Akademia Górniczo – Hutnicza

WIMiIP Inżynieria Obliczeniowa

20.11.2017, Kraków

Podstawy Sztucznej Inteligencji

Budowa i działanie sieci wielowarstwowej

sprawozdanie z projektu 3:

Agnieszka Majkut

nr indeksu 286116

Wprowadzenie:

Celem projektu było poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie z użyciem algorytmu wstecznej propagacji błędu pozpoznawania konkretnych liter alfabetu.

Zadania do wykonania:

- wygenerowanie danych uczących i testujących zawiaracjących 20 wielkich dowolnie wybranych liter alfabetu
- przygotowanie wielowarstwowej sieci oraz algorytmu wstecznej propagacji błędu
- uczenie sieci dla różnych współczynników uczenia
- testowanie sieci

Sieć neuronowa (sztuczna sieć neuronowa) – ogólna nazwa struktur matematycznych i ich programowych lub sprzętowych modeli, realizujących obliczenia lub przetwarzanie sygnałów poprzez rzędy elementów, zwanych sztucznymi neuronami, wykonujących pewną podstawową operację na swoim wejściu. Oryginalną inspiracją takiej struktury była budowa naturalnych neuronów, łączących je synaps, oraz układów nerwowych, w szczególności mózgu.

Sieć wielowarstwowa - składa się zwykle z jednej warstwy wejściowej, kilku warstw ukrytych oraz jednej warstwy wyjściowej. Warstwy ukryte składają się najczęściej z neuronów McCullocha-Pittsa.

Uczenie sieci neuronowych - wymuszenie na niej określonej reakcji na zadane sygnały wejściowe. Uczenie jest konieczne tam, gdzie brak jest informacji doświadczalnych o powiązaniu wejścia z wyjściem lub jest ona niekompletna, co uniemożliwia szczegółowe zaprojektowanie sieci. Uczenie może być realizowane krok po kroku lub poprzez pojedynczy zapis. Istotnym czynnikiem przy uczeniu jest wybór odpowiedniej strategii (metody) uczenia. Wyróżnić możemy dwa podstawowe podejścia: uczenie z nauczycielem (supervised learning) i uczenie bez nauczyciela (unsupervised learning).

Uczenie metodą wstecznej propagacji błędów - uczenie z nauczycielem - pierwszą czynnością w procesie uczenia jest przygotowanie dwóch ciągów danych: uczącego i weryfikującego. Ciąg uczący jest to zbiór takich danych, które w miarę dokładnie charakteryzują dany problem. Jednorazowa porcja danych nazywana jest wektorem uczącym. W jego skład wchodzi wektor wejściowy czyli te dane wejściowe, które podawane są na wejścia sieci i wektor wyjściowy czyli takie dane oczekiwane, jakie sieć powinna wygenerować na swoich wyjściach. Po przetworzeniu wektora wejściowego, nauczyciel porównuje wartości otrzymane z wartościami oczekiwanymi i informuje sieć czy odpowiedź jest poprawna, a jeżeli nie, to jaki powstał błąd odpowiedzi. Błąd ten jest następnie propagowany do sieci ale w odwrotnej niż wektor wejściowy kolejności (od warstwy wyjściowej do wejściowej) i na jego podstawie następuje taka korekcja wag w każdym neuronie, aby ponowne przetworzenie tego samego wektora wejściowego spowodowało zmniejszenie błędu odpowiedzi. Procedurę taką powtarza się do momentu wygenerowania przez sieć błędu mniejszego niż założony. Wtedy na wejście sieci podaje się kolejny wektor wejściowy i powtarza te czynności. Po przetworzeniu całego ciągu uczącego (proces ten nazywany jest epoką) oblicza się błąd dla epoki i cały cykl powtarzany jest do momentu, aż błąd ten spadnie poniżej dopuszczalnego.

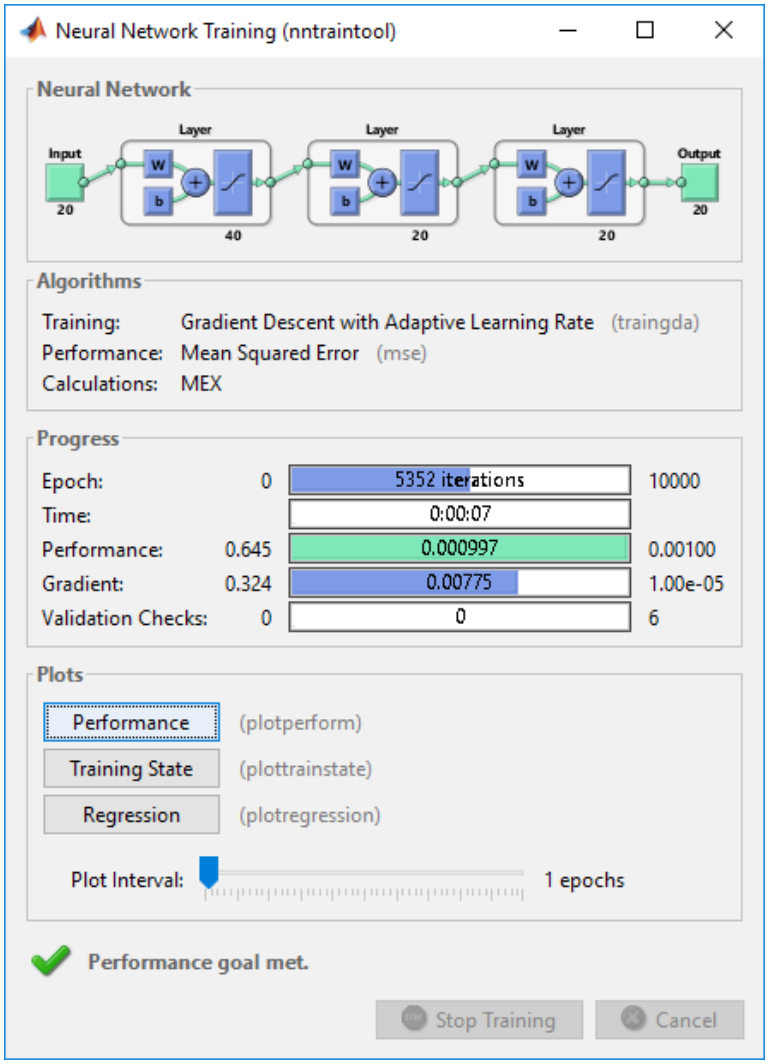
W programie Matlab, za pomocą biblioteki Nueral Network Toolbox, zaimplementowałam sztuczną sieć neuronową. Wykorzystałam do tego funkcje:

- **newff** – tworzy wielowarstwową sieć neuronową
- **sim** – symuluje działanie perceptronu
- **train** – służy do nauki sieci na podstawie wektorów wejściowego i wyjściowego
- **disp** – wyświetla informacje

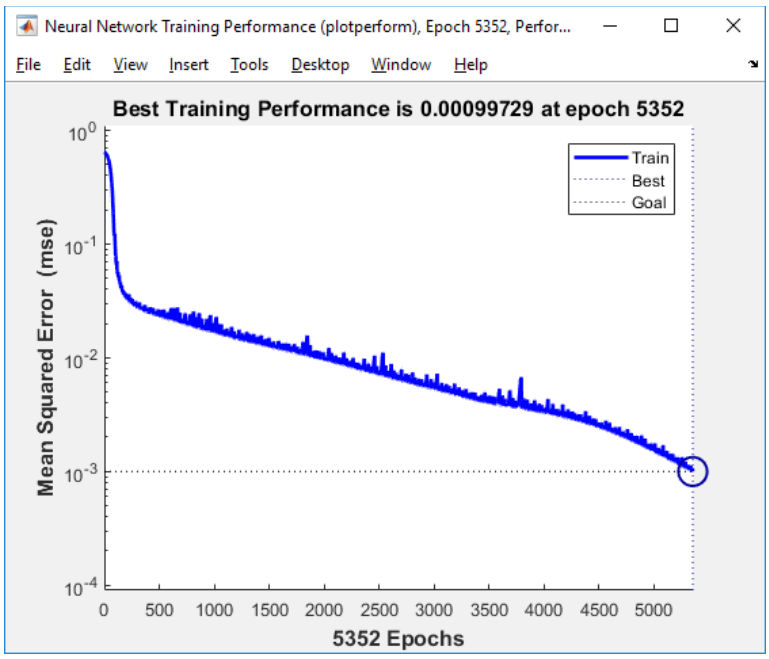
Net jest to struktura zawierająca opis sieci neuronowej.

Celem sieci było rozróżnienie liter, dlatego przygotowałam dane wejściowe. Są nimi litery A, B, C, D, E, F, G, H, I, J, K, L, N, O, P, R, S, T, U, Y, są one reprezentowane za pomocą wartości binarnych (0 i 1) o rozmiarze 4x5. Danymi wyjściowymi jest natomiast macierz diagonalna przyjmująca 1 w momencie dobrego wyniku.

Otrzymane wyniki:



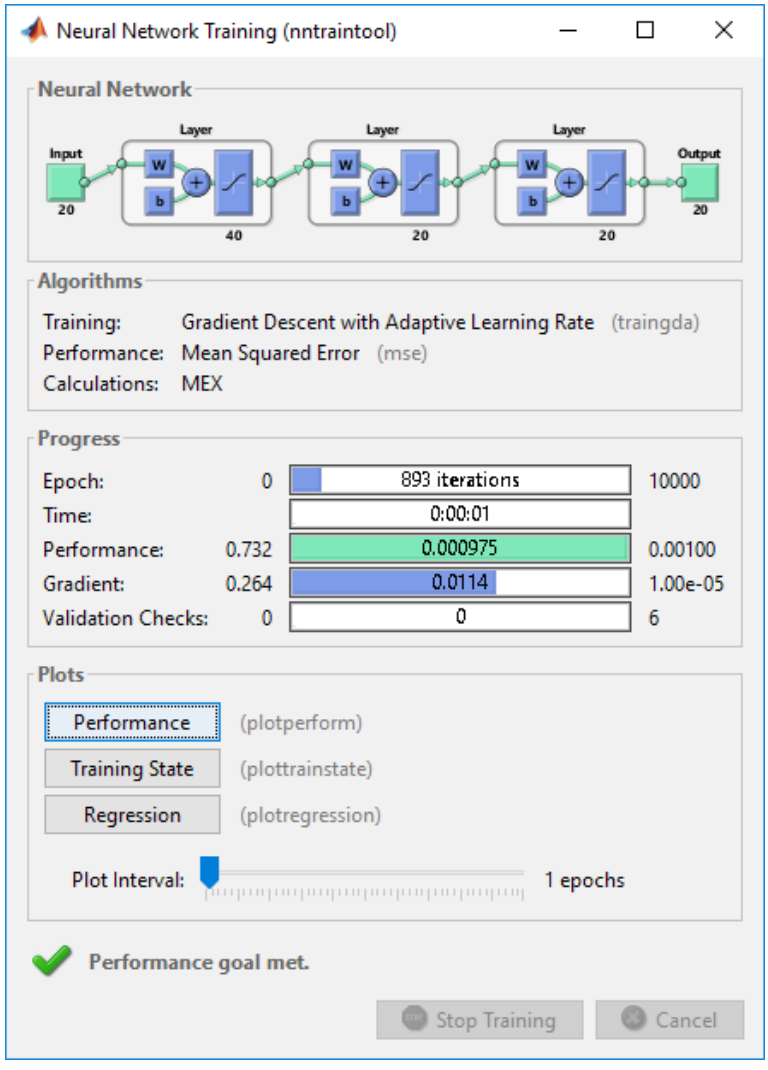
rys1. Wynik działania sieci wielowarstwowej



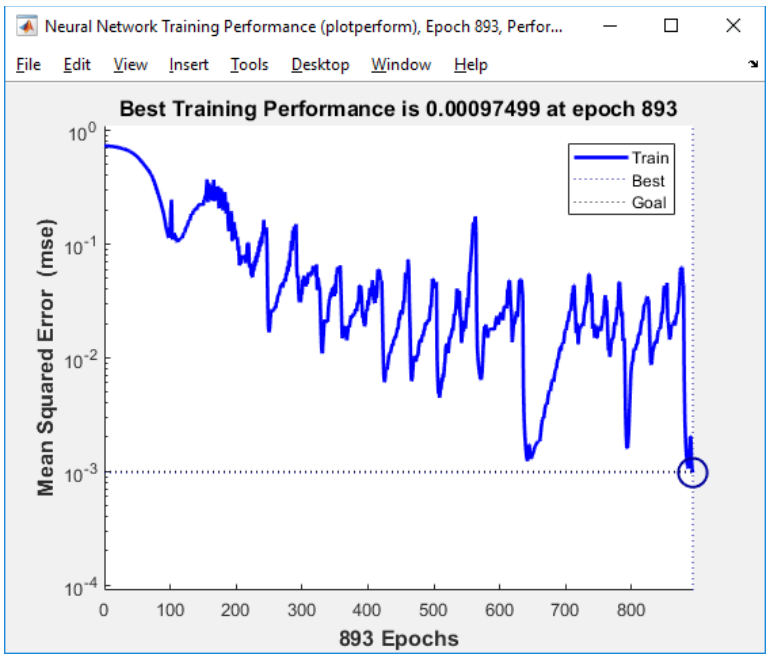
rys2. Wykres wydajności

Ustawiłam maksymalne zwiększenie wydajność na 1,5. Spowodowało to krótszy czas nauki sieci. Ilość iteracji zmalała o ponad 4000.

Oto wyniki, które otrzymałam:

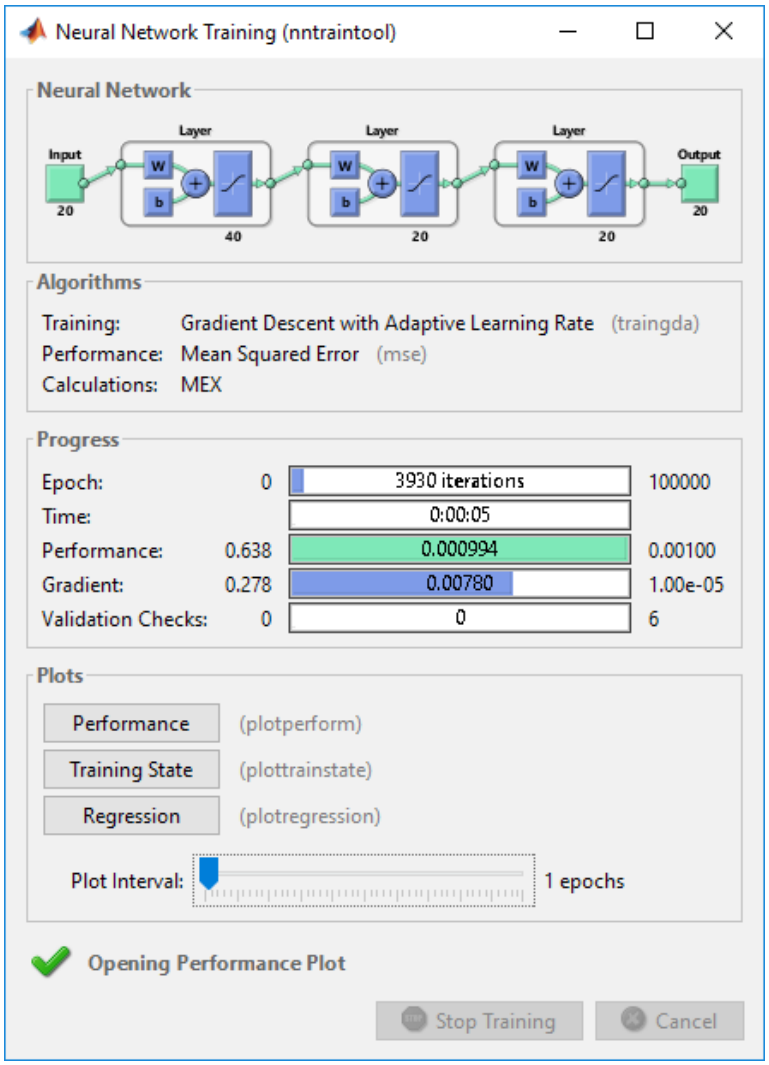


rys3. Wynik działania sieci wielowarstwowej dla zwiększonej wydajności do 1,5

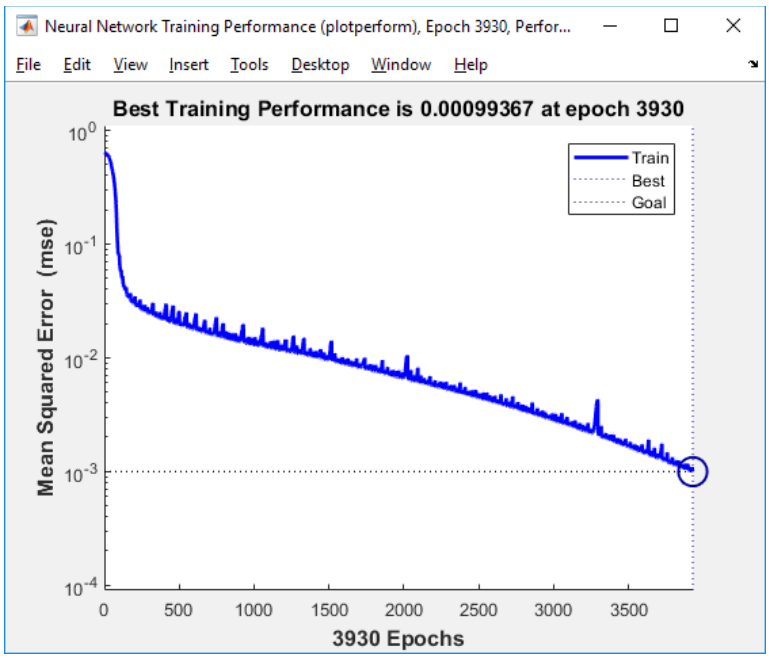


rys5. Wykres wydajności dla zwiększonej wydajności do 1,5

Gdy zwiększymy liczbę epok wykres wydajności jest bardzo podobny jak w przypadku mniejszej liczbie epok. Ilość iteracji nauki się jednak zmniejszyła.

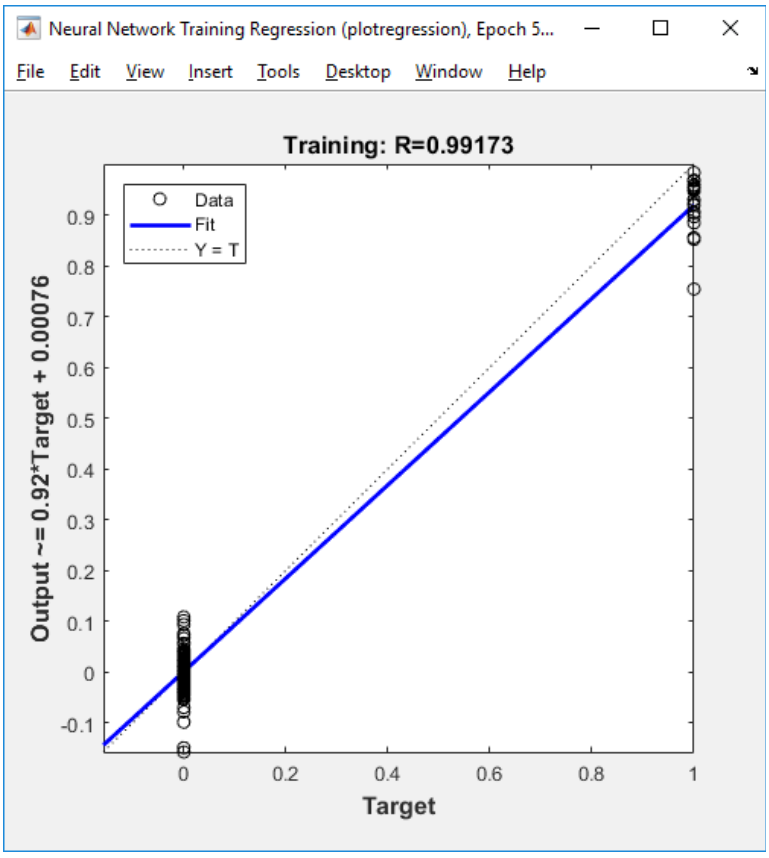


rys6. Wynik działania sieci wielowarstwowej dla maksymalnej liczbie epok 100000

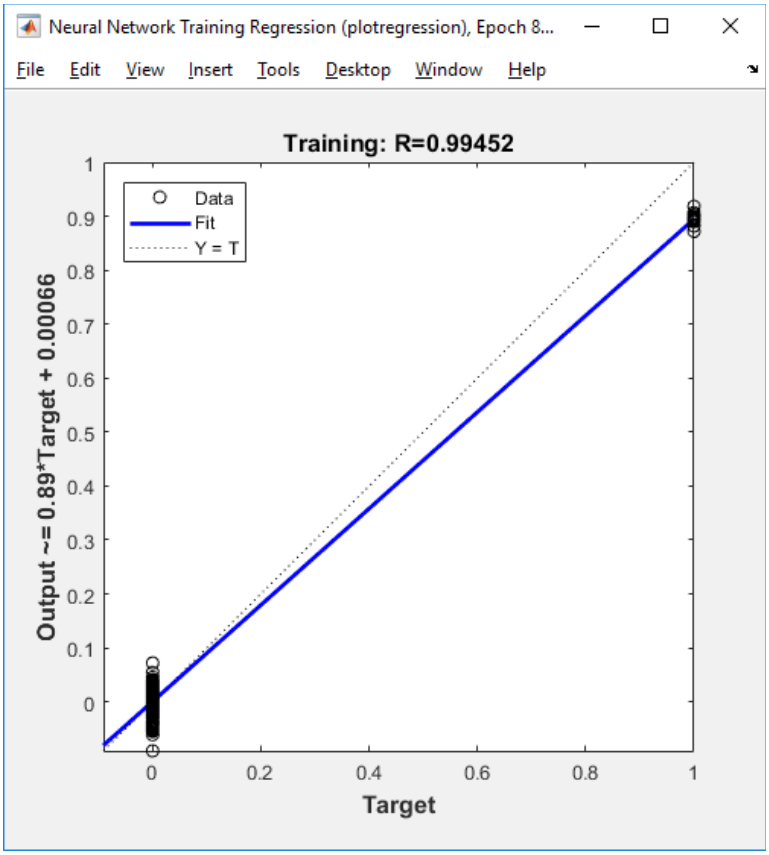


rys7. Wykres wydajności dla maksymalnej liczbie epok 100000

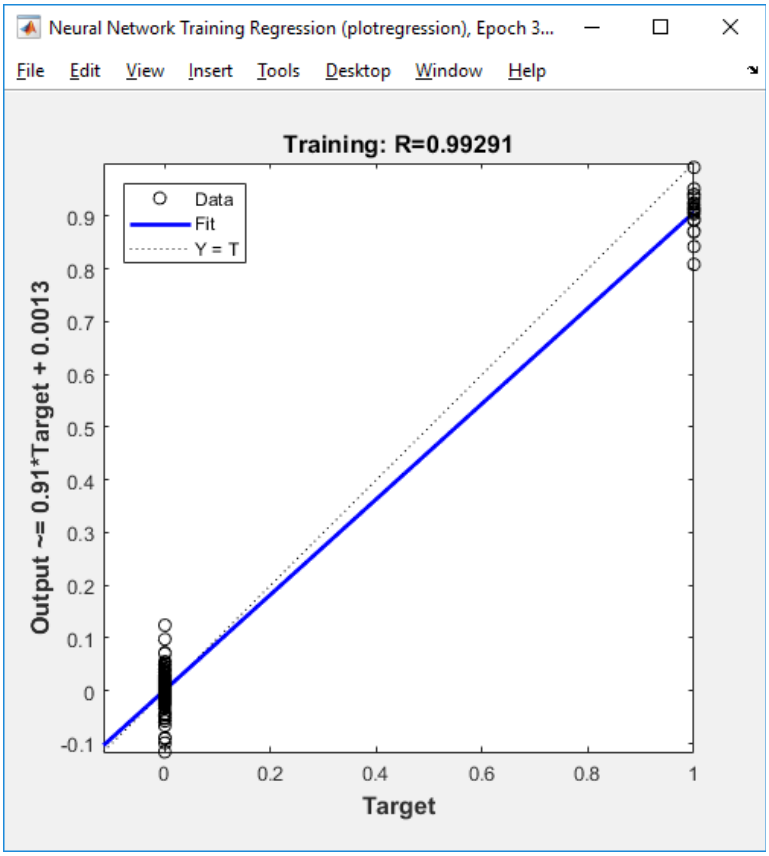
Natomiast regresja dla trzech powyższych przykładów wygląda następująco:



rys8. Regresja dla liczby epok 10000



rys9. Regresja dla zwiększonej wydajności do 1,5



rys10. Regresja dla liczby epok 100000

Wnioski:

Wagi dobrane przy pierwszym orzetestowniu programu były losowe co powoduje, że za każdym uruchormieniem programu liczba iteracji jest różna.

Zwiększenie współczynnika uczenia sieci nie miało znacznego wpływu na czas uczenia, ale miał wpływ na wydajność. Natomiast zwiększenie błędu średnokwadratowego przyśpieszało czas trwania programu oraz wykres wydajności jest znacznie lepszy.

Najlepsza regresja występuje przy pierwszym przypadku, tzn, że wyniki nauki były najlepsze.

Wybór maksymalnego zwiększania wydajności na poziomie 1,5 daje najbardziej prawdopodobne wyniki, zwiększa precyzję.

Ustawienie 40 wektorów wejściowych, 20 ukrytych oraz 20 wyjściowych dawało najbardziej optymalne wyniki.

Kod programu:

```
close all; clear all; clc;

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;];
net = newff(PR,[40 20 20],{'tansig','tansig','tansig'},'traingda');
net.name='Sieæ wielowarstwowa rozpoznaje litery alfabetu';

WE=[0 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1;
    1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 1 0 0;
    1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 1 1 0 1;
    0 0 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 0;
    1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1;
    0 0 0 0 0 0 0 0 0 0 0 0 0 1 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 0 0 1;
    1 1 0 1 0 0 0 1 0 1 0 0 1 1 1 1 0 0 1 0;
    1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0;
    1 1 0 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1;
    1 1 0 0 1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0;
    1 0 0 1 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 0;
    1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1;
    0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0;
    1 1 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0;
    1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 0;
    0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 1;
    0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0;
    1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 1 0 0 0 0;];

WY=[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; ];


% parametry treningu
net.trainParam.epochs = 10000; % maksymalna iloœæ epok
net.trainParam.mu = 0.001; % wspó³czynnik uczenia
net.trainParam.goal = 0.001; % b³¹d kwadratowy
net.trainParam.max_perf_inc = 1.5;

net = train(net, WE, WY); % uczenie sieci

% litery do testu:
A = [0; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 1; 1; 0; 0; 1];
B = [1; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 0];
C = [0; 1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 0; 1; 1; 1];
```

```
D = [1; 1; 1; 0; 1; 0; 0; 1; 1; 0; 0; 1; 1; 0; 0; 1 ;1; 1; 1; 0];
E = [1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 1; 0; 1; 0; 0; 0; 1; 1; 1; 1];
F = [1; 1; 1; 1; 1; 0; 0; 0; 1; 1; 1; 0; 1; 0; 0; 0; 1; 0; 0; 0];
G = [0; 1; 1; 1; 1; 0; 0; 0; 1; 0; 1; 1; 1; 0; 0; 1; 0; 1; 1; 1];
H = [1; 0; 0; 1; 1; 0; 0; 1; 1; 1; 1; 1; 1; 0; 0; 1; 1; 0; 0; 1];
I = [1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0];
J = [1; 1; 1; 1; 0; 0; 0; 1; 0; 0; 0; 1; 1; 0; 0; 1; 0; 1; 1; 1];
K = [1; 0; 0; 1; 1; 0; 1; 0; 1; 1; 0; 0; 1; 0; 1; 0; 1; 0; 0; 1];
L = [1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 1; 1; 1];
N = [1; 0; 0; 1; 1; 1; 0; 1; 1; 0; 1; 1; 1; 0; 0; 1; 1; 0; 0; 1];
O = [0; 1; 1; 0; 1; 0; 0; 1; 1; 0; 0; 1; 1; 0; 0; 1; 0; 1; 1; 0];
P = [1; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 0; 1; 0; 0; 0; 1; 0; 0; 0];
R = [1; 1; 1; 0; 1; 0; 0; 1; 1; 1; 1; 0; 1; 0; 1; 0; 1; 0; 0; 1];
S = [0; 1; 1; 1; 1; 0; 0; 0; 0; 1; 1; 0; 0; 0; 0; 1; 1; 1; 1; 0];
T = [1; 1; 1; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0];
U = [1; 0; 0; 1; 1; 0; 0; 1; 1; 0; 0; 1; 1; 0; 0; 1; 0; 1; 1; 0];
Y = [1; 0; 1; 0; 1; 0; 1; 0; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 0];

efekt=sim(net, A); % symulacja sieci

% funkcja Max
max=1;
for i=1:1:20
    if (efekt(max)<efekt(i))
        max=i;
    end;
end

% wypisywanie liter
switch max
    case 1
        disp('Wpisana litera to A')
        disp(efekt(1))
    case 2
        disp('Wpisana litera to B')
        disp(efekt(2))
    case 3
        disp('Wpisana litera to C')
        disp(efekt(3))
    case 4
        disp('Wpisana litera to D')
        disp(efekt(4))
    case 5
        disp('Wpisana litera to E')
        disp(efekt(5))
    case 6
        disp('Wpisana litera to F')
        disp(efekt(6))
    case 7
        disp('Wpisana litera to G')
        disp(efekt(7))
    case 8
        disp('Wpisana litera to H')
        disp(efekt(8))
    case 9
        disp('Wpisana litera to I')
        disp(efekt(9))
    case 10
        disp('Wpisana litera to J')
        disp(efekt(10))
    case 11
        disp('Wpisana litera to K')
        disp(efekt(11))
    case 12
```

```
        disp('Wpisana litera to L')
        disp(efekt(12))
    case 13
        disp('Wpisana litera to N')
        disp(efekt(13))
    case 14
        disp('Wpisana litera to O')
        disp(efekt(14))
    case 15
        disp('Wpisana litera to P')
        disp(efekt(15))
    case 16
        disp('Wpisana litera to R')
        disp(efekt(16))
    case 17
        disp('Wpisana litera to S')
        disp(efekt(17))
    case 18
        disp('Wpisana litera to T')
        disp(efekt(18))
    case 19
        disp('Wpisana litera to U')
        disp(efekt(19))
    case 20
        disp('Wpisana litera to Y')
        disp(efekt(20))
    otherwise
        disp('BLAD!')
end
```

Krótki opis zmiennych:

PR – zmienna wejściowa dla funkcji tworzących sieć neuronową

net – struktura zawierająca sieć neuronową

net=newff(PR,[40 20 20],{'tansig','tansig','tansig'},'traingda') – utworzenie wielowarstwowej sieci neuronowej, składającej się z 3 warstw: tansig – funkcja aktywacji (tangens hiperboliczny), traingda – funkcja wykorzystana przy treningu sieci (algorytm wstecznej propagacji błędu)

WE– tablica znaków wejściowych

WY – tablica znaków wyjściowych

net.trainParam.* - określenie parametrów treningu

*epochs – maksymalna liczba epok

*goal – błąd średniokwadratowy

*mu – współczynnik uczenia

*max_perf_inc – maksymalne zwiększenie wydajności

test_(litera) – dane testujące reprezentowane przez 0 i 1; w nawiasie: odpowiednia litera, która odpowiada danej zmiennej

efekt – symulcja sieci net za pomocą funkcji sim()

Cały kod znajduje się również na Githubie.