Marcelina Bałamut, IO, gr. 1

Temat ćwiczenia:

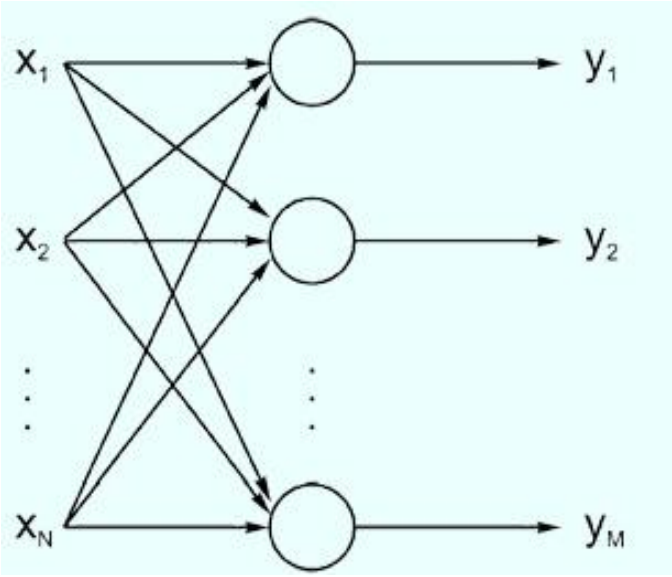
Uczenie sieci regułą Hebba.

Cel ćwiczenia

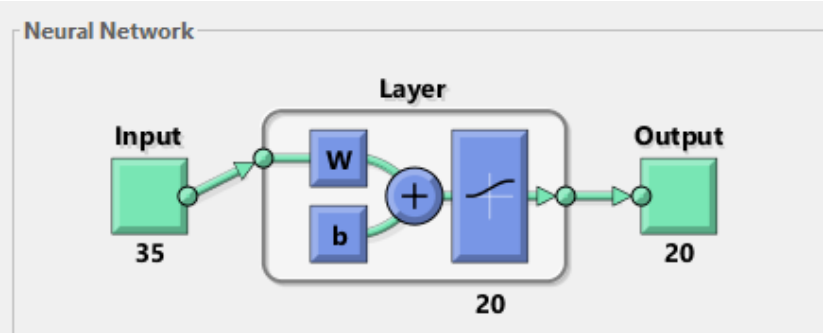
Celem ćwiczenia jest poznanie działania reguły Hebba dla sieci jednowarstwowej na przykładzie grupowania liter alfabetu.

Opis budowy oraz wykorzystanych sieci i algorytmów uczenia:

Sieć jednowarstwowa jest siecią jednokierunkową, gdzie przepływ sygnałów odbywa się w jednym kierunku, od wejścia do wyjścia. Neurony ułożone są w jeden warstwie zasilanej z węzłów wejściowych w których nie zachodzi żaden proces obliczeniowy, więc nie tworzą one warstwy neuronów. Ponadto w sieci jednowarstwowej każdy węzeł jest połączony z każdym neuronem. Działanie perceptronu opiera się na klasyfikowaniu danych na wejściu i ustawianiu do tego wartości wyjścia. Sieć jednowarstwowa jak sama nazwa wskazuje składa się z jednej warstwy o ustalonej funkcji aktywacji. Przed używaniem perceptron należy wytrenować, podając mu przykładowe dane na wejście i modyfikując w odpowiedni sposób wagi wejść i połączeń między warstwami neuronów, tak aby wartość na wyjściu przybierała pożądane wartości. Perceptrony mogą klasyfikować dane na zbiory.



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().

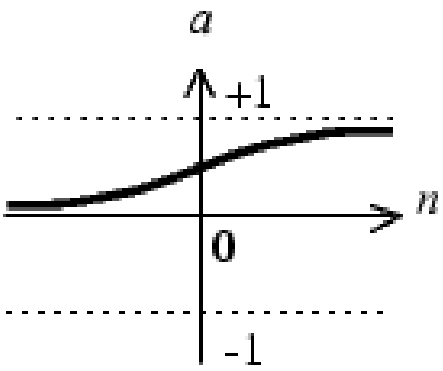


Funkcja newff() tworzy jednowarstwową sieć neuronową, gdzie implementowana warstwa składa się z zadanej liczby neuronów o nieliniowej funkcji aktywacji. Funkcja ta tworzy kaskadową jednokierunkową sieć jednowarstwową uczoną algorytmem Hebba. Po wywołaniu funkcji tworzony jest w przestrzeni obiekt net w którym zapisane są wszystkie informacje na temat utworzonej sieci.

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

gdzie:
minmax(wejscie) - spodziewane zakresy poszczególnych wejść perceptronu,
20- liczba neuronów w warstwie wyjściowej
logsig –funkcja aktywacji
learnh-funkcja uczenia
trainr-algorytm uczenia

Newff wykorzystuje w zadanym problemie nieliniową funkcję aktywacji. 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.



$$a = logsig(n)$$

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 ;
          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 ;
          ];
```

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), wartość współczynnika uczenia(lr). Kolejno ustalane są wagi każdego wejścia oraz wagi w warstwie metodą Hebba.

```
net.trainParam.epochs = 100;
net.trainParam.goal = 0.001;
lp.lr = 0.01;
dw=learnh([0],wejscie,[0],[0],wyjscie,[0],[0],[0],[0],[0],lp,[0]);
net.inputWeights{:,:}.learnFcn = 'learnh';
net.layerWeights {:,:} .learnFcn = 'learnh';
net=train(net, wejscie,wyjscie);
```

Bardzo ważnym fragmentem kodu jest linijka 4 na powyższym screenie. Jest to wywołanie funkcji learnh która implementuje uczenie metodą Hebba obliczając zmianę wagi. Przyjmuje wiele parametrów, z których najważniejsze to wejście, wyjście i parametr uczenia.

Do uczenia sieci została wykorzystana reguła Hebba, która w pakiecie Matlab oznaczana jest poprzez „trainr" i zostaje przesyłana jako parametr podczas tworzenia obiektu net. Reguła ta opiera się na zasadzie, że jeżeli aktywny neuron jest cyklicznie pobudzany przez inny neuron to staje się on jeszcze bardziej czuły na pobudzenie tego właśnie neuronu. Metoda ta odnosi się do nauki bez nauczyciela, a sygnałem uczącym jest sygnał wyjściowy neuronu. Wzmocnieniu ulegają te wagi, których wejścia są aktywne w sytuacji gdy duże jest wzbudzenie neuronu. Reguła prowadzi do uzyskania najlepszej korelacji pomiędzy sygnałami wejściowymi, a zapamiętanym w wagach wzorcem.

Podczas procesu uczenia, w miarę napływających wzorców (wektorów) sieć uczy się tych danych, czyli adaptuje sekwencyjnie swoje wagi – aby umieć prawidłowo rozpoznać te dane. Adaptacja ta jest iteracyjna, tzn. po przedstawieniu k-tego wzorca x(k) wytworzone już wagi zmieniają się według zasady:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k).$$

Ogólna reguła Hebba mówi, że zmiany wag powinny odbywać się według reguły

$$\Delta\mathbf{w}(k) = F(\mathbf{x}(k), y(k)),$$

czyli ogólnie, że przyrost wag Δw(k) powinien zależeć zarówno od wielkości wzorca x(k) jak i od wytworzonego wzorca y(k).

Następnie został stworzony zestaw testujący, dzięki któremu możliwe stało się sprawdzenie wyniku uczenia dla każdej litery. W zestawie zostały zawarte przykłady zmodyfikowanych liter, które później sieć miała przyporządkować do odpowiedniej grupy. Kolejno została zaimplementowana symulacja dla konkretnego testu oraz pętla gdzie przechodząc po wszystkich elementach wyszukuje największej wartości, dzięki której dany test jest przyporządkowany do konkretnej litery oraz wypisanie wartości dla wszystkich liter.

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

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

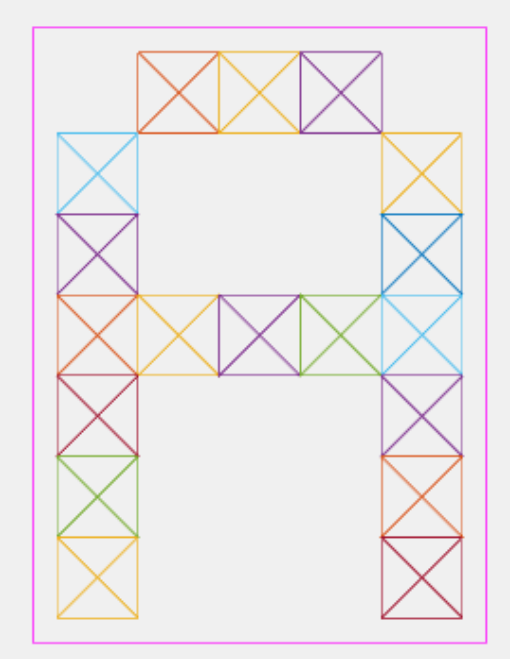
Zestawienie otrzymanych wyników

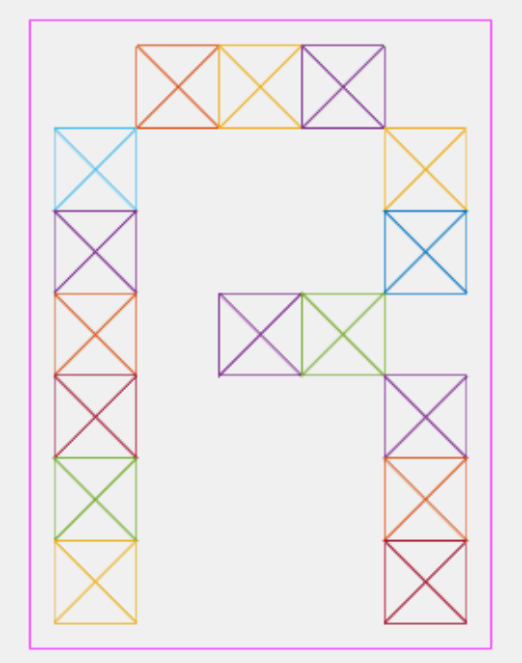
Zależność wartości współczynnika uczenia oraz błędu na wynik i liczbę epok.

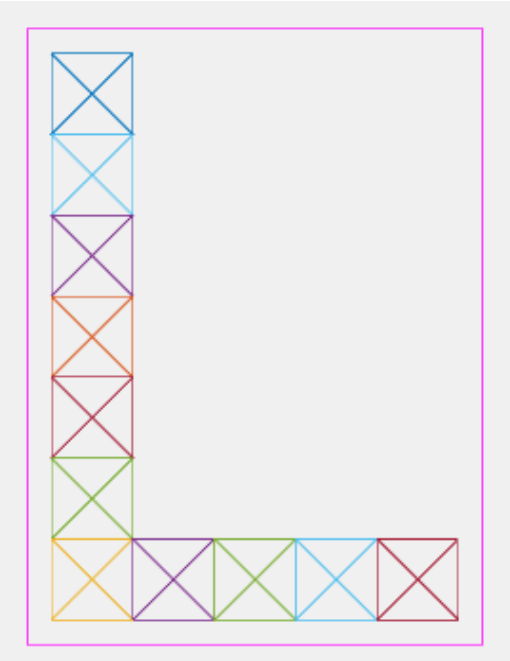
Współczynnik uczenia	Liczba epok	Czas
0.8	11	0:0:05
0.1	12	0:0:03
0.01	17	0:0:02
0.001	12	0:0:03
0.0001	8	0:0:02

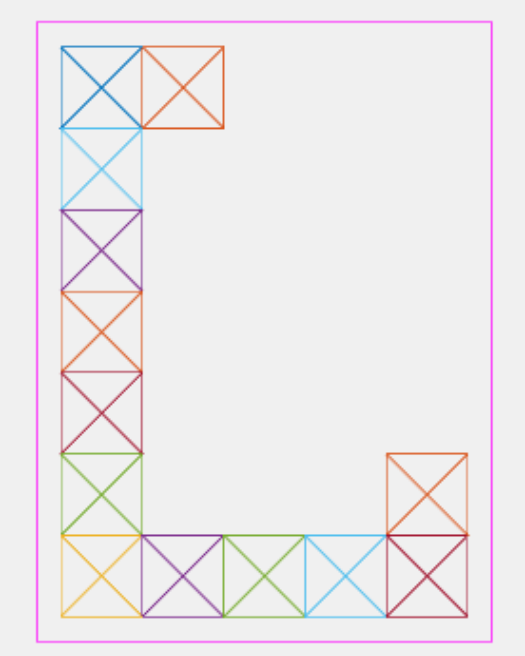
Wartości podobieństwa konkretnej litery po poszczególnych liter.

Lp.	A	Zmodyfikowane A	L	Zmodyfikowane L
A	**0.9936**	**0.96642**	0.00000050073	0.0000018337
B	0.00011164	0.0043502	0.0011595	0.011926
C	0.00000031350	0.00053236	0.0013254	0.1863
D	0.00024777	0.00013548	0.0019375	0.0046009
E	0.0012	0.000070183	0.002522	0.00013919
F	0.00041917	0.011062	0.0000030674	0.00004564
G	0.0014	0.0023045	0.000057765	0.0002475
H	0.0033	0.00018058	0.00015391	0.0000006331
I	0.00000011897	0.000015142	0.00055089	0.00015945
J	0.0000011897	0.000018447	0.000057761	0.0012454
K	0.00000062228	0.0000052552e	0.0010604	0.00025646
L	0.00000038183	0.00000037124	**0.99823**	**0.83352**
M	0.0010	0.0011837	0.00025019	0.00004801
N	0.00064948	0.001813	0.00092317	0.0026418
O	0.0022	0.000010439	0.0000037874	0.0000000026334
P	0.0019	0.031444	0.00010212	0.0000020513
Q	0.00032977	0.0028623	0.00046409	0.0018273
R	0.0011	0.011384	0.0000085539	0.00024447
S	0.00070335	0.0093146	0.0000071243	0.00019857
T	0.0000059410	0.00000054094	0.000000013464	0.000025357









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. Z zależności liczby epok i czasu nauki od wartości współczynnika uczenia nie da się wyciągnąć jednolitych wniosków. Widać, że na początku pomiarów podczas zmniejszania współczynnika liczba epok rosła tak samo jak czas, jednak w późniejszych wywołaniach wartości te zaczęły maleć. Wynika więc z tego, że podczas wywołań programu dla różnej wartości współczynnika uczenia liczba epok oraz czas była zróżnicowana. Niezależnie od wartości współczynnika uczenia sieć jest w stanie się nauczyć. W kolejnej tabeli zostały przedstawione wartości za pomocą których sieć przyporządkowuje konkretny test do litery. Dzieje się to poprzez wybranie największej wartości spośród wszystkich. Bardzo łatwo można dostrzec, że wartości zmodyfikowanych liter są prawie identyczne jak tych poprawnych. Natomiast widać też że im struktury liter są bardziej podobne tym większa wartość, np. dla litery L największą wartością oprócz tej do której została przyporządkowana jest E, co nawet intuicyjnie jest poprawne. W tabeli na czerwono zostały zaznaczone największe wartości dla wybranych struktur. Widać, że są ona największe oraz są bardzo podobne do siebie. Można jednak zauważyć, że różnica pomiędzy A oraz zmodyfikowaną A

jest mniejsza niż między L a L zmodyfikowaną. Wynika to z faktu, że na literze A została wykonana mniejsza modyfikacja niż na L, stąd i różnica jest mniejsza.

Wnioski

Sieć jednowarstwowa to sieć zbudowana z jednej warstwy o zadniej liczbie neuronów. W rozpatrywanym zadaniu sieć posiada wejście w postaci liter złożonych z 35 zero-jedynkowych pól oraz warstwę wyjściową zawierającą 20 neuronów. Sygnały przesyłane są od warstwy wejściowej do warstwy wyjściowej. Do implementacji została używa nieliniowa funkcja aktywacji logsig. Do implementacji został użyty algorytm Hebba wykorzystujący zasadę, że aktywny neuron, który jest cyklicznie pobudzany przez inny neuron staje się jeszcze bardziej czuły na pobudzenie. 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, a w szczególności na proces uczenia. Niezależnie od wartości współczynnika uczenia sieć zawsze się nauczy. Do identyfikacji konkretnego wzorca sieć wykorzystuje wartości obliczane dla każdej litery wybierając ten największy. Sieć uczy się intuicyjnie. Współczynnik jest tym większy im większe jest podobieństwo do danej litery. Natomiast różnica pomiędzy poprawnym przedstawieniem litery a zmodyfikowanym jest tym większa im modyfikacja jest większa.

Listing kodu wykonanego programu

```
close all; clear all; clc;

%litery przedstawione za pomocą 35 pól( 20 dużych liter alfabetu)

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];
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];

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

%wektor wyjściowy składający się z zero-jedynkowych pól
%miejsce wystąpienia 1 identyfikuje literę
wyjscie= [1 0 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 0 1 0 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 0 1 0 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 0 1 0 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 0 1 0 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 0 1 0 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 0 1 0 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 0 1 0 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 0 1 0 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 0 1 0 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 0 1 0 ;
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 ;];
```

```
     % 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;];

  %zmodyfikowany zestaw testujący dla przykładowych liter
 testA2=[0; 1; 1; 1; 1;
     1; 0; 0; 0; 1;
     1; 0; 0; 0; 1;
     1; 1; 1; 1; 1;
     1; 0; 0; 0; 0;
     1; 0; 0; 0; 1;
     1; 0; 0; 0; 1;
     ];

 testA3=[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;];

  testL1=[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; 1; 1; 1;];

 %implementacja sieci jednowarstwowej za pomocą funkcji newff
 net=newff( minmax(wejscie),  20, {'logsig','learnh' ,'trainr'});
 %minmax(wejscie ) - spodziewane zakresy poszczególnych wejść perceptronu,
 %20- liczba neuronów w warstwie wyjściowej
 %logsig –funkcja aktywacji
 %learnh-funkcja uczenia
 %trainr-algorytm uczenia
```

```
net.trainParam.epochs = 100;     %maksymalna liczba epok
net.trainParam.goal = 0.0001;    %błąd
lp.lr = 0.001;                   %współczynnik uczenia
%zmiana wartości wag
dw=learnh([0],wejscie,[0],[0],wyjscie,[0],[0],[0],[0],[0],lp,[0]);
net.inputWeights{:,:}.learnFcn = 'learnh';    %wagi dla wejścia regułą Hebba
net.layerWeights {:,:} .learnFcn = 'learnh'; %wagi dla warstwy regułą Hebba
net=train(net, wejscie,wyjscie);             %trening sieci

%symulacja sieci dla konkretnego testu
pokaz=sim(net, testL1);

% wyszukiwanie największej wartości
max=1;
for i=1:1:20;
    if(pokaz(max)<pokaz(i))
        max=i;
    end;
end

% wypisanie wartości wyjściowych dla każdej litery
disp('wartosci wysciowe dla wszytskich liter')
disp('A'), disp(pokaz(1));
disp('B'), disp(pokaz(2));
disp('C'), disp(pokaz(3));
disp('D'), disp(pokaz(4));
disp('E'), disp(pokaz(5));
disp('F'), disp(pokaz(6));
disp('G'), disp(pokaz(7));
disp('H'), disp(pokaz(8));
disp('I'), disp(pokaz(9));
disp('J'), disp(pokaz(10));
disp('K'), disp(pokaz(11));
disp('L'), disp(pokaz(12));
disp('M'), disp(pokaz(13));
disp('N'), disp(pokaz(14));
disp('O'), disp(pokaz(15));
disp('P'), disp(pokaz(16));
disp('Q'), disp(pokaz(17));
disp('R'), disp(pokaz(18));
disp('S'), disp(pokaz(19));
disp('T'), disp(pokaz(20));


%wypisanie komunikatu
switch max
    case 1
       test=testA;
       disp('wpisana liter to A')
    case 2
       test=testB;
        disp('wpisana liter to B')
    case 3
       test=testC;
        disp('wpisana liter to C')
    case 4
       test=testD;
        disp('wpisana liter to D')
    case 5
```

```
        test=testE;
        disp('wpisana liter to E')
    case 6
        test=testF;
        disp('wpisana liter to F')
     case 7
        test=testG;
        disp('wpisana liter to G')
    case 8
        test=testH;
        disp('wpisana liter to H')
    case 9
        test=testI;
        disp('wpisana liter to I')
    case 10
        test=testJ;
        disp('wpisana liter to J')
    case 11
        test=testK;
        disp('wpisana liter to K')
    case 12
        test=testL;
        disp('wpisana liter to L')
    case 13
        test=testM;
        disp('wpisana liter to M')
    case 14
        test=testN;
        disp('wpisana liter to N')
    case 15
        test=testO;
        disp('wpisana liter to O')
    case 16
        test=testP;
        disp('wpisana liter to P')
    case 17
        test=testQ;
        disp('wpisana liter to Q')
    case 18
        test=testR;
        disp('wpisana liter to R')
    case 19
        test=testS;
        disp('wpisana liter to S')
    case 20
        test=testT;
        disp('wpisana liter to T')
    otherwise
        disp('nie ma takiej litery')
end

%wyświetlenie testu
     plotchar(testL1)
```