Marcelina Bałamut, IO, gr. 1

Temat ćwiczenia:

Budowa i działanie sieci wielowarstwowej.

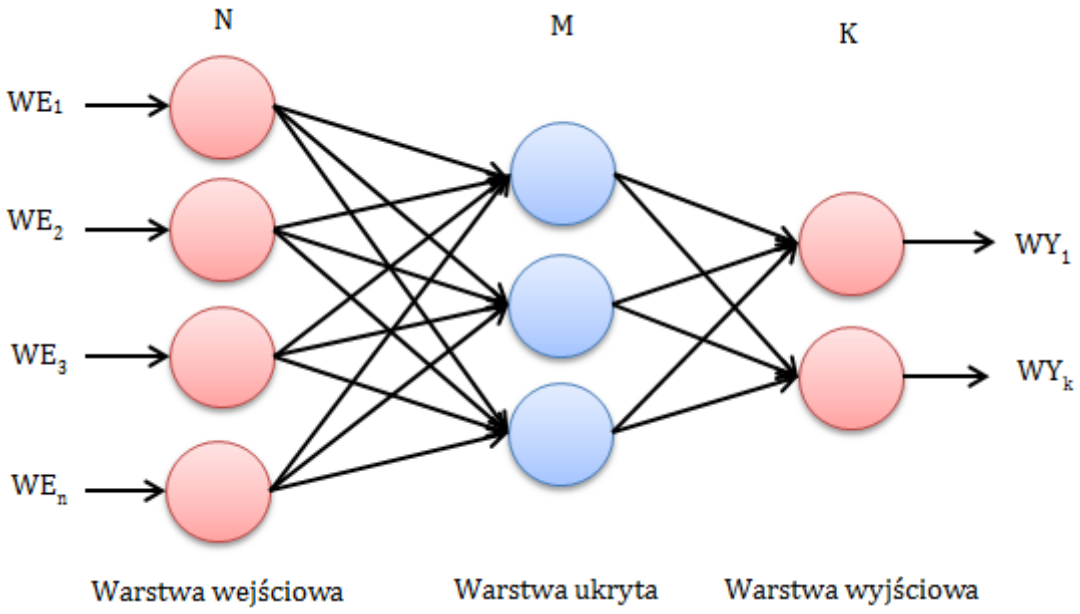
Cel ćwiczenia

Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez implementację oraz uczenie z użyciem algorytmu wstecznej propagacji błędu rozpoznawania konkretnych liter alfabetu.

Opis budowy oraz wykorzystanych sieci i algorytmów uczenia:

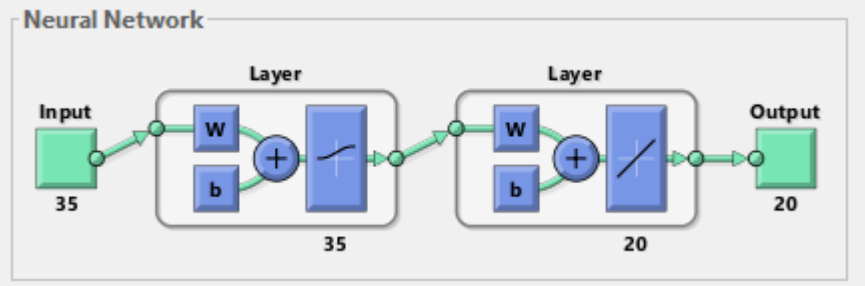
Możliwości pojedynczych sztucznych neuronów są bardzo ograniczone, jednak poprzez połączenie sztucznych neuronów w sieci ich możliwości wzrastają i ujawnia się rzeczywista moc obliczeń neuronowych. Jest ona wynikiem jednoczesnej pracy wielu neuronów połączonych w sieci.

Sieci jednokierunkowe wielowarstwowe słyną z prostej struktury dlatego są tak powszechnie stosowane, ale też dlatego że ich budowa przypomina strukturę mózgu.



Jak widać na powyższym rysunku sieci wielowarstwowe składają się z uporządkowanych warstw neuronów- warstwa wejściowa, wyjściowa i warstwy ukryte. Liczba neuronów w każdej z warstw może być różna, przy czym każda z nich musi posiadać tę samą funkcję przejścia, natomiast neurony z różnych warstw mogą mieć inną funkcję przejścia. Połączenia występują jedynie między neuronami z sąsiednich warstw i są to połączenia asymetryczne na zasadzie „każdy z każdym". Sygnały przesyłane są od warstwy wejściowej do warstwy wyjściowej przez warstwy ukryte. Zadaniem warstwy wejściowej jest wstępna obróbka sygnału, neurony z warstw ukrytych są odpowiedzialne za przetwarzanie decyzyjne, natomiast za udzielanie odpowiedzi odpowiedzialna jest warstwa wyjściowa. Bardzo ważnym aspektem są funkcje aktywacji. Najczęściej wykorzystuje się funkcje typu sigmoidalnego. Natomiast typ funkcji aktywacji neuronów warstwy wyjściowej zależy od przeznaczenia sieci. Zbudowanie sieci wielowarstwowej w której neurony każdej warstwy posiadają liniowe funkcje przejścia jest bezcelowe ponieważ taką sieć wielowarstwową można zastąpić siecią jednowarstwową.

Działanie perceptronu opiera się na klasyfikowaniu danych na wejściu i ustawianiu do tego wartości wyjścia. W wykonywanym zadaniu jako dane uczące zostało wykorzystane 20 dużych liter alfabetu przedstawionych w wersji binarnej za pomocą cyfr 1 lub 0 jako tablica o wymiarze 5x7 (35 wejść). Natomiast wektor wyjściowy składa się również z tablicy gdzie zawarta jest jedna 1 i dziewiętnaście 0, co kwalifikuje konkretna literę. Do implementacji danego problemu użyłam wbudowanej funkcji pakietu Matlab o nazwie newff().

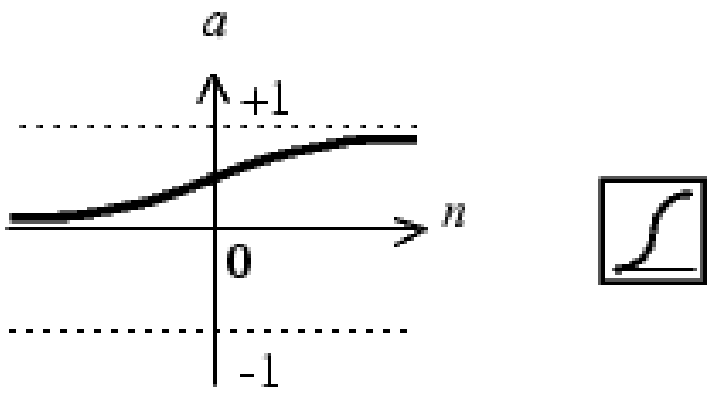


Funkcja newff() tworzy wielowarstwową sieć neuronową gdzie każda warstwa składa się z zadanej liczby neuronów o nieliniowych i liniowych funkcjach aktywacji. Funkcja ta tworzy kaskadową jednokierunkową sieć wielowarstwową uczoną metodą propagacji wstecznej błędu. Po wywołaniu funkcji tworzony jest w przestrzeni obiekt net w którym zapisane są wszystkie informacje na temat utworzonej sieci.

```
L=[35 20];
net = newff(minmax(wejscie),L,{'logsig','purelin'},'traingda');
```

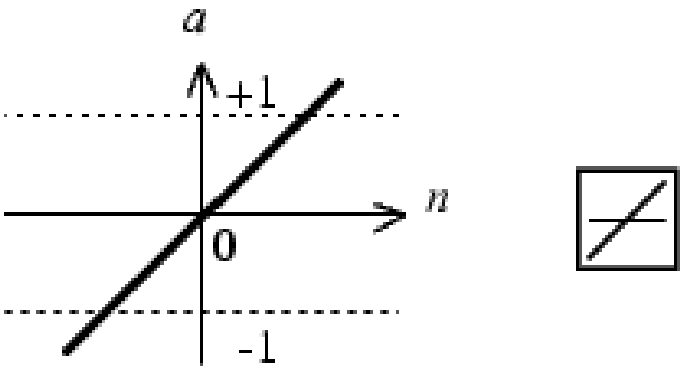
gdzie:
minmax(wejscie) - spodziewane zakresy poszczególnych wejść perceptronu,
L- liczba neuronów w warstwie ukrytej i wyjściowej
logsig, purelin –funkcje aktywacji
traingd-funkcja uczenia

Newff wykorzystuje w zadanym problemie dwie funkcje aktywacji- liniową i nieliniową. Parametr 'logsig' oznacza że warstwa ukryta wykorzystuje nieliniową funkcję aktywacji – funkcję logarytmiczno sigmoidalną unipolarną (niesymetryczna) przyjmującą wartości 0 ,1. Następuje więc klasyfikacja danych wejściowych na takie dla których stan wyjścia równa się 0 -1.



$$a = logsig(n)$$

Natomiast druga funkcja aktywacji – „purelin" jest wykorzystywana przez warstwę wyjściową i jest to funkcja liniowa.



$$a = purelin(n)$$

Funkcja działa na danych wejściowych zobrazowanych jako wektor składający się z przedstawienia binarnego każdej litery.

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

Natomiast wyjście jest to tablica zero-jedynkowa, gdzie 1 wpisane w odpowiednim miejscu identyfikuje literę.

```
wyjscie= [1 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 1 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 1 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 1 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 1 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 1 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 1 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 1 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 1 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 1 ;
          ];
```

Dla funkcji newff() 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, wektor wejścia i wyjścia. Przed wywołaniem ustalona została docelowa wartość błędu uczenia(goal), maksymalna liczba kroków uczących(epochs) oraz wartość współczynnika uczenia(mu).

```
net.trainParam.epochs = 10000;
net.trainParam.goal =0.01;
net.trainParam.mu=1;
net=train(net, wejscie, wyjscie);
```

Do uczenia sieci została wykorzystana metoda wstecznej propagacji błędu wraz z adaptacyjną zmianą współczynnika szybkości uczenia, która w pakiecie Matlab oznaczana jest poprzez „traingda" i zostaje przesyłana jako parametr podczas tworzenia obiektu net. 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.

Cykl uczenia metodą wstecznej propagacji błędu (backpropagation) składa się z następujących etapów:

1. Ustalenie liczby warstw i neuronów i podanie wektora uczącego $\mathbf{u}^m$ na wejście sieci.
2. Wyznaczenie wartości wyjść u_j^{mm} każdego elementu dla kolejnych warstw przetwarzania neuronowego, od pierwszej warstwy ukrytej do warstwy wyjściowej.
3. Obliczenie wartości błędów d_j^{Mm} dla warstwy wyjściowej.
4. Obliczenie wartość sumy kwadratów błędów x_m.
5. Dokonanie propagacji wstecznej błędu wyjściowego d_j^{Mm} do poszczególnych elementów warstw ukrytych wyznaczając d_j^{mm} według wzoru:

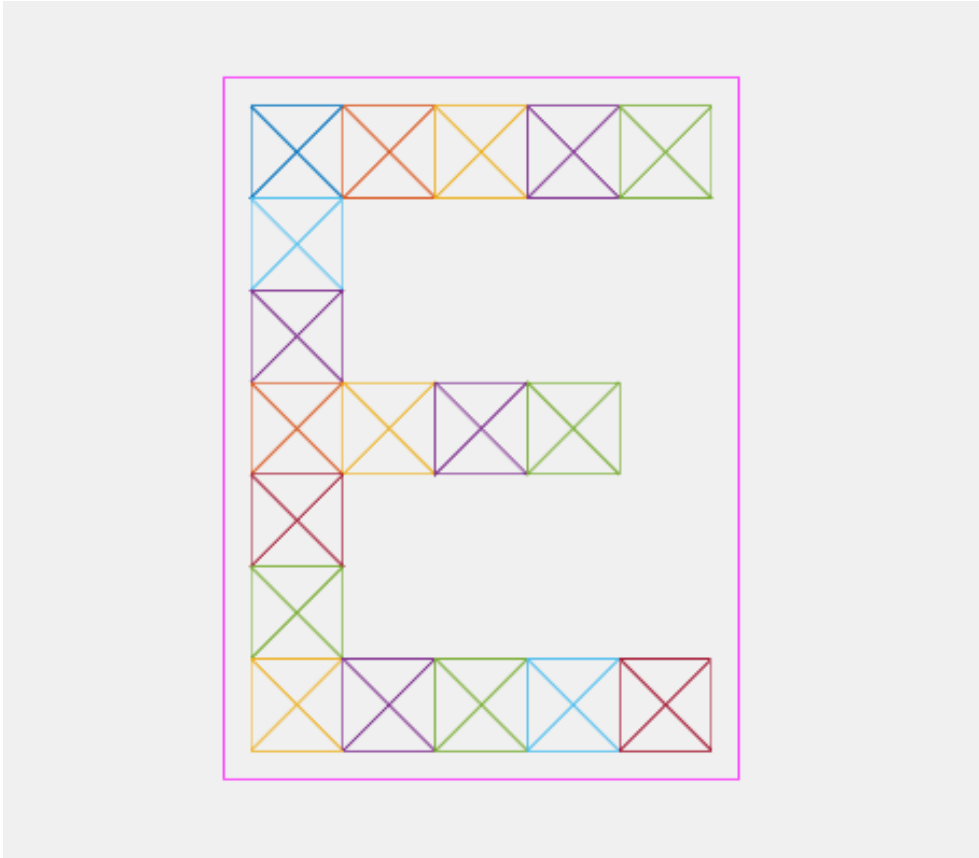
$$\delta_j^{m\mu} = f'(\varphi_j^{m\mu}) \sum_{l=1}^{n_{m+1}} \delta_l^{(m+1)\mu} w_{lj}^{(m+1)}$$

6. Dokonanie aktualizacji wag kolejno pomiędzy warstwą wyjściową i ukrytą, a następnie pomiędzy warstwami ukrytymi przesuwając się w kierunku warstwy wejściowej.

Następnie został stworzony zestaw testujący, dzięki któremu możliwe stało się sprawdzenie wyniku uczenia dla każdej litery. Aby to sprawdzić należy z klawiatury wprowadzić wybrana literę i dzięki mechanizmowi switch…case zostaje wywołany odpowiedni fragment kodu.

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

Poprzez procedurę ‘sim’ wykonuje się symulacja sieci dla konkretnej litery wpisanej z klawiatury. Następnie poprzez funkcję find znajdowana jest odpowiednia część tablicy wyjść, która odpowiada wpisanej przez użytkownika literze. Działa to na zasadzie poszukiwania wartości wynoszącej 1 i na tej podstawie identyfikacja litery której odpowiada to wyjście. Poprzez wywołanie mechanizmu figure i plotchar wyświetlana jest narysowana figura (litera), która została pozyskana w trakcie uczenia sieci. Natomiast jeśli zostanie podana mała litera lub inny znak, którego sieć się nie nauczyła zostaje wyświetlony odpowiedni komunikat.



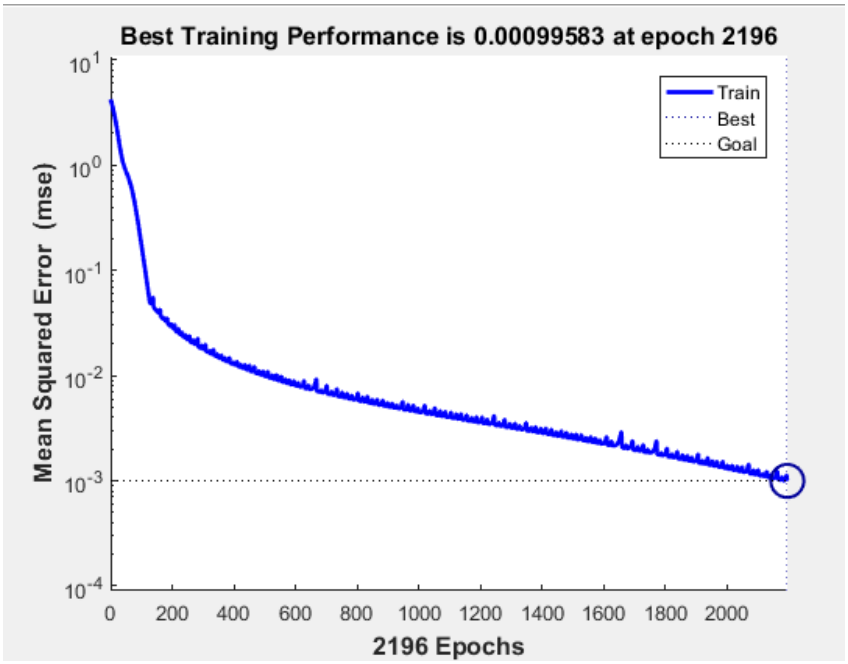
Zestawienie otrzymanych wyników

Zależność liczby epok i czasu od liczby danych:

Liczba danych	Liczba epok	Czas
20	2264	0.00:09
17	1439	0.00:06
12	987	0.00:05
8	902	0.00:05
4	523	0.00:04
2	125	0.00:03

Zależność wartości współczynnika uczenia oraz błędu na wynik i liczbę epok.

Współczynnik uczenia	Błąd	Liczba epok	Wynik
0.0001	0.0001	2217	poprawny
0.1	0.01	1352	poprawny
0.01	0.1	114	niepoprawny
0.0000001	0.1	533	niepoprawny
1	0.01	1316	poprawny



Analiza i dyskusja błędów uczenia i testowania opracowanego perceptronu w zależności od wartości współczynnika uczenia oraz liczby danych uczących.

Jak widać na załączonych wyżej tabelach oraz wykresach po wykonaniu testów działania sieci tworzonej funkcją newff wynika, że zmiana poszczególnych czynników ma wypływ na parametry sieci. Jednak podczas wywołań programu dla różnej wartości współczynnika uczenia liczba epok oraz czas była bardzo zróżnicowana. Nie można więc wyciągnąć z tego żadnej zależności. Po analizie wpływu ilości danych na proces uczenia przedstawionych w tabelach widać, że wraz ze wzrostem ilości danych liczba epok rośnie, tak samo dzieje się też z czasem. Jednak nie są to do końca miarodajne wyniki, ponieważ wszystko zależy od danych jakie zostały wprowadzone. Jeśli wprowadzone litery maja podobną strukturę to czas oraz liczba epok jest większa niż jeśli ich przedstawienie jest całkiem różne od siebie. Z tabeli przedstawiającej wpływ błędu i współczynnika uczenia na wynik oraz liczbę epok można stwierdzić, iż wartość błędu znacząco wpływa na wartości wynikowe, natomiast wartość współczynnika już niespecjalnie. Niezależnie od jego wartości sieć jest w stanie się nauczyć. Natomiast ma duży wpływ idąc w parze z błędem. Aby wynik był poprawny współczynnik uczenia musi być odpowiednio większy od błędu kiedy ten jest duży. Wraz ze wzrostem liczby warstw ukrytych liczba epok oraz czas nauki wydłuża się.

Wnioski

Sieć wielowarstwowa to sieć zbudowana z wielu warstw. W tym przypadku sieć posiada wejście, jedną warstwę ukrytą z 35 neuronami oraz warstwę wyjściową z 20 neuronami. Sygnały przesyłane są od warstwy wejściowej do warstwy wyjściowej przez warstwy ukryte. Sieci te mogą zawierać różne funkcje aktywacji, a algorytmem który zajmuje się nauką sieci jest metoda wstecznej propagacji błędu. Wykorzystując funkcje wbudowane do programu Matlab udało mi się przeprowadzić stosowne testy oraz na ich podstawie przeprowadzić analizę z której wynika wiele wniosków. Pierwszym jest fakt, iż funkcja newff bardzo dobrze nadaje się do rozwiązania problemu rozpoznawania konkretnych liter. Analizując wyniki działania funkcji można stwierdzić, iż ustawiane parametry mają znaczący wpływ na działanie sieci, a w szczególności na proces uczenia. Wraz ze wzrostem ilości danych czas nauki nieznacznie się wydłuża, za to liczba epok rośnie proporcjonalnie do liczby liter w zestawie uczącym. Jednak zależy to głównie od liter zawartych w zestawie uczącym. Natomiast wynik końcowy zależy znacząco od doboru parametru procesu uczenia, takiego jak

błąd, który musi być odpowiednio dopasowany, lub dobrany do współczynnika uczenia. Ważnym wnioskiem jest fakt, że liczba warstw wpływa na proces uczenia.

Listing kodu wykonanego programu

```
close all; clear all; clc;
  %% dane wejściowe: 20 dużych liter zapisanych 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 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];

% transponowanie każdej wektora 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 wejściowy składający się z transponowanych liter

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

```
% wektor wyjściowy, identyfikujący literę po miejscu wystąpienia 1
wyjscie=  [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 ; %Q
           0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ; %R
           0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 ; %S
           0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 ; %T
           ];

    % zestaw testujący dla każdej litery

 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;];
testQ=[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;];
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;];

            % ustalenie liczby neuronów w warstwach
L=[35 20];  % 35 w warstwie ukrytej, 20 w warstwie wyjściowej

%% tworzenie sieci za pomocą funkcji newff
net = newff(minmax(wejscie),L,{'logsig','purelin'},'traingda');
% minmax(wejscie) - spodziewane zakresy poszczególnych wejść perceptronu,
% L- liczba neuronów w warstwie ukrytej i wyjściowej
% logsig, purelin –funkcje aktywacji
% traingd-funkcja uczenia
```

```
net.trainParam.epochs = 10000;                  % maksymalna liczba epok
net.trainParam.goal =0.001;                     % błąd uczenia
net.trainParam.mu=1;                            % współczynnik uczenia
net=train(net, wejscie, wyjscie);               % trening sieci



 % sprawdzanie wyniku uczenia dla każdej litery
litera=input('podaj litere do rozpoznania', 's'); % wprowadzenie litery z
                                                    klawiatury

switch litera
    case 'A'
       test=testA;                  % przypisane 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 'Q'
      test=testQ;
      pause(2);
      disp('Wpisana litera to Q')
   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')

   otherwise
       disp('nie ma takiej litery')
   end

% symulacja sieci dla każdej litery
      pokaz=sim(net, test);
      pokaz=compet(pokaz);
% poszukiwanie litery za pomocą gotowej funkcji odnajdującej wartość 1 w
% tablicy wyjsciowej
      answer=find(compet(pokaz)==1);
% rysowanie nauczonej litery
      figure;
      plotchar(wejscie(:, answer))

 % wydajność sieci
  e = wyjscie*pokaz;
  wydajnosc = msereg (e, net)
```