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 warstwy ukryte. Z funkcjonalnego punktu widzenia układ taki można traktować jako układ aproksymacji funkcji nieliniowej wielu zmiennych y = f(u).

- **Algorytm Levenberga – Marquardta** - modyfikuje wagi w sposób grupowy, czyli po podaniu wszystkich wektorów uczących. Jest on jednym z najbardziej efektywnych algorytmów do uczenia sieci, który łączy w sobie zbieżność algorytmu Gaussa - Newtona blisko minimum, z metodą najszybszego spadku, która bardzo szybko zmniejsza błąd, gdy rozwiązanie jest dalekie.

  Schemat algorytmu :
  1. Ustawić początkową wartość μ oraz β (np. μ = 0.01 i β = 10),
  2. Podawać na sieć wektory uczące:
     a. podać wektor m,
     b. obliczyć wyjścia sieci,
     c. obliczyć błąd sieci dla danej próbki i neuronu w ostatniej warstwie - $, \varepsilon_i^I(m)$
     Zapamiętać te błędy w wektorze :

$$\boldsymbol{\varepsilon}(\mathbf{W}) = \begin{bmatrix} \varepsilon_1^L(1) \\ \varepsilon_2^L(1) \\ \vdots \\ \varepsilon_{N^L}^L(1) \\ \varepsilon_1^L(2) \\ \vdots \\ \varepsilon_{N^L}^L(2) \\ \vdots \\ \varepsilon_{N^L}^L(P) \end{bmatrix}$$

d. jeżeli podany został ostatni wektor uczący, należy obliczyć miarę błędu sieci :

$$E(\mathbf{W}) = \sum_{m=1}^{P} Q_m = \sum_{m=1}^{P} \sum_{i=1}^{N^L} \left[d_i^{(L)}(m) - y_i^{(L)}(m) \right]^2$$

i przejść do kroku 3.

3. Zapamiętać wszystkie wagi sieci neuronowej.

4. Obliczyć jakobian sieci dany wzorem :

$$\mathbf{J}(\mathbf{W}) = \begin{bmatrix} \frac{\partial \varepsilon_1^L(1)}{\partial w_{10}^{(1)}} & \frac{\partial \varepsilon_1^L(1)}{\partial w_{11}^{(1)}} & \cdots & \frac{\partial \varepsilon_1^L(1)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial \varepsilon_1^L(1)}{\partial w_{N^L N^{L-1}}^{(L)}} \\ \frac{\partial \varepsilon_2^L(1)}{\partial w_{10}^{(1)}} & \frac{\partial \varepsilon_2^L(1)}{\partial w_{11}^{(1)}} & \cdots & \frac{\partial \varepsilon_2^L(1)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial \varepsilon_2^L(1)}{\partial w_{N^L N^{L-1}}^{(L)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \varepsilon_{N^L}^L(1)}{\partial w_{10}^{(1)}} & \frac{\partial \varepsilon_{N^L}^L(1)}{\partial w_{11}^{(1)}} & \cdots & \frac{\partial \varepsilon_{N^L}^L(1)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial \varepsilon_{N^L}^L(1)}{\partial w_{N^L N^{L-1}}^{(L)}} \\ \frac{\partial \varepsilon_1^L(2)}{\partial w_{10}^{(1)}} & \frac{\partial \varepsilon_1^L(2)}{\partial w_{11}^{(1)}} & \cdots & \frac{\partial \varepsilon_1^L(2)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial \varepsilon_1^L(2)}{\partial w_{N^L N^{L-1}}^{(L)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \varepsilon_{N^L}^L(2)}{\partial w_{10}^{(1)}} & \frac{\partial \varepsilon_{N^L}^L(2)}{\partial w_{11}^{(1)}} & \cdots & \frac{\partial \varepsilon_{N^L}^L(2)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial \varepsilon_{N^L}^L(2)}{\partial w_{N^L N^{L-1}}^{(L)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \varepsilon_{N^L}^L(P)}{\partial w_{10}^{(1)}} & \frac{\partial \varepsilon_{N^L}^L(P)}{\partial w_{10}^{(1)}} & \cdots & \frac{\partial \varepsilon_{N^L}^L(P)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial \varepsilon_{N^L}^L(P)}{\partial w_{N^L N^{L-1}}^{(L)}} \end{bmatrix}$$

5. Obliczyć nowe wagi sieci używając równania :

$$\mathbf{W}(n+1) = \mathbf{W}(n) - \left[\mathbf{J}^T(\mathbf{W}(n)) \mathbf{J}(\mathbf{W}(n)) + \mu \mathbf{I} \right]^{-1} \mathbf{J}^T(\mathbf{W}(n)) \boldsymbol{\varepsilon}(\mathbf{W}(n))$$

6. Podawać na sieć wektory uczące i obliczyć miarę błędu daną wzorem :

$$E(\mathbf{W}) = \sum_{m=1}^{P} Q_m = \sum_{m=1}^{P} \sum_{i=1}^{N^L} \left[d_i^{(L)}(m) - y_i^{(L)}(m) \right]^2$$

7. Jeżeli nowy błąd jest mniejszy, niż obliczony w punkcie 2d, to: zachować wagi sieci, wykonać i wrócić do kroku 2.

Jeżeli błąd z punktu 2d jest większy niż nowy to wykonać i przejść do punktu 5.

8. Algorytm osiągnął rozwiązanie, gdy błąd sieci zmniejszył się do określonej wcześniej wartości.

- **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.**

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

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:

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

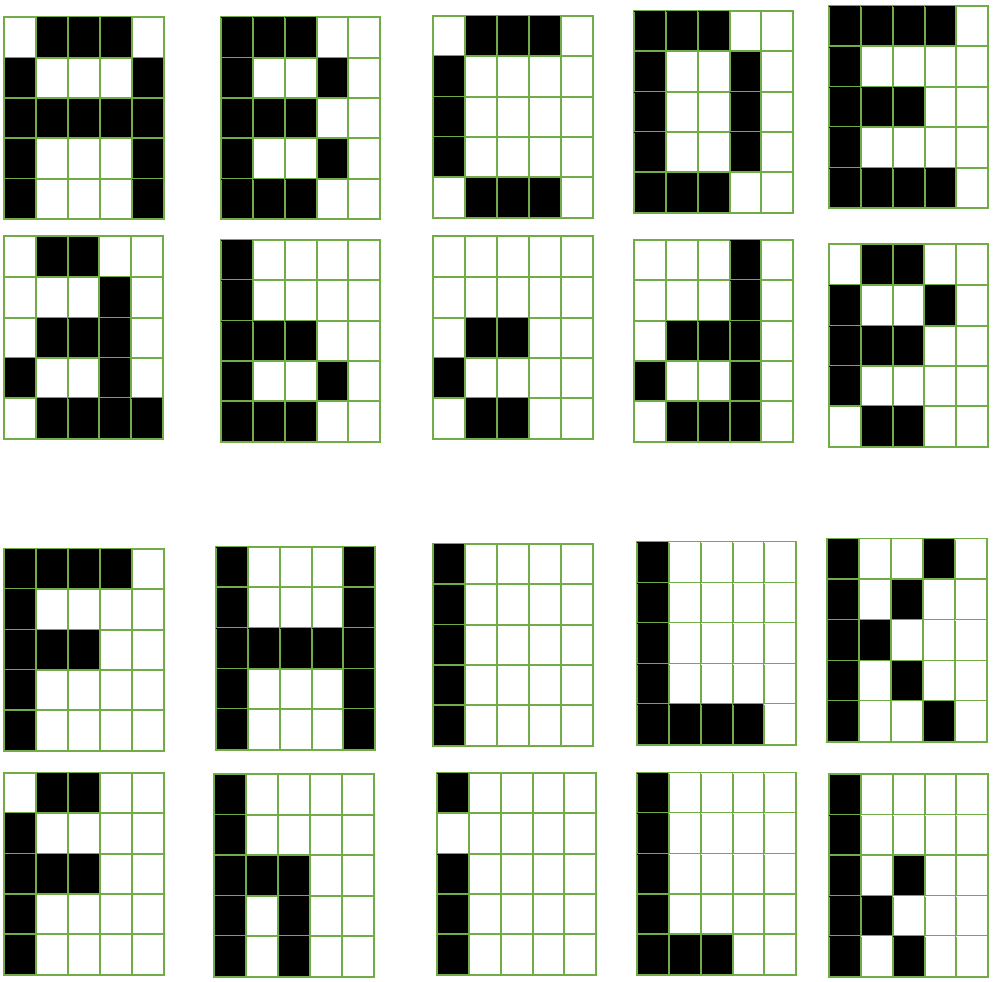
3. Wykonanie :

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

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

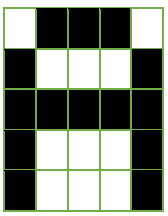
a, b, c, d, e, f, h, i, m, n.

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 :



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 :

```
%         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 0;
          1 1 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 1 0;
          1 1 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 1 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 1 0;
          1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 0 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 0;
          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 1 0 0 0 0 0 1 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 0 0 0 0 0 1;
          1 1 1 1 0 1 0 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 1 0 0 0 0 0 1 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 0;
          0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1;
          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 1 0 0 0 0 0 1 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 1 1 0 0 0 0 0 0 1 1 1;
          0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 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 1 0; ];
```

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; 0 1; 0 1; ];

S = 1; % ilosc wyjsc z sieci

net = newlin(PR,S); % Algorytm 1
%net = newff(PR,S,{'tansig'},'trainlm'); % Algorytm 2

%         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 0;
       1 1 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 1 0;
       1 1 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 1 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 1 0;
```

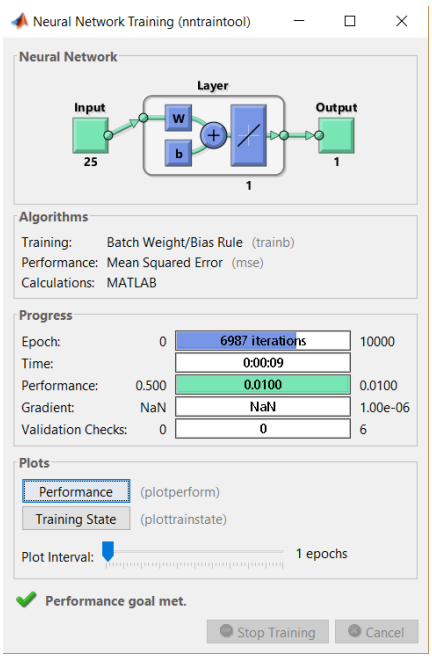
```
        1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 0;
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
        0 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0;
        1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0;
        1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1;
        1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 0 1;
        1 1 1 1 0 1 0 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 1 0 0 0 0 0 1 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 0;
        0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1;
        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 1 0 0 0 0 0 1 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 1 0 0 0 0 0 0 1 1 1 1;
        0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 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 1 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 = 0.01; % Blad sredniokwadratowy
net.trainParam.mu = 0.001; % 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; 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; 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; 1; 0; 0 ];
b = [ 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 0; 0; 1; 0; 0; 1; 0; 1; 1; 1; 0; 0 ];
C = [ 0; 1; 1; 1; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 0; 1; 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; 1; 0; 0 ];
D = [ 1; 1; 1; 0; 0; 1; 0; 0; 1; 0; 1; 0; 0; 1; 0; 1; 0; 0; 1; 0; 1; 1; 1; 0; 0 ];
d = [ 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 0; 1; 1; 1; 0; 1; 0; 0; 1; 0; 0; 1; 1; 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; 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; 1; 0; 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; 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; 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; 0; 1 ];
h = [ 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1; 0; 0; 1; 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; 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; 0; 0 ];
L = [ 1; 1; 1; 1; 0; 1; 0; 0; 1; 0; 1; 1; 1; 1; 0; 1; 1; 0; 0; 0; 1; 0; 1; 0; 0 ];
l = [ 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 0; 0; 1; 0; 0; 1; 0; 1; 0; 0; 0; 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; 1; 0 ];
k = [ 1; 0; 0; 0; 0; 1; 0; 0; 0; 0; 1; 0; 1; 0; 0; 1; 1; 0; 0; 0; 1; 0; 1; 0; 0 ];
% Symulacje :
Sym = sim(net, A);
if round (Sym) == 0, disp('Mala litera');
else disp('Duza litera');
disp (Sym)
end;
```

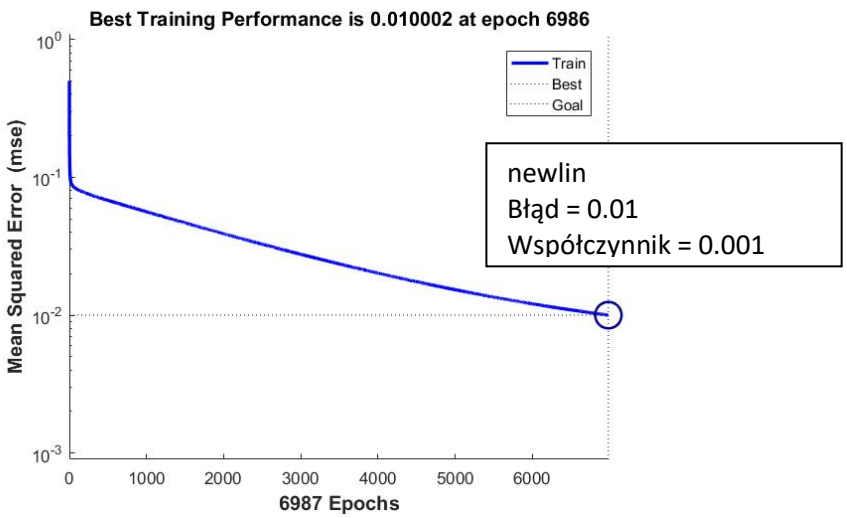
Opis zmiennych :

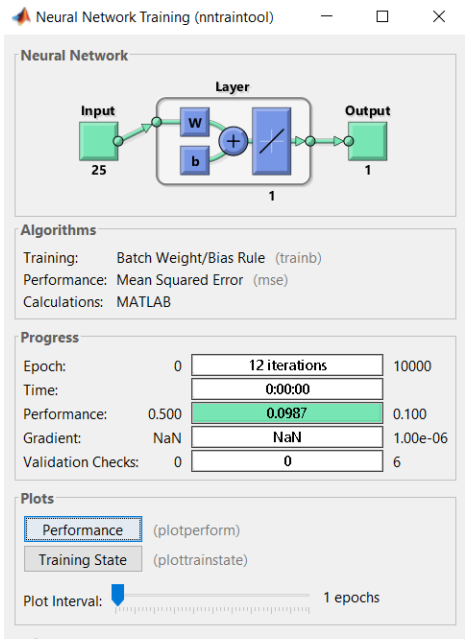
PR - zmienna przechowująca minimalną i maksymalną wartość dla każdego z 25 wejść
S - zmienna przechowująca ilość wyjść z sieci
net - zmienna przechowująca sieć neuronową
Input - dane wejściowe - 10 dużych i 10 małych liter zapisanych w postaci 0 i 1
Output - dane wyjściowe - na przemian 1 i 0 (Możliwe wyjścia)
newlin(PR,S) - funkcja tworząca jednowarstwową sieć neuronową o szybkości uczenia 0.01
newff(PR,S,{'tansig'},'trainlm') - stworzenie jednowarstwowej sieci neuronowej :
tansig - funkcja aktywacji neutronów - tangens hiperboliczny,
trainlnm - metoda Levenberga-Marquardta
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

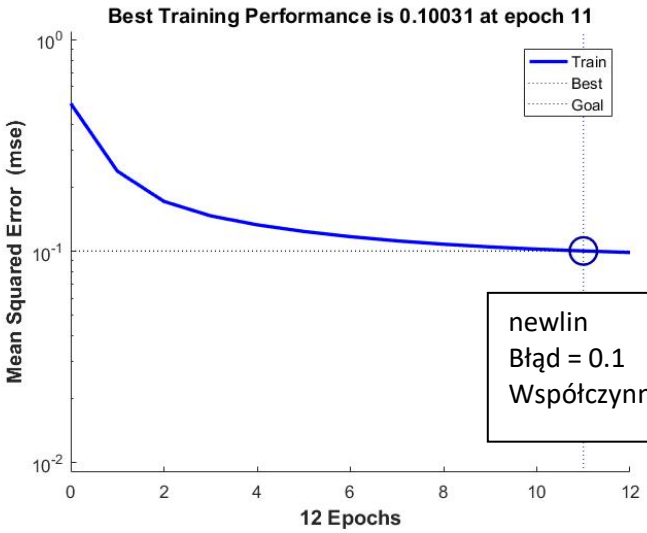
4. Wyniki działania :

Funkcja :	newlin		newff		
Błąd średniokw :	0.01	0.1	0.01	0.1	0.001
Wsp uczenia się :	0.001	0.01 i 0.001	0.001	0.01	0.001
Ilość epochs :	6987	12	3	2	6
Dokładność (Litera A) :	0.9824	0.9140	0.9211	0.8910	0.9883
Litera b :	0.1486	0.3419	-4.21439017017633 × 10^-9	-0.1054	1.910649416458909×10^-11
Litera H :	0.9020	0.7457	0.9450	0.7473	0.9950
Litera f :	0.0890	0.4795	3.177837104573200×10^-9	0.2419	3.171889417785678×10^-10
Litera C :	0.8438	0.7174	0.9997	0.8422	0.9948
Litera a :	0.01796	0.2248	3.430652511109411×10^-4	0.0417	2.944307908592236×10^-9

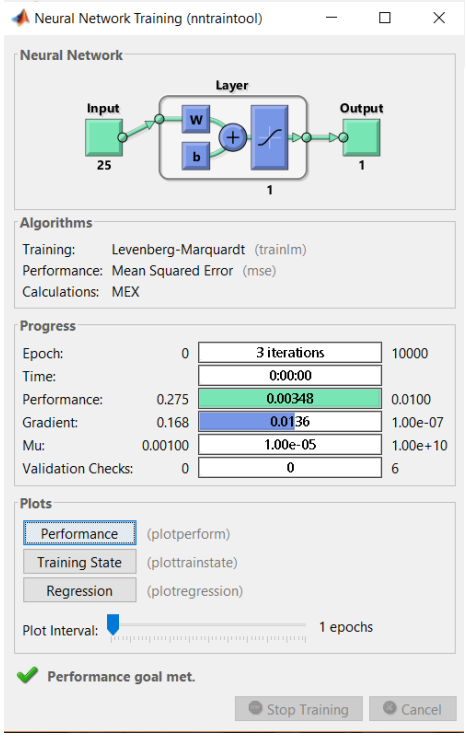


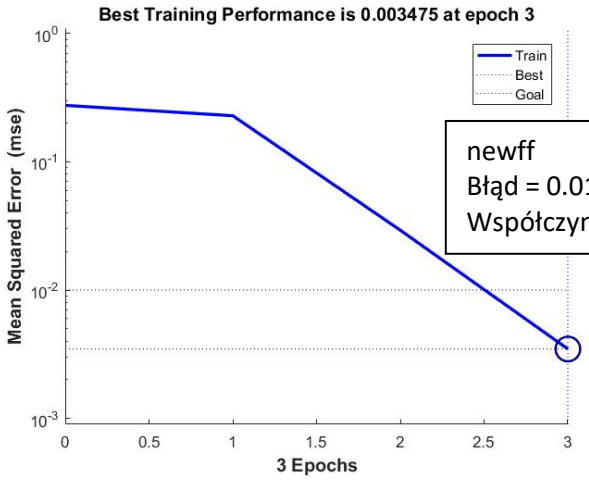




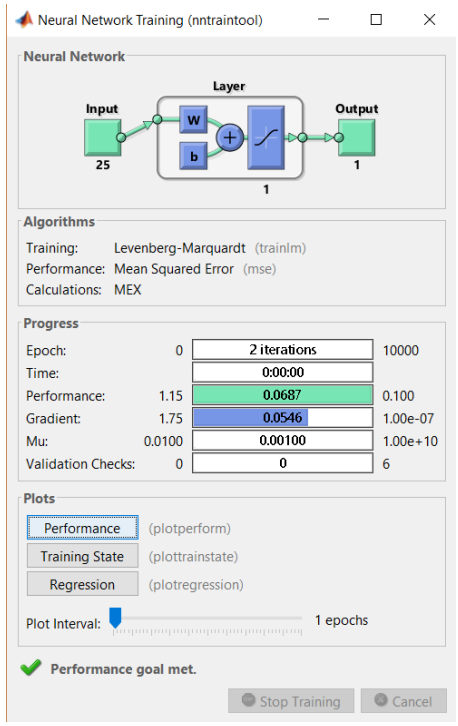


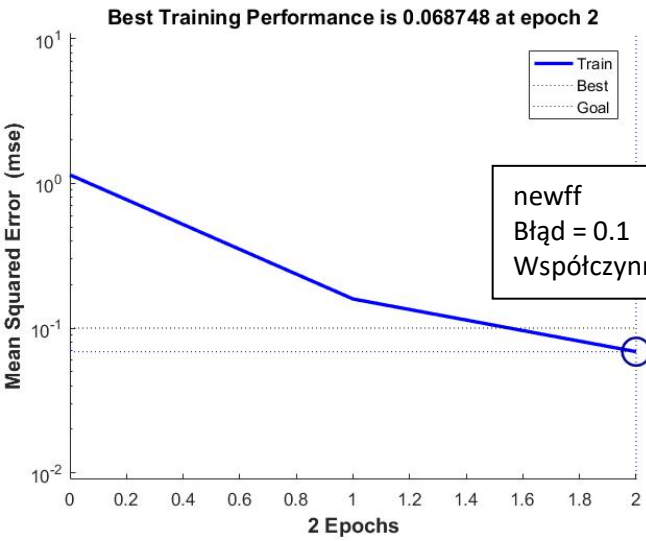
newlin
Błąd = 0.1
Współczynnik = 0.001 i 0.01



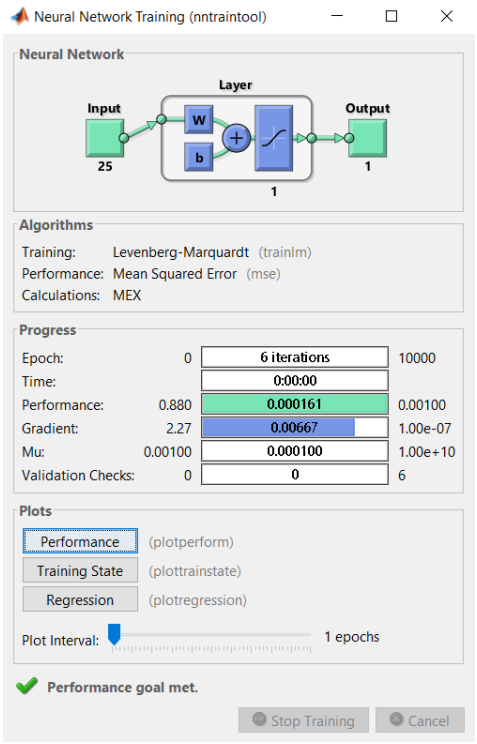


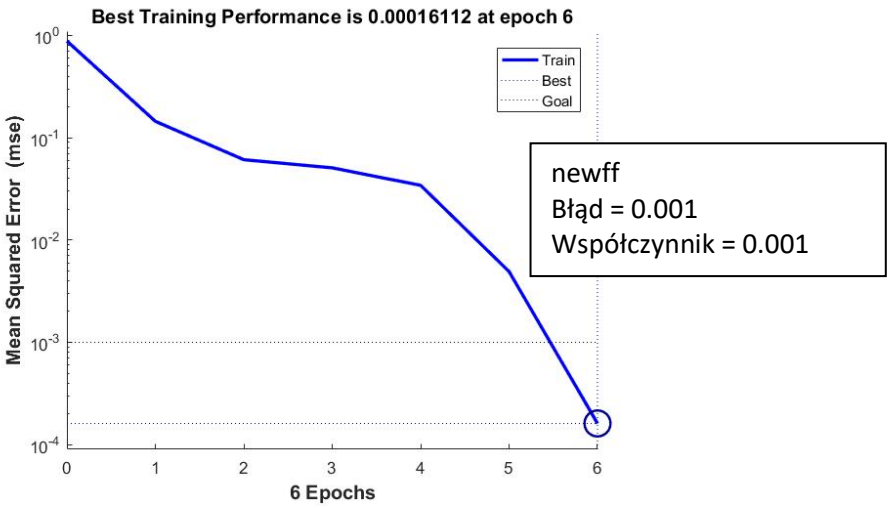
newff
Błąd = 0.01
Współczynnik = 0.001





newff
Błąd = 0.1
Współczynnik = 0.01





5. Analiza :

Po przeprowadzeniu pomiarów dla dwóch algorytmów i różnych parametrów uczenia możemy zaważyć, że współczynnik uczenia się nie ma żadnego wpływu na funkcję newlin. Zmiana błędu średniokwadratowego wpływa już na proces nauki. Przy mniejszym błędzie średniokwadratowym wynoszącym 0.01 dokładność nauki wzrasta do 0.9824. Lecz ilość iteracji (epochs) wynosi aż 6987. Gdy zwiększymy błąd średniokwadratowy do 0.1 to już dokładność wynosi 0.9140 a ilość iteracji spada do 12.

Natomiast podczas użycia funkcji new, która wykorzystuje algorytm Levenberga-Marquardta, możemy zauważyć, że proces nauki jest dużo efektywniejszy i przebiega wielokrotnie szybciej. Dla błędu średniokwadratowego równego 0.01 i współczynnika uczenia się na poziomie 0.001 dokładność wynosi 0.9211. Ilość potrzebnych iteracji (epochs) do nauki wynosi już tylko 3. Po zwiększeniu błędu do 0.1 a współczynnika do 0.01 ilość iteracji spada do 2. Spada również dokładność, która wynosi wtedy 0.8910. Podczas ostatniej próby ustaliłem najmniejszy błąd 0.001 oraz taki sam współczynnik uczenia się. Iteracje wzrosły do 6, lecz uzyskaliśmy wyższą dokładność uczenia równą 0.9883.

6. Wnioski :

Po przeprowadzonej analizie otrzymanych wyników, możemy stwierdzić, że algorytm Levenberga-Marquardta jest znacznie szybszy i dokładniejszy od algorytmu wykorzystującego błąd średniokwadratowy (funkcja newlin). Potrzebuje on zaledwie kilka iteracji do pełnego nauczenia sieci, podczas gdy funkcja newlin potrzebowała na to zależnie od parametrów nawet 6987 iteracji. Algorytm Levenberga-Marquardta 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. Jednak obydwa algorytmy rozpoznały każdą literę prawidłowo z lepszą lub gorszą dokładnością. Jeżeli chcemy uzyskać lepsze dokładności nauczania, powinniśmy zmniejszać błąd średniokwadratowy. Wpłynie to na zwiększenie liczby iteracji, lecz podczas użycia algorytmu Levenberga-Marquardta zmiany te będą niezauważalne a dokładność nauki znacznie wzrośnie.