Marcelina Bałamut, IO, gr. 1

Temat ćwiczenia:

Budowa i działanie sieci jednowarstwowej.

Cel ćwiczenia

Celem ćwiczenia jest poznanie budowy i działanie jednowarstwowych sieci neuronowych poprzez implementację oraz uczenie rozpoznawania wielkości liter.

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 jednej warstwie zasilanej z węzłów wejściowych w których nie zachodzi żaden proces obliczeniowy. Ponadto w sieci jednowarstwowej każdy węzeł jest połączony z każdym neuronem. W wykonywanym zadaniu jako dane uczące zostało wykorzystane 20 dowolnych liter alfabetu: 10 dużych i 10 małych przedstawionych w wersji binarnej za pomocą cyfr 1 lub 0 jako tablica o wymiarze 5x7 (35 wejść). Natomiast wektor docelowy składa się z 20 cyfr gdzie 1 oznacza literę dużą, zaś 0 literę małą. Do implementacji danego problemu użyłam wbudowanych funkcji pakietu Matlab opisanych niżej.

1. *Funkcja newp()*

Tworzy ona jednowarstwową sieć neuronową, złożoną z zadanej liczby „twardych" neuronów (która w opisanym zadaniu wynosi 1), o progowej funkcji aktywacji oraz inicjalizuje jej współczynniki wagowe. Po wywołaniu funkcji tworzony jest w przestrzeni obiekt net w którym zapisane są wszystkie informacje na temat utworzonej sieci. Instrukcja init inicjalizuje również początkowe wartości wag i progów.

```
net=newp(minmax(wejscie), 1);
net=init(net);
```

gdzie:
minmax(wejscie) - spodziewane zakresy poszczególnych wejść perceptronu,
1 - liczba perceptronów w sieci,
net = init(net) – inicjalizuje sieć neuronową.

Funkcja ta może przyjmować jeszcze dwa parametry, z których pierwszy to funkcja aktywacji, która domyślnie ustawiana jest na ‘hardlim’. Każde wejście posiada odpowiednią wagę, a odpowiedź perceptronu określona jest za pomocą funkcji aktywacji Heaviside’a, która daje wartość 0 dla argumentu mniejszego od zera i 1 dla większego od zera, następuje więc klasyfikacja danych wejściowych na takie dla których stan wyjścia równa się 0 i na takie gdzie równa się 1.

Natomiast drugi parametr to metoda uczenia, domyślnie ustawiana na ‘learnp”, która jest funkcją uczącą, służącą do realizacji pojedynczego cyklu uczenia. Perceptron jest szkolony na podstawie pożądanego zachowania, które można podsumować zestawem danych wejściowych i wyjściowych.

```
wejscie=[ zz(:) ii(:) uu(:) kk(:) oo(:) cc(:) hh(:) xx(:) vv(:) ww(:)
          AA(:) BB(:) CC(:) DD(:) EE(:) FF(:) GG(:) HH(:) II(:) JJ(:)];

wyjscie=[ 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1];
```

Celem jest zmniejszenie błędu, czyli różnicy między odpowiedzią neuronu, a wektorem docelowym(„wyjscie”). Learnp zmienia wagi i progi przy pomocy metody Rosenblatt’a, gdzie modyfikowane są w odpowiedni sposób wagi wejść, tak aby wartość na wyjściu przybierała pożądane wartości. Podczas każdego wywołania funkcji learnp perceptron ma większą szansę na wyprodukowanie poprawnych wyjść.

2. *Funkcja newlin()*

Tworzy ona jednowarstwową sieć złożoną z liniowych neuronów oraz inicjalizuje współczynniki wagowe wartościami losowymi. Podczas tworzenia sieci ustala liczbę neuronów, stałą szybkości uczenia i wektor opóźnień. Newlin wykorzystuje liniową funkcję aktywacji.

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

gdzie:
minmax(wejscie) - spodziewane zakresy poszczególnych wejść perceptronu,
1 - liczba perceptronów w sieci,
0 - wektor opóźnień poszczególnych elementów wektora wejść sieci, domyślnie ustawiany na 0
0.01 - stała szybkości uczenia (treningu) sieci liniowej, domyślnie LR ustawiany na 0.01
net2 = init(net2) – inicjalizuje sieć neuronową.

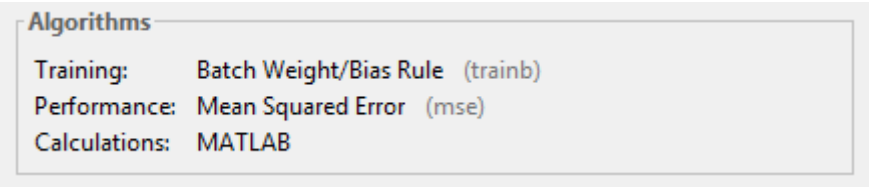
Po wywołaniu funkcji tworzona jest w przestrzeni struktura net2 w której zapisane są wszystkie informacje na temat utworzonej sieci np. opis architektury, model treningu, czy wartości wag. Wartości wag są inicjowane podczas wywołania procedury init.

Dla obu sposobów uczenia został wykorzystany ten sam zestaw danych uczących zawarty w kodzie, który jest przedstawiony za pomocą wektorów.

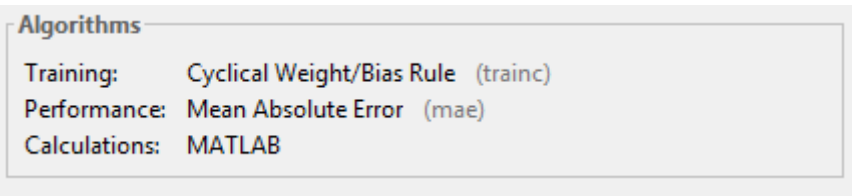
Zarówno dla funkcji newp jak i newlin została wywołana procedura train, dokonująca treningu zbudowanej sieci. Jest ona uniwersalna, wywoływana w jednolity sposób dla wszystkich typów sieci neuronowych. Działa ona dla sieci jednokierunkowych obliczając wartości współczynników wagowych metodą iteracyjną. Przed wywołaniem ustalona została docelowa wartość błędu uczenia(goal), maksymalna liczba kroków uczących(epchos), wartość współczynnika uczenia(lr- learning rate) oraz wartość wag(IW{1}). Poprzez procedurę ‘sim’ wykonuje się symulacja sieci przed treningiem oraz po nim, co pozwala na wyznaczenie wartości wyjściowych które są zależne od wektora wejść.

```
przedterningiem=sim(net, wejscie);
net.trainParam.epochs = 200;
net.trainParam.goal =  0.001;
net.trainParam.lr=0.0001;
net.IW{1}=[1 1 6 1 1 1 1 5 1 1 4 2 2 3 1 1 2 4 2 1 3 1 7 1 1 3 2 1 1 4 1 1 3 1 1];
net=train(net, wejscie, wyjscie);
potreningu= sim(net, wejscie);
```

Różnicą jest fakt, że funkcja newlin realizuje trening za pomocą funkcji trainb, która oblicza wartości współczynników wagowych metodą wsadową, zakładając minimalizację funkcji błędu średniokwadratowego wyznaczanego dla wszystkich obserwacji ze zbioru uczącego. Modyfikacja wag następuje po prezentacji całego zbioru treningowego i jest obliczana jako średnia z wartości gradientów dla poszczególnych obserwacji.



Natomiast funkcja newp realizuje trening za pomocą funkcji trainc, która realizuje cykliczne szkolenie przyrostowe. Dla każdej epoki wszystkie wektory są przedstawiane wraz z wagą, która jest aktualizowana po każdej prezentacji. Trening kończy się, gdy zostanie osiągnięta maksymalna liczba epok, zostanie przekroczony maksymalny czas, lub oczekiwany wynik zostanie spełniony.

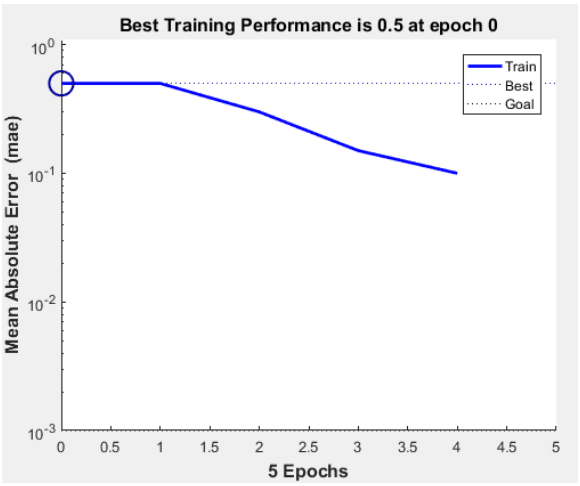


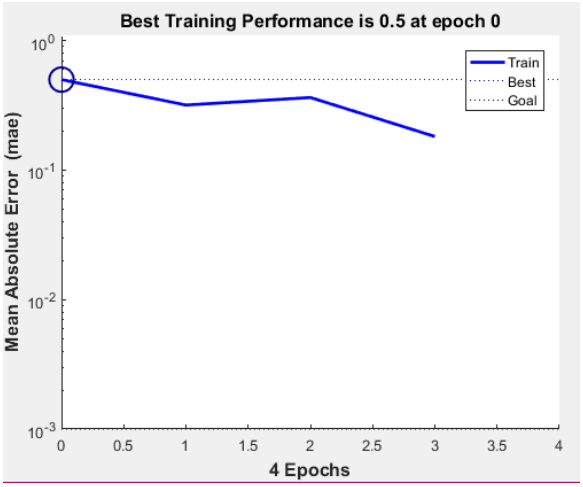
Zestawienie otrzymanych wyników

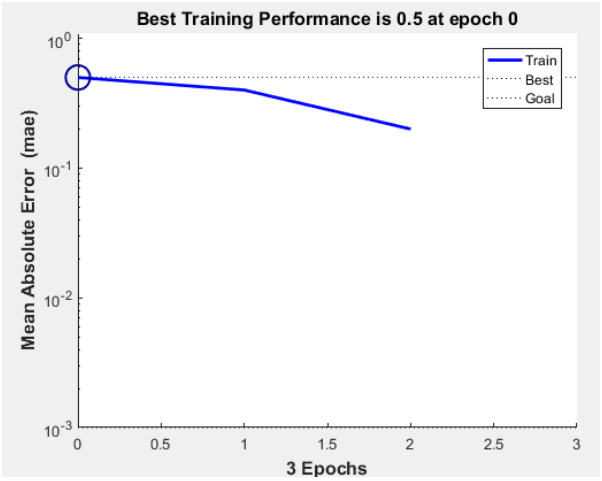
1. *Funkcja newp*

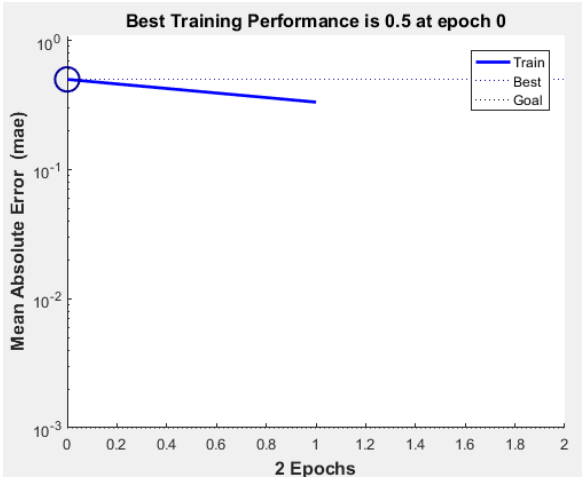
Zależność liczby epok i czasu od liczby danych:

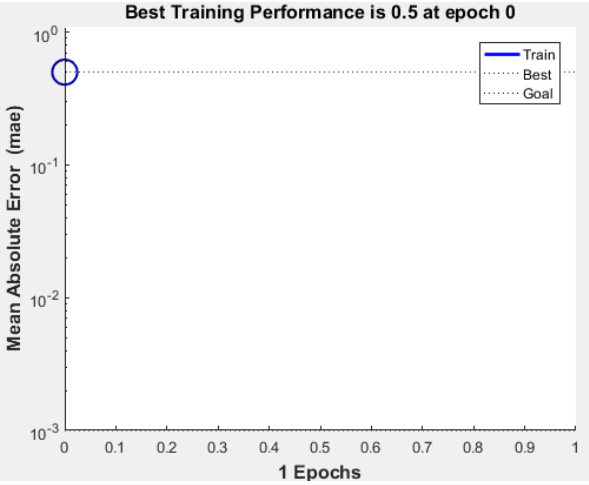
Liczba danych	Liczba epok	Czas
20	5	0.00:03
16	4	0.00:02
10	3	0.00:02
6	2	0.00:02
2	1	0.00:02
26	5	0.00:06

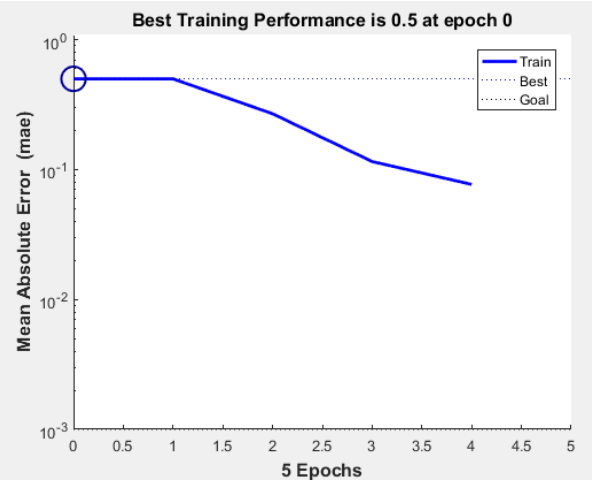












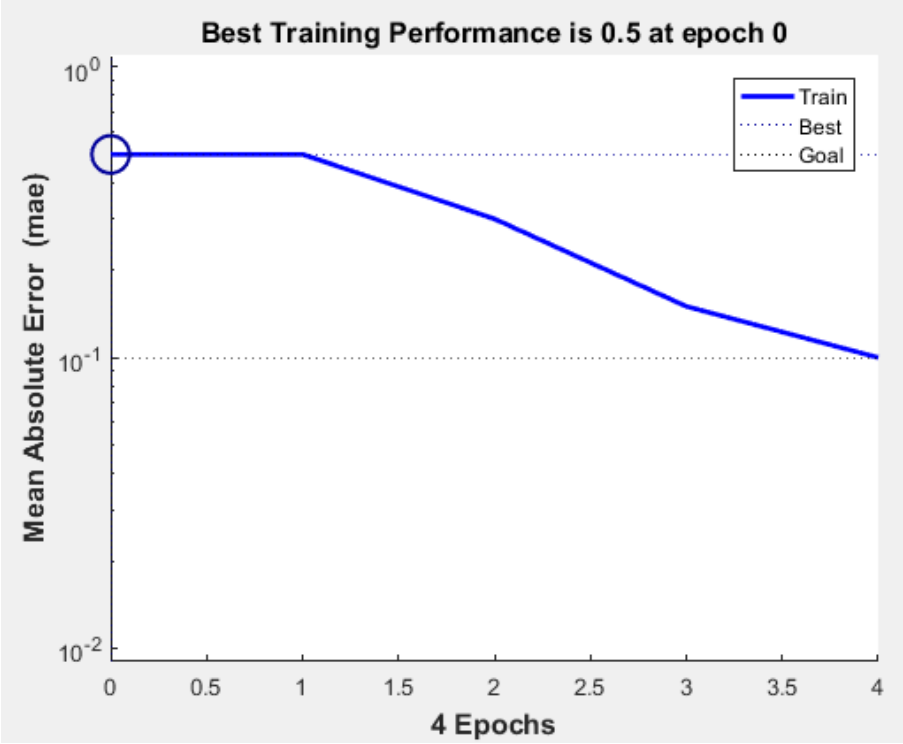
Zależność wyniku od współczynnika uczenia i błędu

Współczynnik uczenia	Błąd	Wynik
0.001	0.1	błędny
0.1	0.001	prawidłowy
0.01	0.01	prawidłowy
0.8	0.1	błędny
0.00001	0.001	prawidłowy

Przykładowy błędny wynik dla współczynnika wynoszącego 0.8 i błędu 0.1:

```
przedterningiem =

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

potreningu =

     0     0     0     0     0     1     1     0     0     0     1     1     1     1     1     1     1     1     1     1

>>
```

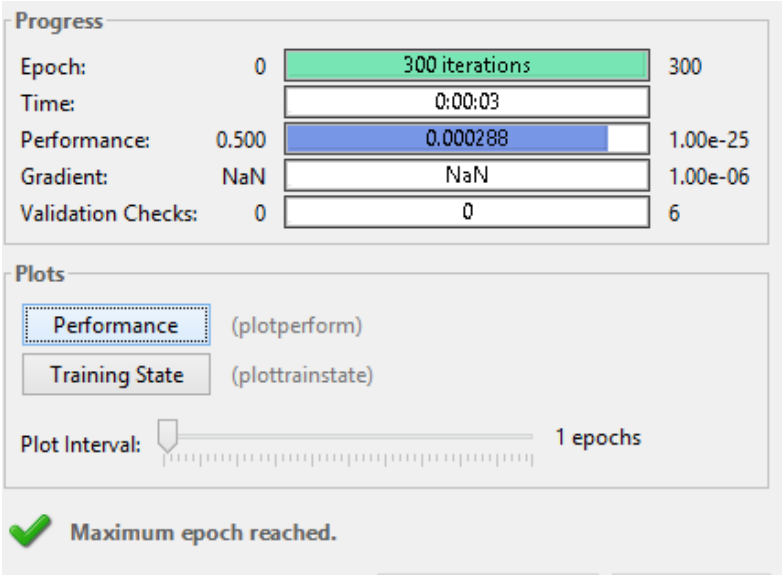
Wykres wydajności:



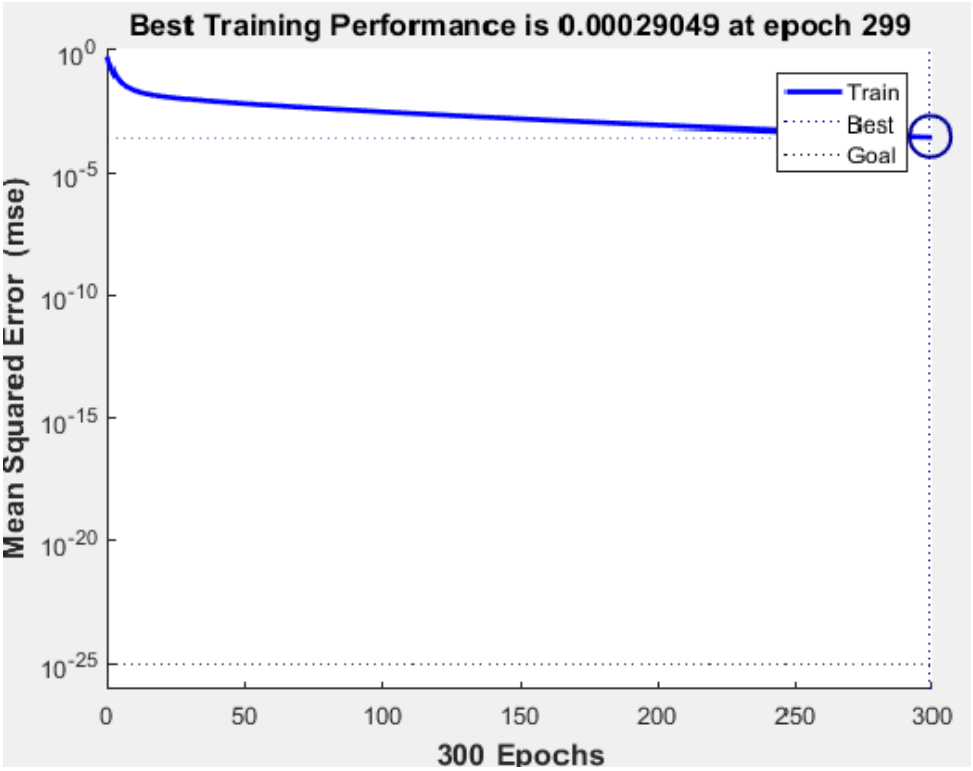
2. Funkcja newlin

Niestety wykonując funkcję newlin nie udało się uzyskać pożądanych wyników, dla żadnych parametrów.

```
przed2 =

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

po2 =

po2 =

  Columns 1 through 15

   -0.0018    0.0113    0.0172    0.0079   -0.0138   -0.0039    0.0095    0.0056    0.0021   -0.0168    1.0125    0.9979    0.9954    0.9854    1.0006

  Columns 16 through 20

    0.9963    1.0054    0.9848    0.9944    1.0139
```



Nawet wykonując program z dowolną ilością maksymalnych epok nie udało się uzyskać poprawnych wyników. Wyświetla się komunikat o osiągnięciu maksymalnej liczby epok, a mimo to uzyskane wartości są tylko zbliżone do pożądanych.



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 newp wynika, że zmiana poszczególnych czynników ma wypływ na parametry sieci. Jednak podczas wywołań programu dla współczynnika uczenia o mniejszej wartości czas uczenia oraz liczba epok nie zmieniała się znacząco, a nawet ilość iteracji wcale nie uległa zmianie. Z przedstawionych wykresów wynika, że najlepsza wydajność tez nie różni się znacząco. Po analizie wpływu ilości danych na proces uczenia przedstawionych w tabelach oraz na wykresach widać, że wraz ze wzrostem ilości danych liczba epok rośnie, natomiast czas w zasadzie utrzymuje się taki sam, jedynie dla skrajnych wartości jest odpowiednio mniejszy i większy. Natomiast najlepsza wydajność nie zależy od liczby danych, zawsze jest 0.5 i występuje w epoce 0. W każdym z przypadków błąd zmniejsza się wraz z kolejną iteracją, jedynie dla liczby danych równych 16 wartość między pierwszą a drugą epokę rośnie, ale ostatecznie błąd zmniejsza się dochodząc do epoki 4. Z tabeli przedstawiającej wpływ błędu i współczynnika uczenia na wynik 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 jeśli błąd jest zbyt duży wyniki będą niepoprawne. Z tabeli wynika, że już przy błędzie równym 0.1 wartości będą różne od pożądanych. Natomiast ilość epok oraz czas nie ma w tym przypadku większego znaczenia, ponieważ zwykle są to równe wartości. Przeważnie jest to 5 iteracji wykonujących się w czasie 2 sekund. Niestety jak widać z powyższych wyników działania oraz wykresu, tworząc sieć funkcją newlin nie udało się osiągnąć pożądanych wyników. Wykres dąży do wartości oczekiwanej, jednak jej nie osiąga, dlatego nie uzyskujemy prawidłowych wyników.

Wnioski

Siec jednowarstwowa to sieć gdzie perceptron zbudowany jest wyłącznie z warstwy wejściowej i wyjściowej. Zasada jego działania polega na dostosowywaniu wartości wyjścia do danych wejściowych. 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 newp bardzo dobrze nadaje się do rozwiązania problemu rozpoznawania wielkości liter, natomiast używając funkcji newlin nie

udało się uzyskać prawidłowych wyników. Wynika z tego, że algorytm wykorzystujący metodę wsadową jest gorszy od algorytmu cyklicznego w przypadku problemu rozpoznawania wielkości liter. Analizując wyniki działania funkcji newp 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. Natomiast wynik końcowy zależy znacząco od doboru parametru procesu uczenia, takiego jak błąd, natomiast współczynnik uczenia nie ma na to wpływu.

Listing kodu wykonanego programu

```
close all; clear all; clc;


  % zestaw uczący 10 małych i 10 dużych liter, gdzie nazwa oznacza
  % reprezentacje konkretnej litery


  % małe litery:
 z= [0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 1 1 1 1 1; 0 0 1 0 0; 0 1 0 0 0; 1 1 1 1 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];
 u= [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 0 0 1; 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];
 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];
 c= [0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 0 1 1 0 0; 1 0 0 0 0; 1 0 0 0 0; 0 1 1 0 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];
 x= [0 0 0 0 0; 0 0 0 0 0; 1 0 0 0 1; 0 1 0 1 0; 0 0 1 0 0; 0 1 0 1 0; 1 0 0 0 1];
 v= [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; 0 1 0 1 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];


 %duże litery:
 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];

  % transpozycja każdej tablicy(każdej litery)

  zz=transpose(z);
  ii=transpose(i);
```

```
  uu=transpose(u);
  kk=transpose(k);
  oo=transpose(o);
  cc=transpose(c);
  hh=transpose(h);
  xx=transpose(x);
  vv=transpose(v);
  ww=transpose(w);
  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);

   % wektor wejściowy złożony z dużych i małych liter
 wejscie=[zz(:) ii(:) uu(:) kk(:) oo(:) cc(:) hh(:) xx(:) vv(:) ww(:) AA(:) BB(:)
          CC(:) DD(:) EE(:) FF(:) GG(:) HH(:) II(:) JJ(:)];

  % wektor docelowy, gdzie 1 oznacza dużą literę, natomiast 0 małą
 wyjscie=[ 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1];

 % implementacja pierwszej funkcji newp tworzącej siec jednokierunkową
 % net-struktura(obiekt)w której zapisane są wszystkie informacje na temat
   utworzonej sieci

net=newp(minmax(wejscie), 1);

 % minmax(wejscie)- spodziewany zakresy poszczególnych wejść perceptronu,
 % 1-liczba perceptronów

 % inicjalizacja sieci
net=init(net);
 % wypisanie wartości przed treningiem (jeszcze nie nauczonych)
przedterningiem=sim(net, wejscie)
 %ustawienie liczby epok(iteracji)
net.trainParam.epochs = 200;
 % ustawienie wartości błędu
net.trainParam.goal =  0.0001;
 % ustawienie wartości współczynnika uczenia
net.trainParam.lr=0.01;
 %ustawianie wartości wag
net.IW{1}=[5 1 3 1 1 1 1 4 2 2 1 5 1 3 1 1 3 2 1 1 2 3 1 1 3 2 5 1 2 5 7 3 1 4 2];
 %wywołanie funkcji trenującej siec( użyta funkcja trainc)
net=train(net, wejscie, wyjscie);
 % wypisanie wartości po treningu
potreningu= sim(net, wejscie)

 % tworzenie sieci funkja newlin
 % net2-struktura(obiekt)w której zapisane są wszystkie informacje na temat
   utworzonej sieci


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

```
 % minmax(wejscie) -spodziewany zakresy poszczególnych wejść perceptronu,
 % 1-liczba perceptronów
 % 0-wektor opóźnień poszczególnych elementów wektora wejść sieci,
 % 0.01- stała szybkości uczenia

 % inicjalizacja sieci
net2 = init(net2);
 % wypisanie wartości przed treningiem
przed2= sim(net2, wejscie)
 % ustawienie liczby epok
net2.trainParam.epochs = 10000;
 % ustawienie wartości błędu
net2.trainParam.goal = 0.0001;
 %ustawianie wartości wag
net2.IW{1}=[5 1 3 1 1 1 1 4 2 2 1 5 1 3 1 1 3 2 1 1 2 3 1 1 3 2 5 1 2 5 7 3 1 4 2];
 % wywołanie funkcji trenującej siec(użyta funkcja trainb)
net2 = train(net2, wejscie, wyjscie);
 % wypisanie wartości po treningu
po2= sim(net2, wejscie)
```