MONASH University
Engineering

Assessment cover sheet

Unit and student details

Unit code	TRC 4901	**Unit title**	Artificial Intelligence		
If this is a group assignment, each student must include their name and ID number and sign the student statement.					
Student ID	29392004	**Surname**	Ranjan	**Given names**	Rivyesch
Student ID		**Surname**		**Given names**	
Student ID		**Surname**		**Given names**	
Student ID		**Surname**		**Given names**	

Assessment details

Title of assignment /lab	Lab 2 - Biometric Identification using Neural Network	**Authorised group assignment**	Yes ☐ [✓ No]
Lecturer/tutor	Dr. Parasuraman	**Tutorial day and time**	Wednesday 8 am
Due date	21/04/21	**Date submitted**	~~09/04/21~~ 19/04/21
Has any part of this assessment been previously submitted as part of another unit/course?			Yes ☐ [✓ No]

Submission date and extensions

All work must be submitted by the due date. If an extension of work is required, please complete and submit a Special Consideration application (in-semester assessment task) form to your examiner/lecturer/tutor.

Plagiarism and collusion

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations.

Plagiarism: Plagiarism means taking and using another person's ideas or manner of expressing them and passing them off as one's own. For example, by failing to give appropriate acknowledgement. The material used can be from any source (staff, students or the internet, published and unpublished works).

Collusion: Collusion means unauthorised collaboration with another person on assessable written, oral or practical work and includes paying another person to complete all or part of the work.

Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or delegate, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student statement and signature

Student Statement

I have read the university's Student Academic Integrity Policy and Procedures.
I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations http://www.monash.edu/legal/legislation/current-statute-regulations-and-related-resolutions
I have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
No part of this assignment has been previously submitted as part of another unit/course.
I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:

i. provide to another member of faculty and any external marker; and/or
ii. submit it to a text matching software; and/or
iii. submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.

I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.

Student signature	Rivyesch Ranjan	**Date**	05/04/21
Student signature		**Date**	
Student signature		**Date**	
Student signature		**Date**	

Please note that it is your responsibility to retain a copy of your assignment/laboratory work

Privacy Statement
The information on this form is collected for the primary purpose of assessing your assignment/laboratory work and ensuring the academic integrity requirements of the University are met. Other purposes of collection include recording your plagiarism and collusion declaration, attending to course and administrative matters, and statistical analyses. If you choose not to complete all the questions on this form Monash University it may not be possible for Monash University to assess your assignment/laboratory work. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. You have a right to access personal information that Monash University holds about you, subject to any exception in relevant legislation. If you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer: privacyofficer@adm.monash.edu.au

V1.3 October 2016

Contents

1.0 Introduction 3

2.0 Preliminary Studies 3

3.0 Methodology 3

 3.1 Preparation of Inputs for Training 3

 3.2 Designation of BIS Target 4

 3.3 Network Architecture 4

 3.4 Training Neural Network 5

4.0 Results 5

 4.1 Variation to Layers and Neurons 5

 4.2 Variation to Training Algorithm 6

 4.3 Variation to Transfer Function 6

5.0 Discussion 6

6.0 Conclusion 7

7.0 References 8

8.0 Appendix 8

 8.1 Important Figures 8

  8.1.1 Case 1 8

  8.1.2 Case 2 9

  8.1.3 Case 3 10

  8.1.4 Case 4 11

  8.1.5 Case 5 12

  8.1.6 Case 6 13

  8.1.7 Case 7 14

  8.1.8 Case 8 15

  8.1.9 GD 16

  8.1.10 GDM 17

  8.1.11 CGF 18

  8.1.12 CGB 19

  8.1.13 SCG 20

  8.14 Logsig 21

  8.15 Purelin 22

 8.2 MATLAB Code 23

1.0 Introduction

The objective of this lab is to design a back-propagation neural network for Biometric authentication purposes capable of identifying the identity of a person from an image of their fingerprint. The Biometric Identification system (BIS) which is what the model designed on MATLAB is called is trained with two fingerprint samples as the input. Since input images are rarely of perfect quality, the BIS is trained with and without noise. Two separate studies were conducted on the BIS model designed. Firstly, the effect of varying the number of layers and the number of neurons in each layer is investigated and the optimum hyper-parameters of the model is determined. Secondly, the accuracy of the BIS was tested using various training algorithms.

2.0 Preliminary Studies

An artificial neural network (ANN) is a network that replicates the working mechanisms of the human nervous system. It comprises of numerous interconnected neurons that act as parallel information-processors to solve classification or regression problems. An ANN only requires historical data to learn and train itself, after which it can then make predictions based on new input data.

ANNs as shown in Figure 1 consist of three types of layers – input, hidden and output. In the input layer a neuron receives input from the historical data. Within the hidden layer each neuron is connected to other neurons in other layers by a number called a weight. These neurons receive inputs from the input layer and produces output based on its internal activation function as seen in Figure 2. The functions as well as the weights are altered by a process known as learning. In this layer information is passed back-and-forth between layers. Meanwhile in the output layer the processed information is relayed out.

These models in recent years have garnered a lot of attention of researchers. They have shown better performance due to their capacity and flexibility to map non-linear relationships from data given their deep structure. Another advantage would be its ability to generalise information and extract temporal patterns automatically [1].

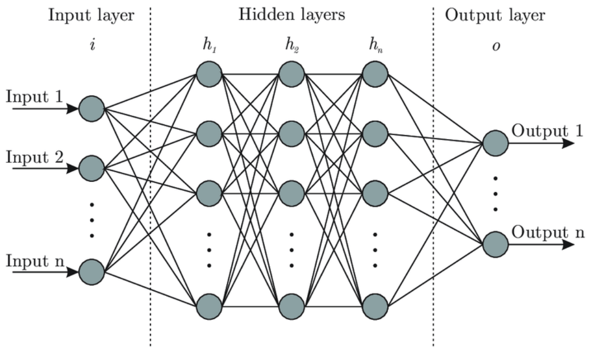


Figure 1: Network Architecture of an ANN

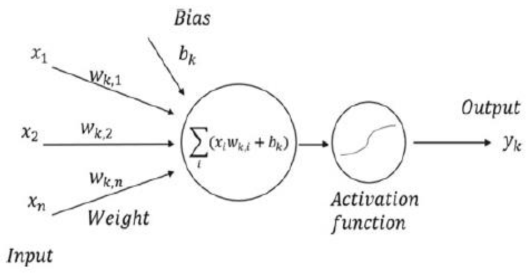


Figure 2: Structure of an Artificial Neuron

3.0 Methodology

3.1 Preparation of Inputs for Training

Two fingerprints shown below were used in the training process of the neural network. They were first loaded as images into MATLAB and then converted to black and white using the in-built function "im2bw". A threshold of 0.5 was used for both fingerprint images.

Figure 3: Left and Right Fingerprint Image used as Input

Instead of using the whole fingerprint as the input which would require high computational memory, a small window of size 21x21 was cropped and used as an input for the training of the model. As often is the case, input images are rarely perfect or ideal. To make the BIS model robust and able to deal with inputs of varying quality, the small window cropped from each fingerprint sample was subjected to noise. This is achieved by using the custom MATLAB function provided called AddNoise. It essentially selects a specified number of random pixels and inverts its' binary values, thus creating noise. Overall, there are five samples for each fingerprint, four with noise and one without noise introduced to it shown on the far left of each row in Figure 4. Each row corresponds to each fingerprint sample. Each training input images is then reshaped to be one column. The first five columns or the left half of the picture in Figure 5 belongs to the binary inputs corresponding to the first fingerprint while the rest belong to the second fingerprint.

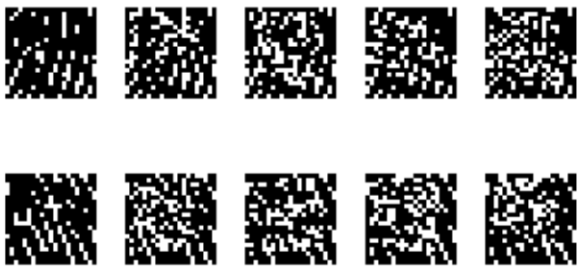


Figure 4: Input Images which include Noise Samples

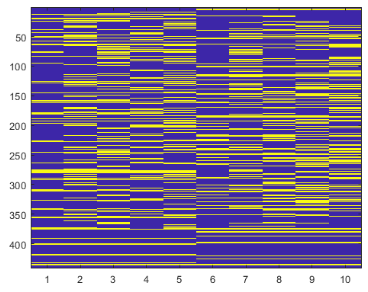


Figure 5: Biometric Input after Reshaping

3.2 Designation of BIS Target

A target is then set to determine which set of images correspond to which fingerprint. The point of specifying a target value is to give the BIS model a basis for comparison between the predicted and actual value which it then can use to reduce the error using the back-propagation technique. The following Figure 6 below represents the target for the BIS.

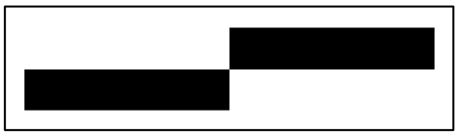


Figure 6: Target Output

3.3 Network Architecture

As previously mentioned, the neural network utilises back-propagation training for learning. Initially, a model would not give the most accurate possible results due to its weights not being tuned yet. The basis of back-propagation is that it determines the difference between the actual output and the simulated output, called the loss, and propagates this loss back into the neural network. By doing this the weights of the network are then fine-tuned based on the loss obtained during the previous iteration. Through multiple iterations, the weights are optimised to ensure the lowest error rate. Hence, this creates a highly reliable model [1].

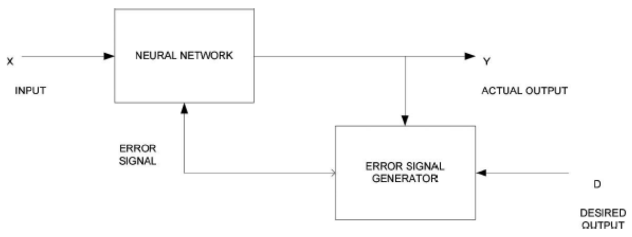


Figure 7: Block Diagram of Supervised Learning Methods

3.4 Training Neural Network

Numerous models with different network architecture and hyper-parameters were trained and compared to identify and analyse the effects of these changes on the performance. For this lab the only parameters that are varied are the training algorithms, the number of layers and the number of neurons in each of the layers, and the transfer function. The results from these studies are presented and analysed below in the results and discussion section respectively.

In order to create a controlled study environment for fair comparisons between the various models trained, certain standard parameters were fixed across all the models trained and are as follows:

- Maximum number of Epochs: 1000
- Learning Rate: 0.08
- Goal: 0
- Training Function: Sigmoid function (Tansig)

4.0 Results

4.1 Variation to Layers and Neurons

Table 1: Variation of Number of Neurons for a Two Layer Network

	Levenberg-Marquardt Backpropagation Algorithm					
Case	Layers	No. of Neurons	Iterations	Elapsed Time (s)	MSE	Regression
1	1	10	10	6	7.58E-8	1
	2	2				
2	1	20	8	30	5.82E-9	1
	2	2				
3	1	30	9	90	6.28E-4	0.99977
	2	2				
4	1	40	9	260	3.30E-5	0.99999
	2	2				

Table 2: Variation of Number of Neurons for a Three Layer Network

	Levenberg-Marquardt Backpropagation Algorithm					
Case	Layers	No. of Neurons	Iterations	Elapsed Time (s)	MSE	Regression
5	1	10	10	6	5.48E-6	1
	2	10				
	3	2				
6	1	20	10	54	2.01E-8	1

	2	20				
	3	2				
7	1	30	6	59	1.36E-2	0.51671
	2	30				
	3	2				
8	1	40	10	405	1.93E-7	1
	2	40				
	3	2				

4.2 Variation to Training Algorithm

The following networks all have two layers with 20 neurons in the hidden layers and two in the output layer. This corresponds to the network in Case 2 with the training algorithm varied.

Table 3: Variation of Training Algorithms for a Two Layer Network

Algorithm Type	Iterations	Time Elapsed (s)	MSE	Regression
Gradient Descent Backpropagation	1000	2	6.64E-2	0.95916
Gradient Descent with Momentum Backpropagation	1000	2	4.50E-2	0.97904
Fletcher-Powell Conjugate Gradient Backpropagation	0	0	1.47E-16	1
Powell-Beale Conjugate Gradient Backpropagation	0	0	1.13E-13	1
Scaled Conjugate Gradient Backpropagation	5	0	1.13E-13	1
Levenberg-Marquardt Backpropagation	8	30	5.82E-9	1

4.3 Variation to Transfer Function

All three of the networks trained below correspond to the network used for Case 2 with the Scaled Conjugate Gradient Backpropagation training algorithm. The only changes made is to the transfer function of the sole hidden layer which consist of 20 neurons.

Table 4: Variation of Transfer Function of Hidden Layer for a Two Layer Network

Transfer Function Type	Iterations	Time Elapsed (s)	MSE	Regression
Hyperbolic Tangent Sigmoid (Tansig)	1000	2	6.64E-2	0.95916
Log-Sigmoid (Logsig)	1000	2	4.50E-2	0.97904
Linear (Purelin)	0	0	1.47E-16	1

5.0 Discussion

From the results obtained in section 4.1 there are two obvious trends. The first is that using more layers increases the computational complexity of the trained model which results in generally higher training time. The networks with two layers could be trained faster than networks with three layers. Furthermore, referring to Table 1 and Table 2 comparing the variation of neurons for a trained network it is evident that increasing the number of neurons also increase the time required to train the model. This is the case for both networks with two and three layers.

As for the performance of the neural networks trained both two-layer networks and three-layer networks perform best when there are 20 neurons in each of its hidden layer. For two-layer networks the results illustrate that the performance in terms of Mean Squared Error (MSE) is best for those which have fewer neurons in its layers. The performance worsens slightly as the number of neurons increases. When the network has three layers, there is no clear pattern as the MSE tends to fluctuate. It is low when the number of neurons is few, however, this is only true up to a certain point after which the performance begins to degrade and the error increases. When the number of neurons is increased further the MSE once more begins to decrease and reaches the low level it was at before.

Comparing the networks with two layers and three layers showed that for the same number of neurons in each hidden layer, the performance of the two-layer networks is significantly better. The only exception to this is when comparing Case 4 and Case 8 which both have 40 neurons in each hidden layer. Here the three-layer network which corresponds to Case 8 performs much better and has a lower MSE.

Overall, the optimal network architecture corresponds to Case 2 which has two layers, one of which is a hidden layer consisting of 20 neurons. The MSE for this network is 5.82 E-9 which is the lowest MSE obtained out of the eight cases trained and studied. The network in Case 2 uses the Levenberg-Marquardt training algorithm and Tansig transfer function. These parameters were fixed beforehand. However, since these parameters have not been optimised yet the subsequent studies conducted in section 4.2 and 4.3 aimed to determine the best training algorithm and transfer function for Case 2.

A separate study in section 4.2 was done to analyse the effect of different training algorithms on the performance of the network. The architecture of Case 2 was chosen and kept constant throughout this study. Only the training algorithms were varied with all other parameters fixed. The type of training algorithms used can be viewed in Table 3. The best training algorithm for this experiment was found to be Fletcher-Powell Conjugate Gradient Backpropagation. It gave the lowest MSE at 1.47 E-16 which indicates a high level of accuracy. The next best algorithm was Powell-Beale Conjugate Gradient Backpropagation and Scaled Conjugate Gradient Backpropagation which both gave an extremely low MSE of 1.13 E-13. The worst model was the Gradient Descent Backpropagation which not only gave the highest MSE but also the lowest regression.

Lastly, the influence of the transfer functions on the performance of the network is investigated in section 4.3. The results clearly shows that the Purelin transfer function provides the best model with the lowest error obtained. The MSE was only at 1.47E-16 and had a perfect regression of 1. It is evident that using this transfer function would give the most accurate results for this lab.

6.0 Conclusion

This lab provided an in-depth understanding to the use of neural networks and the importance of various parameters to the performance of the neural network. All of the studies conducted shows that the optimisation and fine tuning of parameters and architecture of network is crucial to designing an optimal model. For every different scenario, the number of neurons and hidden layers needed for an optimal model is unique.

For this particular lab the optimal model consisted of two layers in total, one of which is hidden consisting of 20 neurons. The Fletcher-Powell Conjugate Gradient Backpropagation is the best training algorithm while the Purelin transfer function is the most ideal transfer function. The fully optimised model gives an MSE of 1.47 E-16 and a regression of 1. Furthermore, it is extremely fast to train. This means the fully optimised model can work really well as a Biometric Authentication system for identifying a person based on their fingerprint.

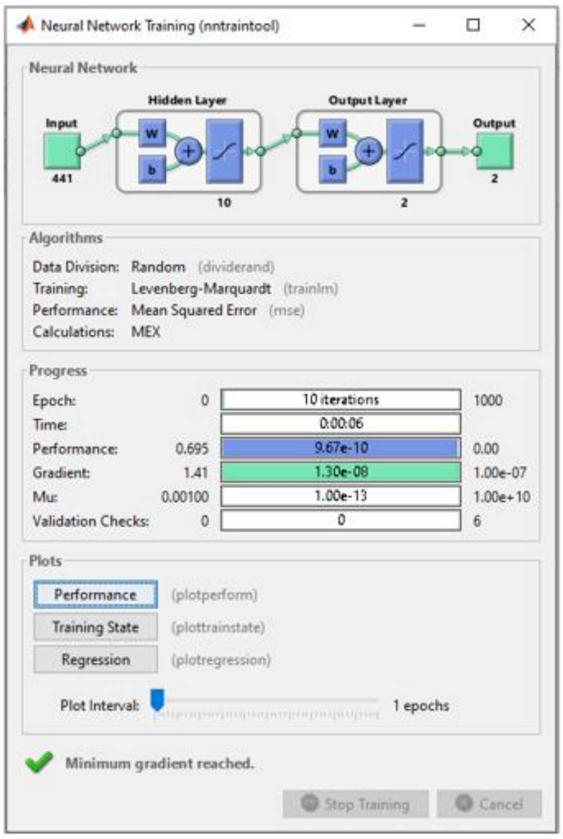
7.0 References

[1] S. Sengupta et al., "A review of deep learning with special emphasis on architectures, applications and recent trends", *Knowledge-Based Systems*, vol. 194, p. 105596, 2020. Available: 10.1016/j.knosys.2020.105596.

8.0 Appendix

8.1 Important Figures

8.1.1 Case 1



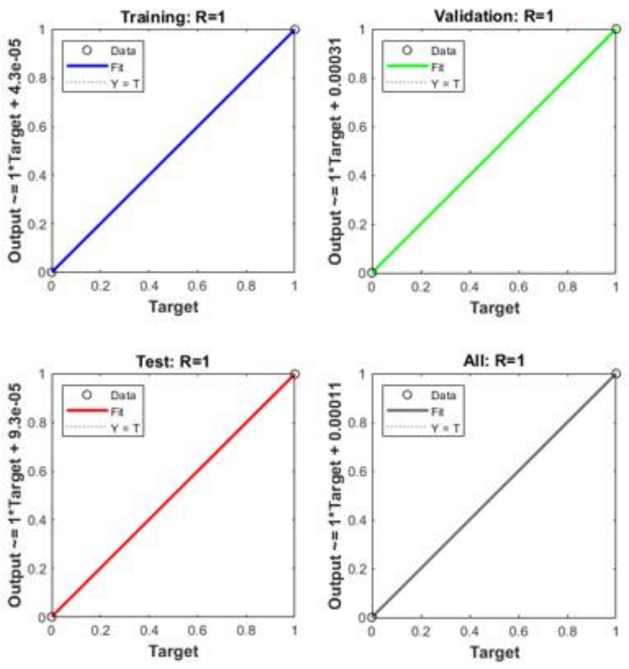
Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

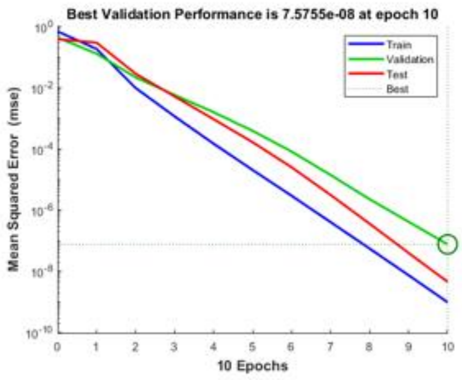
Progress

Epoch:	0	10 iterations	1000
Time:		0:00:06	
Performance:	0.695	9.67e-10	0.00
Gradient:	1.41	1.30e-08	1.00e-07
Mu:	0.00100	1.00e-13	1.00e+10
Validation Checks:	0	0	6

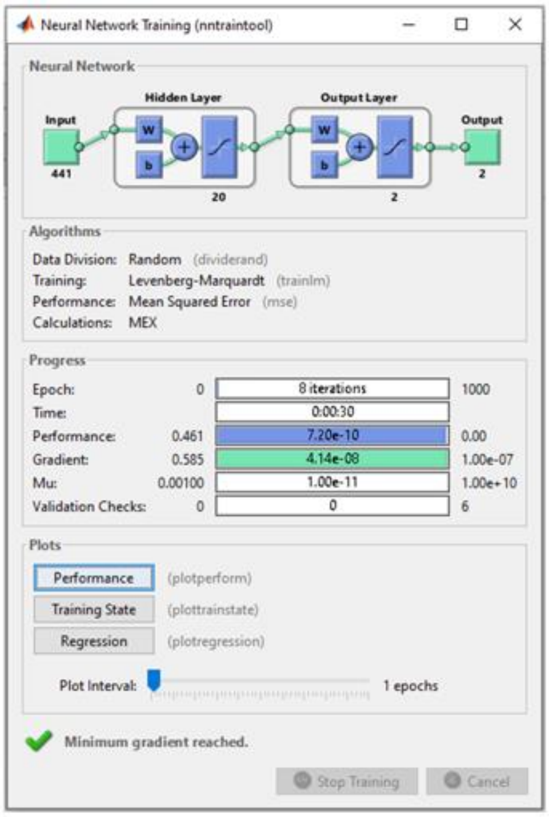
Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs

Minimum gradient reached.





8.1.2 Case 2



Neural Network Training (nntraintool)

Neural Network

Input
441
Hidden Layer
W
b
20
Output Layer
W
b
2
Output
2

Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

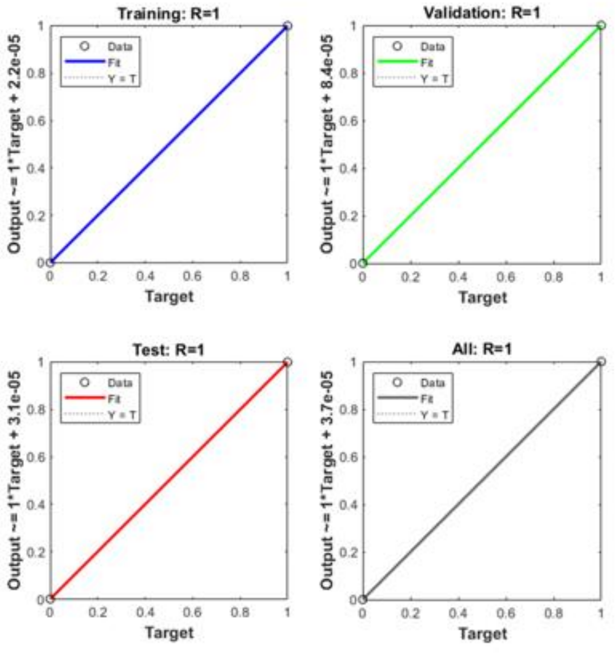
Progress

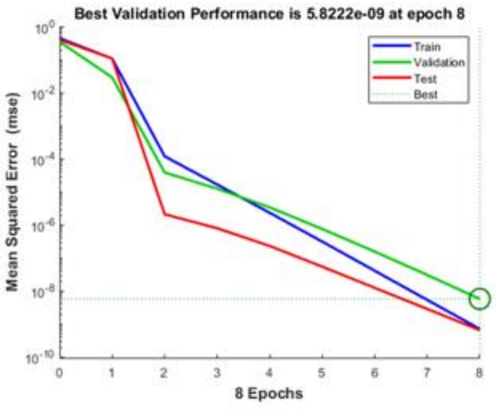
Epoch:	0	8 iterations	1000
Time:		0:00:30	
Performance:	0.461	7.20e-10	0.00
Gradient:	0.585	4.14e-08	1.00e-07
Mu:	0.00100	1.00e-11	1.00e+10
Validation Checks:	0	0	6

Plots

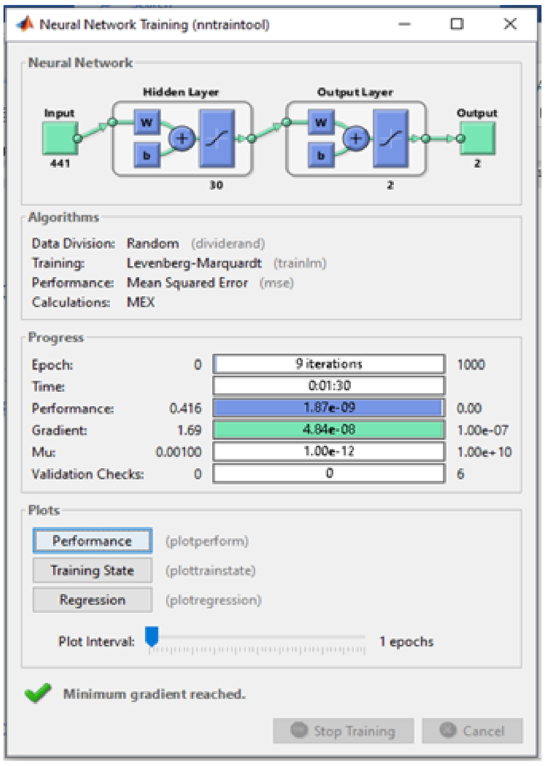
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs

Minimum gradient reached.





8.1.3 Case 3



Neural Network Training (nntraintool)

Neural Network

Input 441 — Hidden Layer 30 — Output Layer 2 — Output 2

Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

Progress

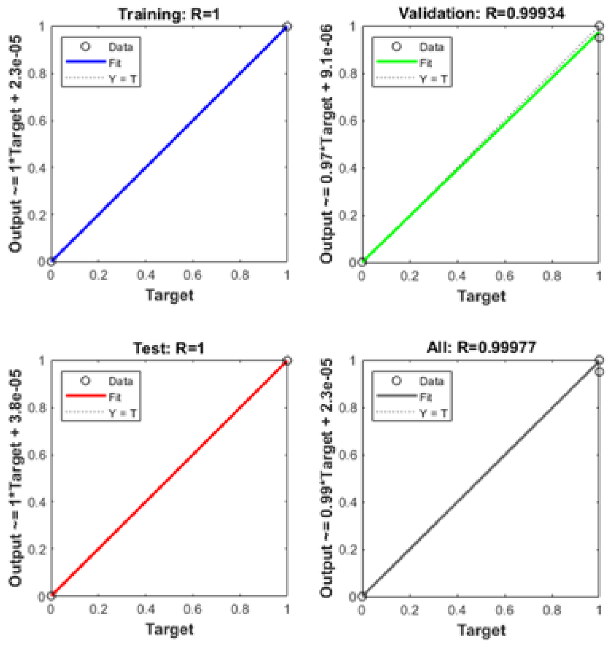
Epoch:	0	9 iterations	1000
Time:		0:01:30	
Performance:	0.416	1.87e-09	0.00
Gradient:	1.69	4.84e-08	1.00e-07
Mu:	0.00100	1.00e-12	1.00e+10
Validation Checks:	0	0	6

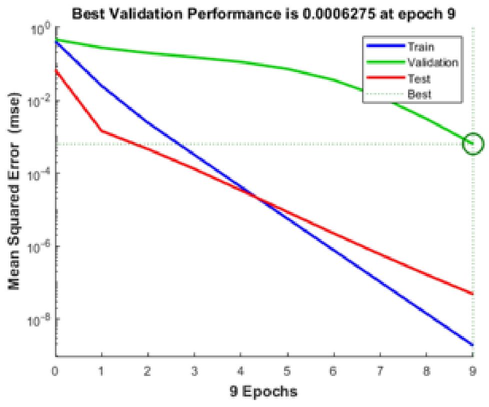
Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

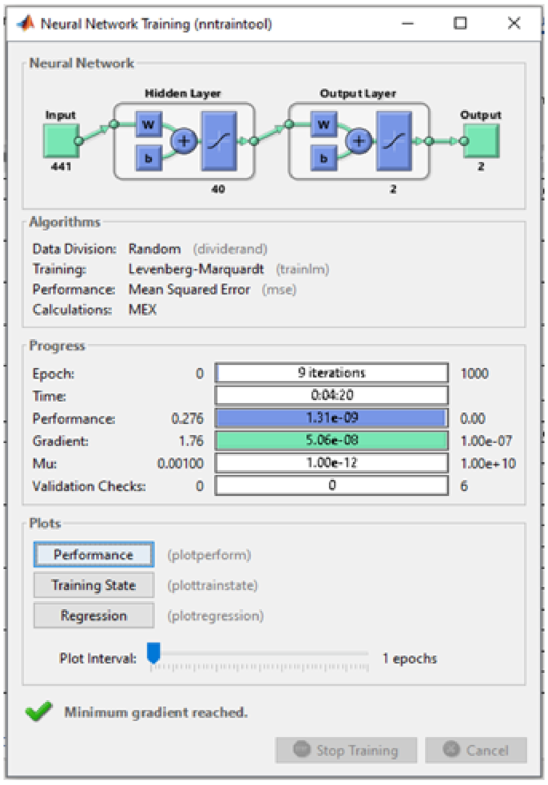
Plot Interval: 1 epochs

Minimum gradient reached.





8.1.4 Case 4


Neural Network Training (nntraintool)

Neural Network

Hidden Layer
Output Layer
Input
Output
W
b
441
40
2
2

Algorithms

```
Data Division:  Random  (dividerand)
Training:       Levenberg-Marquardt  (trainlm)
Performance:    Mean Squared Error  (mse)
Calculations:   MEX
```

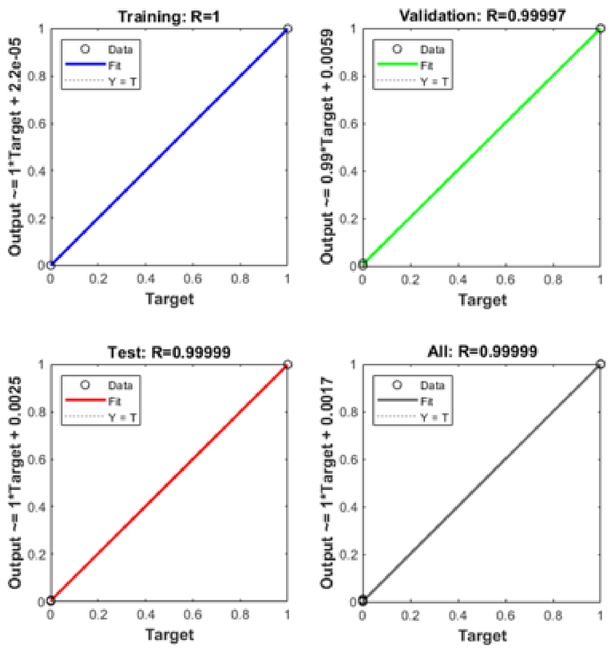
Progress

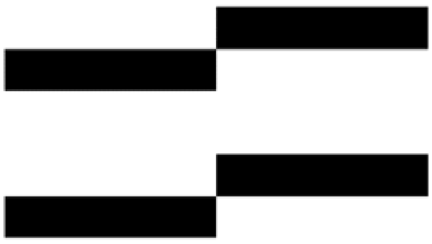
```
Epoch:                 0   9 iterations   1000
Time:                      0:04:20
Performance:       0.276   1.31e-09       0.00
Gradient:           1.76   5.06e-08       1.00e-07
Mu:              0.00100   1.00e-12       1.00e+10
Validation Checks:     0   0              6
```

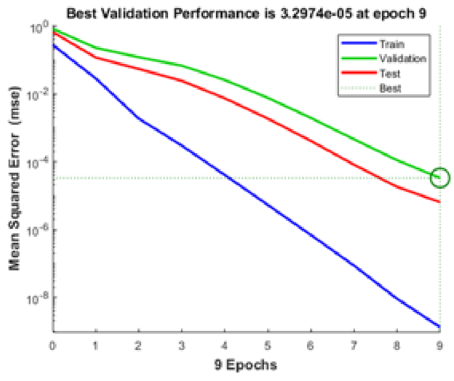
Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval:
1 epochs

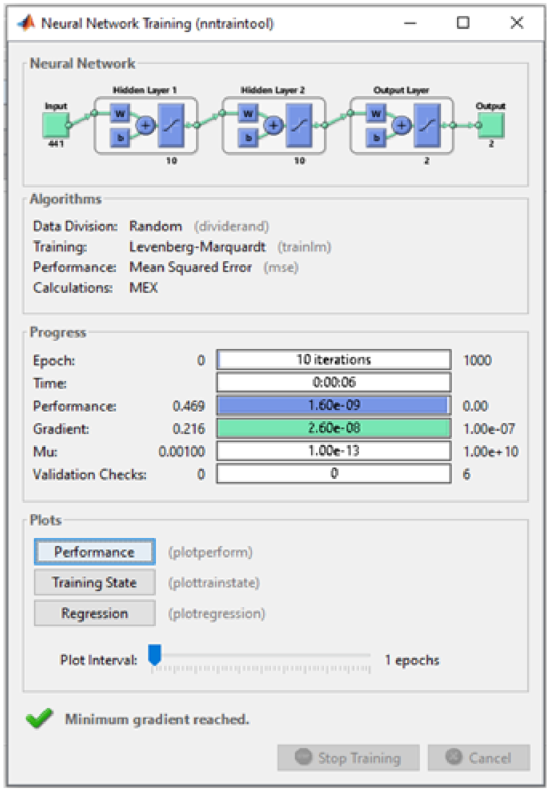
Minimum gradient reached.







8.1.5 Case 5


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

Progress

Epoch:	0	10 iterations	1000
Time:		0:00:06	
Performance:	0.469	1.60e-09	0.00
Gradient:	0.216	2.60e-08	1.00e-07
Mu:	0.00100	1.00e-13	1.00e+10
Validation Checks:	0	0	6

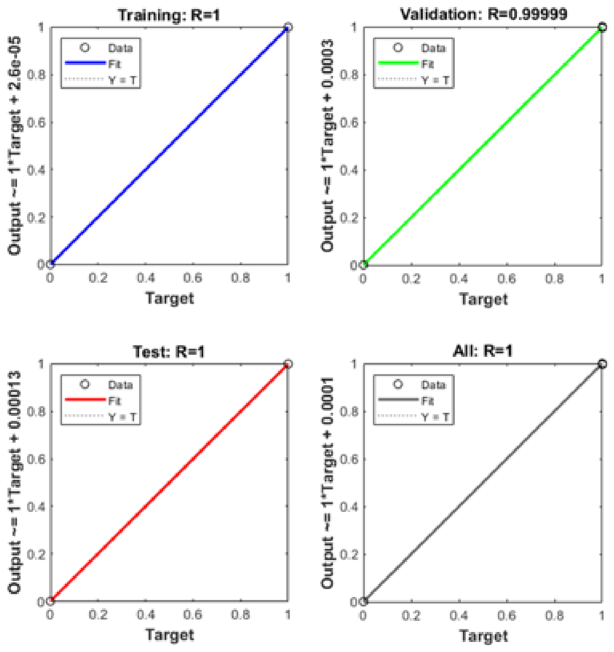
Plots

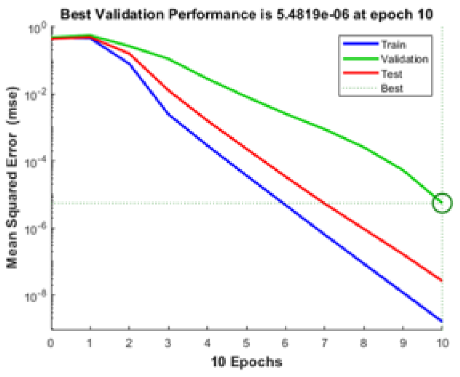
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

Plot Interval: 1 epochs

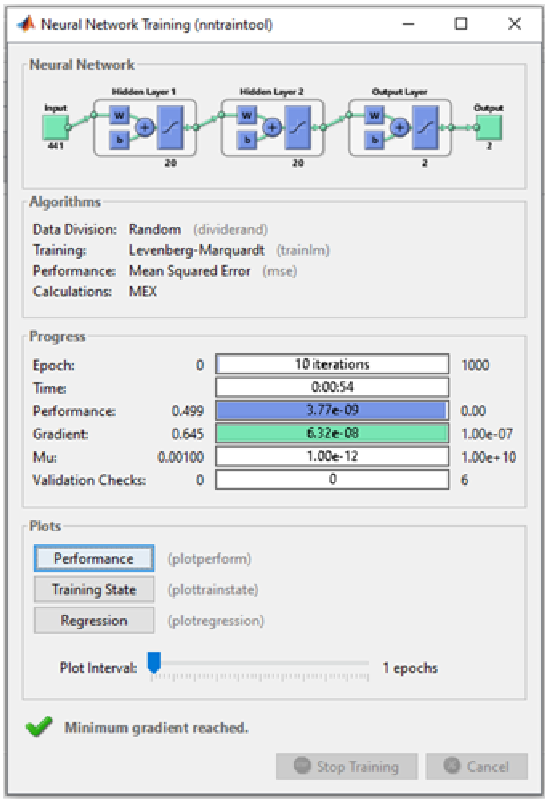
✓ Minimum gradient reached.

Stop Training Cancel





8.1.6 Case 6


Neural Network Training (nntraintool)

Neural Network

Input
441
Hidden Layer 1
20
Hidden Layer 2
20
Output Layer
2
Output
2

Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

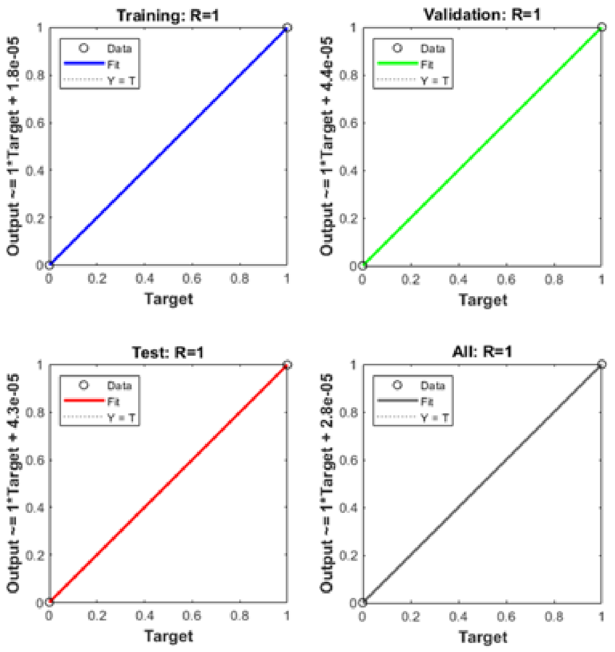
Progress

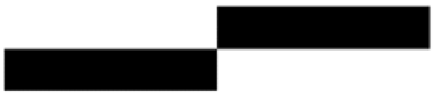
```
Epoch:                0     10 iterations     1000
Time:                          0:00:54
Performance:      0.499       3.77e-09        0.00
Gradient:         0.645       6.32e-08        1.00e-07
Mu:             0.00100       1.00e-12        1.00e+10
Validation Checks:    0           0           6
```

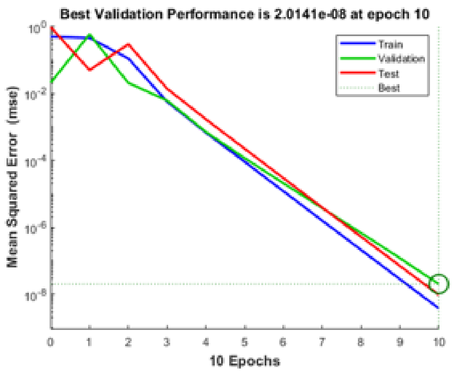
Plots

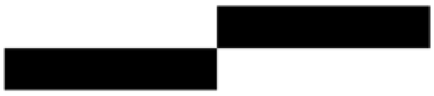
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs

Minimum gradient reached.

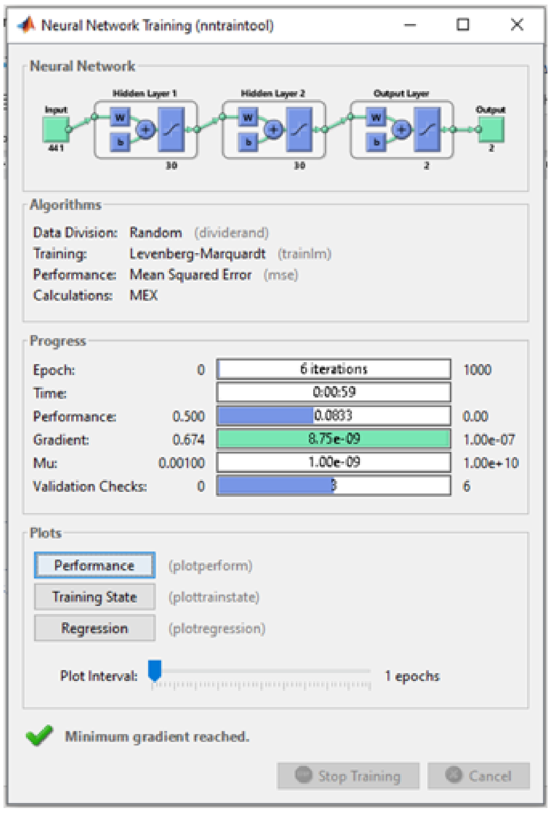








8.1.7 Case 7


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division:	Random	(dividerand)
Training:	Levenberg-Marquardt	(trainlm)
Performance:	Mean Squared Error	(mse)
Calculations:	MEX	

Progress

Epoch:	0	6 iterations	1000
Time:		0:00:59	
Performance:	0.500	0.0833	0.00
Gradient:	0.674	8.75e-09	1.00e-07
Mu:	0.00100	1.00e-09	1.00e+10
Validation Checks:	0	3	6

Plots

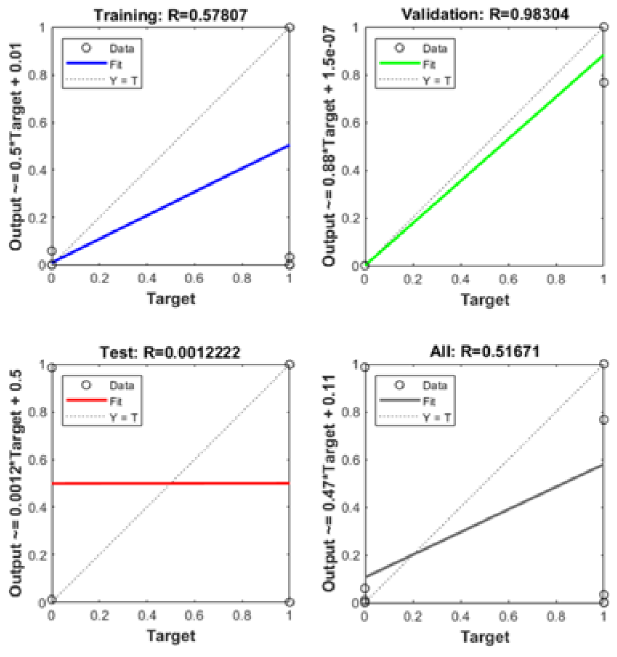
Performance (plotperform)

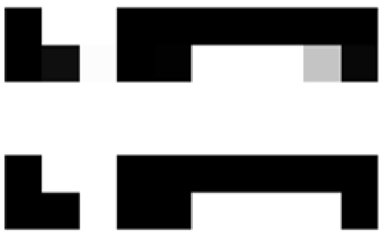
Training State (plottrainstate)

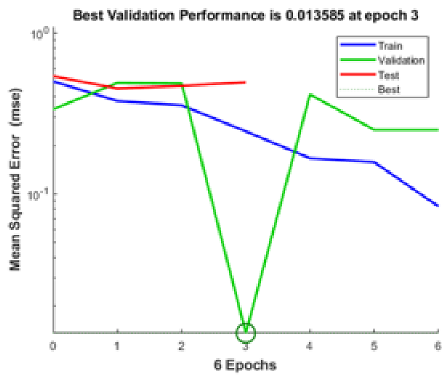
Regression (plotregression)

Plot Interval: 1 epochs

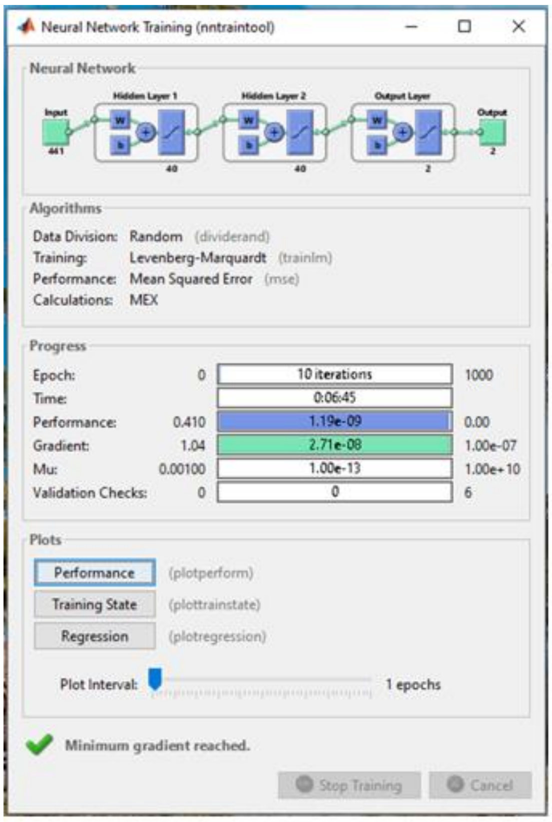
Minimum gradient reached.







8.1.8 Case 8


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

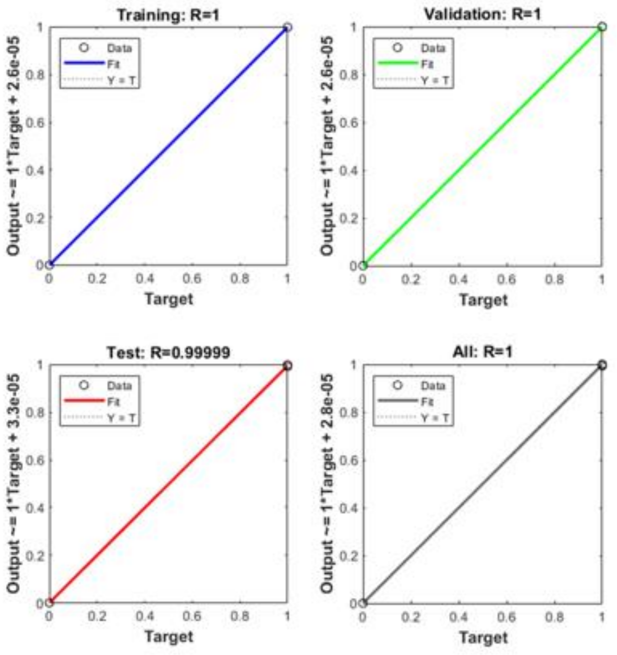
Progress

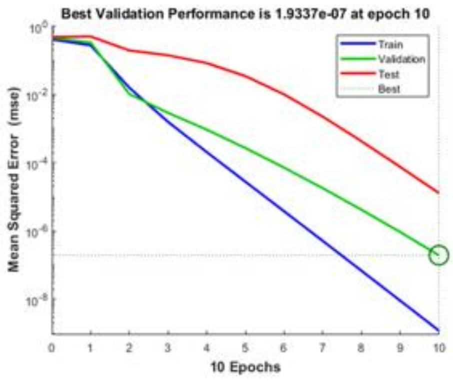
Epoch:	0	10 iterations	1000
Time:		0:06:45	
Performance:	0.410	1.19e-09	0.00
Gradient:	1.04	2.71e-08	1.00e-07
Mu:	0.00100	1.00e-13	1.00e+10
Validation Checks:	0	0	6

Plots

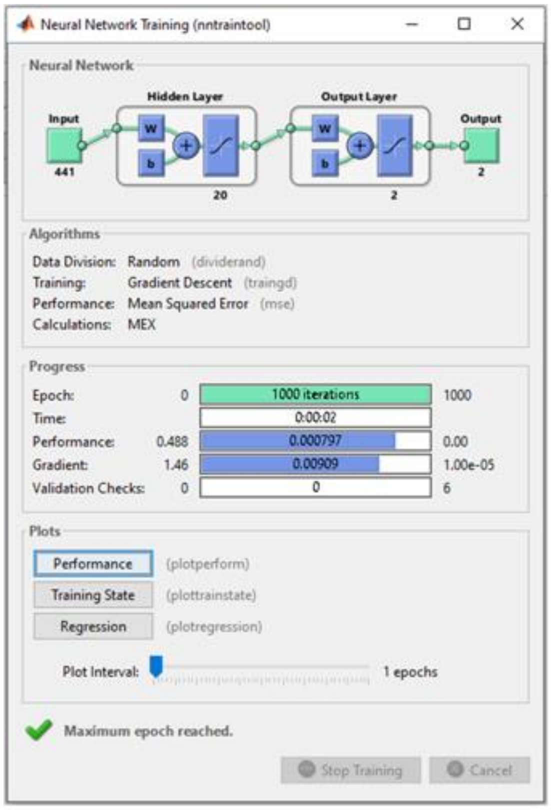
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs
Stop Training
Cancel

Minimum gradient reached.





8.1.9 GD


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division:	Random	(dividerand)
Training:	Gradient Descent	(traingd)
Performance:	Mean Squared Error	(mse)
Calculations:	MEX	

Progress

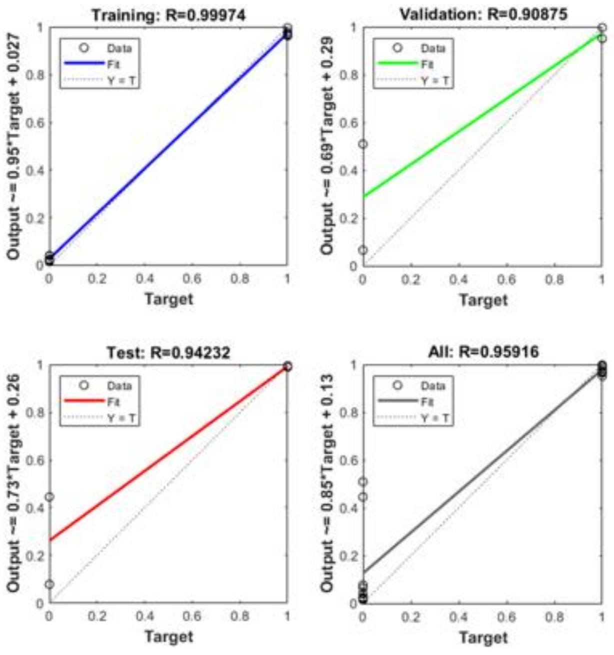
Epoch:	0	1000 iterations	1000
Time:		0:00:02	
Performance:	0.488	0.000797	0.00
Gradient:	1.46	0.00909	1.00e-05
Validation Checks:	0	0	6

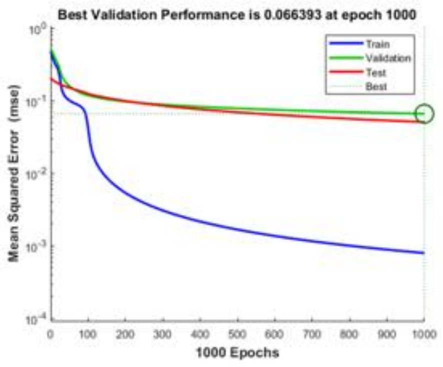
Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

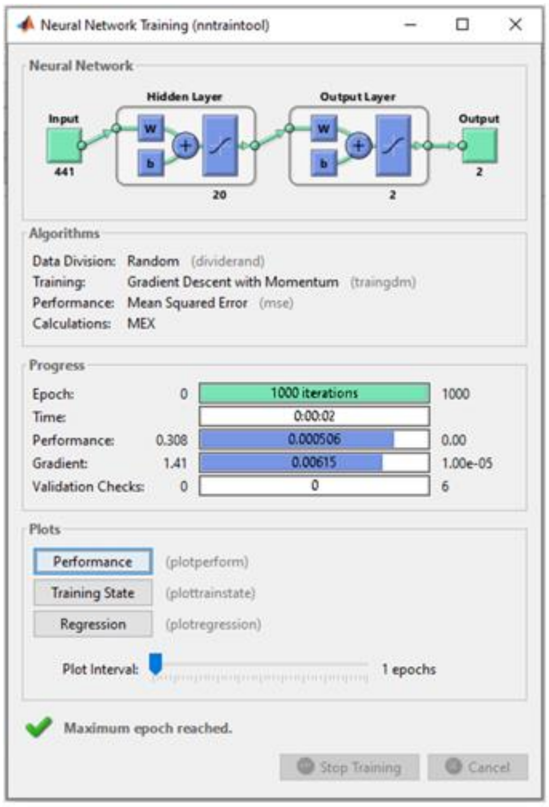
Plot Interval: 1 epochs

Maximum epoch reached.





8.1.10 GDM


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division: Random (dividerand)
Training: Gradient Descent with Momentum (traingdm)
Performance: Mean Squared Error (mse)
Calculations: MEX

Progress

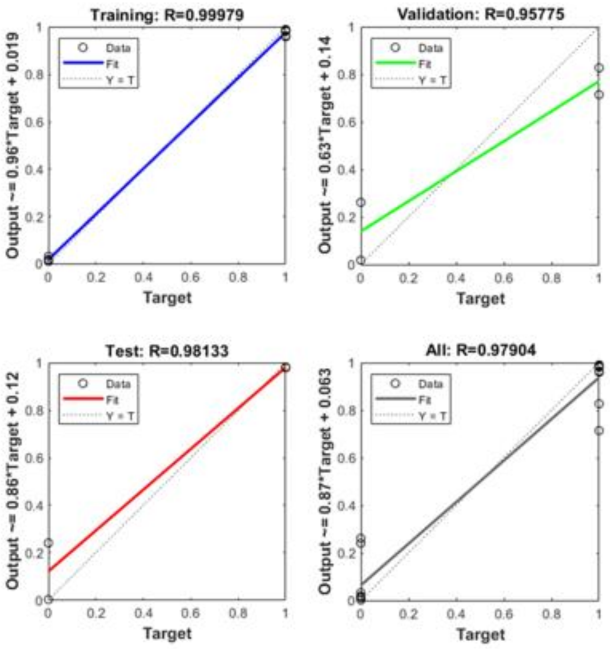
Epoch:	0	1000 iterations	1000
Time:		0:00:02	
Performance:	0.308	0.000506	0.00
Gradient:	1.41	0.00615	1.00e-05
Validation Checks:	0	0	6

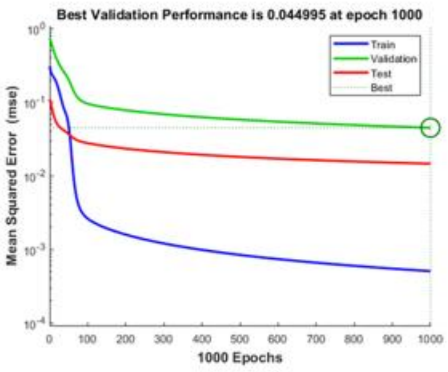
Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

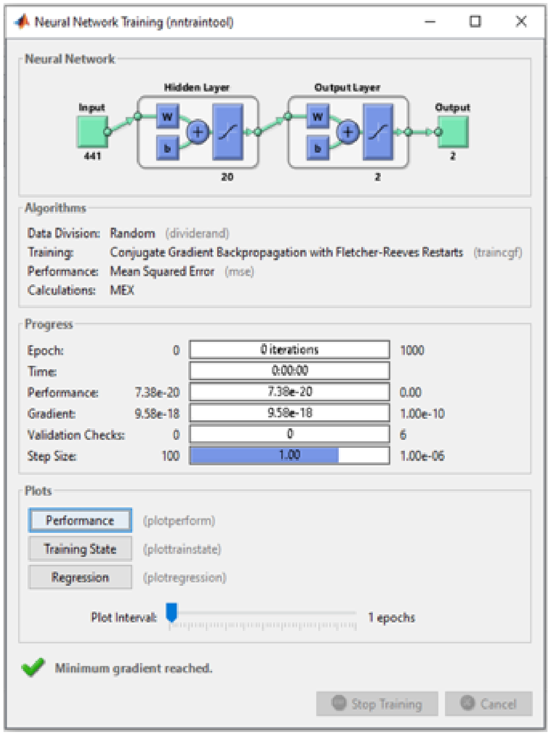
Plot Interval: 1 epochs

Maximum epoch reached.





8.1.11 CGF


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division: Random (dividerand)
Training: Conjugate Gradient Backpropagation with Fletcher-Reeves Restarts (traincgf)
Performance: Mean Squared Error (mse)
Calculations: MEX

Progress

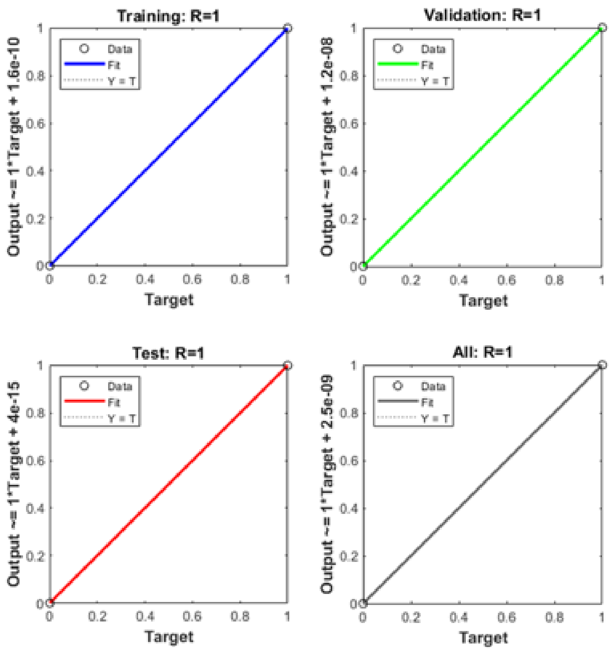
```
Epoch:                0      0 iterations     1000
Time:                        0:00:00
Performance:   7.38e-20      7.38e-20         0.00
Gradient:      9.58e-18      9.58e-18         1.00e-10
Validation Checks:    0      0                6
Step Size:          100      1.00             1.00e-06
```

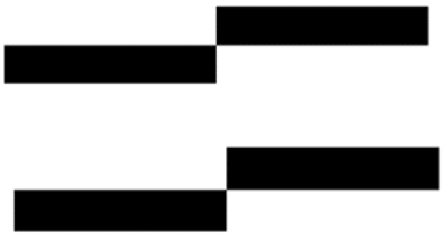
Plots

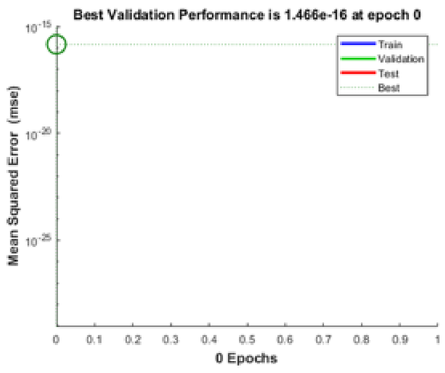
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

Plot Interval: 1 epochs

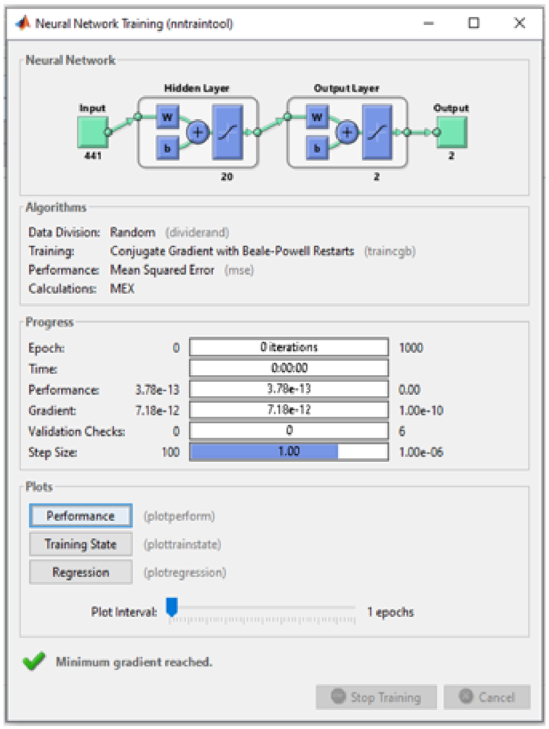
✓ Minimum gradient reached.







8.1.12 CGB



Neural Network Training (nntraintool)

Neural Network

Algorithms

```
Data Division:  Random  (dividerand)
Training:       Conjugate Gradient with Beale-Powell Restarts  (traincgb)
Performance:    Mean Squared Error  (mse)
Calculations:   MEX
```

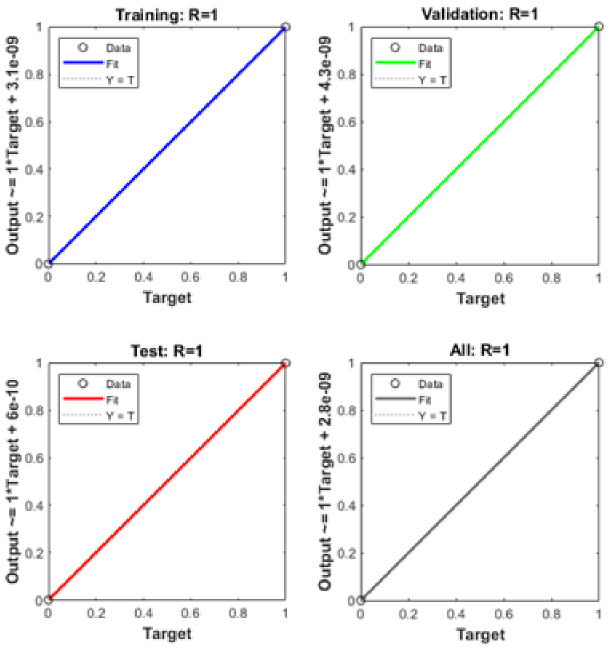
Progress

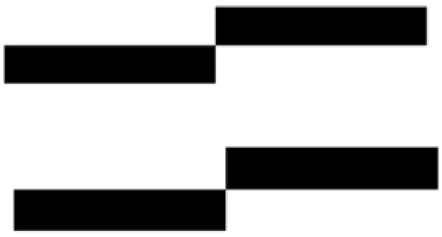
```
Epoch:                  0    0 iterations    1000
Time:                        0:00:00
Performance:     3.78e-13    3.78e-13        0.00
Gradient:        7.18e-12    7.18e-12        1.00e-10
Validation Checks:      0    0               6
Step Size:            100    1.00            1.00e-06
```

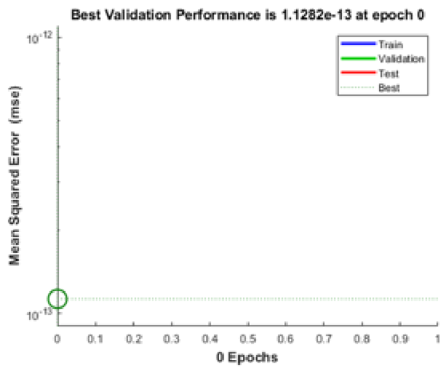
Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs

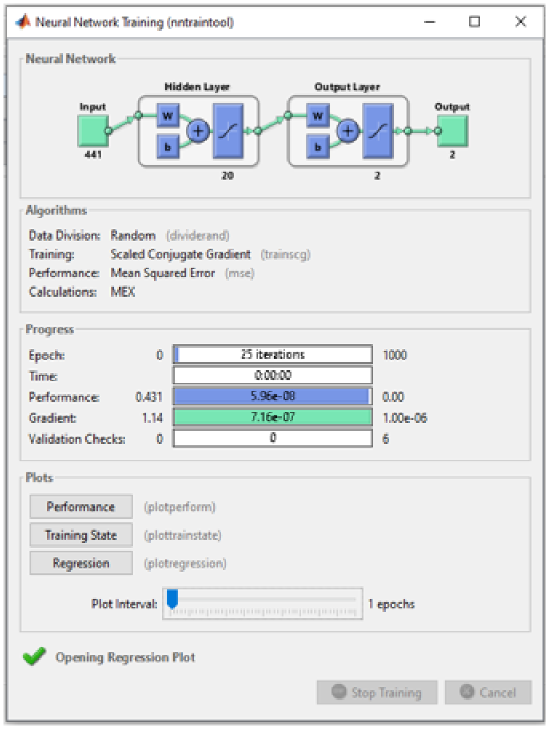
Minimum gradient reached.







8.1.13 SCG


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division:	Random (dividerand)
Training:	Scaled Conjugate Gradient (trainscg)
Performance:	Mean Squared Error (mse)
Calculations:	MEX

Progress

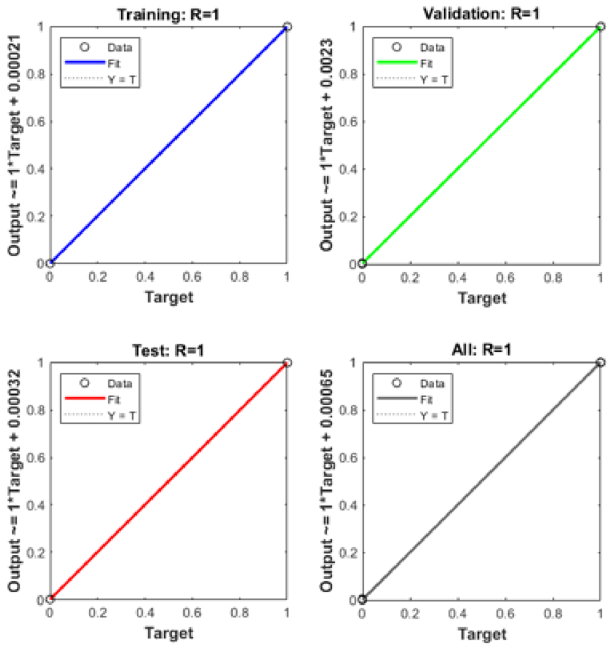
Epoch:	0	25 iterations	1000
Time:		0:00:00	
Performance:	0.431	5.96e-08	0.00
Gradient:	1.14	7.16e-07	1.00e-06
Validation Checks:	0	0	6

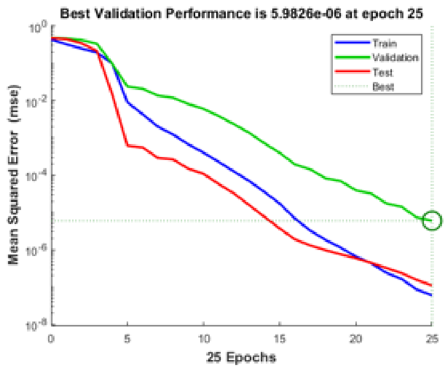
Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

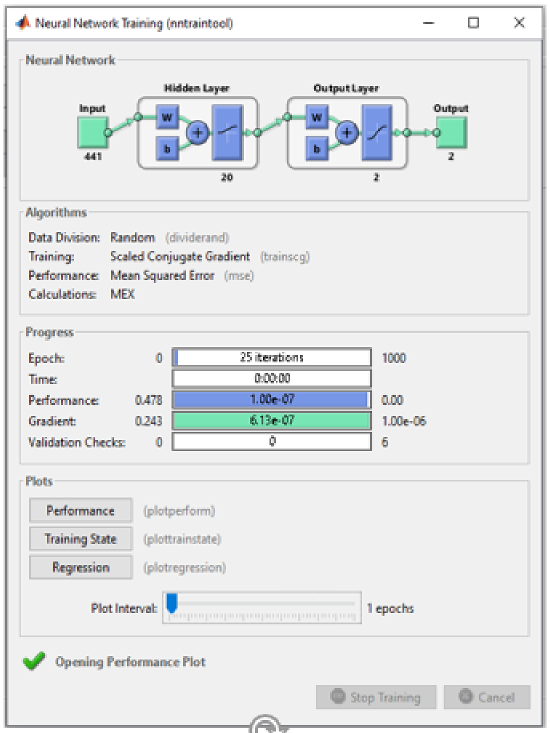
Plot Interval: 1 epochs

✓ Opening Regression Plot





8.14 Logsig


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division: Random (dividerand)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Mean Squared Error (mse)
Calculations: MEX

Progress

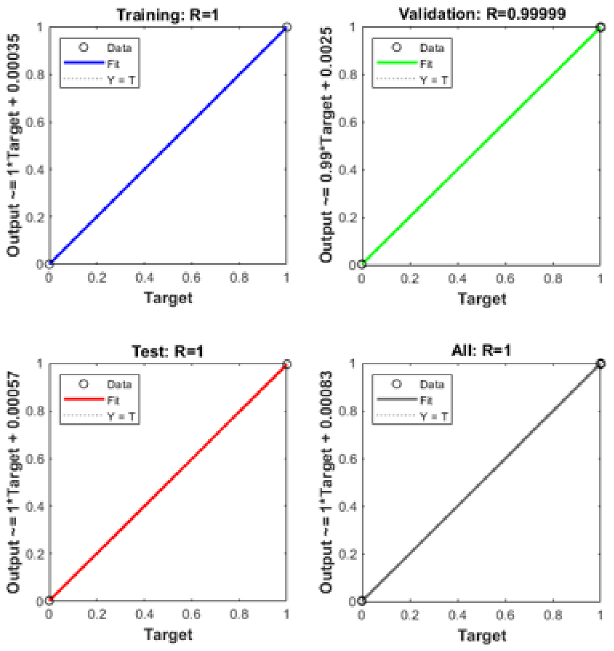
Epoch:	0	25 iterations	1000
Time:		0:00:00	
Performance:	0.478	1.00e-07	0.00
Gradient:	0.243	6.13e-07	1.00e-06
Validation Checks:	0	0	6

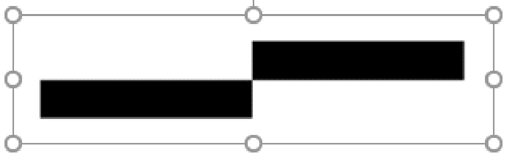
Plots

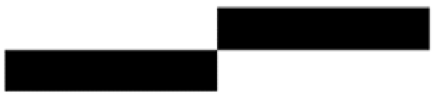
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

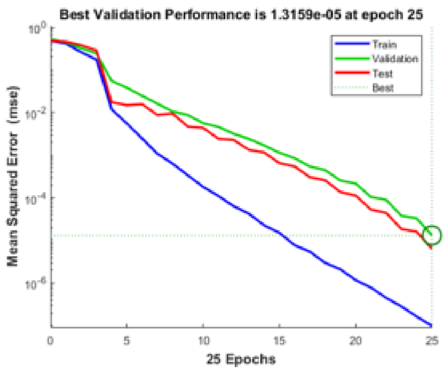
Plot Interval: 1 epochs

Opening Performance Plot

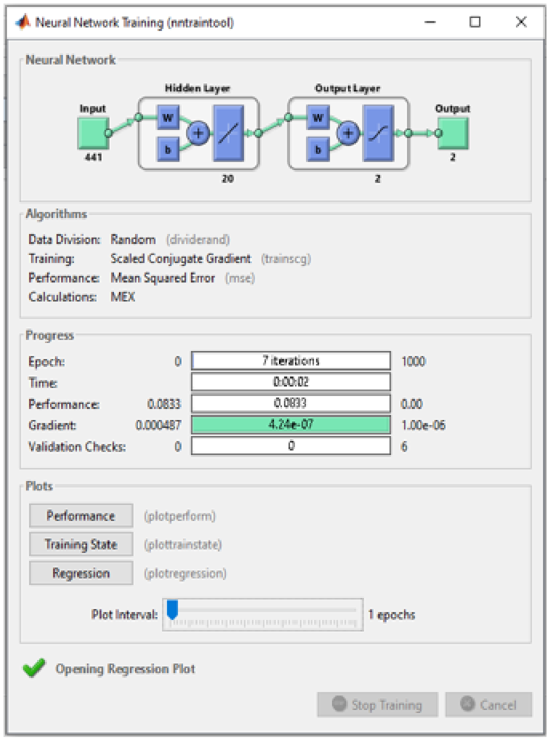








8.15 Purelin


Neural Network Training (nntraintool)

Neural Network

Algorithms

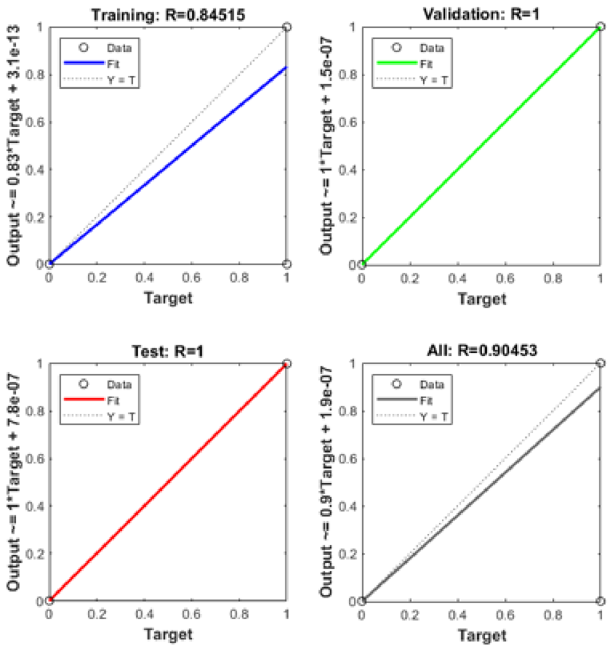
Data Division: Random (dividerand)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Mean Squared Error (mse)
Calculations: MEX

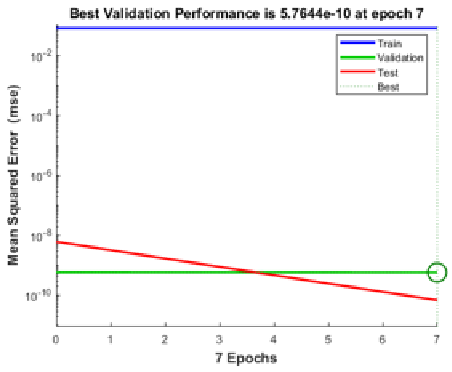
Progress

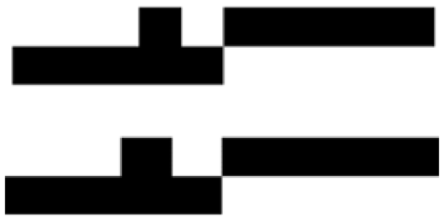
Epoch:	0	7 iterations	1000
Time:		0:00:02	
Performance:	0.0833	0.0833	0.00
Gradient:	0.000487	4.24e-07	1.00e-06
Validation Checks:	0	0	6

Plot Interval: 1 epochs

Opening Regression Plot







8.2 MATLAB Code

```matlab
% Name: Rivyesch Ranjan
% ID: 29392004
% Date Modified: 31/03/2021

clc;clear all;close all;
%% Loading Fingerprint Images
pic1 = imread('leftthumb.png');
pic2 = imread('rightthumb.png');

%% Convert Images to Black and White
bw1 = im2bw(pic1,0.5);
bw2 = im2bw(pic2,0.5);
figure('Name','Fingerprint Images after Thresholding')
subplot(1,2,1),imshow(bw1,'InitialMagnification','fit'),axis square;
subplot(1,2,2),imshow(bw2,'InitialMagnification','fit'),axis square;
% Image size (For resizing purpose)
imrow = 192;
imcol = 112;

%% Create Duplicate of Specified Regions with Randomised Pixels
if exist('FingerprintSamples.mat','file')
    fprintf('The Variables Exist. Loading...\n');
    load('FingerprintSamples.mat');
else fprintf('Variables Do Not Exist. Creating Variables...\n');
    N = 100; % Number of pixels to be inverted

    % Fingerprint 1
    f11_og = bw1; % Original Fingerprint
    f11_resize = imresize(f11_og,[imrow,imcol]);
    f11 = f11_resize((96:116),(56:76));
    f12 = AddNoise(f11,N);
    f13 = AddNoise(f11,N);
    f14 = AddNoise(f11,N);
    f15 = AddNoise(f11,N);
    % Fingerprint 2
    f21_og = bw2; % Original Fingerprint
    f21_resize = imresize(f21_og,[imrow,imcol]);
    f21 = f21_resize((96:116),(56:76));
    f22 = AddNoise(f21,N);
    f23 = AddNoise(f21,N);
    f24 = AddNoise(f21,N);
    f25 = AddNoise(f21,N);


save('FingerprintSamples.mat','f11','f12','f13','f14','f15','f21','f22','f2
3','f24','f25');
    % Save variables for future training usage
end

%% Plot Fingerprint Images
figure('name','Input Images')
subplot(2,5,1),imshow(f11,'InitialMagnification','fit'),axis square;
subplot(2,5,2),imshow(f12,'InitialMagnification','fit'),axis square;
subplot(2,5,3),imshow(f13,'InitialMagnification','fit'),axis square;
subplot(2,5,4),imshow(f14,'InitialMagnification','fit'),axis square;
subplot(2,5,5),imshow(f15,'InitialMagnification','fit'),axis square;
subplot(2,5,6),imshow(f21,'InitialMagnification','fit'),axis square;
subplot(2,5,7),imshow(f22,'InitialMagnification','fit'),axis square;
subplot(2,5,8),imshow(f23,'InitialMagnification','fit'),axis square;
subplot(2,5,9),imshow(f24,'InitialMagnification','fit'),axis square;
subplot(2,5,10),imshow(f25,'InitialMagnification','fit'),axis square;

%% Prepare Input for Training
biometric_in =
[reshape(f11,1,[]);reshape(f12,1,[]);reshape(f13,1,[]);reshape(f14,1,[]);re
shape(f15,1,[]);
reshape(f21,1,[]);reshape(f22,1,[]);reshape(f23,1,[]);reshape(f24,1,[]);res
hape(f25,1,[])];
traininput = biometric_in';
figure('Name','Training Input')
imagesc(traininput)

%% ANN Target
figure('Name','Target Output')
target = zeros(2,10);
target(1,1:5) = 1;
target(2,6:10) = 1;
imshow(target,'InitialMagnification','fit');

%% Train Network
train1 = newff( traininput, target, [25,25], {'tansig','tansig', 'tansig'},
'trainscg' );
train1.trainParam.show = 100;
train1.trainParam.epochs = 1000; % Number of epochs
train1.trainParam.lr = 0.08; % Learning Rate
train1.trainParam.goal = 0; % Goal
[net_out,train_rec,train_out,nn_err] = train(train1,traininput,target);

%% Simulate Output
NN_Out = sim(net_out,traininput);
NN_Out_round = round(SCG_logsig_sim);
figure('name','Neural Network Output')
imshow(SCG_logsig_sim,'InitialMagnification','fit')
figure('name','NN Output(Rounded)')
imshow(NN_Out_round,'InitialMagnification','fit')
```