Body Fat Estimation

This example illustrates how a function fitting neural network can estimate body fat percentage based on anatomical measurements.

View MATLAB Command

The Problem: Estimate Body Fat Percentage

In this example we attempt to build a neural network that can estimate the body fat percentage of a person described by thirteen physical attributes:

- Age (years)
- Weight (lbs)
- Height (inches)
- Neck circumference (cm)
- Chest circumference (cm)
- Abdomen circumference (cm)
- Hip circumference (cm)
- Thigh circumference (cm)
- Knee circumference (cm)
- Ankle circumference (cm)
- Biceps (extended) circumference (cm)
- Forearm circumference (cm)
- Wrist circumference (cm)

This is an example of a fitting problem, where inputs are matched up to associated target outputs, and we would like to create a neural network which not only estimates the known targets given known inputs, but can generalize to accurately estimate outputs for inputs that were not used to design the solution.

Why Neural Networks?

Neural networks are very good at function fit problems. A neural network with enough elements (called neurons) can fit any data with arbitrary accuracy. They are particularly well suited for addressing nonlinear problems. Given the nonlinear nature of real world phenomena, like body fat accretion, neural networks are a good candidate for solving the problem.

The thirteen physical attributes will act as inputs to a neural network, and the body fat percentage will be the target.

The network will be designed by using the anatomical quantities of bodies whose body fat percentage is already known to train it to produce the target valuations.

Preparing the Data

Data for function fitting problems are set up for a neural network by organizing the data into two matrices, the input matrix X and the target matrix T.

Each ith column of the input matrix will have thirteen elements representing a body with known body fat percentage.

Each corresponding column of the target matrix will have one element, representing the body fat percentage.

Here such a dataset is loaded.

```
[X,T] = bodyfat_dataset;
```

We can view the sizes of inputs X and targets T.

Note that both X and T have 252 columns. These represent 252 physiques (inputs) and associated body fat percentages (targets).

The input matrix X has thirteen rows, for the thirteen attributes. The target matrix T has only one row, as for each example we only have one desired output, the body fat percentage.

```
size(X)
size(T)
```

```
ans =

    13   252


ans =

     1   252
```

Fitting a Function with a Neural Network

The next step is to create a neural network that will learn to estimate body fat percentages.

Since the neural network starts with random initial weights, the results of this example will differ slightly every time it is run. The random seed is set to avoid this randomness. However this is not necessary for your own applications.

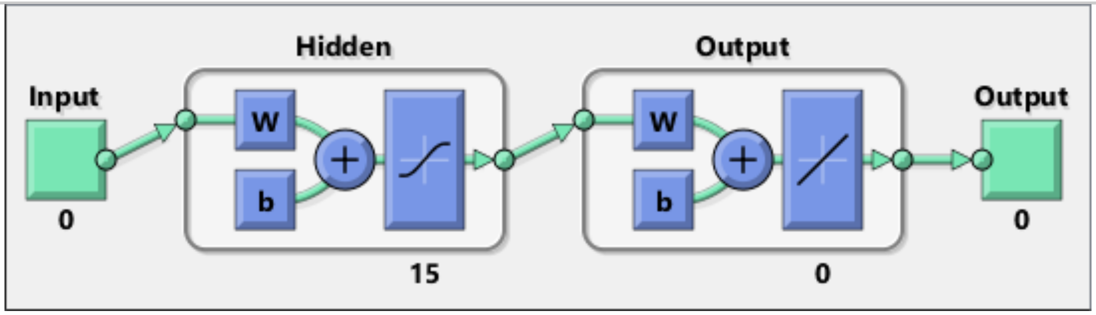
```
setdemorandstream(491218382)
```

Two-layer (i.e. one-hidden-layer) feed forward neural networks can fit any input-output relationship given enough neurons in the hidden layer. Layers which are not output layers are called hidden layers.

We will try a single hidden layer of 15 neurons for this example. In general, more difficult problems require more neurons, and perhaps more layers. Simpler problems require fewer neurons.

The input and output have sizes of 0 because the network has not yet been configured to match our input and target data. This will happen when the network is trained.

```
net = fitnet(15);
view(net)
```

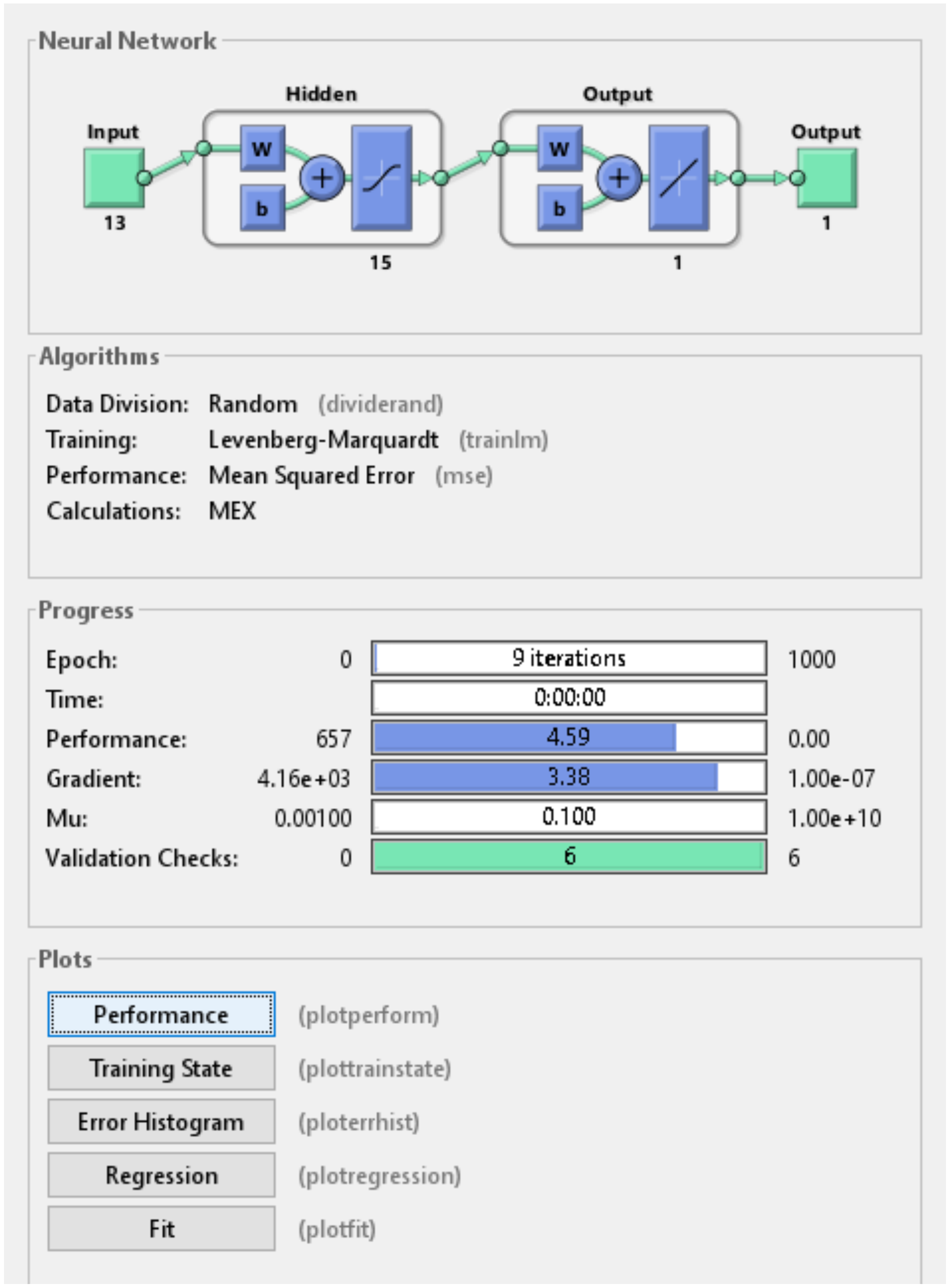


Now the network is ready to be trained. The samples are automatically divided into training, validation and test sets. The training set is used to teach the network. Training continues as long as the network continues improving on the validation set. The test set provides a completely independent measure of network accuracy.

The Neural Network Training Tool shows the network being trained and the algorithms used to train it. It also displays the training state during training and the criteria which stopped training will be highlighted in green.

The buttons at the bottom open useful plots which can be opened during and after training. Links next to the algorithm names and plot buttons open documentation on those subjects.

```
[net,tr] = train(net,X,T);
nntraintool
```


Neural Network
Hidden
Output
Input
W
b
13
15
W
b
1
Output
1

Algorithms

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

Progress

Epoch:	0	9 iterations	1000
Time:		0:00:00	
Performance:	657	4.59	0.00
Gradient:	4.16e+03	3.38	1.00e-07
Mu:	0.00100	0.100	1.00e+10
Validation Checks:	0	6	6

Plots

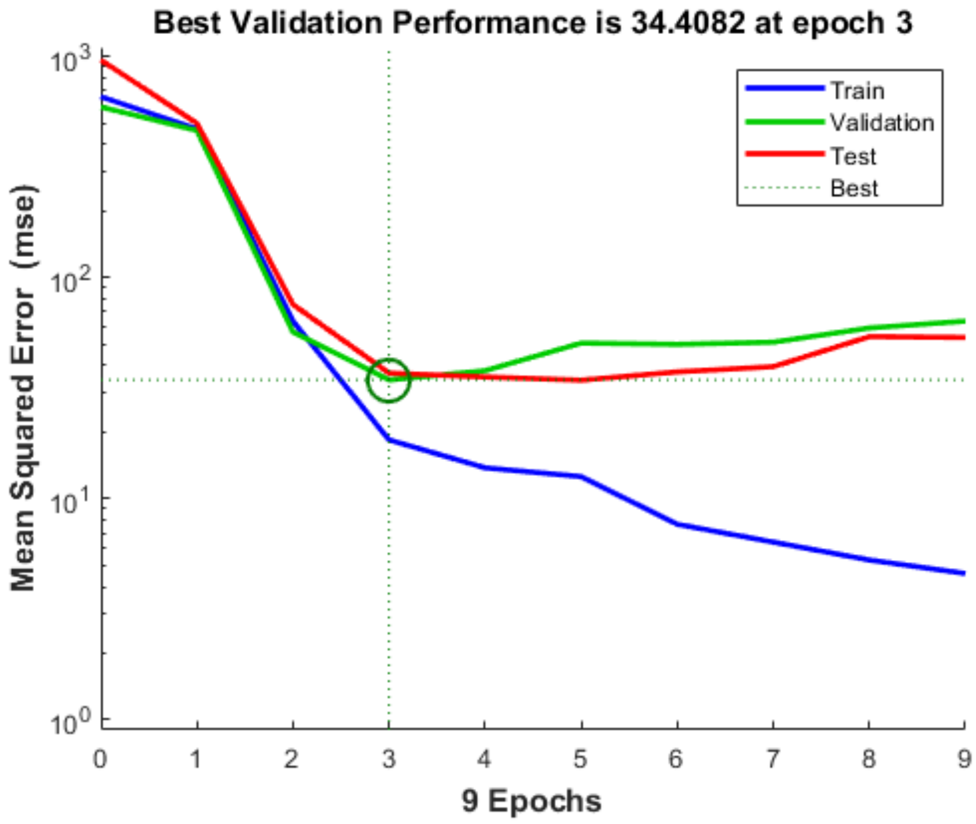
Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Regression (plotregression)
Fit (plotfit)

To see how the network's performance improved during training, either click the "Performance" button in the training tool, or call PLOTPERFORM.

Performance is measured in terms of mean squared error, and shown in log scale. It rapidly decreased as the network was trained.

Performance is shown for each of the training, validation, and test sets. The final network is the network that performed best on the validation set.

```
plotperform(tr)
```



Testing the Neural Network

The mean squared error of the trained neural network can now be measured with respect to the testing samples. This will give us a sense of how well the network will do when applied to data from the real world.

```
testX = X(:,tr.testInd);
testT = T(:,tr.testInd);

testY = net(testX);

perf = mse(net,testT,testY)
```

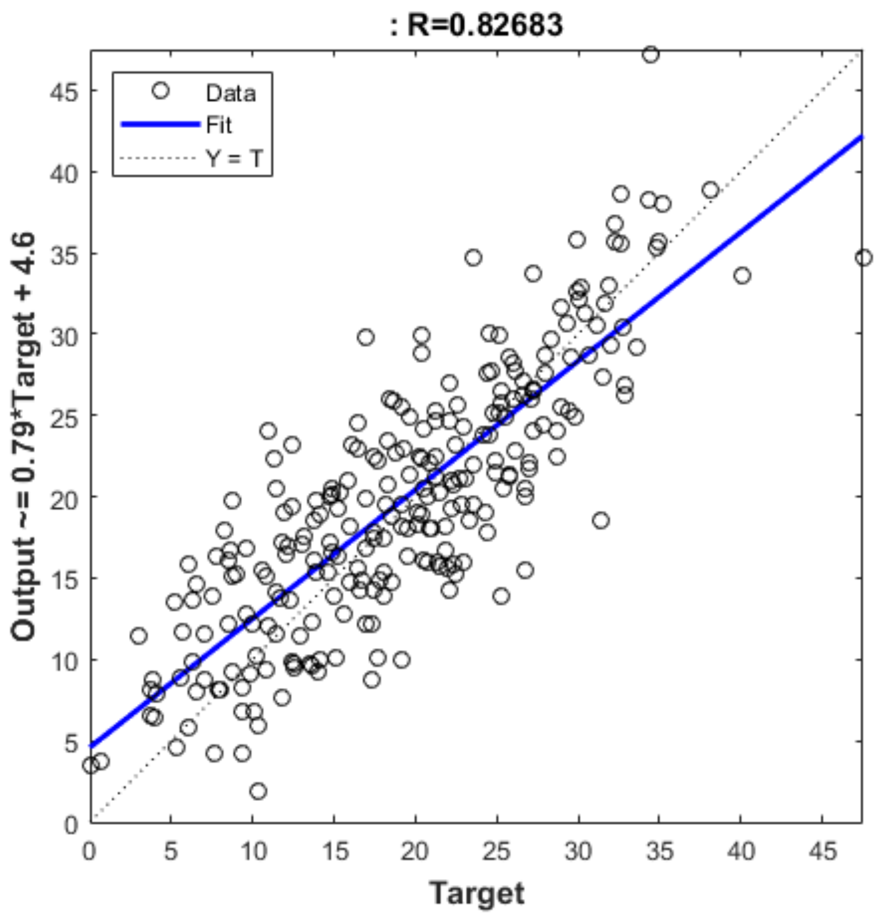
```
perf =

   36.9404
```

Another measure of how well the neural network has fit the data is the regression plot. Here the regression is plotted across all samples.

The regression plot shows the actual network outputs plotted in terms of the associated target values. If the network has learned to fit the data well, the linear fit to this output-target relationship should closely intersect the bottom-left and top-right corners of the plot.

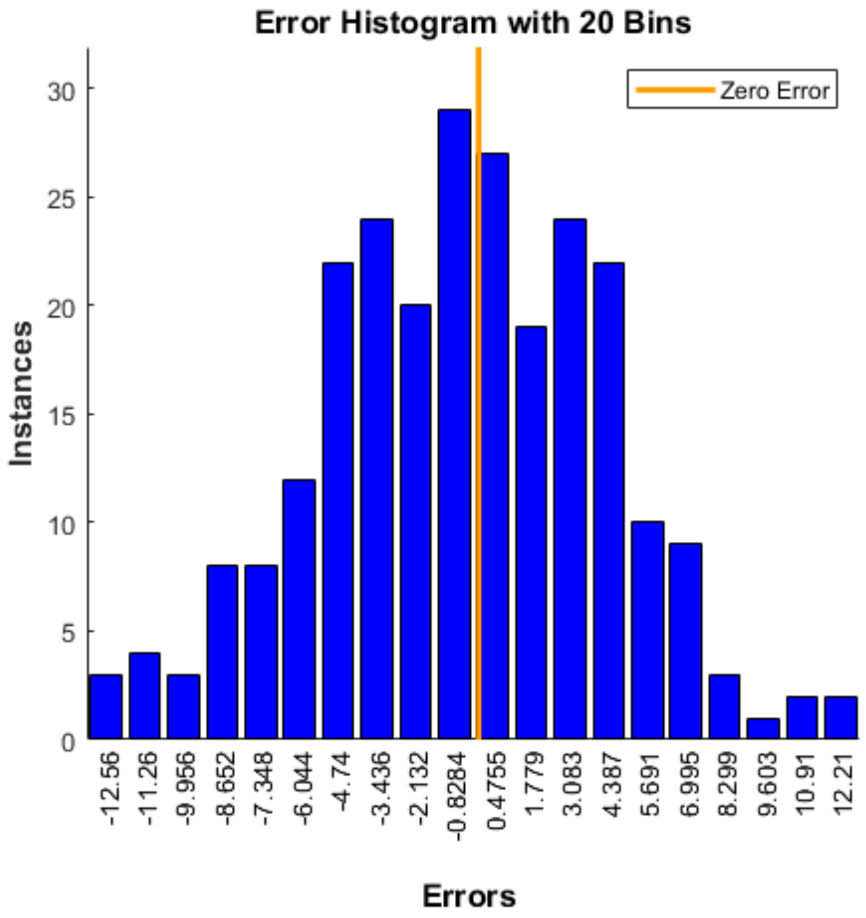
If this is not the case then further training, or training a network with more hidden neurons, would be advisable.

```
Y = net(X);

plotregression(T,Y)
```



Another third measure of how well the neural network has fit data is the error histogram. This shows how the error sizes are distributed. Typically most errors are near zero, with very few errors far from that.

```
e = T - Y;

ploterrhist(e)
```



This example illustrated how to design a neural network that estimates the body fat percentage from physical characteristics.

Explore other examples and the documentation for more insight into neural networks and their applications.