Maciej Krzywda,
Inżynieria Obliczeniowa, IMiIP
Podstawy Sztucznej Inteligencji
nr albumu: 293102

Sprawozdanie 3
Tytuł projektu:
Budowa i działanie sieci wielowarstwowej typu feedforward.

- **Cel projektu:**

Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie kształtu funkcji matematycznej z użyciem algorytmu wstecznej propagacji błędu.

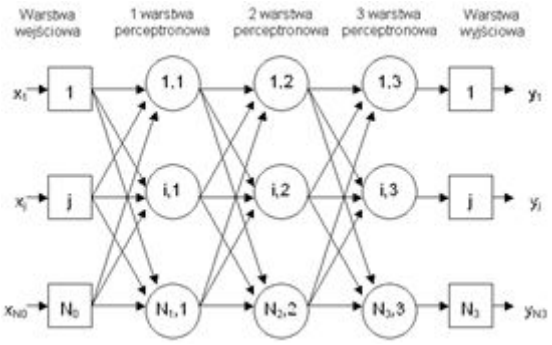
- **Przebieg ćwiczenia**

1. **Wygenerowanie danych uczących i testujących dla funkcji Rastrigin 3D dla danych**
2. **wejściowych z przedziałów od -2 do 2. formuła funkcji: https://en.wikipedia.org/wiki/Rastrigin_function**
3. **Przygotowanie (implementacja lub wykorzystanie gotowych narzędzi) wielowarstwowej sieci oraz algorytmu wstecznej propagacji błędu.**
4. **Uczenie sieci dla różnych współczynników uczenia (np. 0.5, 0.1, 0.01) i bezwładności (np. 0, 0.5, 1).**
5. **Testowanie sieci.**

- **Część Teoretyczna**

Neuron jego schemat został opracowany przez McCullocha i Pittsa w 1943 roku i oparty został na budowie komórki nerwowej. Jego działanie jest następujące: Do wejść doprowadzane są sygnały dochodzące z neuronów warstwy poprzedniej. Każdy sygnał mnożony jest przez odpowiadającą mu wartość liczbową zwaną wagą. Wpływa ona na percepcję danego sygnału wejściowego i jego udział w tworzeniu sygnału wyjściowego przez neuron. Waga może być pobudzająca - dodatnia lub opóźniająca – ujemna; jeżeli nie ma połączenia między neuronami to waga jest równa zero. Zsumowane iloczyny sygnałów i wag stanowią argument funkcji aktywacji neuronu

Sieć wielowarstwowa – Jednokierunkowe wielowarstwowe sieci neuronowe, czyli inaczej sieć typu MLP (Multi Layered Perceptron) jest upowszechniona ze względu na upowszechnienie algorytmu wstecznej propagacji błędu (Error Back Propagation), który umożliwił skuteczne trenowanie sieci w stosunkowo łatwy sposób.



Schemat sieci wielowarstwowej

Algorytm wstecznej propagacji (Rumelhart, Hinton, Williams 1986) – Algorytm wstecznej propagacji błędu jest uogólnieniem reguły delta na potrzeby wielowarstwowych sieci neuronowych. Trening takiej sieci opiera się na numerycznym optymalizacji z wykorzystaniem metod gradientowych mówiących iż gradient funkcji wskazuje jej kierunek najszybszego wzrostu, a dla znaków składowych przeciwnych czyli wymnożeniu przez (-1) kierunek jest najszybszego spadku.
Algorytm wstecznej propagacji błędu modyfikuje każdą wagę proporcjonalnie do wartości pochodnej cząstkowej funkcji celu. Modyfikacja wag w k-tej iteracji polega na odjęciu od wartości wag z iteracji poprzedniej (k - 1) wektora gradientu obliczonego dla bieżącej obserwacji. Algorytm jest sparametryzowany dzięki użyciu tzw. współczynnika uczenia

Uczenie metodą wstecznej propagacji błędów.
Jest to uczenie z nadzorem lub inaczej – 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. Jak to już było zasygnalizowane wcześniej, SSN wykazują tolerancję na nieciągłości, przypadkowe zaburzenia lub wręcz niewielkie braki w zbiorze uczącym. Jest to wynikiem właśnie zdolności do uogólniania wiedzy.

Jeżeli mamy już nauczoną sieć, musimy zweryfikować jej działanie. W tym momencie ważne jest podanie na wejście sieci wzorców z poza zbioru treningowego w celu zbadania czy sieć może efektywnie generalizować zadanie, którego się nauczyła. Do tego używamy ciągu weryfikującego, który ma te same cechy co ciąg uczący tzn. dane dokładnie charakteryzują problem i znamy dokładne odpowiedzi. Ważne jest jednak, aby dane te nie były używane uprzednio do uczenia. Dokonujemy zatem prezentacji ciągu weryfikującego z tą różnicą, że w tym procesie nie rzutujemy błędów wstecz a jedynie rejestrujemy ilość odpowiedzi poprawnych i na tej podstawie orzekamy, czy sieć spełnia nasze wymagania czyli jak została nauczona.

Wagi początkowe, z którymi sieć rozpoczyna naukę z reguły stanowią liczby wygenerowane przypadkowo. Po nauczeniu sieci zawsze warto dla sprawdzenia otrzymanych wyników powtórzyć całą procedurę od wygenerowania wag początkowych. Dla dużych sieci i ciągów uczących składających się z wielu tysięcy wektorów uczących ilość obliczeń wykonywanych podczas całego cyklu uczenia jest gigantyczna a więc i czasochłonna. Nie zdarza się także aby sieć została dobrze zbudowana od razu. Zawsze jest ona efektem wielu prób i błędów. Ponadto nigdy nie mamy gwarancji, że nawet prawidłowa sieć nie utknie w minimum lokalnym podczas gdy interesuje nas znalezienie minimum globalnego. Dlatego algorytmy realizujące SSN wyposaża się mechanizmy dające nauczycielowi możliwość regulacji szybkości i jakości uczenia. Są to tzw. współczynniki: uczenia i momentu. Wpływają one na stromość funkcji aktywacji i regulują szybkość wpływu zmiany wag na proces uczenia.

Zwykle błąd liczony jest jako kwadrat odchylenia: d = 1/2 (y-t)2, co możemy rozpisać jako:

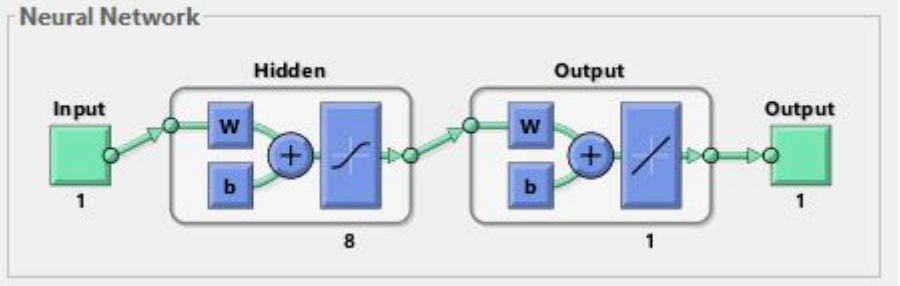
$$d(w_1,\ldots,w_K)=\frac{1}{2}\left(f(w_1z_1+\ldots+w_Kz_K)-t\right)^2$$

- **Część Praktyczna**

Zadanie polegało na wygenerowaniu danych uczących i testujących dla funkcji Rastrign 3D dla danych wejściowych z przedziału [-2; 2].
Funkcja **Rastrign3D** przyjmowała następującą postać:

$$f(\mathbf{x}) = An + \sum_{i=1}^{n}\left[x_i^2 - A\cos(2\pi x_i)\right]$$

gdzie: A = 10, xi∈ [-5.12; 5.12] oraz dla f(0) = 0.



Schemat działania sieci w programie Matlab

- **Przebieg ćwiczenia**

Dane wejściowe: 21 elementowa tablica liczb zmiennoprzecinkowych z przedziału [-2; 2] z krokiem 0.2, wraz z końcami przedziału.

Dane wyjściowe: 21 elementowa zawierającą wyniki działania funkcji Rastrign 3D dla danych wejściowych.

Do zaimplementowania sieci wielowarstwowej użyłam polecenia feedforwardnet(8), które tworzy sieć wielowarstwową zawierającą 8 warstw ukrytych. Można jednak utworzyć sieć o większej lub mniejszje ilości warstw.

Do wersji Matlab2010 istniała funkcja newff która domyślnie tworzyła wielowarstwowa sieć neuronowa z algorytmem wstecznej propagacji błedu.
W swoim projekcie użyłem już nowszej wersji feedforwardnet()
link do dokumentacji Matworks:

1. https://www.mathworks.com/help/deeplearning/release-notes.html?searchHighlight=newff&s_tid=doc_srchtitle
2. https://www.mathworks.com/help/deeplearning/ref/feedforwardnet.html

W przypadku użycia więcej niż 2 warstw istnieje ryzyko przeuczenia sieci więc po kilku próbach zmieniłem na 2 warstwy.Domyślnie Matlab przyjmuje 10 warstw ukrytych co można przeczytać w dokumentacji

1. https://www.mathworks.com/help/deeplearning/ref/feedforwardnet.html

Nasz problem nie jest na tyle złożony, by korzystać z dużej ilości warstw ukrytych.

Testowanie działania sieci, która uczyła bez i z algorytmem wstecznej propagacji – aby wprowadzić ten algorytm należy dodać parametr treningu net.trainFcn = 'traingd'.

Działanie sieci było uzależnione również od współczynnika uczenia oraz bezwładności (tzw. momentum).

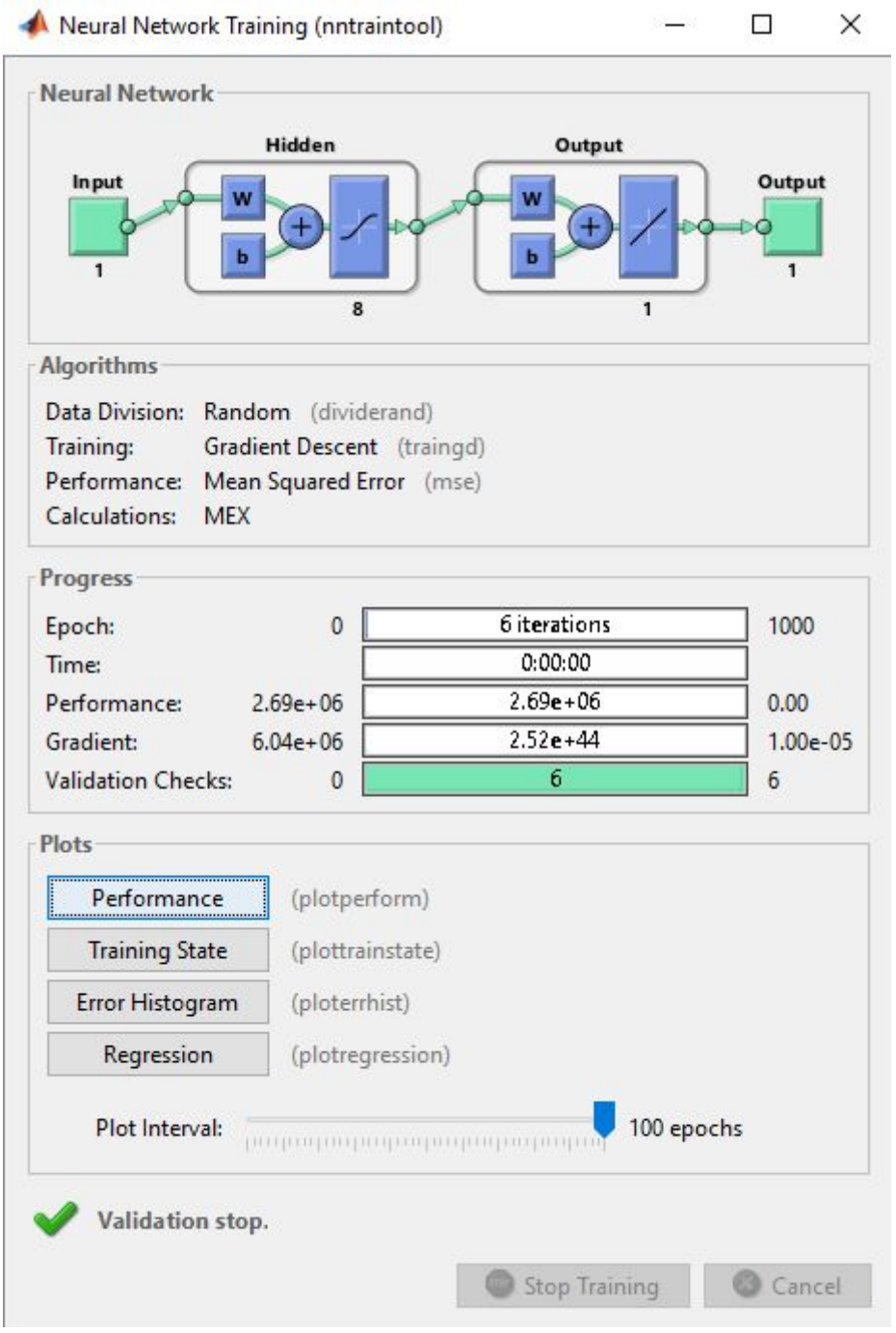
- **Listing programu**

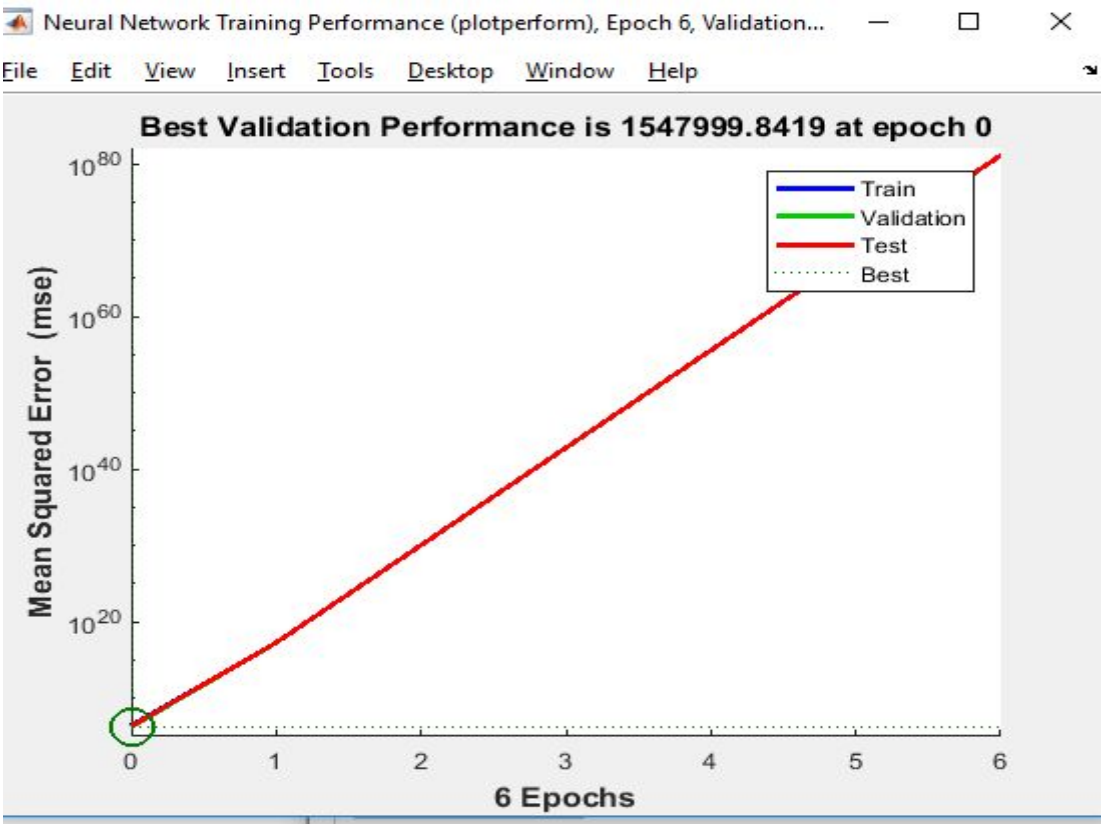
```
close all; clear all; clc;

%nasz dataset
in = [-2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8
1.0 1.2 1.4 1.6 1.8 2.0];
out = [1663.3144032672324 1690.1132374370839 1688.034998342087
1674.099059643066 1686.7210309127113 1732.9225480289258
1776.4412985191298 1786.0418485521361 1779.9521156718527
1803.5731229031892 0 1928.860217750531 1949.5489457731007
1948.9857411333403 1982.511002459129 2065.507872849039 2147.692111352819
```

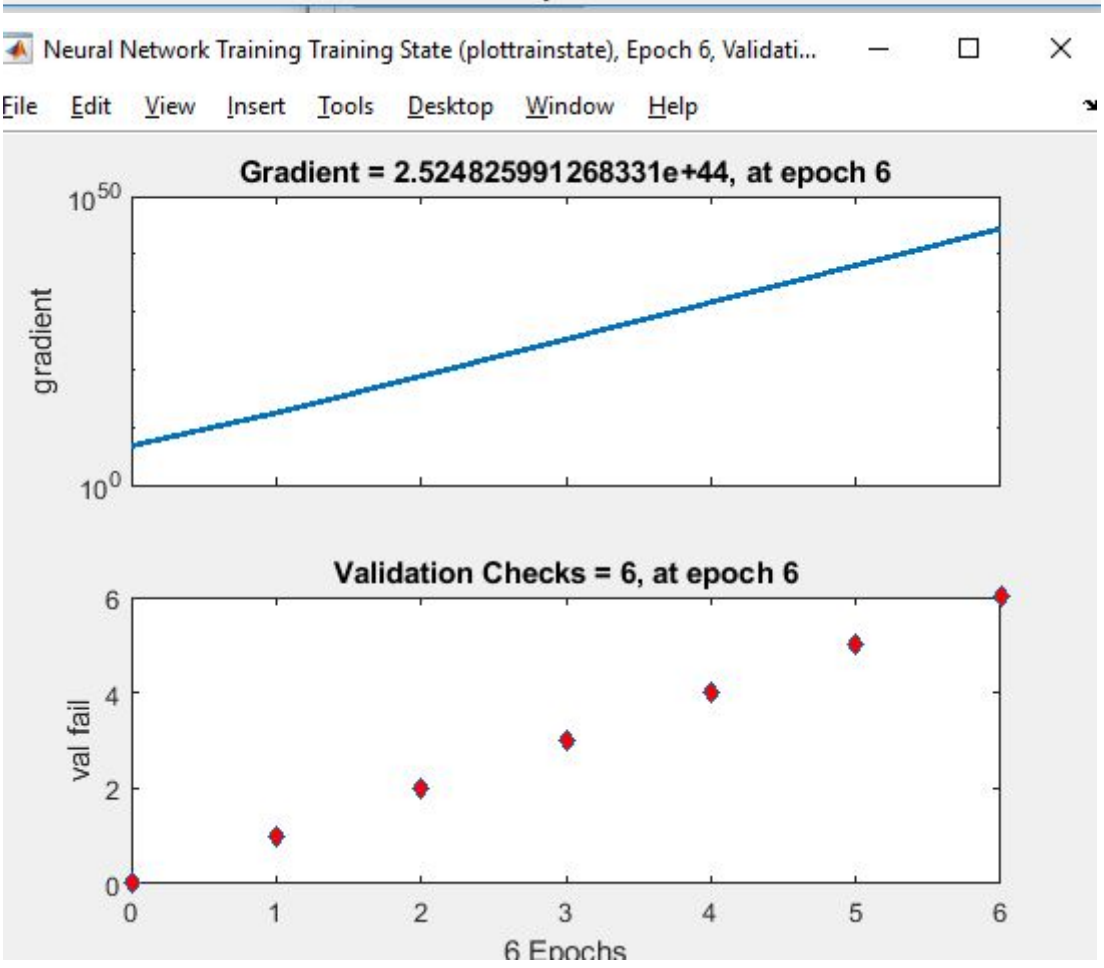
```
2178.4193810575716 2179.078110374434 2220.2957248379535 2326.150912968639
];
test = zeros(1);

%tworzymy siec z 8 warstwami ukrytymi
net = feedforwardnet(8);
%algorytm wstecznej propagacji
net.trainFcn = 'traingd';
% manipulujemy wspolczynnikiem uczenia 0.5 0.1 0.01
net.trainParam.lr = 0.1;
% manipulujemy wspolczynnikiem bezwladnośsci 0.0 0.5 1
net.trainParam.mc = 0.5;
net = train(net, in, out);
efect = zeros(size(net));
% genereujemy wyniki dla funkcji rstrigin_3d oraz testujemy dzialanie
sieci.

for k = 1:21
    test(k) = rstrigin_3d(in(k));
      efect(k) = sim(net, in(k));
end

function fx = rstrigin_3d(x)
    if x == 0
        fx = 0;
    else
      x1 = x;
        A = 10;
        n = 100;
        dx = (5.12-x)/n;
        fx = A * n;

        for i = 1:n
            x = x1 + (i * dx);
            fx = fx + (x^2) - (A * cos(2 * pi * x));
        end
    end
end
```

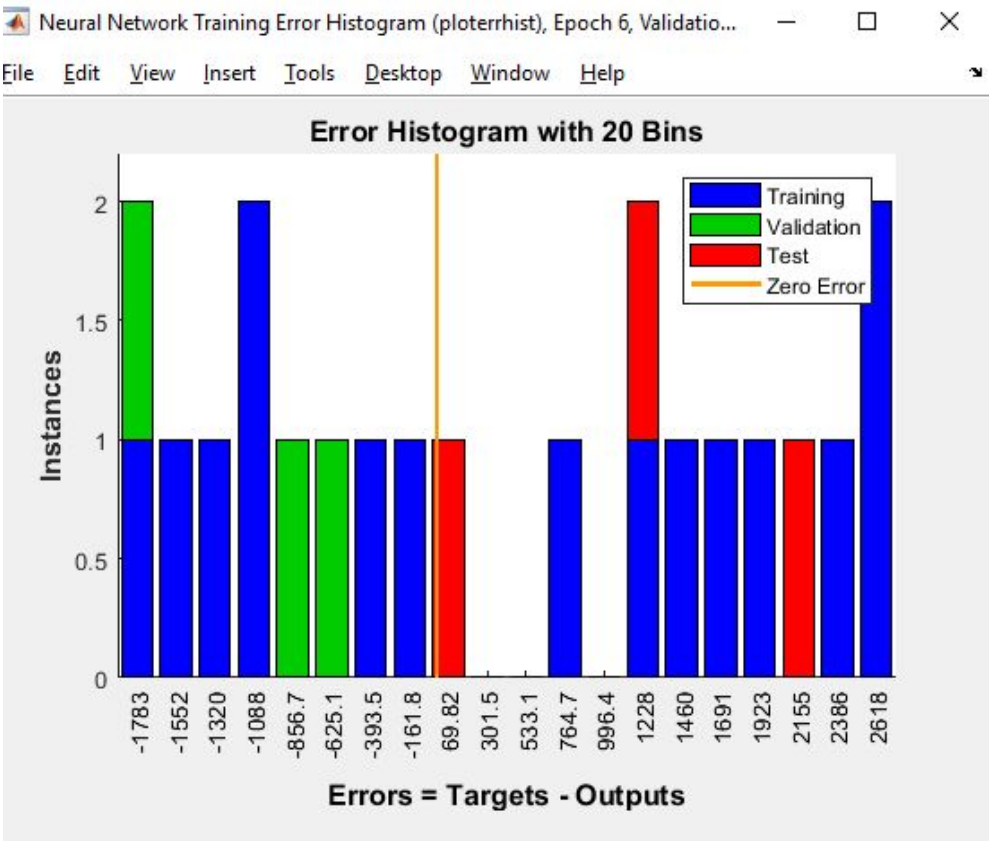
- **Analiza wyników**

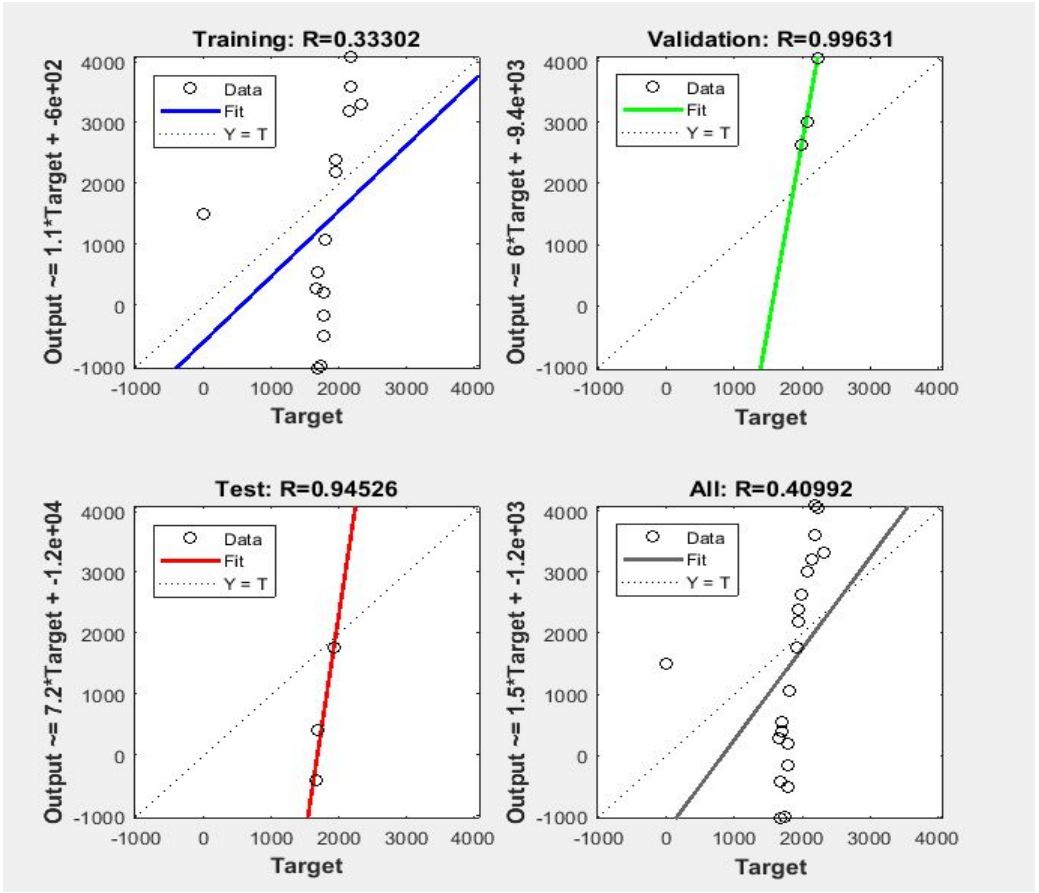
```
net = feedforwardnet(8);
net.trainParam.lr = 0.1;
net.trainParam.mc = 0.5;
```



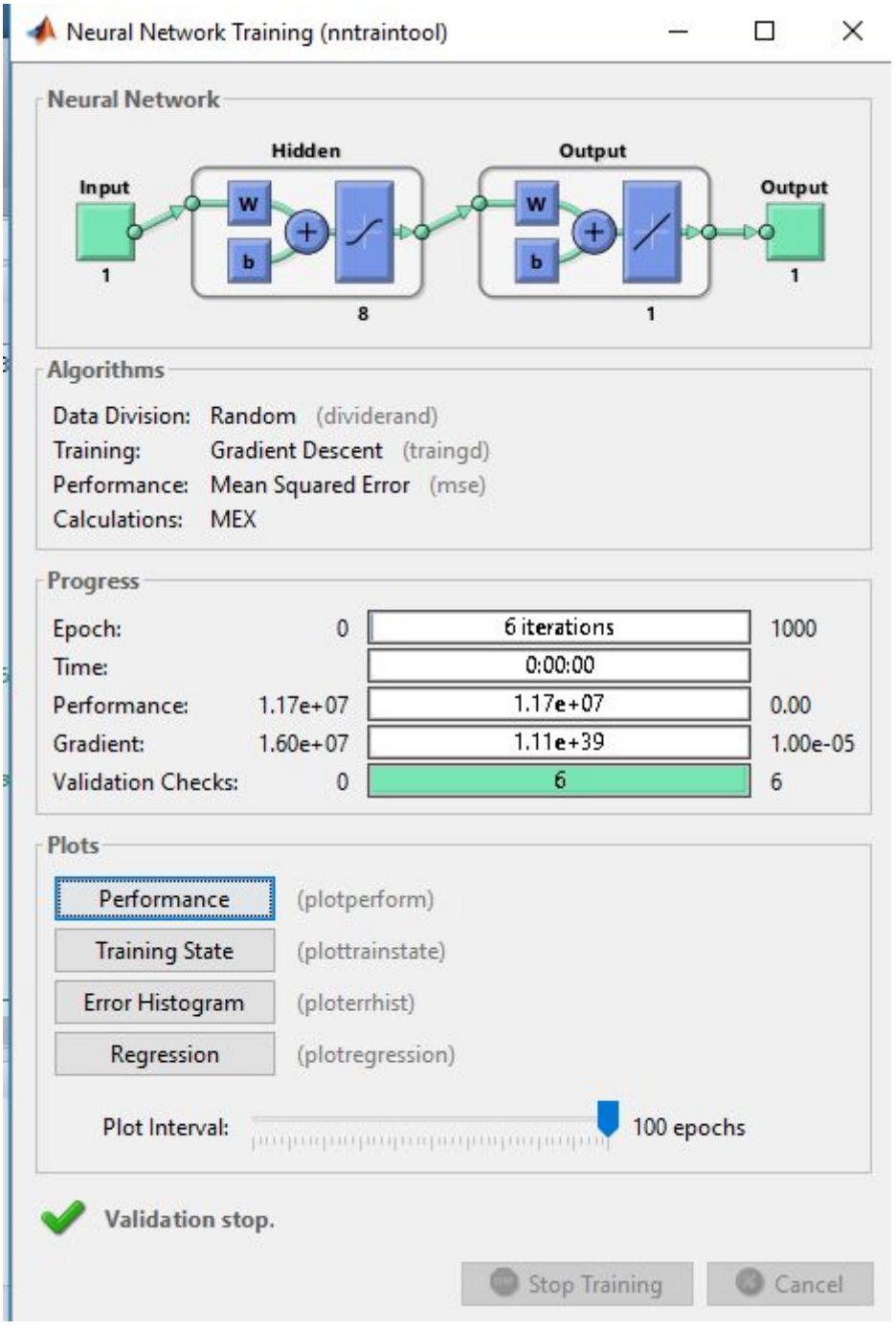

Neural Network Training Performance (plotperform), Epoch 6, Validation...
File Edit View Insert Tools Desktop Window Help
Best Validation Performance is 1547999.8419 at epoch 0
Mean Squared Error (mse)
10^{80}
10^{60}
10^{40}
10^{20}
Train
Validation
Test
Best
0 1 2 3 4 5 6
6 Epochs

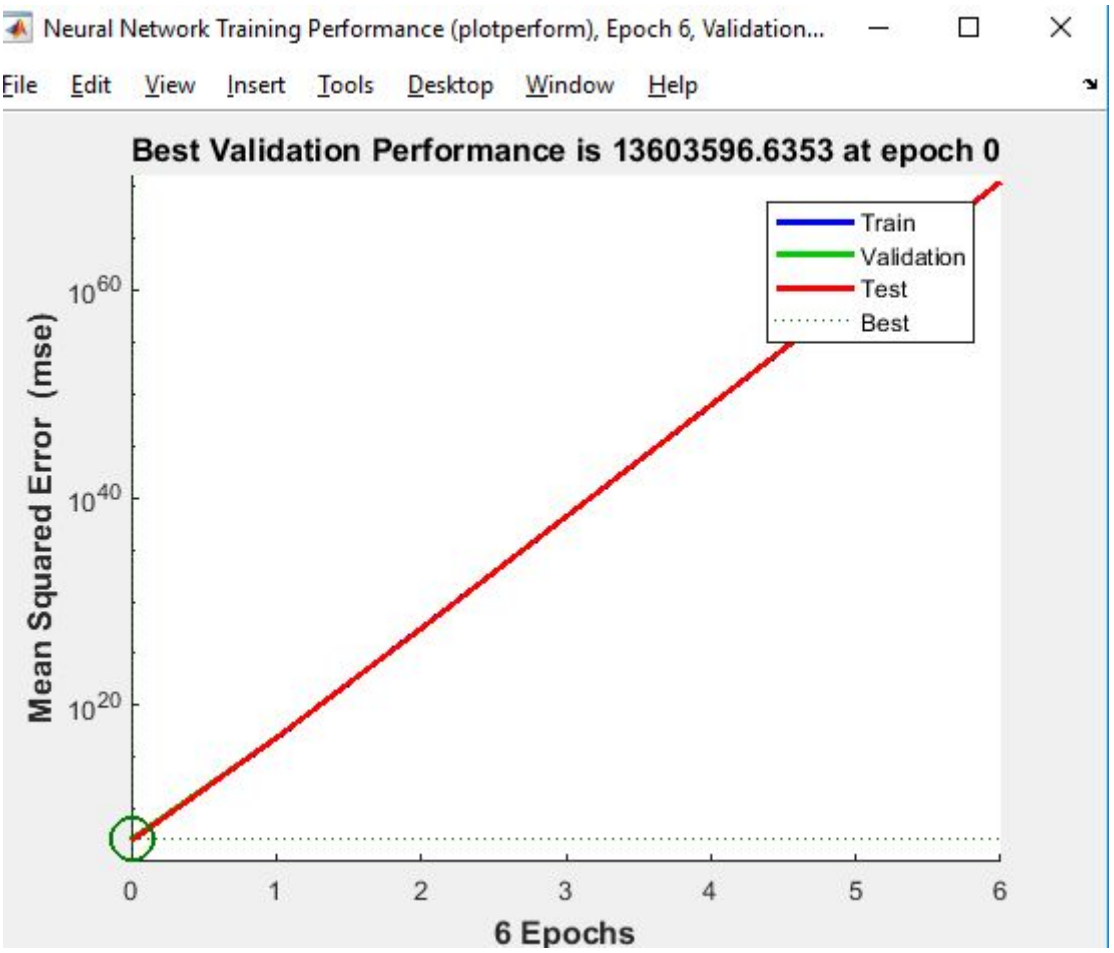

Neural Network Training Training State (plottrainstate), Epoch 6, Validati...
File Edit View Insert Tools Desktop Window Help
Gradient = 2.524825991268331e+44, at epoch 6
gradient
10^{50}
10^{0}
Validation Checks = 6, at epoch 6
val fail
6
4
2
0
0 1 2 3 4 5 6
6 Epochs

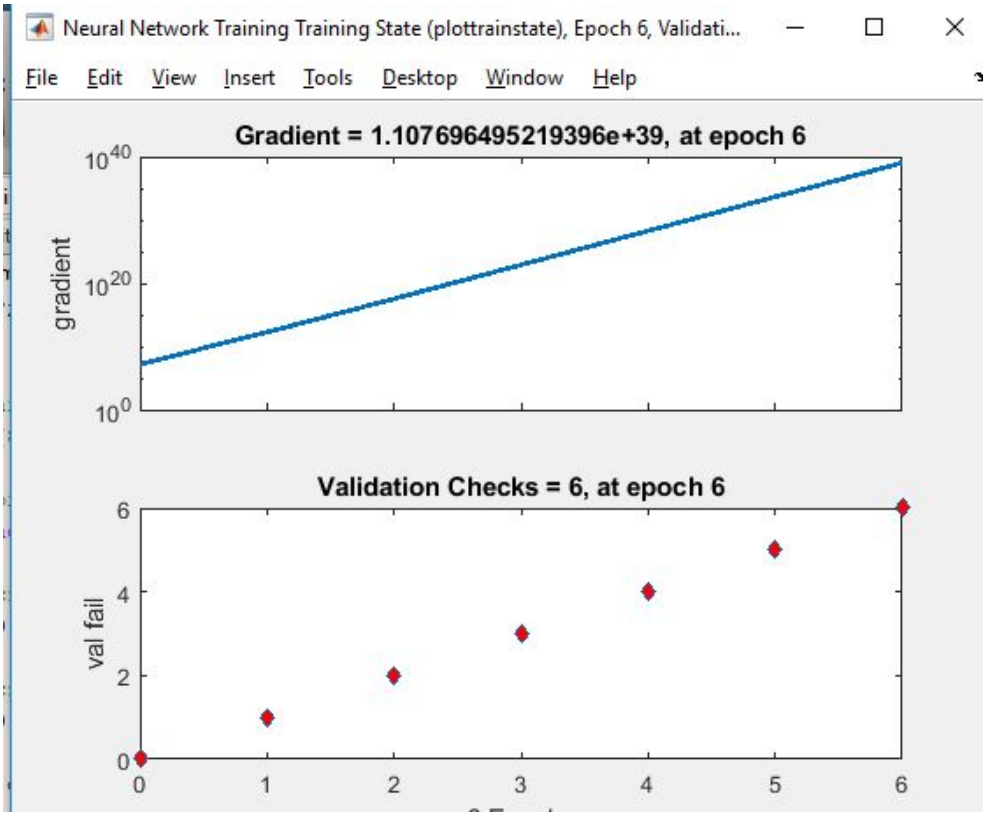

Neural Network Training Error Histogram (ploterrhist), Epoch 6, Validatio...
File Edit View Insert Tools Desktop Window Help
Error Histogram with 20 Bins
Training
Validation
Test
Zero Error
Instances
0
0.5
1
1.5
2
-1783
-1552
-1320
-1088
-856.7
-625.1
-393.5
-161.8
69.82
301.5
533.1
764.7
996.4
1228
1460
1691
1923
2155
2386
2618
Errors = Targets - Outputs

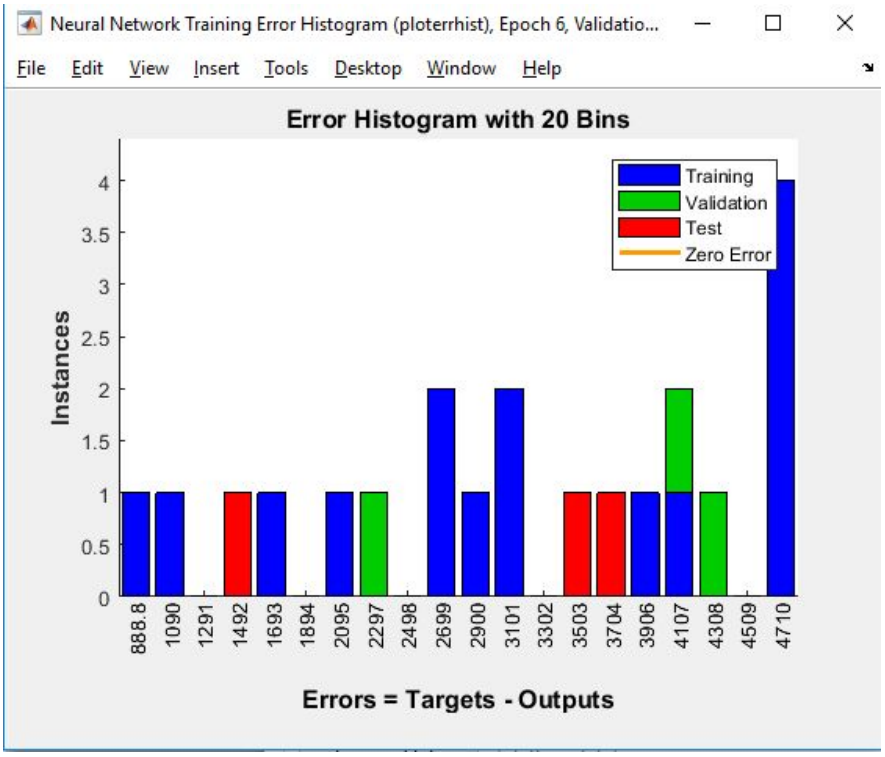

Training: R=0.33302
Output ~= 1.1*Target + -6e+02
Validation: R=0.99631
Output ~= 6*Target + -9.4e+03
Test: R=0.94526
Output ~= 7.2*Target + -1.2e+04
All: R=0.40992
Output ~= 1.5*Target + -1.2e+03
Data
Fit
Y = T
Target
-1000
0
1000
2000
3000
4000

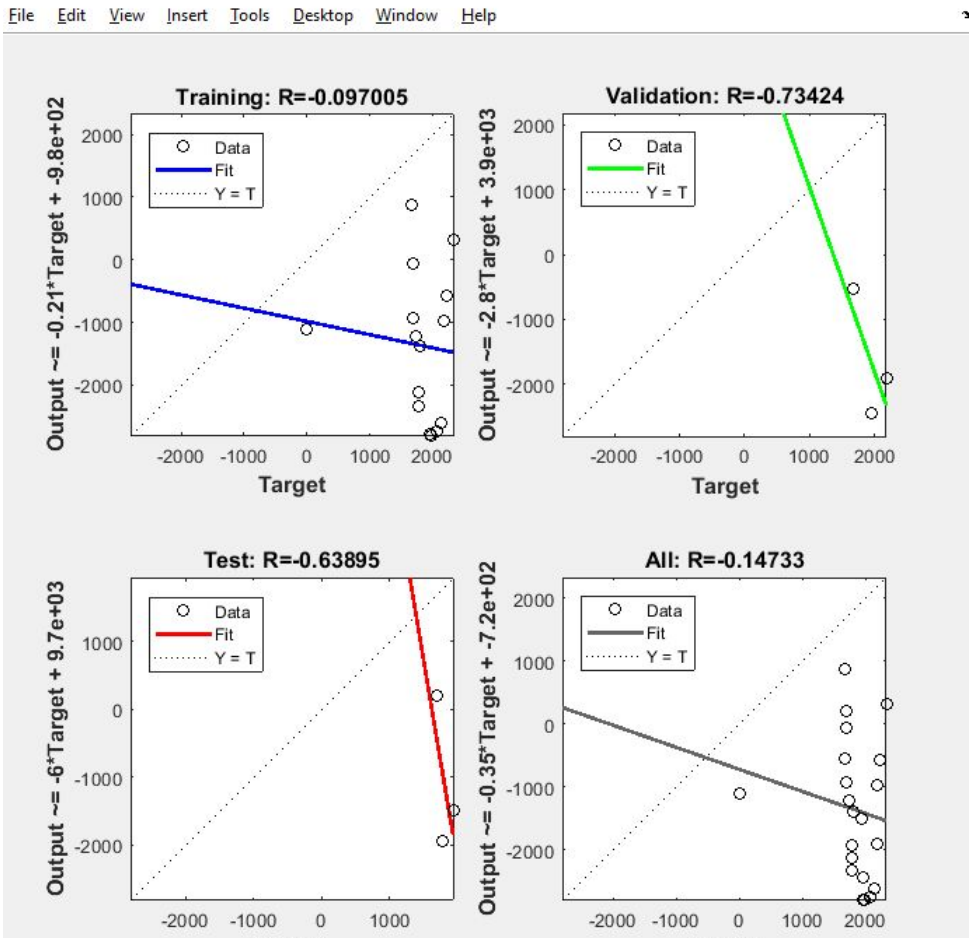
```
net = feedforwardnet(8);
net.trainParam.lr = 0.01;
net.trainParam.mc = 0.0;
```



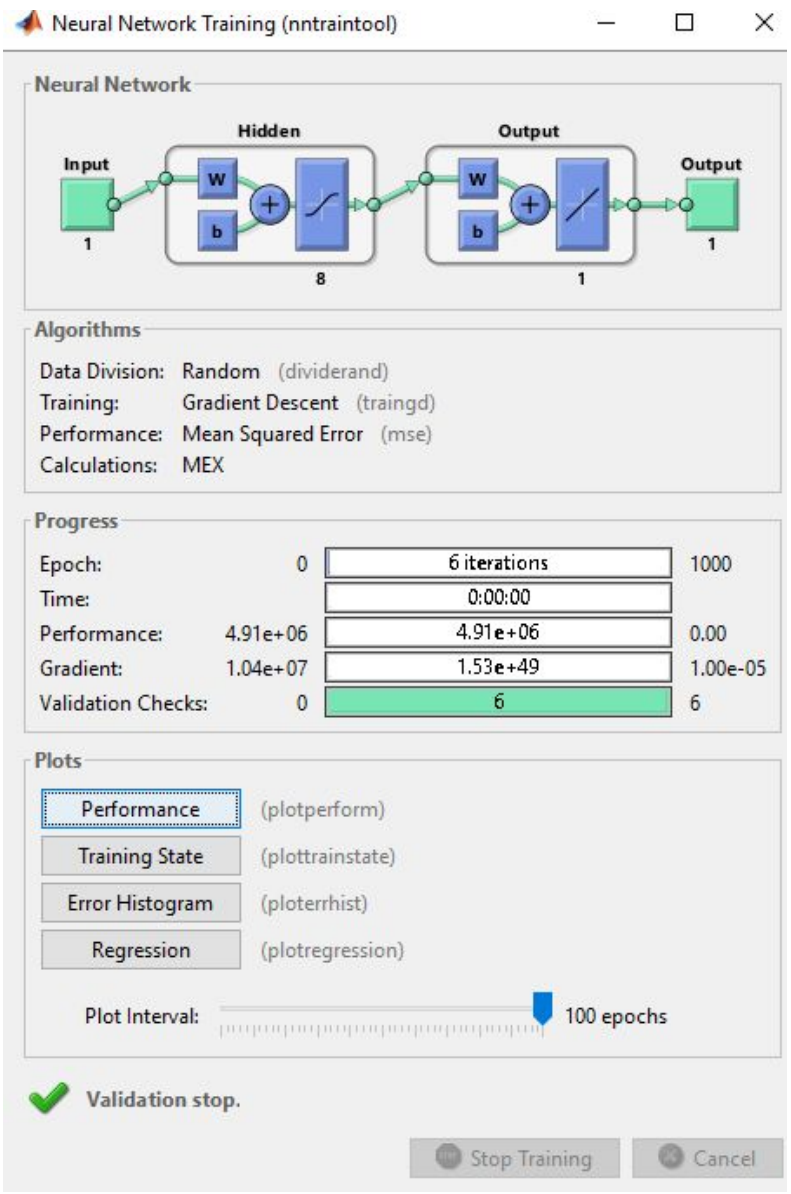

Neural Network Training Performance (plotperform), Epoch 6, Validation...
File Edit View Insert Tools Desktop Window Help
Best Validation Performance is 13603596.6353 at epoch 0
Train
Validation
Test
Best
Mean Squared Error (mse)
10^60
10^40
10^20
0 1 2 3 4 5 6
6 Epochs

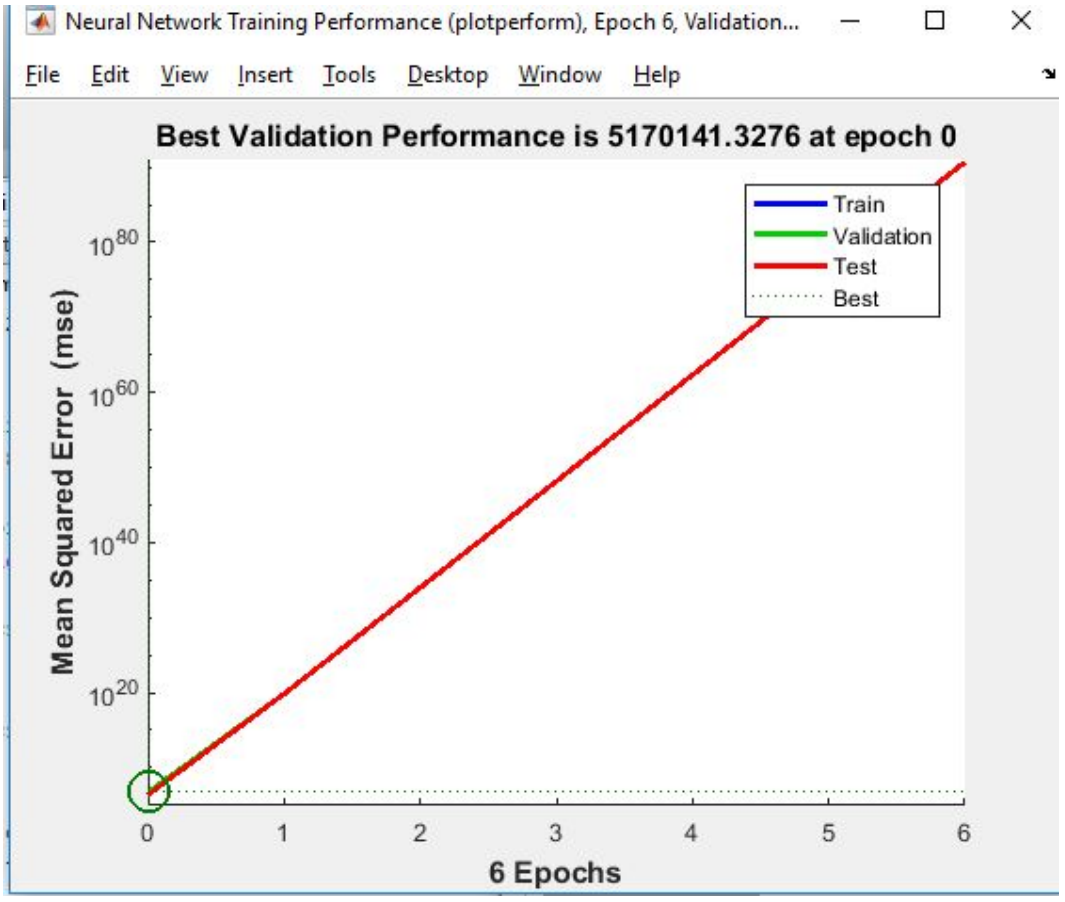

Neural Network Training Training State (plottrainstate), Epoch 6, Validati...
File Edit View Insert Tools Desktop Window Help
Gradient = 1.107696495219396e+39, at epoch 6
gradient
10^40
10^20
10^0
Validation Checks = 6, at epoch 6
val fail
0 2 4 6
0 1 2 3 4 5 6

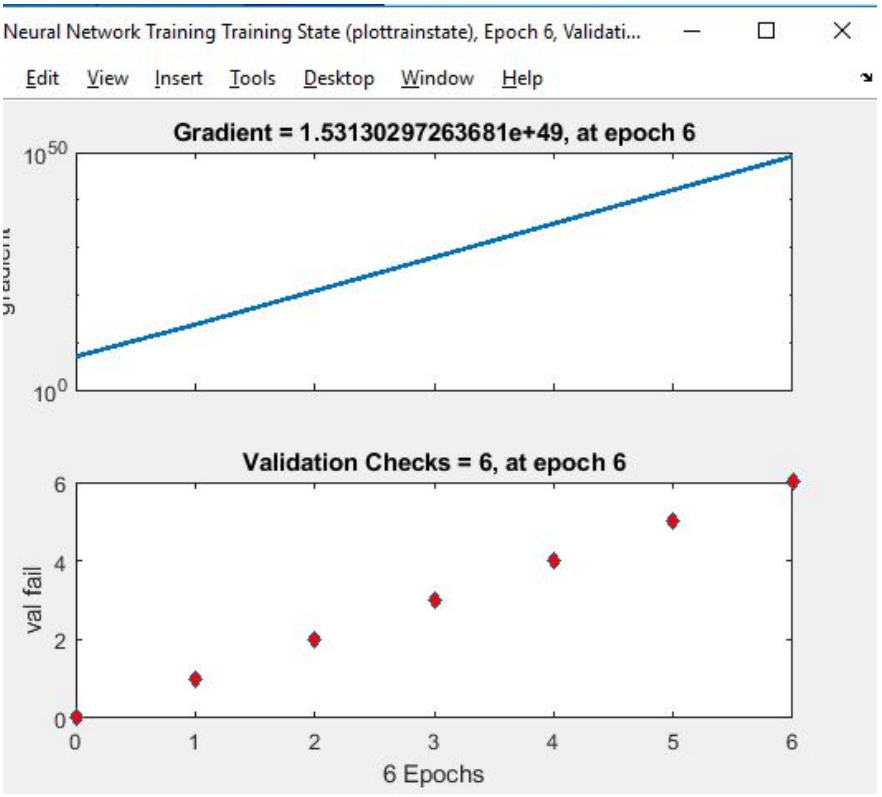

Neural Network Training Error Histogram (ploterrhist), Epoch 6, Validatio...
File Edit View Insert Tools Desktop Window Help
Error Histogram with 20 Bins
Training
Validation
Test
Zero Error
Instances
0 0.5 1 1.5 2 2.5 3 3.5 4
888.8 1090 1291 1492 1693 1894 2095 2297 2498 2699 2900 3101 3302 3503 3704 3906 4107 4308 4509 4710
Errors = Targets - Outputs

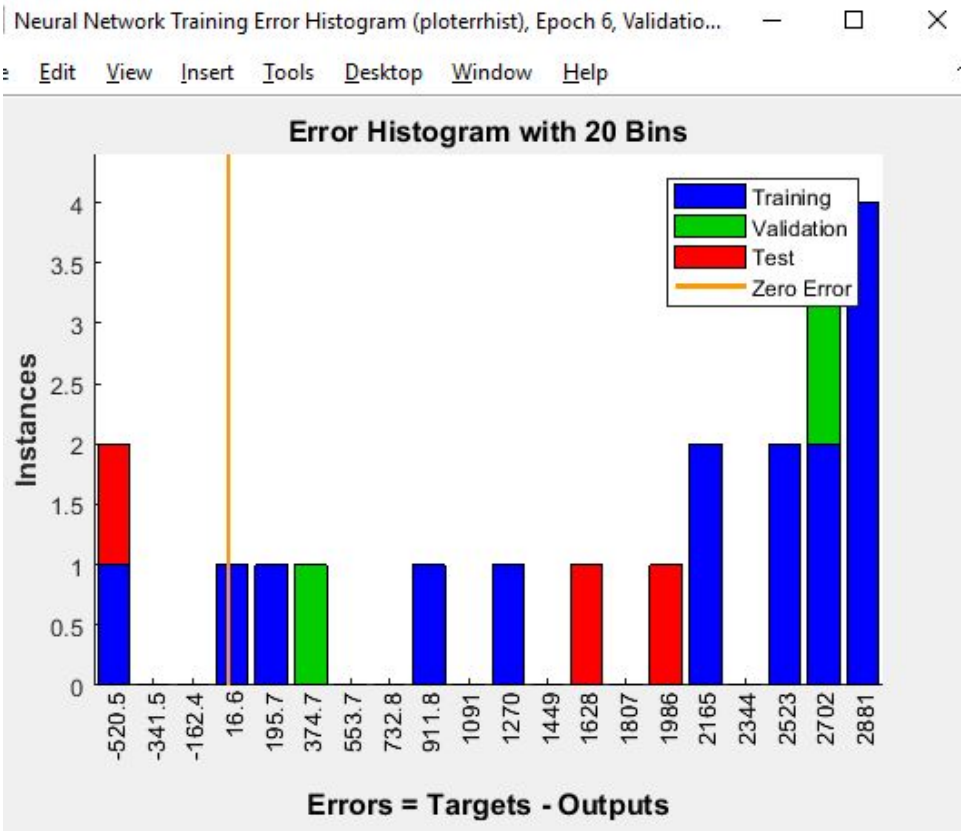

File Edit View Insert Tools Desktop Window Help
Training: R=-0.097005
Output ~= -0.21*Target + -9.8e+02
Data
Fit
Y = T
Target
Validation: R=-0.73424
Output ~= -2.8*Target + 3.9e+03
Test: R=-0.63895
Output ~= -6*Target + 9.7e+03
All: R=-0.14733
Output ~= -0.35*Target + -7.2e+02

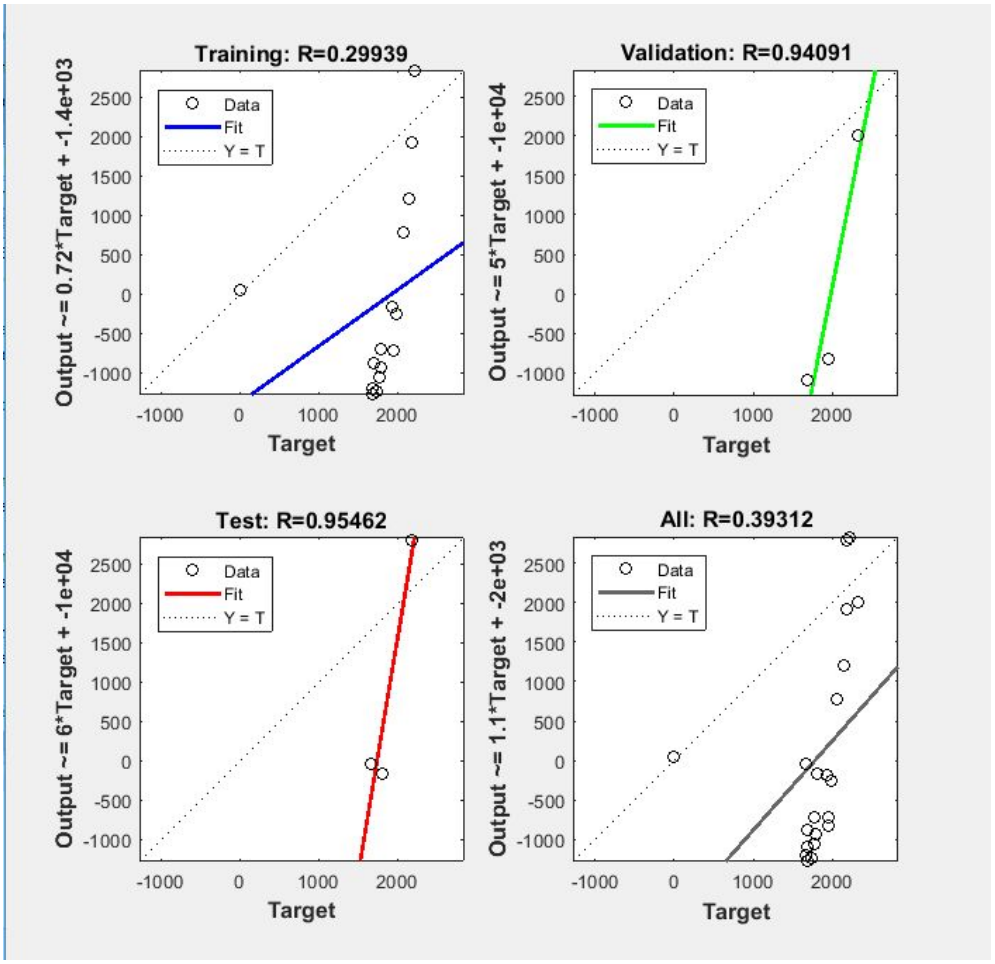
```
net = feedforwardnet(8);
net.trainParam.lr = 0.5;
net.trainParam.mc = 1.0;
```



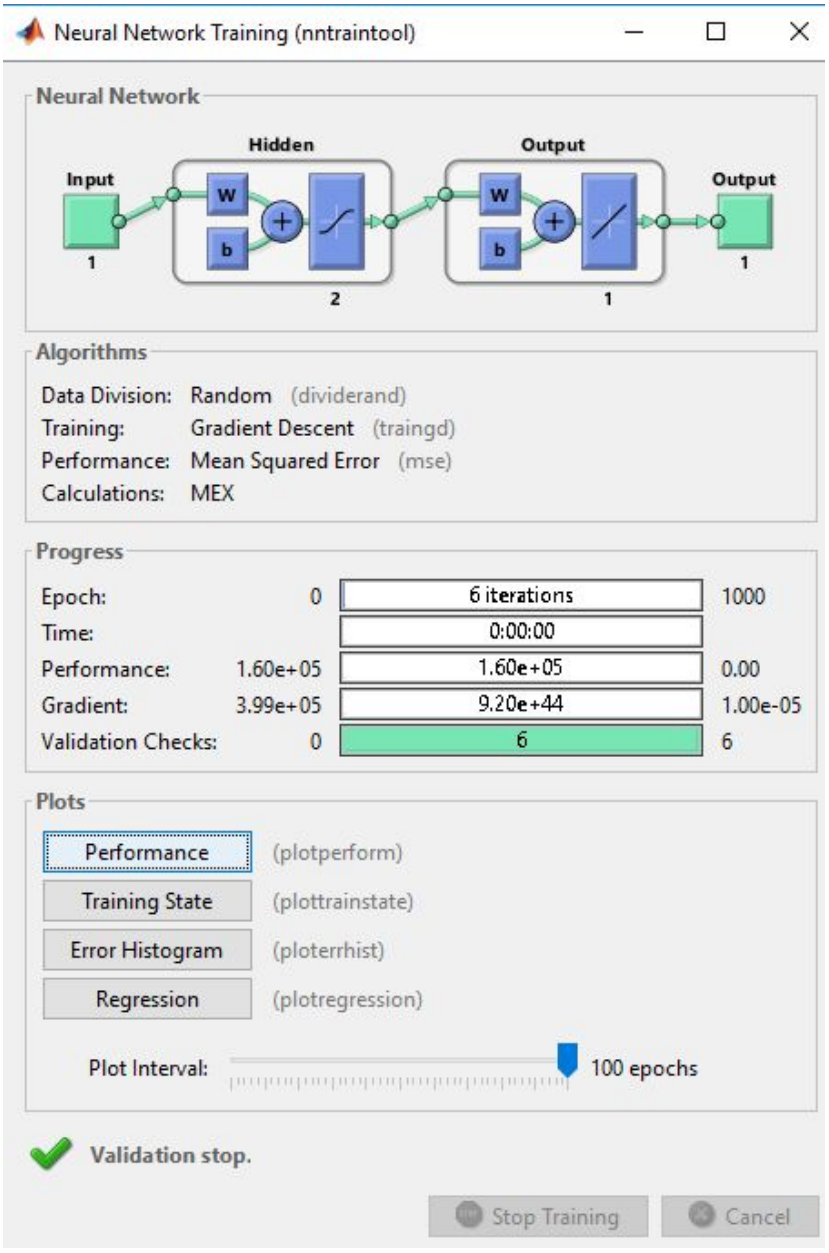

Neural Network Training Performance (plotperform), Epoch 6, Validation...
File Edit View Insert Tools Desktop Window Help
Best Validation Performance is 5170141.3276 at epoch 0
Train
Validation
Test
Best
Mean Squared Error (mse)
10^80
10^60
10^40
10^20
0 1 2 3 4 5 6
6 Epochs

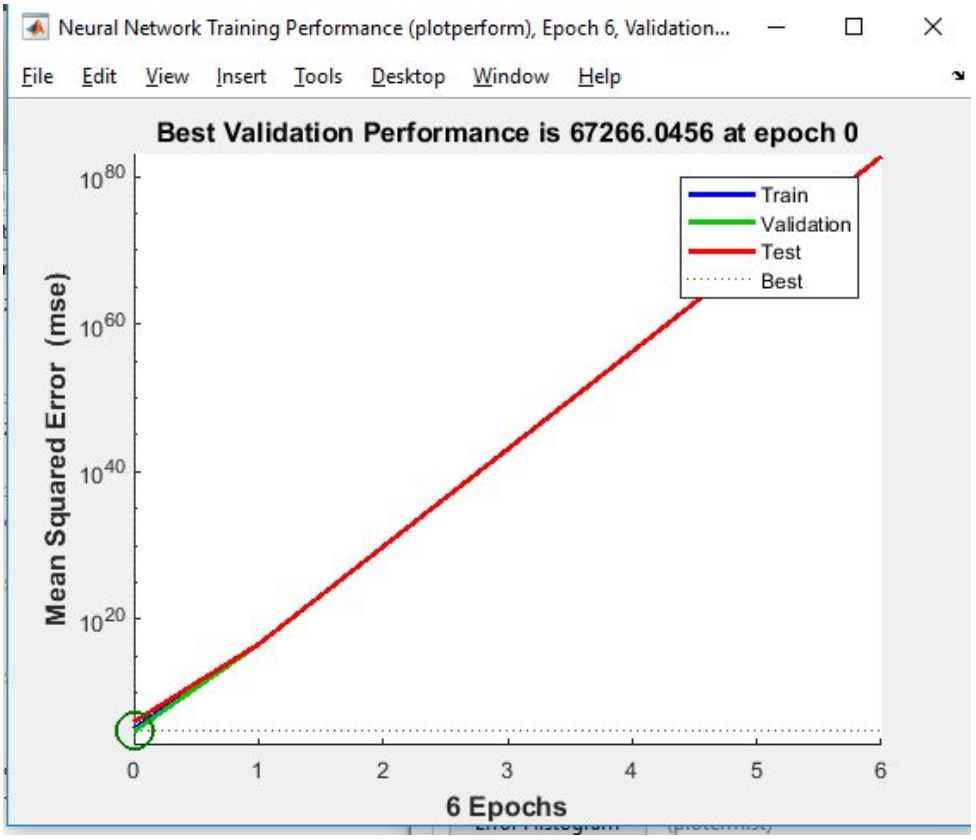

Neural Network Training Training State (plottrainstate), Epoch 6, Validati...
Edit View Insert Tools Desktop Window Help
Gradient = 1.53130297263681e+49, at epoch 6
10^50
10^0
Validation Checks = 6, at epoch 6
val fail
0 2 4 6
0 1 2 3 4 5 6
6 Epochs

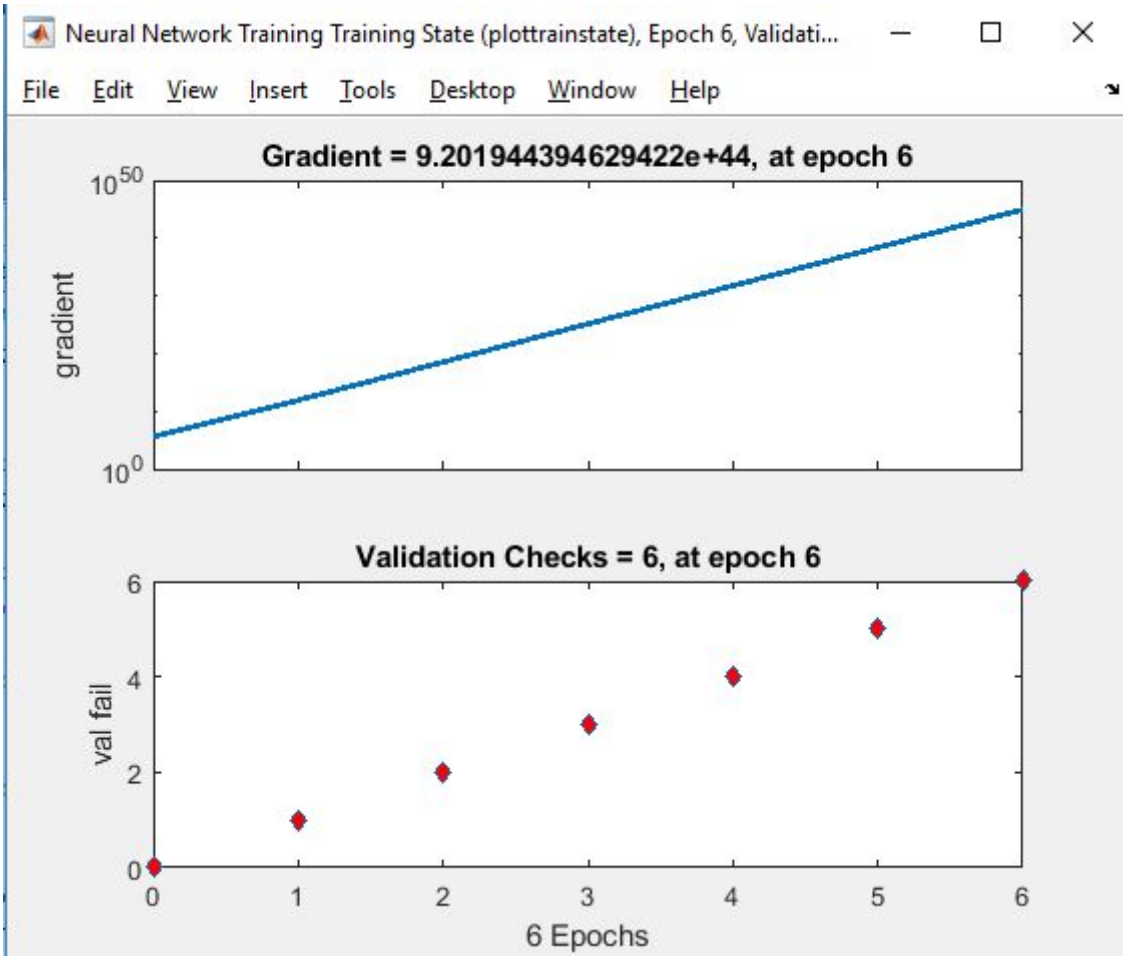

Neural Network Training Error Histogram (ploterrhist), Epoch 6, Validatio...
Edit View Insert Tools Desktop Window Help
Error Histogram with 20 Bins
Training
Validation
Test
Zero Error
Instances
0 0.5 1 1.5 2 2.5 3 3.5 4
-520.5 -341.5 -162.4 16.6 195.7 374.7 553.7 732.8 911.8 1091 1270 1449 1628 1807 1986 2165 2344 2523 2702 2881
Errors = Targets - Outputs

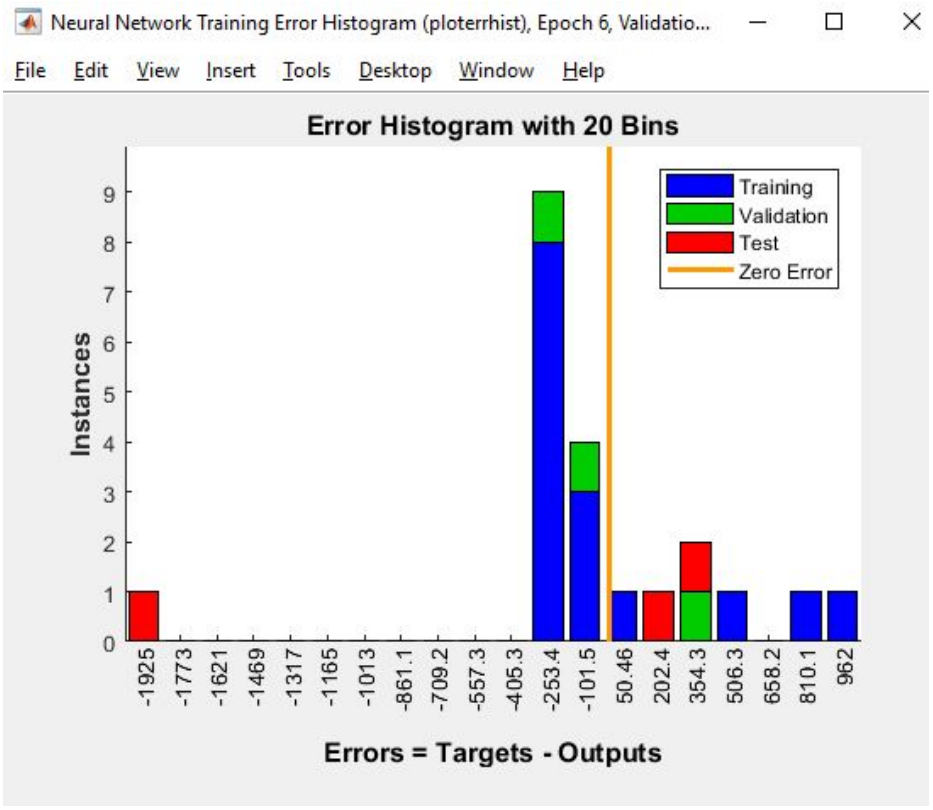

Training: R=0.29939
Output ~= 0.72*Target + -1.4e+03
Validation: R=0.94091
Output ~= 5*Target + -1e+04
Test: R=0.95462
Output ~= 6*Target + -1e+04
All: R=0.39312
Output ~= 1.1*Target + -2e+03
Data
Fit
Y = T
Target
-1000 0 1000 2000
-1000 -500 0 500 1000 1500 2000 2500

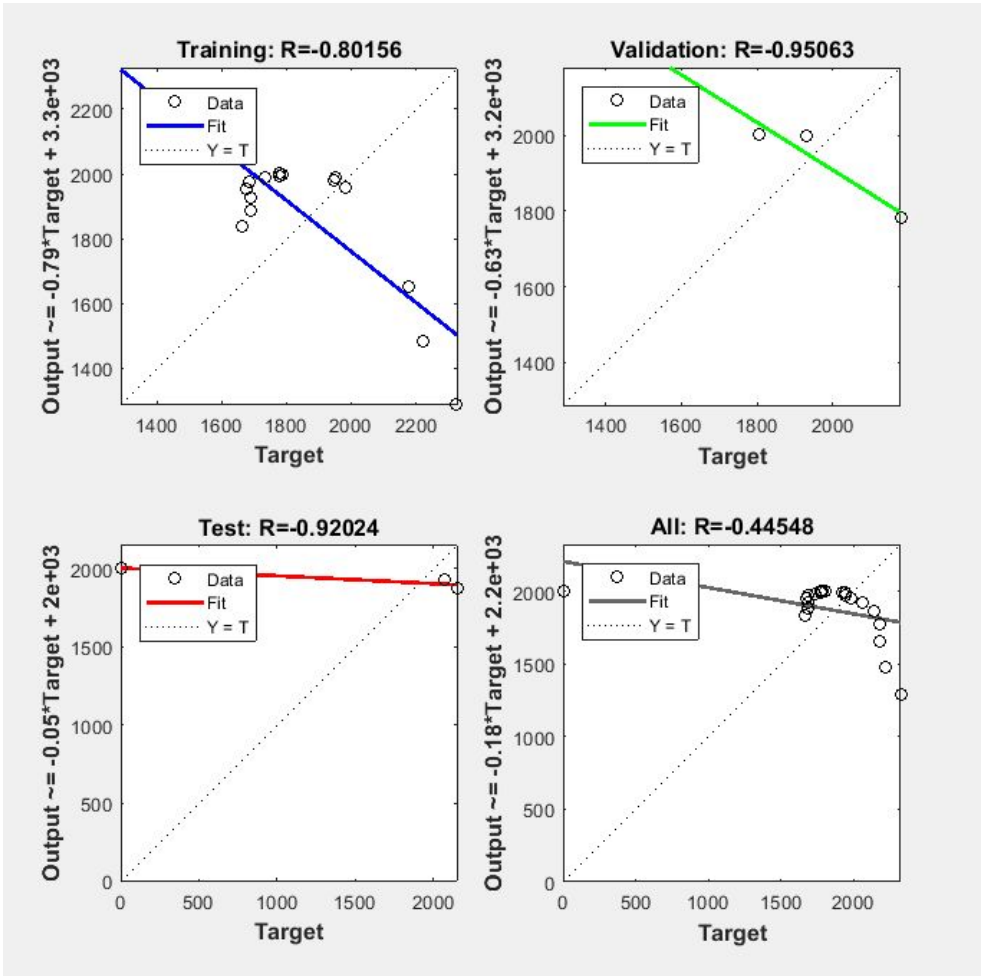
```
net = feedforwardnet(2);
net.trainParam.lr = 0.5;
net.trainParam.mc = 0.5;
```



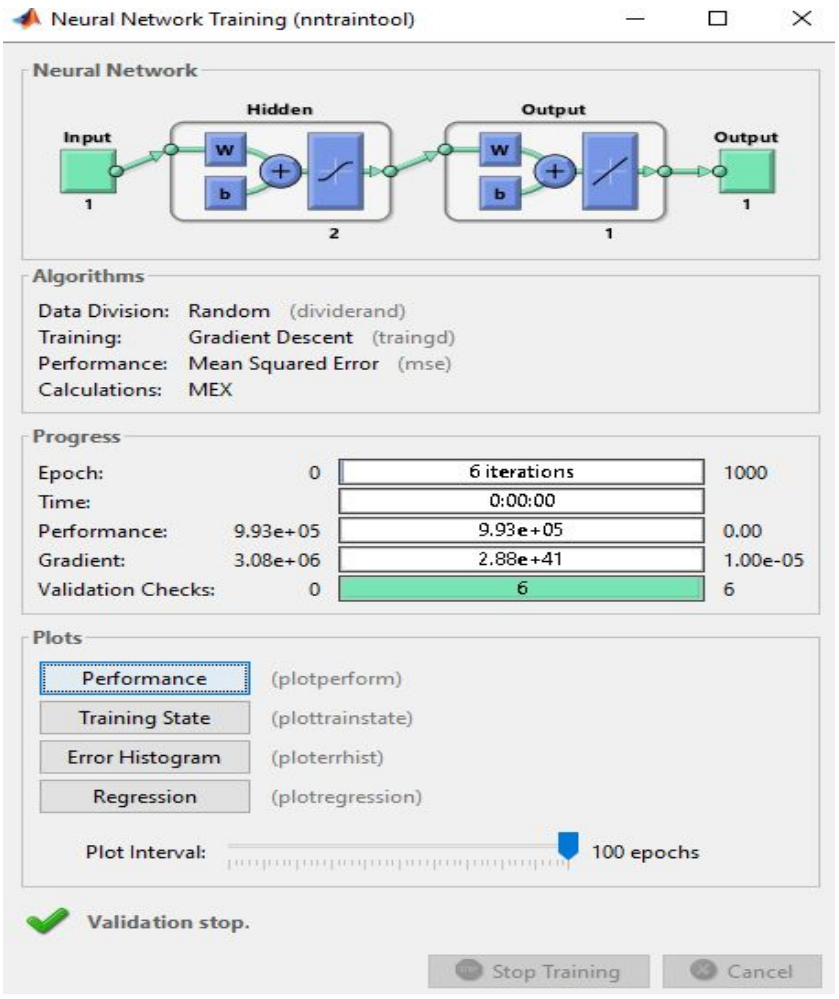

Neural Network Training Performance (plotperform), Epoch 6, Validation...
File Edit View Insert Tools Desktop Window Help
Best Validation Performance is 67266.0456 at epoch 0
Mean Squared Error (mse)
10^{80}
10^{60}
10^{40}
10^{20}
Train
Validation
Test
Best
0 1 2 3 4 5 6
6 Epochs

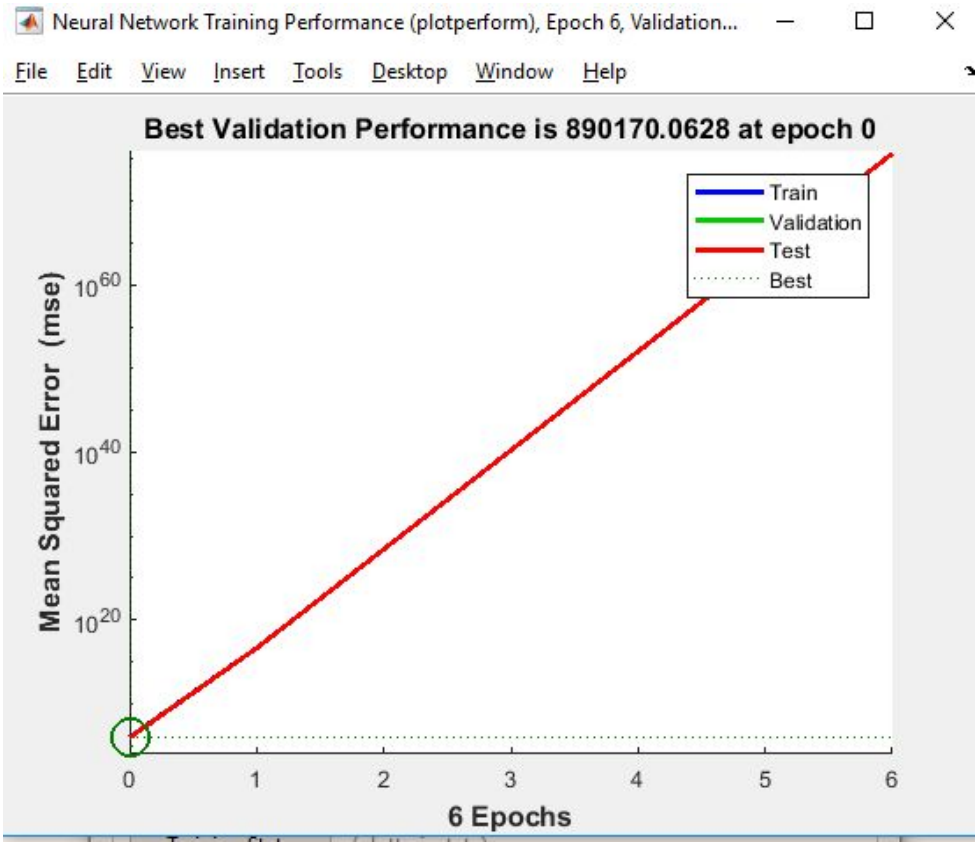

Neural Network Training Training State (plottrainstate), Epoch 6, Validati...
File Edit View Insert Tools Desktop Window Help
Gradient = 9.201944394629422e+44, at epoch 6
gradient
10^{50}
10^{0}
Validation Checks = 6, at epoch 6
val fail
0 2 4 6
0 1 2 3 4 5 6
6 Epochs

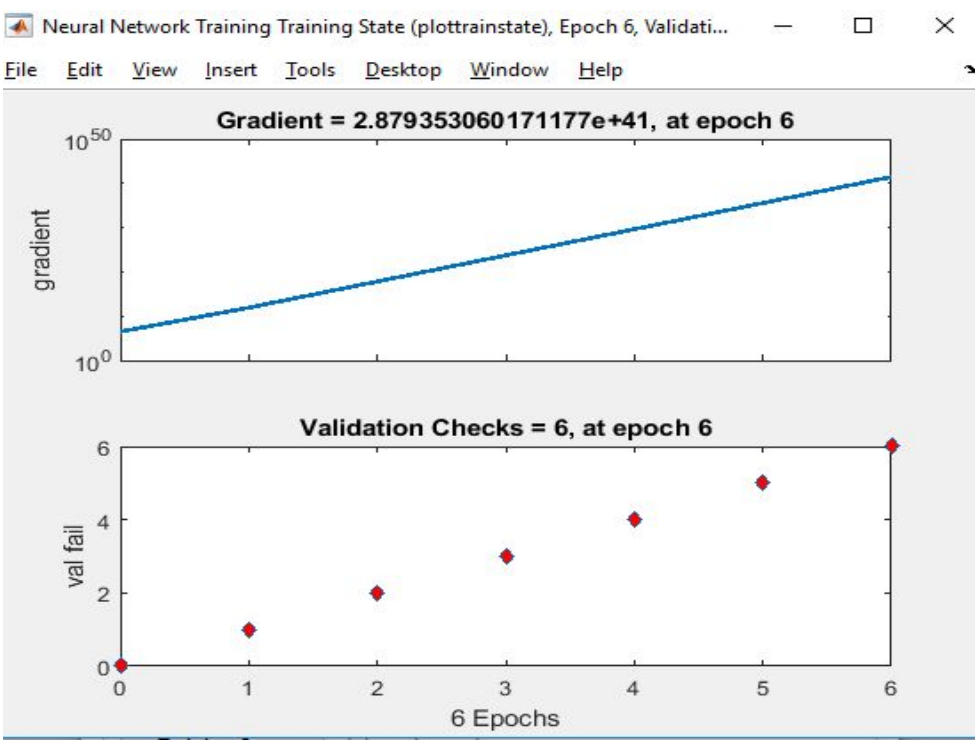

Neural Network Training Error Histogram (ploterrhist), Epoch 6, Validatio...
File Edit View Insert Tools Desktop Window Help
Error Histogram with 20 Bins
Training
Validation
Test
Zero Error
Instances
0 1 2 3 4 5 6 7 8 9
-1925 -1773 -1621 -1469 -1317 -1165 -1013 -861.1 -709.2 -557.3 -405.3 -253.4 -101.5 50.46 202.4 354.3 506.3 658.2 810.1 962
Errors = Targets - Outputs

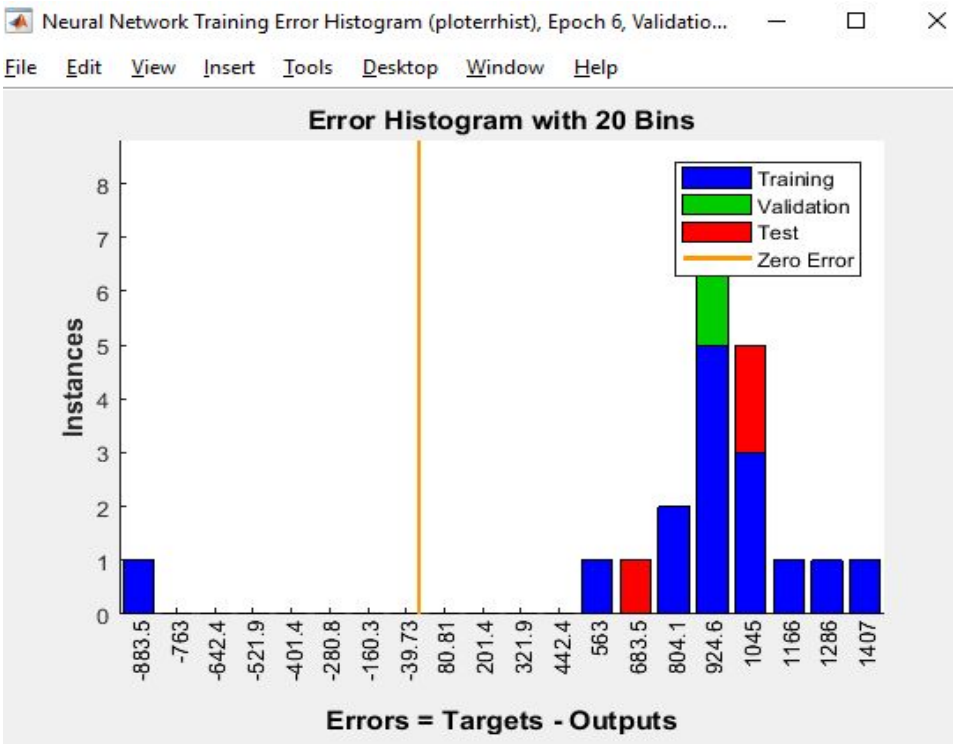

Training: R=-0.80156
Data
Fit
Y = T
Output ~= -0.79*Target + 3.3e+03
Target
Validation: R=-0.95063
Output ~= -0.63*Target + 3.2e+03
Test: R=-0.92024
Output ~= -0.05*Target + 2e+03
All: R=-0.44548
Output ~= -0.18*Target + 2.2e+03

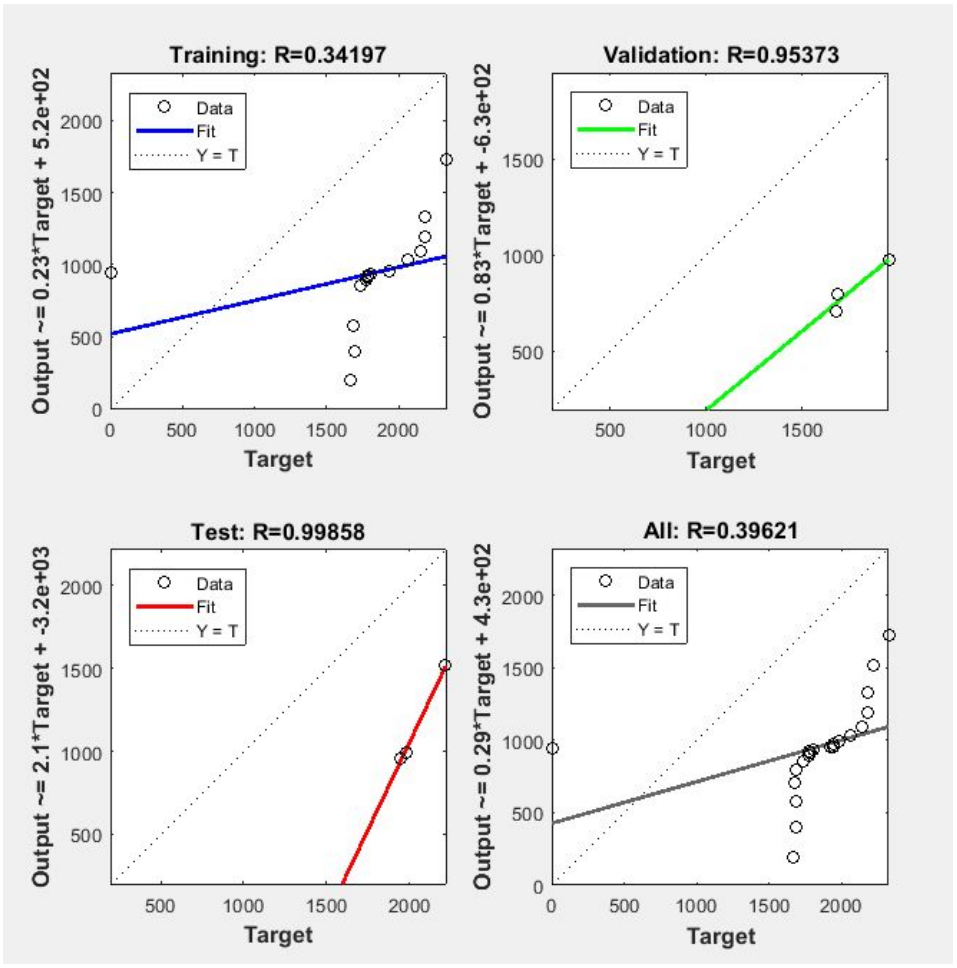
```
net = feedforwardnet(2);
net.trainParam.lr = 0.1;
net.trainParam.mc = 0.5;
```




Neural Network Training Performance (plotperform), Epoch 6, Validation...
File Edit View Insert Tools Desktop Window Help
Best Validation Performance is 890170.0628 at epoch 0
Train
Validation
Test
Best
Mean Squared Error (mse)
10^60
10^40
10^20
0 1 2 3 4 5 6
6 Epochs


Neural Network Training Training State (plottrainstate), Epoch 6, Validati...
File Edit View Insert Tools Desktop Window Help
Gradient = 2.879353060171177e+41, at epoch 6
gradient
10^50
10^0
Validation Checks = 6, at epoch 6
val fail
0 2 4 6
0 1 2 3 4 5 6
6 Epochs


Neural Network Training Error Histogram (ploterrhist), Epoch 6, Validatio...
File Edit View Insert Tools Desktop Window Help
Error Histogram with 20 Bins
Training
Validation
Test
Zero Error
Instances
0 1 2 3 4 5 6 7 8
-883.5 -763 -642.4 -521.9 -401.4 -280.8 -160.3 -39.73 80.81 201.4 321.9 442.4 563 683.5 804.1 924.6 1045 1166 1286 1407
Errors = Targets - Outputs


Training: R=0.34197
Data
Fit
Y = T
Output ~= 0.23*Target + 5.2e+02
Target
Validation: R=0.95373
Output ~= 0.83*Target + -6.3e+02
Target
Test: R=0.99858
Output ~= 2.1*Target + -3.2e+03
Target
All: R=0.39621
Output ~= 0.29*Target + 4.3e+02
Target

- **Wnioski**

Najlepsze wyniki zostały uzyskane w momencie gdy współczynnik uczenia był różny od współczynnika bezwładności.

Sieć wielowarstwowa bez algorytmu propagacji błędu działają lepiej niż sieci z wspomnianym algorytmem.

Błędy, które na wejściu i w trakcie działania wydają się być pomijalne mogą na końcu spowodować duży rozrzut między tym, co zostało zasymulowane a tym, co powinna dać sieć w swoim wyniku.

Sieci wielowarstwowe z algorytmem propagacji danych wykazują duże odsetki błędów.

Im większy jest wsp. uczenia od bezwładności, tym częściej mogą zdarzać się błędy.

Algorytmy z propagacją danych przyspieszają proces uczenia nawet o połowę epok w stosunku do sieci nie zawierających tego algorytmu.

Gradient uczenia (z algorytmem propagacji błędu) posiada charakterystykę niemal że liniową, dzięki czemu można łatwiej przewidzieć, jakich współczynników do nauki sieci najlepiej użyć, by otrzymane rezultaty były zadowalające.

Jeżeli nie korzystamy z algorytmu propagacji danych należy pamiętać, by były duże różnice między wsp. uczenia a bezwładnością.