Fabiola Dąbroś , IO gr.1 rok III

Sprawozdanie z zagadnienia nr 2.

Temat ćwiczenia:
Budowa i działanie sieci jednowarstwowej.
Cel ćwiczenia:
Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

1. Opis budowy 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 korzystam 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. Perceptron zbudowany jest jedynie z warstwy wejściowej oraz warstwy wyjściowej. Jego działanie polega na klasyfikowaniu danych pojawiających się na wejściu i ustawianiu stosownie do tego wartości wyjścia. 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.
Do wykonania ćwiczenia wykorzystałam gotowe narzędzia z pakietu Matlab.

- Funkcja newp()

Służy ona do tworzenia jednowarstwowej sieci neuronowej złożonej z zadanej liczby neuronów – „twardych” perceptronów. Na wejściu ustalamy liczbę wejść sieci oraz liczbę neuronów sieci. Ponadto mamy funkcję aktywacji neuronów oraz funkcję służącą do trenowania sieci perceptronowej. Natomiast na wyjściu tworzy się struktura zawierająca opis architektury, metod treningu, wartości liczbowe wag i progów oraz inne parametry sieci perceptronowej. Opisana funkcja wygląda następująco:

```
net1=newp(minmax(dane_we),1,'hardlim','learnp');
net1=init(net1);
```

gdzie:
- minmax(dane_we) - spodziewane zakresy dla elementów wejściowych
- 1 - liczba neuronów sieci
- ‘hardlim’ – nazwa funkcji aktywacji neuronów
- ‘learnp’ – nazwa funkcji trenowania sieci perceptronowej
- net1 = init(net1) – inicjalizuje sieć neuronową

Funkcją aktywacji perceptronów może być funkcja unipolarna oraz bipolarna. Domyślnie ustawiana jest funkcja unipolarna. Według tych funkcji określana jest wartość wyjścia

neuronów sieci neuronowej. Każde wejście posiada zadaną wartość progową , natomiast odpowiedź określana jest przy użyciu funkcji Heavyside'a :

$$y(x) = \begin{cases} 0 & dla \quad x < a \\ 1 & dla \quad x \geq a \end{cases}$$

Funkcja wymusza wprowadzenie 0 jeśli próg osiągnie wartość progową, w przeciwnym razie wynosi 1. Umożliwia to neuronowi podjęcie decyzji lub klasyfikacji.

Funkcja trenowania sieci perceptronowej używana jest do modyfikacji wag w oparciu o regułę perceptronową. Perceptrony są szkolone na przykładach pożądanego zachowania. Żądane zachowanie można podsumować zestawem wejściowych oraz wyjściowych par $\mathbf{p}_1\mathbf{t}_1,\mathbf{p}_2\mathbf{t}_1,\ldots,\mathbf{p}_Q\mathbf{t}_Q$ gdzie p jest wejściem do sieci, a t jest docelowym wyjściem. Wektor docelowy t musi zawierać wartości 0 lub 1, ponieważ perceptron (zgodnie z funkcją 'hardlim') może wysyłać takie wartości. Celem funkcji trenowania jest zmniejszenie błędu, czyli różnicy między odpowiedzią neuronu, a wektorem docelowym. Za każdym razem gdy wywołujemy funkcję learnp perceptron ma większą szansę na uzyskanie poprawnych wyjść.

- Funkcja newlin()

Służy ona do tworzenia jednowarstwowej sieci neuronowej złożonej z zadanej liczby neuronów o liniowej funkcji aktywacji. Na wejściu ustalamy liczbę wejść sieci oraz liczbę neuronów sieci. Ponadto mamy wektor opóźnień oraz stałą szybkość uczenia sieci. Natomiast na wyjściu tworzy się struktura zawierająca opis architektury, metod treningu, wartości liczbowe wag i progów oraz inne parametry sieci perceptronowej. Opisana funkcja wygląda następująco:

```
net2=newlin(minmax(dane_we), 1, 0, 0.01);
net2=init(net2);
```

gdzie:

- minmax(dane_we) - spodziewane zakresy dla elementów wejściowych
- 1 - liczba neuronów sieci
- 0 – wektor opóźnień poszczególnych elementów wektora wejść sieci; domyślnie 0
- 0.01 - stała szybkości uczenia (treningu) sieci liniowej; domyślnie *LR* = 0.01
- net2 = init(net2) – inicjalizuje sieć neuronową

Struktura net2, zwracana przez tę funkcję zawiera maksymalną wartość stałej szybkości uczenia, zapewniającą stabilny trening sieci liniowej dla wektorów wejściowych. Wartości wag oraz progi sieci liniowej są inicjowane podczas wywołania procedury init.

Zarówno dla funkcji newp() oraz newlin() wywołane zostały te same dane wejściowe oraz wyjściowe. Prezentują się one następująco:

```
dane_we=[ mb(:) dB(:) mc(:) dC(:) md(:) dD(:) mh(:) dH(:) mi(:) dI(:)
          mk(:) dK(:) ml(:) dL(:) mo(:) dO(:) mt(:) dT(:) mw(:) dW(:)];

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

Zestaw danych uczących składa się z 10 małych oraz 10 dużych liter alfabetu. Przedstawione są one jako matryca 5x7 o wartościach 0 lub 1 co daje nam 35 pól. Jako dane wyjściowe mamy wartości 0 oraz 1, które oznaczają kolejno małą oraz dużą literę.

Ponadto dla obydwóch funkcji wywołane zostały następujące procedury:

```
przed_treningiem=sim(net, dane_we);
net.trainparam.epochs=150;
net.trainparam.goal=1e-25;
net.trainparam.lr=0.001;
net.IW{1}=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ];
net=train(net, dane_we, dane_wy);
po_treningu= sim(net, dane_we);
```

Poprzez operację ‘sim’ wykonuje się symulacja sieci , którą sprawdzamy zarówno przed treningiem jak i po wykonaniu treningu. Funkcja ta służy do wyznaczenia wyjść sieci neuronowej dla zadanej macierzy danych wejściowych. Następnie określamy długość treningu ustalając liczbę epok na 150. Poprzez ‘goal’ określamy kryterium stopu - sumę kwadratów błędów wyjść sieci oraz poprzez ‘lr’ określamy learning rate czyli wartość współczynnika uczenia sieci. Dodatkowo określamy wartości wag poprzez IW{1}. Na końcu wywołujemy funkcję ‘train’ dokonującą treningu sieci. Jej działanie dla sieci jednokierunkowych polega na obliczaniu metodą iteracyjną wartości współczynników wagowych.

Jednak każda z funkcji wykorzystuje inny algorytm uczenia. Są one następujące:

- Algorytm trainc

Funkcja newp() korzysta z tego algorytmu. Polega on na cyklicznym szkoleniu przyrostowym z funkcjami uczenia się. Trenuje sieć z wagą i błędem uczenia się z przyrostowymi aktualizacjami po każdej prezentacji danych wejściowych. Wejścia są prezentowane w kolejności cyklicznej. Szkolenie odbywa się w zależności od wartości parametrów opisanych powyżej (epochs oraz goal). Trening kończy się gdy maksymalna liczba epok zostaje osiągnięta lub gdy wyniki zostają zminimalizowane do wartości goal.

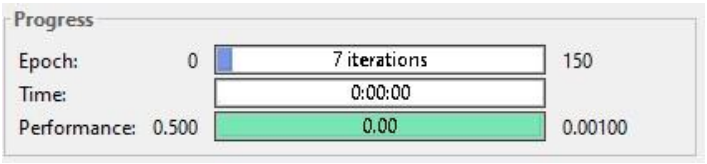
- Algorytm trainb

Funkcja newlin() korzysta z tego algorytmu. Polega on na szkoleniu wsadowym z funkcjami uczenia się. Trenuje sieć z wagą i błędem uczenia się z wsadową aktualizacją, która zakłada minimalizację funkcji błędu średniokwadratowego wyznaczanego dla wszystkich obserwacji ze zbioru uczącego. Wagi oraz błędy są aktualizowane przez dane wejściowe pod koniec całego przebiegu. Szkolenie odbywa się w zależności od wartości parametrów opisanych powyżej (epochs oraz goal). Trening kończy się gdy maksymalna liczba epok zostaje osiągnięta lub gdy wyniki zostają zminimalizowane do wartości goal.

2. Zestawienie otrzymanych wyników

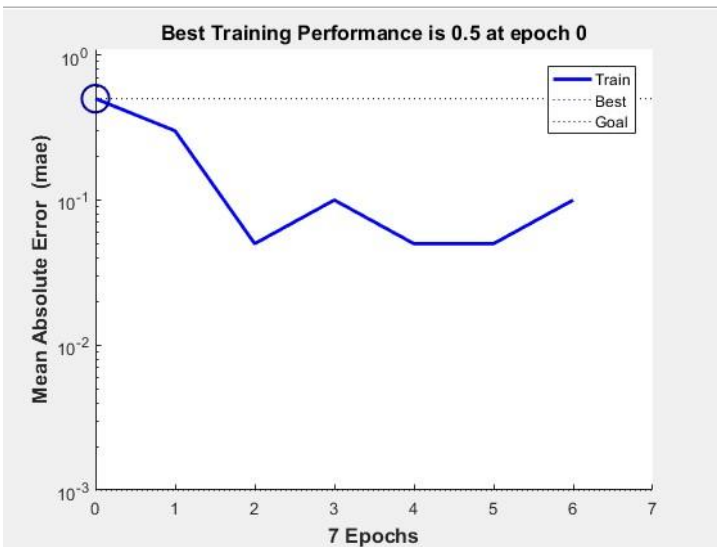
Tabela wyników uczenia dla testowania różnej ilości danych uczących prezentuje się następująco:

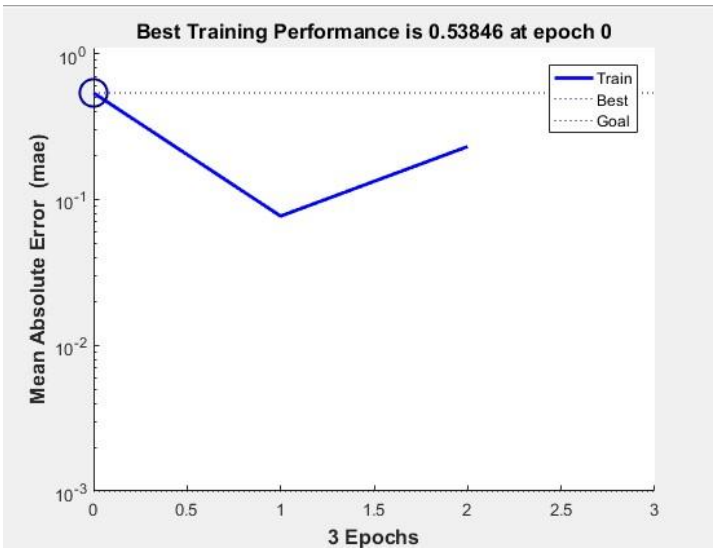
- Funkcja newp()

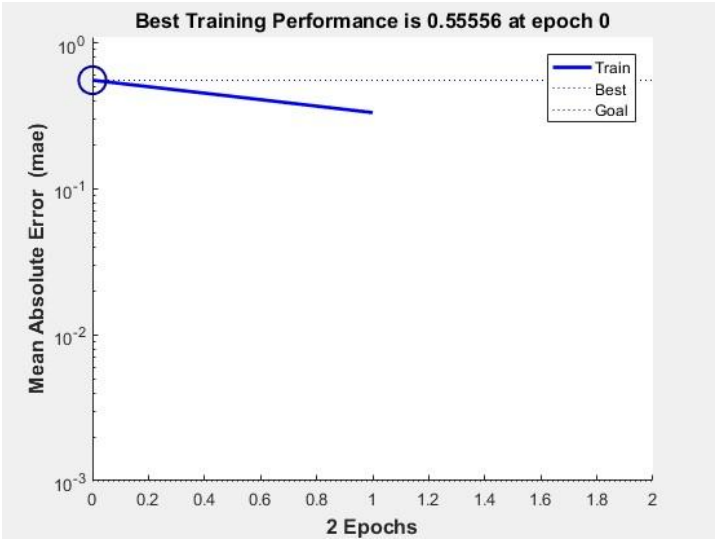


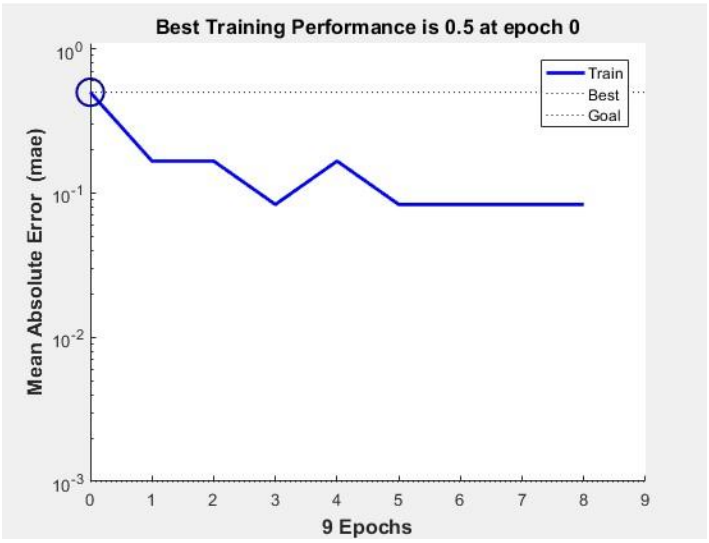
Ilość danych uczących	Liczba epok	Czas realizacji
20	7	0:00:00
15	3	0:00:00
8	2	0:00:00
25	9	0:00:00

Wykresy wydajności prezentują się następująco:









- Funkcja newlin()

Niestety dla tej funkcji nie udało się uzyskać pożądanych wyników. Za każdym razem pomimo wykonania dużej ilości iteracji wyniki są niepoprawne.

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

- Funkcja newp()

Współczynnik uczenia	Błąd	Liczba epok	Wynik
0.001	**0.001**	7	poprawny
0.001	**0.1**	2	niepoprawny
0.1	**0.01**	7	poprawny
0.000001	**0.1**	3	niepoprawny

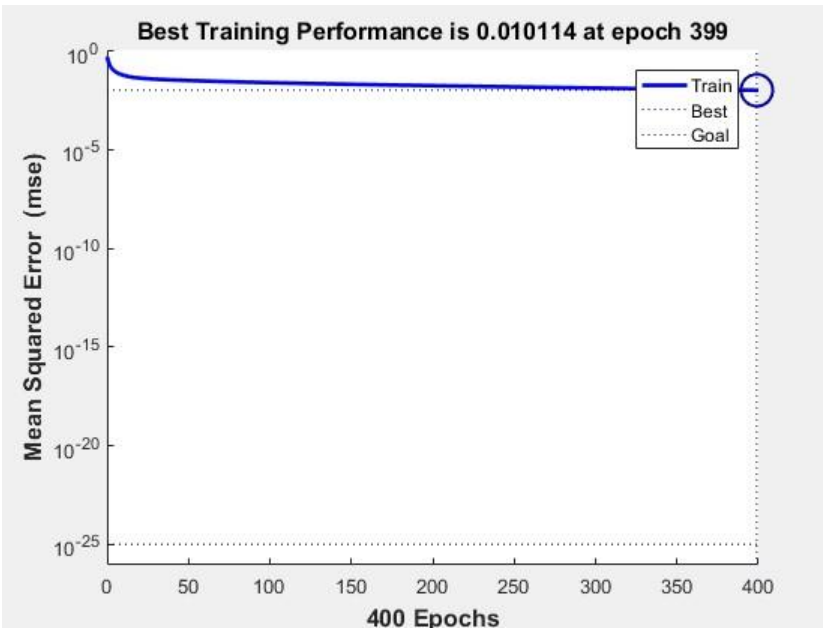
Przykładowy niepoprawny wynik:

```
przed_treningiem1 =

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

po_treningu1 =

     0     1     0     1     0     1     0     1     0     1     0     1     1     1     0     1     0     1     0     1
```

- Funkcja newlin()

Pomimo zmian wartości współczynnika uczenia oraz błędu wyniki są zawsze niepoprawne.

Wykres wydajności:




```
przed_treningiem2 =

     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

po_treningu2 =

  Columns 1 through 12

    0.0821    0.9689   -0.1997    1.0068    0.2513    0.9780    0.0584    0.8246    0.0394    0.9835    0.0467    1.0111

  Columns 13 through 20

    0.1690    0.8864   -0.0515    1.0324   -0.0014    1.0243   -0.0521    0.9393
```

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 newp() wzrasta wraz z powiększaniem zestawu uczącego. Natomiast różnica w ilości danych jest tak znikoma, że nie wpływa ona na czas nauki. W każdej sytuacji jest on natychmiastowy. Dla funkcji

newlin() zestaw danych uczących nie ma wpływu. Bez względu na to czy jest on duży czy mały nigdy nie osiąga poprawnego wyniku.

Wykresy wydajności funkcji newp() potwierdzają, że bez względu na ilość danych uczących wynik jest poprawny. Na każdym z nich widnieje inna ilość epok dzięki czemu widzimy wykres w różnej skali. Jednak po analizie dostrzegamy, że zmienia się on w jednakowy sposób. Ponadto za każdym razem najlepszy wynik zostaje osiągnięty w epoce 0 i wynosi średnio 0.5.

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 przypadku funkcji newlin() zmiana wartości współczynnika uczenia oraz błędu nie ma wpływu na wynik, który jest zawsze niepoprawny. Jak widać na powyższym wykresie najlepszy wynik zostaje osiągnięty w przedostatniej epoce, ale i tak nie jest zadowalający. Analizując wartości ukazane po treningu są one tylko przybliżone do poprawnych.

4. Wnioski

W celu uczenia rozpoznawania wielkości liter korzystałam z funkcji dostępnych w pakiecie Matlab. Podczas testowania sieci zauważyłam, że funkcja newp() idealnie się do tego nadaje. Niestety funkcja newlin() nie spełnia danych oczekiwań. Pomimo faktu, że wraz z powiększaniem maksymalnej ilości epok funkcja wykonuje wszystkie iteracje oraz pojawia się komunikat informujący o tym, że maksymalna ilość epok została osiągnięta, wynik końcowy różni się od oczekiwanego. Wynika z tego fakt, że do rozwiązania problemu rozpoznawania wielkości liter algorytm oparty na cyklicznym szkoleniu jest lepszy od algorytmu opartego na szkoleniu wsadowym.

Skupiając się na funkcji newp() zauważamy, że parametry procesu uczenia mają ogromny wpływ na uzyskiwane wyniki. Zwiększanie zestawu uczącego wpływa na liczbę epok. Natomiast zmiana współczynnika uczenia oraz błąd wpływają na wynik końcowy. Jeśli błąd będzie mało dokładny nie osiągniemy poprawnego rozwiązania.

5. Listing całego kodu

```
close all; clear all; clc;

% Zestaw danych uczących składający się z 10 małych oraz 10 dużych liter.
Przedstawione są jako matryca 5x7 o wartościach 0 lub 1

b= [1 0 0 0 0; 1 0 0 0 0; 1 0 0 0 0; 1 1 1 1 0; 1 0 0 0 1; 1 0 0 0 1; 1 1 1 1 0];
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 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 0 1 1 1 0; 1 0 0 0 0; 1 0 0 0 0; 0 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 1; 0 1 1 1 0];
d= [0 0 0 0 1; 0 0 0 0 1; 0 0 0 0 1; 0 1 1 1 1; 1 0 0 0 1; 1 0 0 0 1; 0 1 1 1 1];
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 1 1 1 0];
h= [1 0 0 0 0; 1 0 0 0 0; 1 0 0 0 0; 1 1 1 0 0; 1 0 0 1 0; 1 0 0 1 0; 1 0 0 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 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 1 0 0; 0 0 1 0 0];
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];
k= [1 0 0 0 0; 1 0 0 0 0; 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];
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 0 0 0];
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];
o= [0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 0 0 1 0 0; 0 1 0 1 0; 0 1 0 1 0; 0 0 1 0 0];
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];
t= [0 0 1 0 0; 0 0 1 0 0; 0 1 1 1 0; 0 0 1 0 0; 0 0 1 0 0; 0 0 1 0 0; 0 0 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= [0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 1 0 0 0 1; 1 0 0 0 1; 1 0 1 0 1; 0 1 0 1 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

```
mb=transpose(b);
dB=transpose(B);
mc=transpose(c);
dC=transpose(C);
md=transpose(d);
dD=transpose(D);
mh=transpose(h);
dH=transpose(H);
mi=transpose(i);
dI=transpose(I);
mk=transpose(k);
dK=transpose(K);
ml=transpose(l);
dL=transpose(L);
mo=transpose(o);
dO=transpose(O);
mt=transpose(t);
dT=transpose(T);
mw=transpose(w);
dW=transpose(W);
```

% Wektor wejściowy złożony z małych oraz dużych liter po transpozycji

```
dane_we=[ mb(:) dB(:) mc(:) dC(:) md(:) dD(:) mh(:) dH(:) mi(:) dI(:)
          mk(:) dK(:) ml(:) dL(:) mo(:) dO(:) mt(:) dT(:) mw(:) dW(:)];
```

% Wektor wyjściowy o wartościach 0 oraz 1, które oznaczają kolejno małą oraz dużą literę

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

% Tworzenie sieci za pomocą funkcji newp()

```
net1=newp(minmax(dane_we),1,'hardlim','learnp');
```

% - minmax(dane_we) – spodziewane zakresy dla elementów wejściowych
% - 1 – liczba neuronów sieci
% - hardlim – funkcja aktywacji neuronów
% - learnp – funkcja trenowania sieci perceptronowej

```
% Inicjalizacja sieci neuronowej
net1=init(net1);
% Symulacja sieci – wypisanie danych przed treningiem
przed_treningiem1=sim(net1, dane_we)
% Określenie długości treningu
net1.trainparam.epochs=150;
% Określenie wartości stopu
net1.trainparam.goal=0.001;
% Określenie współczynnika uczenia
net1.trainparam.lr=0.1;
% Określenie wag
net1.IW{1}=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ];
% Trening sieci ( algorytm trainc)
net1=train(net1, dane_we, dane_wy);
% Symulacja sieci – wypisanie danych po treningu
po_treningu1= sim(net1, dane_we)


% Tworzenie sieci za pomocą funkcji newlin()

net2=newlin(minmax(dane_we), 1, 0, 0.01);
      % - minmax(dane_we) – spodziewane zakresy dla elementów wejściowych
      % - 1 – liczba neuronów sieci
      % - 0 – wektor opóźnień poszczególnych elementów wektora wejść sieci; domyślnie 0
      % - 0.01 – stała szybkości uczenia sieci liniowej; domyślnie 0.01

% Inicjalizacja sieci neuronowej
net2=init(net2);
% Symulacja sieci – wypisanie danych przed treningiem
przed_treningiem2=sim(net2, dane_we)
% Określenie długości treningu
net2.trainparam.epochs=400;
% Określenie wartości stopu
net2.trainparam.goal=0.0001;
% Określenie współczynnika uczenia
net2.trainparam.lr=0.01;
% Trening sieci ( algorytm trainb)
net2=train(net2, dane_we, dane_wy);
% Symulacja sieci – wypisanie danych po treningu
po_treningu2= sim(net2, dane_we)
```