

Patryk Kurzeja

Inżynieria Obliczeniowa

Nr albumu : 286112

Podstawy sztucznej inteligencji

## **Sprawozdanie : Budowa i działanie sieci jednowarstwowej [ Scenariusz 2 ] :**

### **1. Cel ćwiczenia :**

Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

### **2. Pojęcia i algorytmy :**

- **Sieci neuronowe** - są sztucznymi strukturami, których budowa i działanie zostały zaprojektowane w sposób modelujący działanie naturalnego układu nerwowego, w szczególności mózgu. Cieszą się bardzo dużym zainteresowaniem. Jednym z głównych czynników mających na to wpływ jest możliwość stosowania ich w bardzo wielu dziedzinach życia do rozwiązywania problemów, gdzie użycie innych metod jest trudne lub niemożliwe. Znakomicie sprawdzają się w problemach klasyfikacji, predykcji, związanych ze sterowaniem, analizą danych.
- **Sieci jednokierunkowe** - składają się z neuronów ułożonych w warstwach o jednym kierunku przepływu sygnałów i połączeniach między-warstwowych jedynie między kolejnymi warstwami. Sieć tego typu posiada warstwę wejściową, wyjściową i warstwę ukryte. Z funkcjonalnego punktu widzenia układ taki można traktować jako układ aproksymacji funkcji nieliniowej wielu zmiennych  $y = f(u)$ .
- **Funkcje używane do tworzenia jednokierunkowej sieci neuronowej :**
  - **Newp** - Funkcja tworzy jednowarstwową sieć neuronową, złożoną z zadanej liczby neuronów o funkcjach aktywacji „twardego” perceptronu (ang. hardlimit perceptron).

Wywołanie funkcji:  $NET = NEWP(PR, S, TF, LF)$

#### WEJŚCIE:

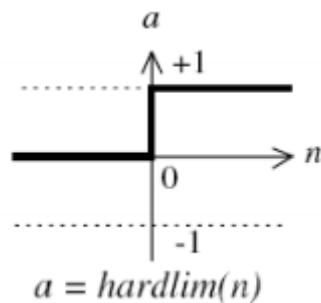
PR - macierz o wymiarach  $R \times 2$ , gdzie R jest liczbą wejść sieci (liczbą współrzędnych wektorów wejściowych); pierwsza kolumna macierzy R zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga kolumna – maksymalne wartości tych współrzędnych

S - liczba neuronów sieci

TF - nazwa funkcji aktywacji neuronów (zmienna tekstowa); nazwa domyślna = 'hardlim'; dopuszczalne wartości parametru TF to: 'hardlim' i 'hardlims'

LF - nazwa funkcji trenowania sieci perceptronowej (zmienna tekstowa); nazwa domyślna = 'learnp'; dopuszczalne wartości parametru LF to: 'learnp' i 'learnpn'

**hardlim** - funkcja skoku jednostkowego



**learnp** – perceptronowa reguła uczenia

- **Newlin** - Funkcja tworzy jednowarstwową sieć neuronową, złożoną z zadanej liczby neuronów o liniowych funkcjach aktywacji. Tego typu sieć jest zwykle wykorzystywana jako filtr adaptacyjny do przetwarzania sygnałów lub predykcji szeregów czasowych.

Wywołanie funkcji:  $\text{NET} = \text{NEWLIN}(\text{PR}, \text{S}, \text{ID}, \text{LF})$

#### WEJŚCIE:

PR - macierz o wymiarach  $R \times 2$ , gdzie R jest liczbą wejść sieci (tj. liczbą współrzędnych wektorów wejściowych); pierwsza kolumna zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga kolumna – maksymalne wartości tych współrzędnych

S - liczba neuronów sieci, tj. wyjść sieci (neurony tworzą automatycznie warstwę wyjściową)

ID - wektor opóźnień poszczególnych elementów wektora wejść sieci; domyślnie ID = [0]

LR - stała szybkości uczenia (treningu) sieci liniowej; domyślnie LR = 0.01

- **Reguła delta** – reguła uczenia z nauczycielem - Polega na tym, że każdy neuron po otrzymaniu na swoich wejściach określone sygnały (z wejść sieci albo od innych neuronów, stanowiących wcześniejsze piętra przetwarzania informacji) wyznacza swój sygnał wyjściowy wykorzystując posiadaną wiedzę w postaci wcześniej ustalonych wartości współczynników wzmocnienia (wag) wszystkich wejść oraz (ewentualnie) progu. Wartość sygnału wyjściowego, wyznaczonego przez neuron na danym kroku procesu uczenia porównywana jest z odpowiedzią wzorcową podaną

przez nauczyciela w ciągu uczącym. Jeśli występuje rozbieżność - neuron wyznacza różnicę pomiędzy swoim sygnałem wyjściowym a tą wartością sygnału, która była by - według nauczyciela prawidłowa. Ta różnica oznaczana jest zwykle symbolem greckiej litery d (delta) i stąd nazwa opisywanej metody.

### Schemat algorytmu :

Reguła delta obowiązuje dla neuronów z ciągłymi funkcjami aktywacji i nadzorowanego trybu uczenia. Regułę delta wyprowadza się jako wynik minimalizacji kryterium **błędu średniokwadratowego Q**.

$$Q = \frac{1}{2} \sum_{j=1}^N (z^j - y^j)^2 = \sum_{j=1}^N Q^j, \quad Q^j = \frac{1}{2} (\delta^j)^2$$

Korekta wag:

$$\Delta w_i = \eta \cdot \delta^j \cdot f'(e^j) \cdot x_i^j$$

gdzie  $f'()$  oznacza pochodną funkcji aktywacji. W przypadku funkcji sigmoidalnej:

$$\Delta w_i = \eta \cdot \delta^j \cdot (1 - y^j) \cdot y^j \cdot x_i^j$$

- **Algorytm wstecznej propagacji błędów** - Jest to podstawowy algorytm uczenia nadzorowanego wielowarstwowych jednokierunkowych sieci neuronowych. Podaje on przepis na zmianę wag w dowolnych połączeniach elementów przetwarzających rozmieszczonych w sąsiednich warstwach sieci. Oparty jest on na minimalizacji sumy kwadratów błędów uczenia z wykorzystaniem optymalizacyjnej metody największego spadku. Dzięki zastosowaniu specyficznego sposobu propagowania błędów uczenia sieci powstałych na jej wyjściu, tj. przesyłania ich od warstwy wyjściowej do wejściowej, algorytm propagacji wstecznej stał się jednym z najskuteczniejszych algorytmów uczenia sieci.

### Kroki algorytmu :

1. Pobierz ze zbioru uczącego parę wejście-oczekiwane wyjście.
2. Podaj na wejście neuronu wartości wejściowe.
3. Przepuść sygnał przez sieć obliczając wyjścia poszczególnych neuronów w kolejnych warstwach.
4. Oblicz sygnał wyjściowy neuronów z warstwy wyjściowej.
5. Oblicz błędy w neuronach warstwy wyjściowej -  $d = (o - y) \cdot f'(z)$ .
6. Korzystając ze wzoru: 
$$\delta_j = f'(z) \cdot \sum_{k=1}^K \delta_{(j+1)k} w_{(j+1)k}$$
 przepropaguj (wstecz) błędy przez całą sieć
7. Uaktualnij ( $w = w + \eta \cdot d \cdot x$ ) wszystkie wagi w sieci.
8. Powyższe kroki powtarzaj dla wszystkich par uczących.
9. Zakończ naukę gdy całkowity błąd dla wszystkich wzorców uczących będzie mniejszy od pewnej ustalonej na początku nauki wartości.

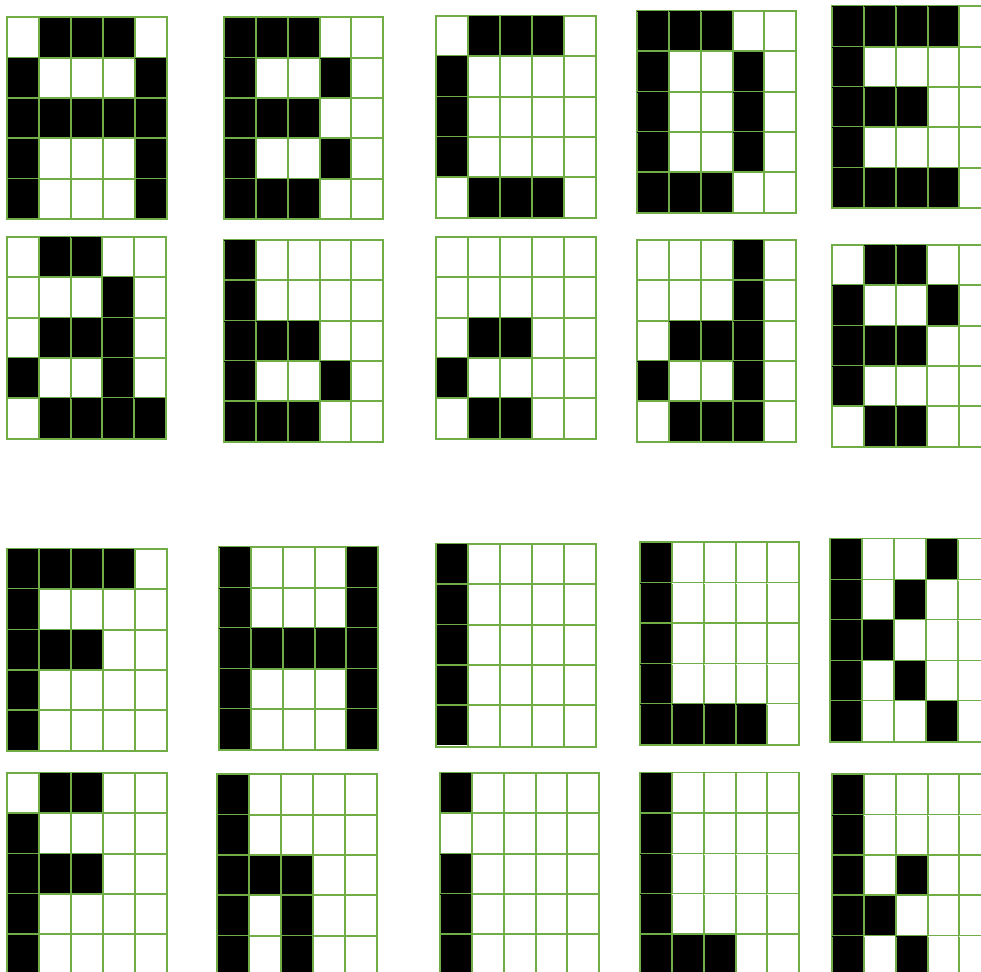
### 3. Wykonanie :

Wygenerowałem dane uczące, czyli 10 dużych i 10 małych liter alfabetu :

A, B, C, D, E, F, H, I, M, N.

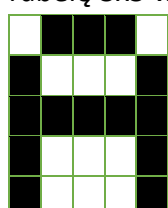
a, b, c, d, e, f, h, i, m, n.

Litery umieściłem w tablicach o rozmiarze 5x5 a następnie tablice przekształciłem w ciąg zer i jedynek, gdzie 1 oznacza pole czarne a 0 – pole białe ( puste ).



#### Procedura przekształcania liter do postaci binarnej :

- Wybieramy literę : np. A
- Tabelę 5x5 wypełniam 0 i 1 :

	<table><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	1	1	1	0	1	0	0	0	1	1	1	1	1	1	1	0	0	0	1	1	0	0	0	1
0	1	1	1	0																						
1	0	0	0	1																						
1	1	1	1	1																						
1	0	0	0	1																						
1	0	0	0	1																						

- A = 0 1 1 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1
- Każdą literę zapisuję w postaci 25 znakowego ciągu

- Dane wejściowe :

```
% Dane uczace :
%
A a B b C c D d E e F f H h I i L l K k
Input = [ 0 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1;
          1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0;
          1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0;
          1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0;
          0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
          1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1;
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0;
          1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
          1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1;
          1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0;
          1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1;
          1 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0;
          1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
          1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 1 0 0 0 0 1 0;
          0 1 1 1 0 0 1 1 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 1 0 0 0 0 0 0 0;
          1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1;
          0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0;
          0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 0 1;
          0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0;
          1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1];
```

Dane wejściowe wprowadziłem do sieci i przeprowadziłem jej uczenie dwoma metodami.

Dane wyjściowe to wartości, które możemy otrzymać, czyli 0 oznacza małą literę a 1 – dużą.

Sieć uczyła się rozróżniać małe i duże litery z zestawu tych wprowadzonych przeze mnie.

Sprawdziłem wydajność i przebieg procesu uczenia dla dwóch różnych metod.

Do funkcji symulacji sieci wprowadzałem po kolei duże i małe litery a program zwracał komunikat czy rozpoznana litera jest duża czy mała. Wyświetlana została również dokładność rozpoznawania.

### Listing kodu programu :

```
%% Sprawozdanie 2
close all; clear all; clc;

% Min i max wejsc sieci :
PR= [0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
      0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; ];

S = 1; % ilosc wyjsc z sieci

net = newlin(PR,S); % Algorytm 1
%net = newp(PR,S); % Algorytm 2

%
Input = [ A a B b C c D d E e F f H h I i L l K k
          0 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1;
          1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0;
          1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0;
          1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0;
          0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
          1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1;
          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1 1 1 0 0 1 1 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 1 0 0 0 0 0 0 0;
          1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1;
          0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0;
          0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 0 1;
          0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0;
          1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1];
```

```

0 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0;
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1;
1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 0;
1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1;
1 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0;
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 1 0 0 0 0 1 0;
0 1 1 1 0 0 1 1 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 1 0 0 0 0 0 0;
1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1;
0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0;
0 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 0 1;
0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0;
1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0; ];

```

```
Output = [ 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 ];
```

```

% Parametry uczenia się :
net.trainParam.epochs = 10000; % Max epochs
net.trainParam.goal = 0.1; % Bład sredniokwadratowy
net.trainParam.mu = 0.1; % Wspolczynnik uczenia

```

```

% Uczenie :
net = train(net, Input, Output);

```

```

% Dane testowe :
A = [ 0; 1; 1; 1; 0; 1; 0; 0; 0; 1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 0; 0; 0; 1 ];
a = [ 0; 1; 1; 0; 0; 0; 0; 0; 1; 0; 0; 1; 1; 1; 0; 1; 0; 0; 1; 0; 0; 1; 1 ];
B = [ 1; 1; 1; 0; 0; 1; 0; 0; 1; 0; 1; 1; 1; 0; 0; 1; 0; 0; 1; 0; 1; 1; 0 ];
b = [ 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 0; 0; 1; 0; 0; 1; 0; 1; 1; 0 ];
C = [ 0; 1; 1; 1; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 0 ];
c = [ 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 0; 0; 1; 0; 0; 0; 0; 0; 1; 0 ];
D = [ 1; 1; 1; 0; 0; 1; 0; 0; 1; 0; 1; 0; 0; 1; 0; 1; 0; 0; 1; 0; 1; 1; 0 ];
d = [ 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 1; 1; 0; 1; 0; 0; 1; 0; 0; 1; 0 ];
E = [ 1; 1; 1; 1; 0; 1; 0; 0; 0; 0; 1; 1; 1; 0; 0; 1; 0; 0; 0; 0; 1; 1; 0 ];
e = [ 0; 1; 1; 0; 0; 1; 0; 0; 1; 0; 1; 1; 1; 0; 0; 1; 0; 0; 0; 0; 0; 1; 0 ];
F = [ 1; 1; 1; 1; 0; 1; 0; 0; 0; 0; 1; 1; 1; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0 ];
f = [ 0; 1; 1; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0 ];
H = [ 1; 0; 0; 0; 1; 1; 0; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 0; 0; 1 ];
h = [ 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 0; 0; 1; 0; 0; 1; 0; 1; 0; 0 ];
I = [ 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0 ];
i = [ 1; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0 ];
L = [ 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 0 ];
l = [ 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 0 ];
K = [ 1; 0; 0; 1; 0; 1; 0; 1; 0; 0; 1; 1; 0; 0; 0; 1; 0; 1; 0; 0; 1; 0; 0 ];
k = [ 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 1; 0; 0; 1; 1; 0; 0; 0; 1; 0; 0 ];

```

```

% Symulacje :
Sym = sim(net, k);
if round (Sym) == 0, disp('Mala litera');
else disp('Duza litera');
end;

```

## Opis zmiennych :

**PR** - zmienna przechowująca minimalną i maksymalną wartość dla każdego z 25 wejść

**S** - zmienna przechowująca ilość wyjść z sieci

**net** - zmienna przechowująca sieć neuronową

**Input** - dane wejściowe - 10 dużych i 10 małych liter zapisanych w postaci 0 i 1

**Output** - dane wyjściowe - na przemian 1 i 0 ( Możliwe wyjścia )

**newlin(PR,S)** - funkcja tworząca jednowarstwową sieć neuronową

**newp(PR,S)** - funkcja tworząca jednowarstwową sieć neuronową, złożoną z zadanej liczby neuronów o funkcjach aktywacji „twardego” perceptronu

**net.trainParam.epochs** - maksymalna liczba epochs

**net.trainParam.goal** - Błąd średniokwadratowy

**net.trainParam.mu** - Współczynnik uczenia

**train(net, Input, Output)** - proces uczenia sieci z wykorzystaniem danych wejściowych oraz wyjściowych

**Sym = sim(net, A)** - symulacja sieci, jako zmienną podajemy literę, która ma zostać rozpoznana

## 4. Wyniki działania :

`Sym = sim(net, A);` -> Wprowadzamy dużą literę A

Wykresy dla wprowadzanej dużej litery A

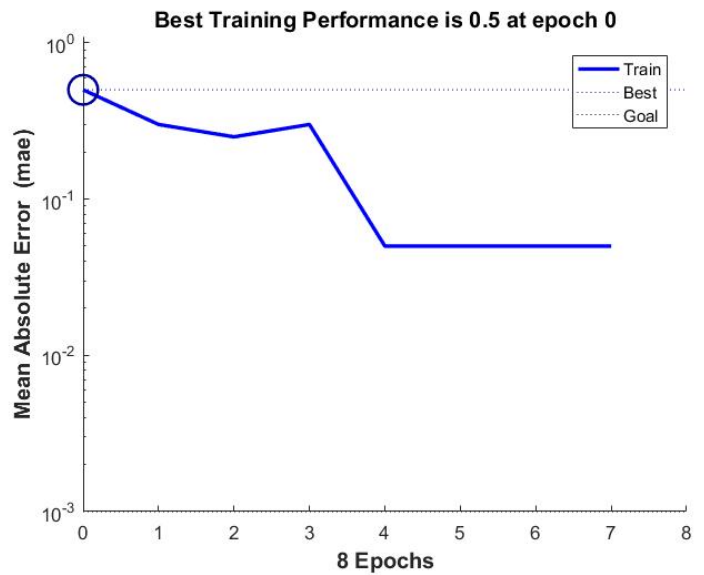
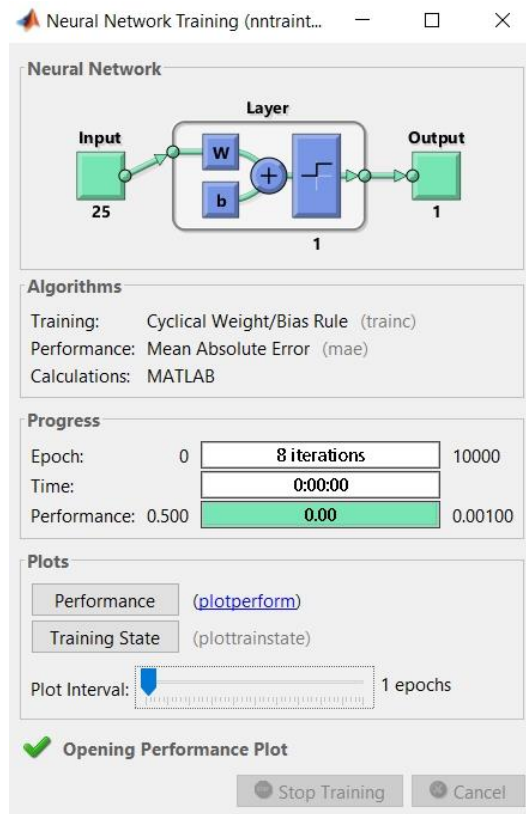
Dla ustalenia innych dokładności uczenia wprowadzamy inne litery

Funkcja :	newp			newlin		
<b>Błąd średniokw</b>	0.001	0.01	0.1	0.001	0.01	0.1
<b>Wsp uczenia się</b>	0.001	0.01	0.1	0.001	0.01	0.1
<b>Ilość epochs :</b>	8	8	4	5406	1792	13
<b>Dokładność Uczenia A :</b>	1	1	1	0.9835	0.9509	0.9642
<b>Dokładność Uczenia a :</b>	0	0	0	0.0152	0.0627	0.2351
<b>Dokładność Uczenia E :</b>	1	1	1	0.9927	1.0106	0.8434
<b>Dokładność Uczenia e :</b>	0	0	0	-0.0149	0.0299	0.3013
<b>Dokładność Uczenia K :</b>	1	1	1	0.9992	1.0041	0.8489
<b>Dokładność Uczenia k :</b>	0	0	0	-0.0005001	0.0010	0.3029

NEWP

Błąd = 0.001

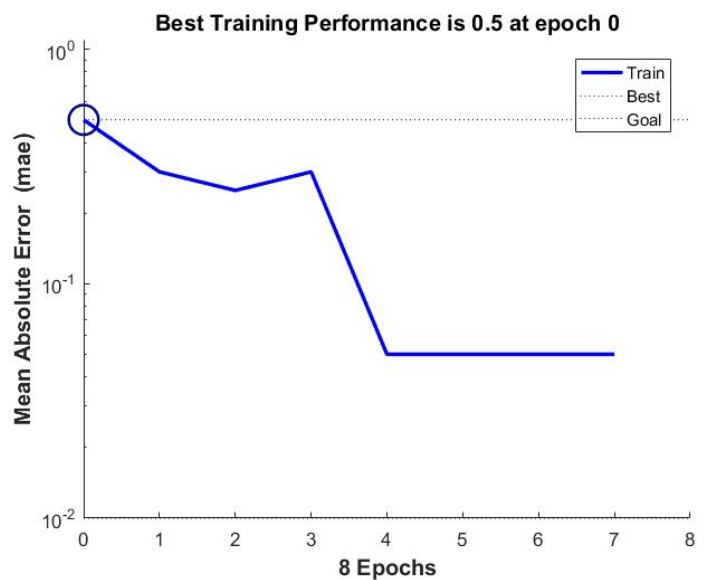
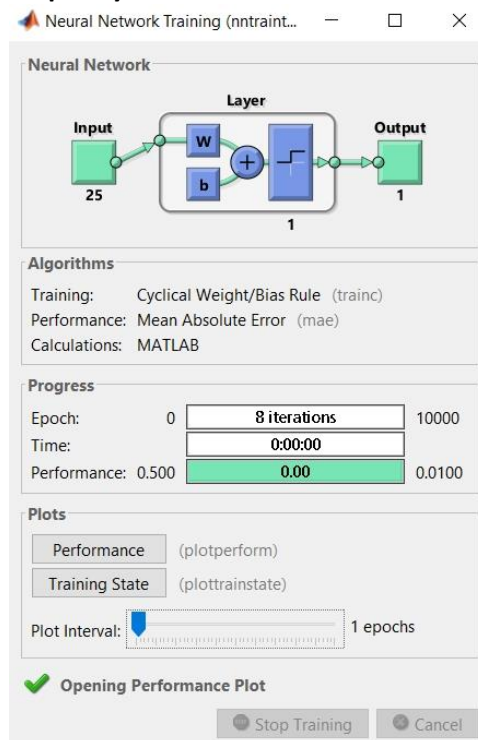
Współczynnik = 0.001



NEWP

Błąd = 0.01

Współczynnik = 0.01

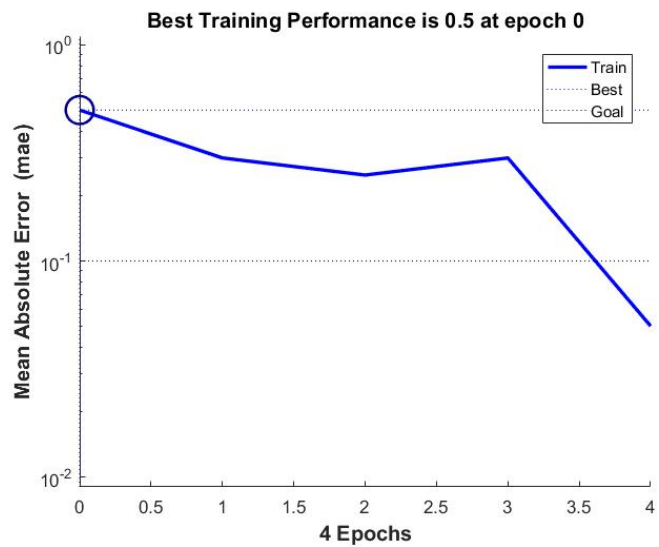
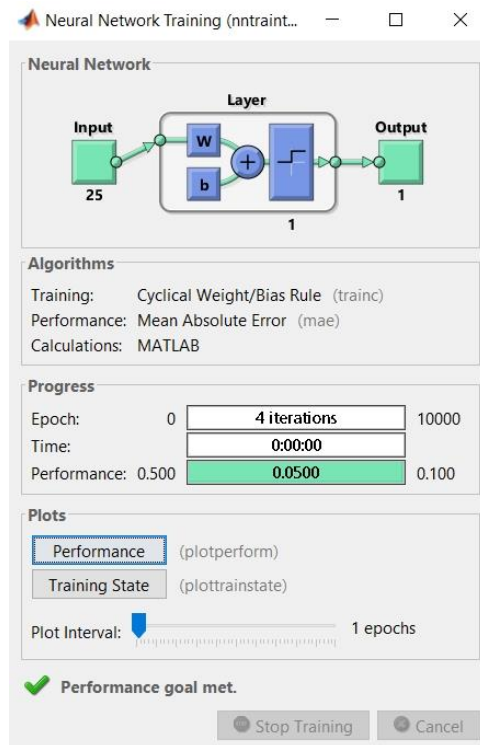




## NEWP

Błąd = 0.1

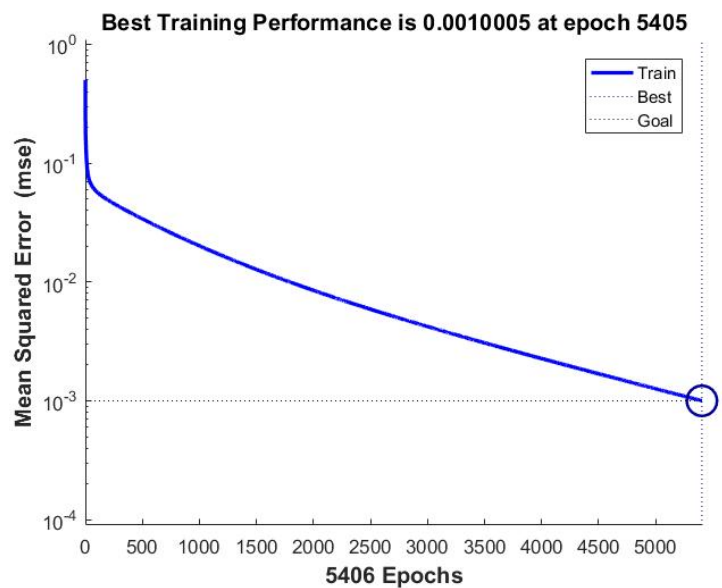
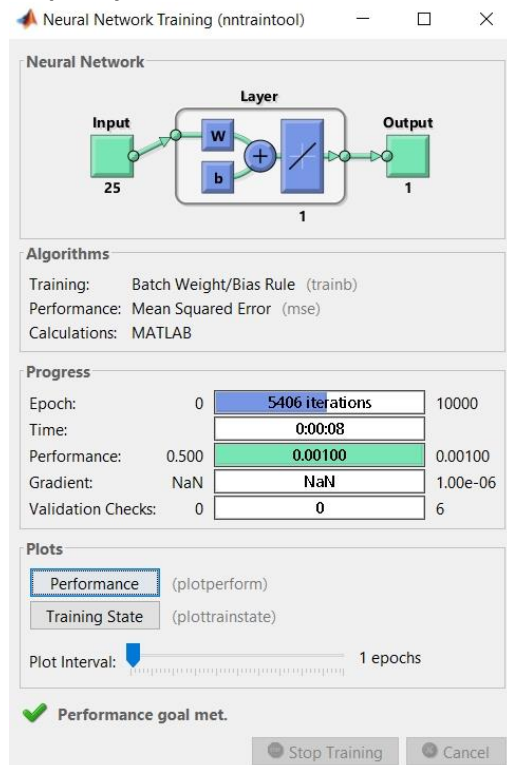
Współczynnik = 0.1



## NEWLIN

Błąd = 0.001

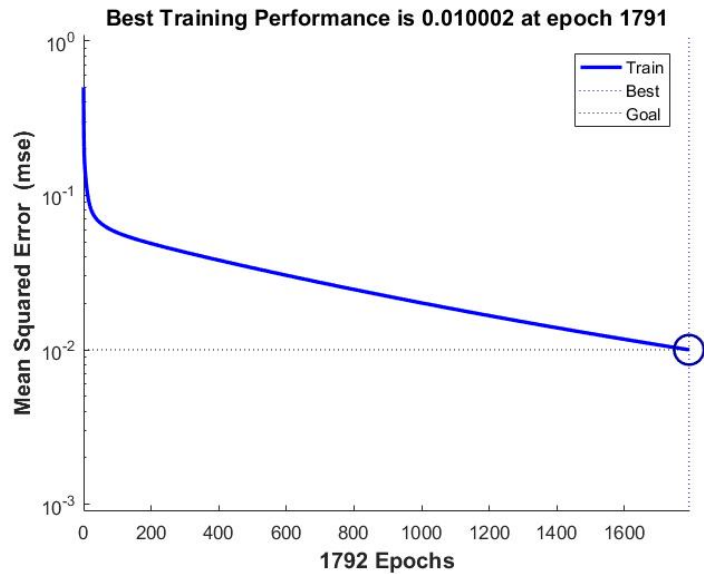
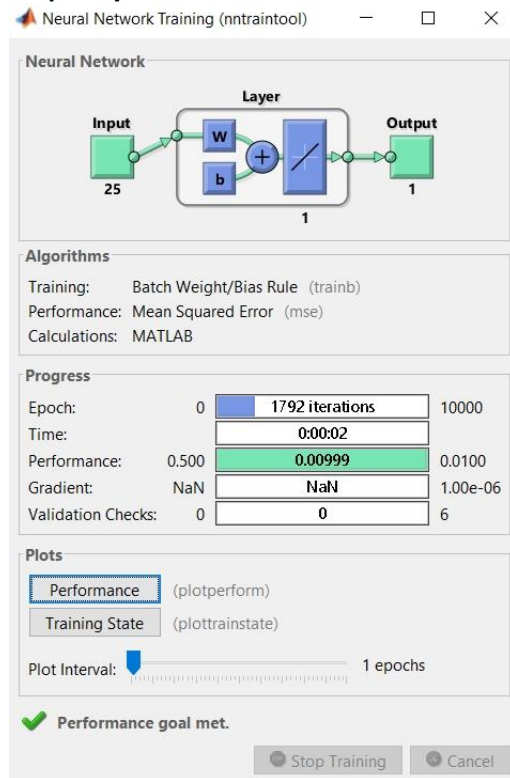
Współczynnik = 0.001



NEWLIN

Błąd = 0.01

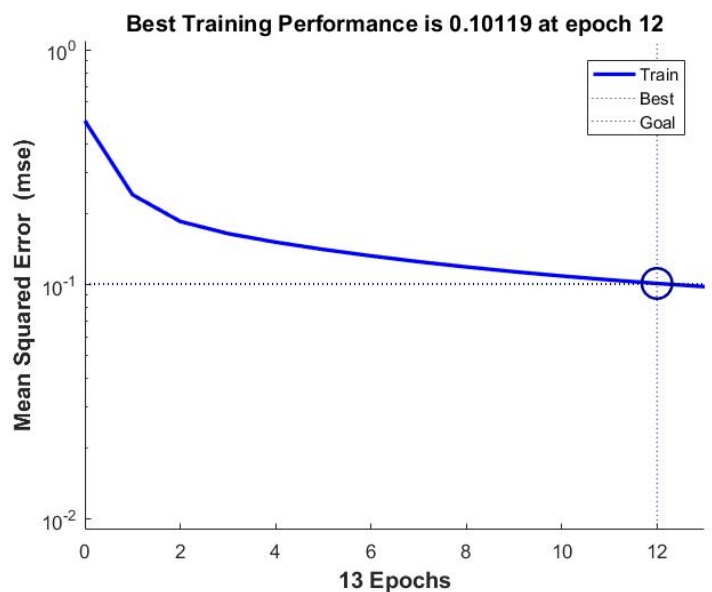
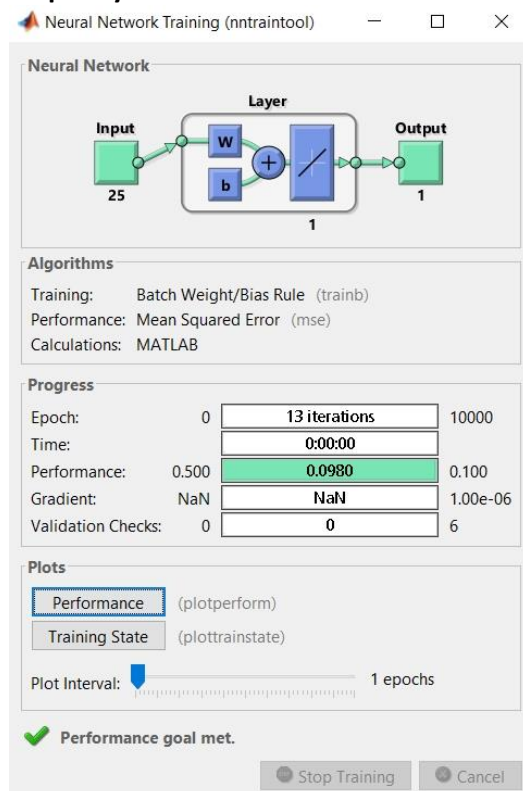
Współczynnik = 0.01



NEWLIN

Błąd = 0.1

Współczynnik = 0.1



## 5. Analiza :

Po przeprowadzeniu pomiarów dla dwóch algorytmów i różnych parametrów uczenia możemy zauważyć, że podczas użycia funkcji newp, dokładność uczenia wynosi 0 lub 1 co oznacza dokładnie otrzymanie małej ( 0 ) lub dużej ( 1 ) litery. Ilość iteracji potrzebnych do nauki dla błędu średniokwadratowego oraz współczynnika uczenia się równych 0.001 wynosi 8. Identycznie jest w przypadku gdy zwiększymy je do 0.01. Natomiast gdy ustalimy ich wartość na 0.1 to zauważamy, że ilość iteracji spada o połowę z 8 do 4.

Natomiast podczas użycia funkcji newlin, możemy zauważyć, że proces nauki przebiega znacznie wolniej i nie jest w stu procentach dokładny. Dla błędu średniokwadratowego równego 0.001 i współczynnika uczenia się na poziomie 0.001 dokładność dla dużej litery A wynosi 0.9835 czyli prawie 1 co oznacza że wprowadzona litera jest duża.

Ilość potrzebnych iteracji ( epochs ) do nauki wynosi w tym przypadku aż 5406.

Po zwiększeniu parametrów do 0.01 ilość iteracji spada do 1792. Spada również dokładność, która wynosi wtedy 0.9509. Podczas ostatniej próby ustaliłem największy błąd równy 0.1 oraz taki sam współczynnik uczenia się. Iteracje spadły do 13, lecz uzyskana dokładność 0.9642 była większa niż w poprzednim przypadku.

## 6. Wnioski :

Po przeprowadzonej analizie otrzymanych wyników, możemy stwierdzić, że funkcja newp jest znacznie szybsza i dokładniejsza od funkcji newlin. Potrzebuje ona zaledwie kilka iteracji do pełnego nauczania sieci, podczas gdy funkcja newlin potrzebowała na to zależnie od parametrów nawet 5406 iteracji. Funkcja newp daje znacznie wyższe dokładności procesu nauki przy tej samej lub mniejszej liczbie iteracji. Sprawdzając poszczególne litery otrzymywaliśmy różne wartości parametru dokładności nauki. Wynika to z podobieństw pomiędzy niektórymi literami. Wartość dokładności procesu nauki dla wprowadzanych małych i dużych liter oscylowała w pobliżu 0 dla małych liter oraz w pobliżu 1 dla liter dużych.

Jednak obydwa algorytmy rozpoznały każdą literę prawidłowo z lepszą lub gorszą dokładnością. Jeżeli chcemy uzyskać lepsze dokładności nauczania, powinniśmy zmniejszać błąd średniokwadratowy oraz skorzystać z decyzji z funkcji newp, która wykorzystuje funkcje aktywacji „twardego” perceptronu . Daje ona pełne wyniki, które są liczbami całkowitymi {0;1} w przeciwieństwie do funkcji newlin, która daje już liczby niecałkowite.