

MATLAB Project#3 (Intelligent Systems-Summer 2019)

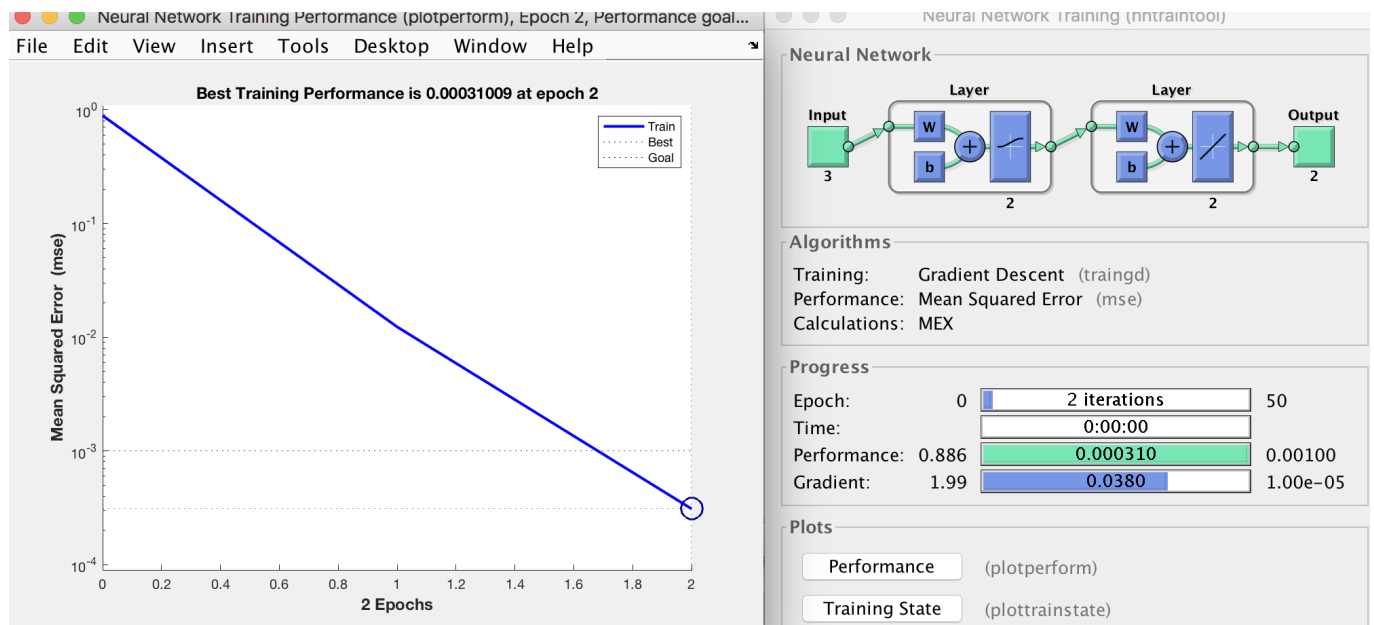
Problem #1 Consider the following network with the inputs and outputs as follow:

(1.1) Use the sample code and run in matlab given the correct inputs and desired output.

```
P=[0.8; 1; 0.9]; % Input Sample
T=[1; 1]; %Desired Output
%Creating the network
net = newff([0 1 ; 0 1 ; 0 1],[2 2],{'logsig' 'purelin'},'traingd');
net.iw{1,1}=[0.3 0.3 0.7;0.9 0.6 0.1]; % Input Weights
net.b{1}=[0; 0]; % Input Threshold Weights = 0
net.lw{2,1}=[0.9 0.4;0.2 0.1]; % Hidden Layer weights
% Printing the Weights
input_layer_weight = net.iw{1,1}
Hidden_layer_weight = net.lw{2,1}
Bias_weights = net.b{1}
Initial_Output=sim(net,P);
% Setting Training Parameters
net.trainParam.lr = 0.5;
net.trainParam.epochs = 50;
net.trainParam.goal = 0.001;
net = train(net,P,T); % Training the Network
% Evaluating the results
Final_Output = sim(net,P)
input_layer_weight = net.iw{1,1}
Hidden_layer_weight = net.lw{2,1}
Bias_weights = net.b{1}
```

Performance :

Using sigmoid activation function in the first layer. The network is trained with a learning rate of 0.5, and max epochs of 50 . The goal is to reach an MSE of 0.001. The system trained achieved that error in only 2 epochs.



Output :

input_layer_weight =

```
0.3000    0.3000    0.7000
0.9000    0.6000    0.1000
```

Hidden_layer_weight =

```
0.9000    0.4000
0.2000    0.1000
```

Bias_weights =

```
0
0
```

Final_Output =

```
0.9960
0.9754
```

input_layer_weight =

```
0.3005    0.3006    0.7006
0.8980    0.5974    0.0977
```

Hidden_layer_weight =

```
0.8374    0.3341
```

```
0.6433    0.5672
```

```
Bias_weights =
```

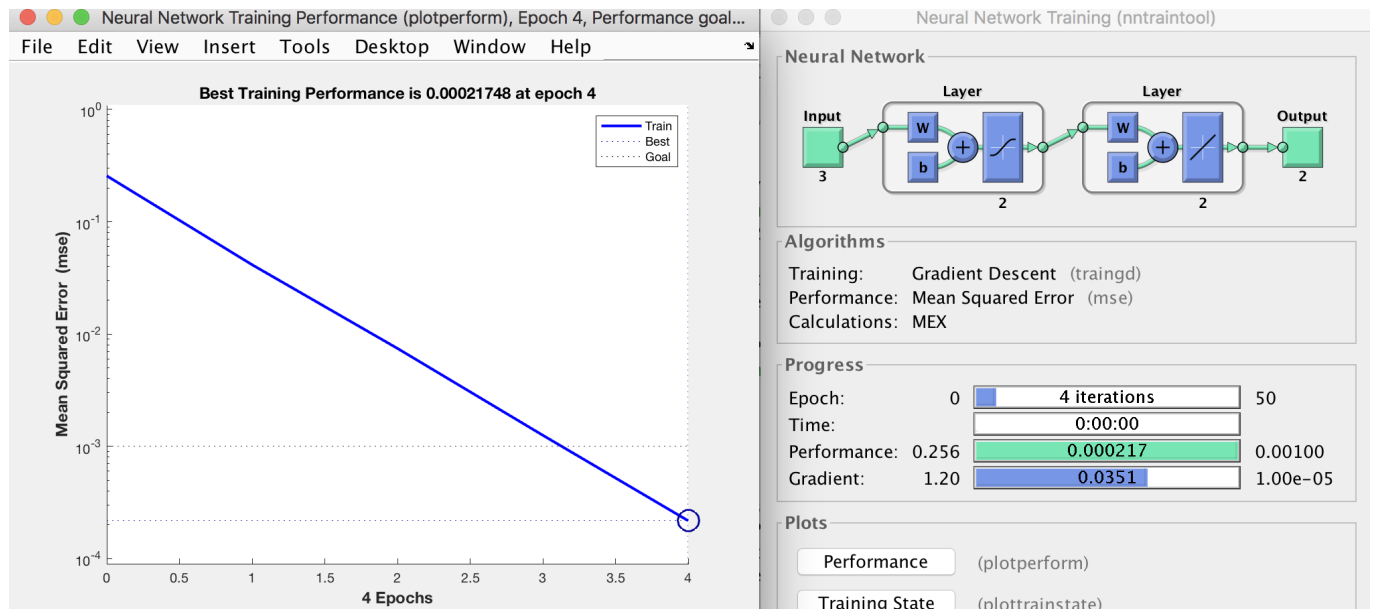
```
0.0006  
-0.0026
```

(1.2) using hyperbolic tangent for first layer and linear activation for second layer.

```
P=[0.8; 1; 0.9]; % Input Sample  
T=[1; 1]; %Desired Output  
%Creating the network  
net = newff([0 1 ; 0 1 ; 0 1],[2 2],{'tansig' 'purelin'},'traingd');  
net.iw{1,1}=[0.3 0.3 0.7;0.9 0.6 0.1]; % Input Weights  
net.b{1}=[0; 0]; % Input Threshold Weights = 0  
net.lw{2,1}=[0.9 0.4;0.2 0.1]; % Hidden Layer weights  
% Printing the Weights  
input_layer_weight = net.iw{1,1}  
Hidden_layer_weight = net.lw{2,1}  
Bias_weights = net.b{1}  
Initial_Output=sim(net,P);  
% Setting Training Parameters  
net.trainParam.lr = 0.5;  
net.trainParam.epochs = 50;  
net.trainParam.goal = 0.001;  
net = train(net,P,T); % Training the Network  
% Evaluating the results  
Final_Output = sim(net,P)  
input_layer_weight = net.iw{1,1}  
Hidden_layer_weight = net.lw{2,1}  
Bias_weights = net.b{1}
```

Performance :

The output shows what the initial weights defined in the problem. The network is trained with a learning rate of 0.5, and max epochs of 50 . The goal is to reach an MSE of 0.001. The system trained achieved that error in only 4 epochs.



Output :

input_layer_weight =

```
0.3000    0.3000    0.7000
0.9000    0.6000    0.1000
```

Hidden_layer_weight =

```
0.9000    0.4000
0.2000    0.1000
```

Bias_weights =

```
0
0
```

Final_Output =

```
0.9854
0.9675
```

input_layer_weight =

```
0.3058    0.3072    0.7065
0.9008    0.6010    0.1009
```

Hidden_layer_weight =

```
0.8966    0.3964
0.3391    0.2507
```

(1.3) Suppose we select different initial conditions. Are the final weights the same?

```
P=[1; 1; 1]; % Input Sample
```

With a different initial condition the neural net will try to train itself to meet the desired output. But with only a single input to start with the system will only learn to provide the output and tweak coefficients for a single input and thus not be able to tweak enough to generalize. The weights will be different to different starting conditions if its all that is used to train with.

Output :

```
input_layer_weight =

    0.3000    0.3000    0.7000
    0.9000    0.6000    0.1000
```

```
Hidden_layer_weight =

    0.9000    0.4000
    0.2000    0.1000
```

```
Bias_weights =

    0
    0
```

```
Final_Output =

    0.9948
    1.0217
```

```
input_layer_weight =

    0.2283    0.2283    0.6283
    0.8716    0.5716    0.0716
```

```
Hidden_layer_weight =

    0.6382    0.1292
    0.6445    0.5674
```

```
Bias_weights =  
  
-0.0717  
-0.0284
```

Problem #2 Suppose for above problem , the inputs and the outputs are as follow:

```
P=[12; 3; 8]; % Input Sample  
T=[9; 1]; %Desired Output
```

How do you solve the problem with appropriate scaling of the inputs and outputs data sets.

By normalizing against the largest value of the input and output then all of the values could then be scaled back to the original.

```
P=[12; 3; 8]; % Input Sample  
T=[9; 1]; %Desired Output  
P_norm =  
  
1.0000  
0.2500  
0.6667  
  
T_norm =  
  
0.7500  
0.0833
```

Problem #3 -Please open the MATLAB application, and do the following steps:

(3.1) Go to NN Fitting application toolbox (nftool):

Neural Fitting (nftool)

Welcome to the Neural Network Fitting app.

Solve an input-output fitting problem with a two-layer feed-forward neural network.

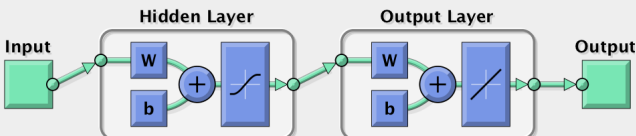
Introduction

In fitting problems, you want a neural network to map between a data set of numeric inputs and a set of numeric targets.

Examples of this type of problem include estimating engine emission levels based on measurements of fuel consumption and speed (`engine_dataset`) or predicting a patient's bodyfat level based on body measurements (`bodyfat_dataset`).

The Neural Fitting app will help you select data, create and train a network, and evaluate its performance using mean square error and regression analysis.

Neural Network



A two-layer feed-forward network with sigmoid hidden neurons and linear output neurons (`fitnet`), can fit multi-dimensional mapping problems arbitrarily well, given consistent data and enough neurons in its hidden layer.

The network will be trained with Levenberg-Marquardt backpropagation algorithm (`trainlm`), unless there is not enough memory, in which case scaled conjugate gradient backpropagation (`trainscg`) will be used.

To continue, click [Next].

Neural Network Start Welcome Back Next Cancel

(3.2) Load Simplefit dataset

Neural Fitting (nftool)

Select Data

What inputs and targets define your fitting problem?

Get Data from Workspace

Input data to present to the network.

Inputs: `simplefitInputs` ...

Target data defining desired network output.

Targets: `simplefitTargets` ...

Samples are: ☒ Matrix columns ☐ Matrix rows

Want to try out this tool with an example data set?

Load Example Data Set

Summary

Inputs 'simplefitInputs' is a 1x94 matrix, representing static data: 94 samples of 1 element.

Targets 'simplefitTargets' is a 1x94 matrix, representing static data: 94 samples of 1 element.

To continue, click [Next].

Neural Network Start Welcome Back Next Cancel

(3.3) Select default value for Training, validation and testing with 10 hidden layers

Validation and Test Data

Set aside some samples for validation and testing.

Select Percentages

Randomly divide up the 94 samples:

Training:	70%	66 samples
Validation:	15%	14 samples
Testing:	15%	14 samples

Restore Defaults

Explanation

Three Kinds of Samples:

- Training:** These are presented to the network during training, and the network is adjusted according to its error.
- Validation:** These are used to measure network generalization, and to halt training when generalization stops improving.
- Testing:** These have no effect on training and so provide an independent measure of network performance during and after training.

Change percentages if desired, then click [Next] to continue.

Neural Network Start Welcome Back Next Cancel

Network Architecture

Set the number of neurons in the fitting network's hidden layer.

Hidden Layer

Define a fitting neural network. (fitnet)

Number of Hidden Neurons: 10

Restore Defaults

Recommendation

Return to this panel and change the number of neurons if the network does not perform well after training.


Neural Network

Change settings if desired, then click [Next] to continue.

Neural Network Start Welcome Back Next Cancel

(3.4) Train the network with BP algorithm

Neural Fitting (nftool)



Train Network

Train the network to fit the inputs and targets.

Train Network

Choose a training algorithm:







Levenberg-Marquardt

This algorithm typically requires more memory but less time. Training automatically stops when generalization stops improving, as indicated by an increase in the mean square error of the validation samples.

Train using Levenberg-Marquardt. (trainlm)

Retrain

Results


	 Samples	 MSE	 R
 Training:	66	9.58695e-6	9.99999e-1
 Validation:	14	3.01549e-5	9.99998e-1
 Testing:	14	6.43137e-6	9.99999e-1


Plot Fit


Plot Error Histogram

Plot Regression

Notes

 Training multiple times will generate different results due to different initial conditions and sampling.

 Mean Squared Error is the average squared difference between outputs and targets. Lower values are better. Zero means no error.

 Regression R Values measure the correlation between outputs and targets. An R value of 1 means a close relationship, 0 a random relationship.

➡ Open a plot, retrain, or click [Next] to continue.

Neural Network Start

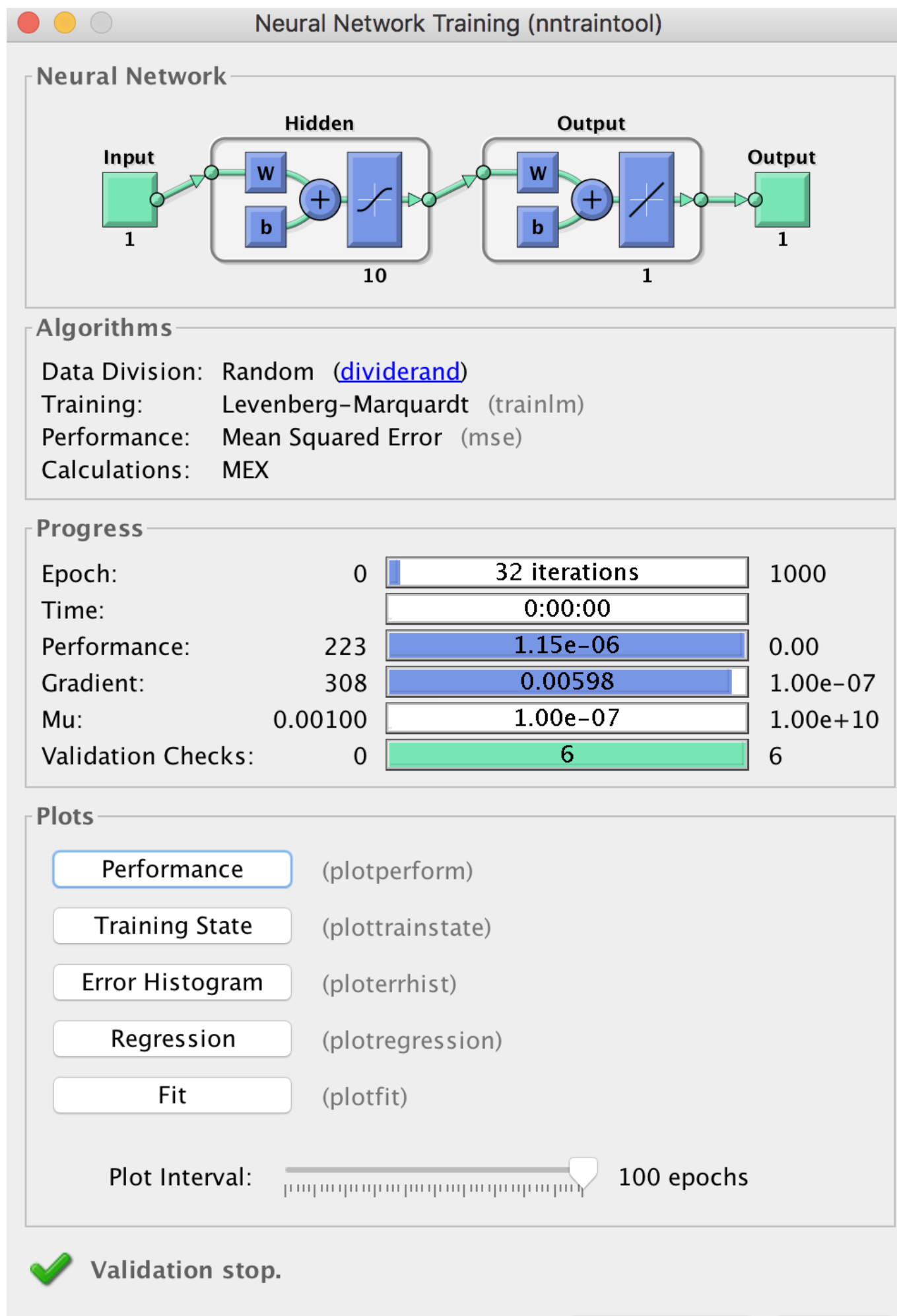
Welcome

➡ Back

➡ Next

✖ Cancel

9 / 13



(3.5) Review the performance of your network. Try at least four different options with the same data. Provide the best NN performance as your solution. Justify your answer.

Train Network
Train the network to fit the inputs and targets.

Train Network

Choose a training algorithm:

Scaled Conjugate Gradient

This algorithm requires less memory. Training automatically stops when generalization stops improving, as indicated by an increase in the mean square error of the validation samples.

Train using Scaled Conjugate Gradient. (trainscg)

Retrain

Results

	Samples	MSE	R
Training:	66	1.13635e-2	9.99383e-1
Validation:	14	1.10541e-2	9.99249e-1
Testing:	14	1.51533e-2	9.98767e-1

Plot Fit Plot Error Histogram

Plot Regression

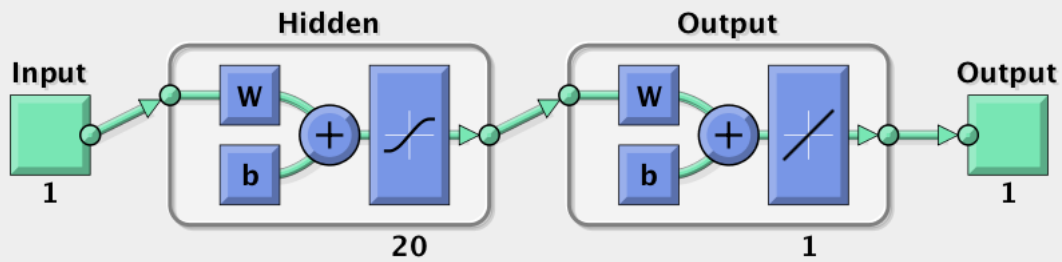
Notes

Training multiple times will generate different results due to different initial conditions and sampling.

Mean Squared Error is the average squared difference between outputs and targets. Lower values are better. Zero means no error.

Regression R Values measure the correlation between outputs and targets. An R value of 1 means a close relationship, 0 a random relationship.

Neural Network



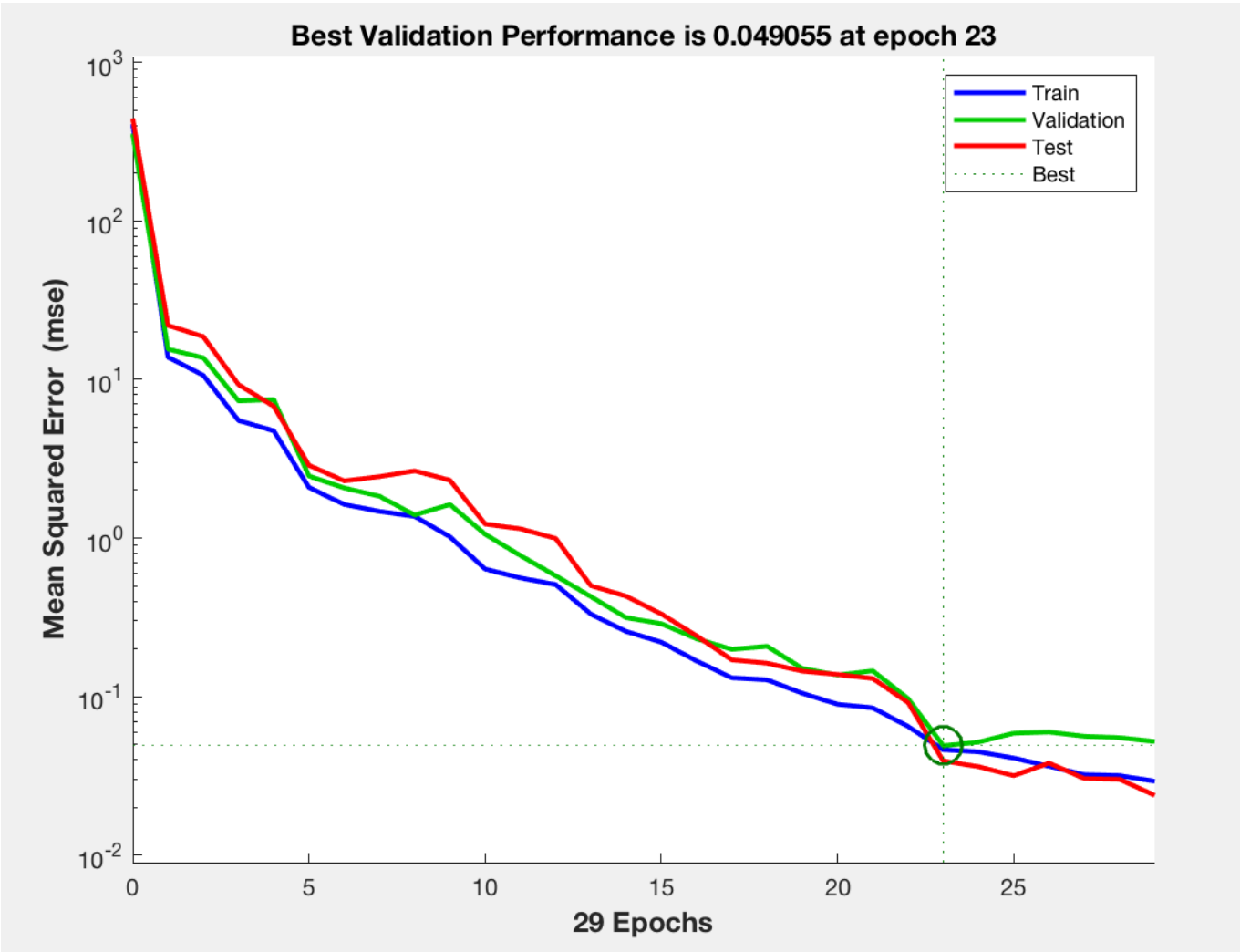
Algorithms

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

Progress

Epoch:	0	29 iterations	1000
Time:		0:00:00	
Performance:	406	0.0293	0.00
Gradient:	537	0.262	1.00e-065
Validation Checks:	0	6	6

Retraining with Scaled Conjugate Gradient yielded a MSE with lesser epochs. Then the hidden layer neurons were increased to 20 from 10 yielding only slightly better performance.



Problem #4 Repeat Problem #3 for building energy data base using the same toolbox.

Multiple training methods were attempted when training. The best was a scaled conjugate gradient to reduce the epoch iterations. This training data set was very different and seemed to converge much better regardless of the training method chosen compared to the simple fit data.

