadzących wiedzę o przybliżanych zależnościach w wektorze wag.
Do wykonania ćwiczenia wykorzystałam gotowe narzędzia z pakietu Matlab.



- Funkcja newff()

Służy ona do tworzenia wielowarstwowej sieci neuronowej złożonej z neuronów o nieliniowej funkcji aktywacji. Jednak funkcje aktywacji w poszczególnych warstwach mogą mieć również postać liniową. Na wejściu ustalamy liczbę wejść sieci oraz liczbę neuronów sieci w każdej z warstw. Ponadto mamy funkcję aktywacji neuronów oraz funkcję służącą do trenowania sieci. Natomiast na wyjściu tworzy się struktura zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry sieci. Opisana funkcja wygląda następująco:

```
S=[40 30 20];
net = newff(minmax(dane_we),S,{'logsig','purelin', 'tansig'},'traingda');
```

gdzie:

- minmax(dane_we) - spodziewane zakresy dla elementów wejściowych
- S - liczba neuronów sieci w każdej z warstw
- 'logsig' 'purelin' 'tansig' – nazwa funkcji aktywacji neuronów
- 'traingda' – nazwa funkcji trenowania sieci

Suma ważonych danych wejściowych i stronniczości tworzy dane wejściowe do funkcji aktywacji *f* . Neurony mogą wykorzystywać dowolną zróżnicowaną funkcję transferu *f,* aby wygenerować swoją wydajność. Sieci wielowarstwowe często korzystają z funkcji transferu 'logsig'. Ta generuje wyjścia od 0 do 1, ponieważ wejście sieci neuronowej przechodzi od ujemnej do dodatniej nieskończoności. Alternatywnie, sieci wielowarstwowe mogą wykorzystywać funkcję transferu 'tansig'. Czasami również funkcja liniowego transferu 'purelin' jest używana w sieciach propagacji wstecznej. Jeśli ostatnia warstwa sieci wielowarstwowej ma sigmoidalne neurony, to wyjścia sieci są ograniczone do małego zakresu. Jeśli liniowe neurony wyjściowe są używane, wyjścia sieciowe mogą przyjąć dowolną wartość.

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.

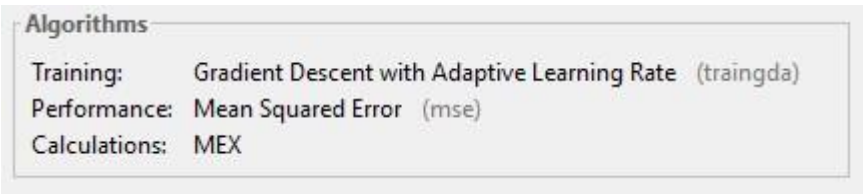
```
dane_wy= [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;
          0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;
          0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;
          0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;
          0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;
          0 0 0 0 0 1 0 0 0 0 0 0 0 0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

Ponadto dla funkcji newff() wywołane zostały następujące procedury:

```
net.trainParam.epochs = 10000;
net.trainParam.goal = 0.001;
net.trainParam.mu=0.01;
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 ‘mu’ określamy wartość współczynnika uczenia sieci. Na końcu wywołujemy funkcję ‘train’ dokonującą treningu sieci. Jej działanie dla sieci jednokierunkowych polega na obliczaniu metodą iteracyjną wartości współczynników wagowych.

- Algorytm traingda



Funkcja newff() opiera swoje działanie na algorytmie traingda. Jest to funkcja szkolenia sieciowego, która opiera się na metodzie propagacji wstecznej błędu z adaptacyjną zmianą

stałej szybkości uczenia. Pozwala ona matematycznie wyznaczyć błąd popełniany przez neurony warstw ukrytych - na podstawie błędu warstwy wyjściowej - i wykorzystanie go do korekty wag neuronów tychże warstw. Jest to efektywne wykorzystanie reguł uczenia nadzorowanego do treningu sieci wielowarstwowych.

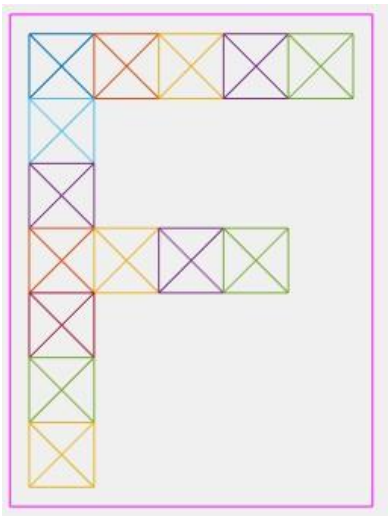
Algorytm wstecznej propagacji błędu składa się z następujących kroków:

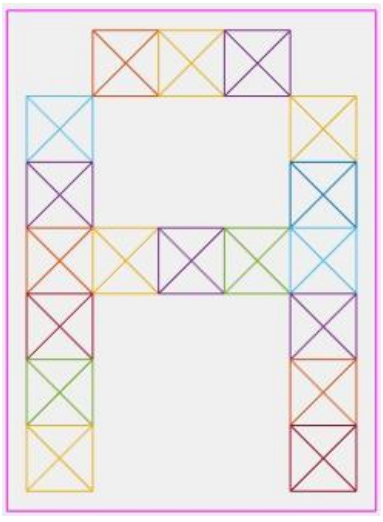
1. Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
2. Inicjujemy wagi losowo (na małe wartości).
3. Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po warstwie).
4. Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią *y* oraz poprawną odpowiedzią *t*.
5. Błędy propagowane są do wcześniejszych warstw.
6. Każdy neuron (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkości przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3. dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
8. Zatrzymujemy się, gdy średni błąd na danych treningowych przestanie maleć.

Po treningu sieci utworzony został zestaw testujący w celu sprawdzenia wyniku uczenia dla każdej litery.

```
rezultat=sim(net, test);
rezultat=compet(rezultat);
answer=find(compet(rezultat)==1);
figure;
plotchar(dane_we(:, answer))
```

Poprzez operację ‘sim’ wykonuje się symulacja sieci dla każdej litery wpisanej z klawiatury. Funkcja ‘find’ umożliwia znalezienie w tablicy wyjść tej części, która odpowiada wpisanej z klawiatury literze poprzez poszukiwanie wartości równej 1 oraz identyfikację litery, której odpowiada to wyjście. Kolejno wyświetla się podana litera dzięki funkcji ‘plotchar’.





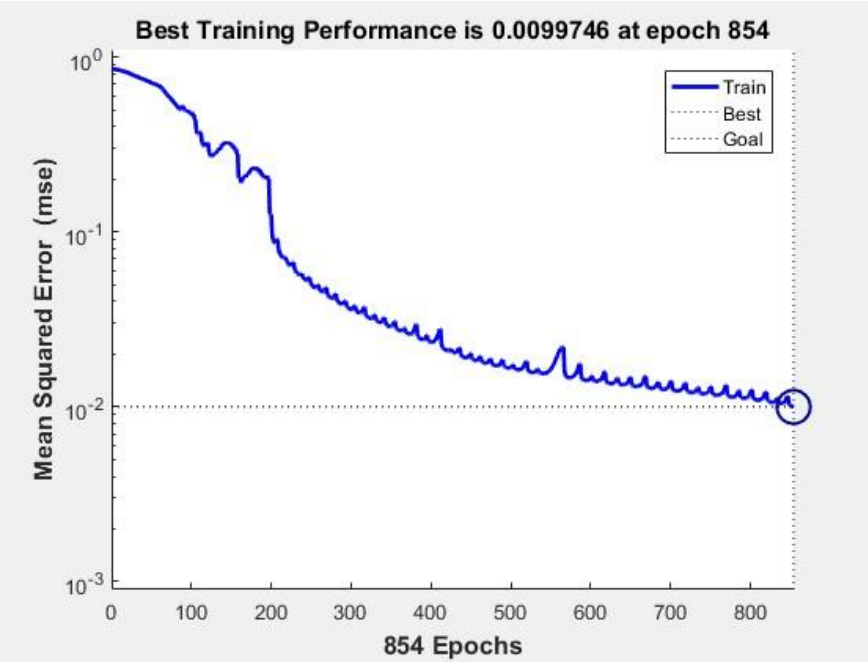
2. Zestawienie otrzymanych wyników.

Tabela wyników uczenia dla testowania różnej ilości danych uczących prezentuje się następująco:

Ilość danych uczących	Liczba epok	Czas realizacji
20	963	0:00:03
15	756	0:00:03
8	521	0:00:02
25	1453	0:00:03

Tabela wyników uczenia dla testowania różnej wartości współczynnika uczenia oraz błędu prezentuje się następująco:

Współczynnik uczenia	Błąd	Liczba epok	Wynik
0.001	**0.001**	3678	poprawny
0.01	**0.1**	181	niepoprawny
0.001	**0.01**	984	poprawny
0.000001	**0.1**	369	niepoprawny
1	**0.01**	875	poprawny



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

Analizując przedstawione powyżej wyniki w tabelach można zauważyć, że ilość danych uczących ma wpływ na ilość iteracji podczas nauki. Liczba epok dla funkcji newff() wzrasta wraz z powiększaniem zestawu uczącego. Natomiast różnica w ilości danych jest tak znikoma, że nie wpływa ona znacząco na czas nauki. W każdym z przykładów trwa on prawie tyle samo. Nie możemy jednak całkowicie opierać się na tych danych, ponieważ litery wchodzące w skład zestawu uczącego mają ogromne znaczenie. Pomniejszając zestaw o litery podobne do siebie jak np. 'O' czy 'D' wyniki będą inne niż w przypadku liter całkiem różnych od siebie jak 'B' czy „W'. Oczywistym jest, że sieć potrzebuje więcej czasu na naukę liter o podobnej strukturze.
Biorąc pod uwagę wartości współczynnika uczenia oraz błędu przedstawione w tabeli drugiej można zauważyć, że im większa dokładność błędu tym lepiej. Gdy mamy małą dokładność błędu to pomimo zwiększania lub zmniejszania współczynnika uczenia wynik nie jest poprawny. Co więcej zauważamy, że ma to wpływ również na liczbę epok. Gdy błąd ma dużą dokładność liczba iteracji jest większa i wtedy osiągamy satysfakcjonujący wynik. W sytuacji, gdy błąd nie jest zbyt dokładny, współczynnik uczenia musi być odpowiednio większy aby osiągnąć poprawny wynik. Jednak już przy minimalnie dokładniejszym błędzie wynik jest poprawny, a współczynnik uczenia wpływa tylko na szybkość nauki. Im jest większy tym wykonuje się mniej epok a co za tym idzie – czas jest krótszy. Dodatkowo wraz ze wzrostem ilości warstw ukrytych proces uczenia trwa dłużej.

4. Wnioski.

W celu uczenia rozpoznawania konkretnych liter alfabetu korzystałam z funkcji dostępnych w pakiecie Matlab. Wybrałam sieć wielowarstwową oraz podczas testowania sieci zauważyłam, że funkcja newff() idealnie się nadaje do tego zadania. Podczas analizy wyników zauważamy, że parametry procesu uczenia mają ogromny wpływ na uzyskiwane wyniki. Zwiększanie zestawu uczącego jak i sam jego skład wpływa na liczbę epok. Natomiast zmiana współczynnika uczenia oraz błąd wpływają na wynik końcowy. Jeśli błąd będzie mało dokładny nie osiągniemy poprawnego rozwiązania. Ponad to modyfikacja współczynnika uczenia pozwala osiągnąć satysfakcjonujące wyniki tylko do pewnego momentu. Zbyt duże zmniejszanie współczynnika nie poprawi znacząco efektywności, a jedynie przyczyni się do znacznego wydłużenia się procesu uczenia.
Wydajność algorytmu jest bardzo wrażliwa na właściwe ustawienie szybkości uczenia się. Jeśli tempo uczenia się jest ustawione zbyt wysoko, algorytm może oscylować i stać się niestabilny. Jeśli szybkość uczenia się jest zbyt mała, algorytm zajmuje zbyt wiele czasu, aby się zjednoczyć.

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 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];
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];
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];

% 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(:)];
```

```
% Tablica zero-jedynkowa – 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;];

   % Tworzenie sieci za pomocą funkcji newff()

S=[40 30 20];
net = newff(minmax(dane_we),S,{'logsig','purelin','tansig'},'traingda');
      % - minmax(dane_we) – spodziewane zakresy dla elementów wejściowych
      % - S – liczba neuronów sieci w każdej z warstw
      % - logsig, purelin, tansig – funkcja aktywacji neuronów
      % - traingda – funkcja trenowania sieci perceptronowej

% Określenie długości treningu
net.trainParam.epochs = 10000;
% Określenie wartości stopu
net.trainParam.goal = 0.01;
% Określenie współczynnika uczenia
net.trainParam.mu = 0.1;
% Trening sieci
net=train(net, dane_we, dane_wy);

% Wczytanie litery z klawiatury
litera=input('Wprowadz litere do rozpoznania', 's');

% Sprawdzanie wyniku uczenia dla każdej litery
switch litera
    case 'A'
       test=testA;          % Przypisanie zmiennej test konkretnego wektora testującego
       pause(2);
       disp('Wpisana litera to A')       % Wypisanie komunikatu
    case 'B'
       test=testB;
       pause(2);
       disp('Wpisana litera to B')
    case 'C'
       test=testC;
       pause(2);
       disp('Wpisana litera to C')
    case 'D'
       test=testD;
       pause(2);
       disp('Wpisana litera to D')
    case 'E'
       test=testE;
       pause(2);
```

```
   disp('Wpisana litera to E')
case 'F'
   test=testF;
   pause(2);
   disp('Wpisana litera to F')
 case 'G'
   test=testG;
   pause(2);
   disp('Wpisana litera to G')
case 'H'
   test=testH;
   pause(2);
   disp('Wpisana litera to H')
case 'I'
   test=testI;
   pause(2);
   disp('Wpisana litera to I')
case 'J'
   test=testJ;
   pause(2);
   disp('Wpisana litera to J')
case 'K'
   test=testK;
   pause(2);
   disp('Wpisana litera to K')
case 'L'
   test=testL;
   pause(2);
   disp('Wpisana litera to L')
case 'M'
   test=testM;
   pause(2);
   disp('Wpisana litera to M')
case 'N'
   test=testN;
   pause(2);
   disp('Wpisana litera to N')
case 'O'
   test=testO;
   pause(2);
   disp('Wpisana litera to O')
case 'P'
   test=testP;
   pause(2);
   disp('Wpisana litera to P')
    case 'R'
   test=testR;
   pause(2);
   disp('Wpisana litera to R')
case 'S'
   test=testS;
   pause(2);
   disp('Wpisana litera to S')
case 'T'
   test=testT;
   pause(2);
   disp('Wpisana litera to T')
case 'W'
   test=testW;
   pause(2);
   disp('Wpisana litera to W')
```

```
    otherwise
        disp('nie ma takiej litery')
    end

% Symulacja sieci dla konkretnej litery wpisanej z klawiatury
    rezultat=sim(net, test);
    rezultat=compet(rezultat);
% Szukanie odpowiedniej litery
    answer=find(compet(rezultat)==1);
% Wyświetlenie litery
    figure;
    plotchar(dane_we(:, answer))

% Obliczenie wydajności sieci
    e = dane_wy*rezultat;
    wydajnosc = msereg (e, net)
```