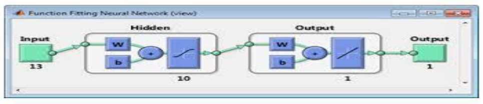


Neural Networks and Fuzzy Logic (630514)

Lecture 7

Perceptrons and Multi-Layer Feedforward Neural Networks using matlab
Part 3

Matlab examples:

1) House Price Estimation using feedforward neural networks (fitting data)

- Build a neural network that can estimate the median price of a home described by ***thirteen attributes***:
  1. *crime rate per town*
  2. *Proportion of residential land zoned for lots over 25,000 sq. ft.*
  3. *proportion of non-retail business acres per town*
  4. *1 if tract bounds Charles river, 0 otherwise*
  5. *Nitric oxides concentration*
  6. *Average number of rooms*
  7. *Proportion of owner-occupied units built prior to 1940*
  8. *Weighted distances to five Boston employment centres*
  9. *Index of accessibility to radial highways*
  10. *-tax rate per $10,000*
  11. *Pupil-teacher ratio by town*
  12. *1000(Bk - 0.63)^2*
  13. *Percent lower status of the population*

- Create a neural network which not only estimates the known targets given known inputs, but can *generalize to estimate outputs for inputs that were not used to design the solution:* **It is a fitting problem.**
  - **Why Neural Networks?**
  - ***Neural networks are very good solution for solving the function fit problems with non-linear nature.***
  - **The thirteen attributes are the inputs to a neural network, and the median home price is the target.**

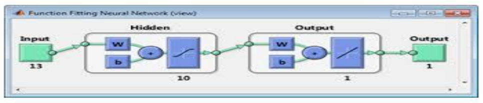
♣ **Preparing the Data**

♠ Load the dataset.

```
[P, T] = house_dataset;
```

♠ To view the sizes of inputs **P** and targets **T**:

```
size (P);
size (T);
ans = 13   506
ans = 1   506
```



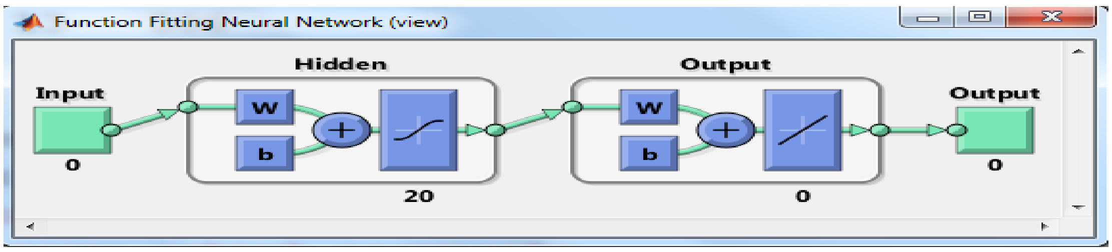
♣ Fitting a Function with a Neural Network

- ♠ Weights initialization, the **random seed** is set to avoid this randomness.

```
setdemorandstream (491218382)
```

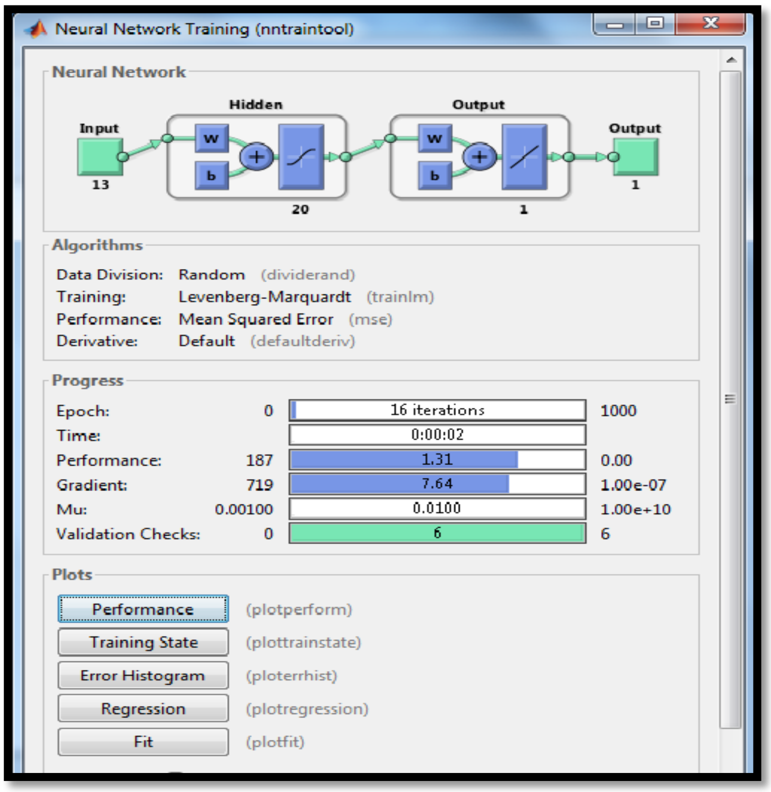
- ♠ Two-layer feed forward neural networks can fit any input-output relationship given enough neurons in the hidden layer.
- ♠ The input and output have sizes of 0 because the network has ***not yet been configured*** to match our input and target data. This will happen when the network is trained.

```
net = fitnet (20);
view (net)
```



- ♠ 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.

```
[net, tr] = train (net, P, T); % tr is the training record information
nntraintool % to display the training tool window
```


Neural Network Training (nntraintool)

Neural Network

Algorithms

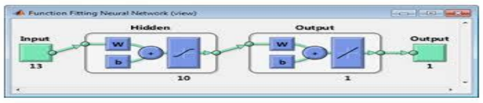
```
Data Division:  Random  (dividerand)
Training:       Levenberg-Marquardt  (trainlm)
Performance:    Mean Squared Error  (mse)
Derivative:     Default  (defaultderiv)
```

Progress

```
Epoch:                   0    16 iterations    1000
Time:                         0:00:02
Performance:           187    1.31             0.00
Gradient:              719    7.64             1.00e-07
Mu:                0.00100    0.0100           1.00e+10
Validation Checks:       0    6                6
```

Plots

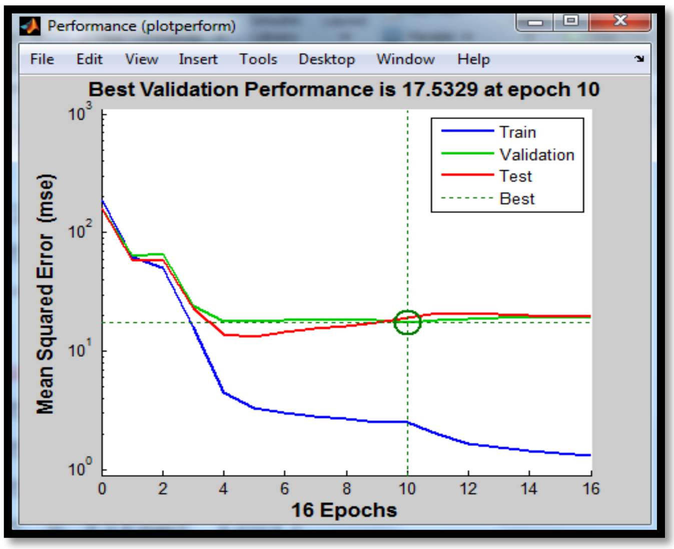
```
Performance      (plotperform)
Training State   (plottrainstate)
Error Histogram  (ploterrhist)
Regression       (plotregression)
Fit              (plotfit)
```



- To display the **network's performance** during training, we can use either:
  - "**Performance**" button in the training tool.
  - Call **plotperform** function:

```
plotperform (tr)
```

- Performance is shown for each of the training, validation and test sets.



Testing the Neural Network using the following measures:

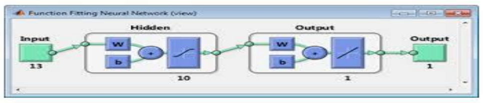
I. **The mean squared error** of the trained neural network is used to how well the network will do when applied to data from the real world.

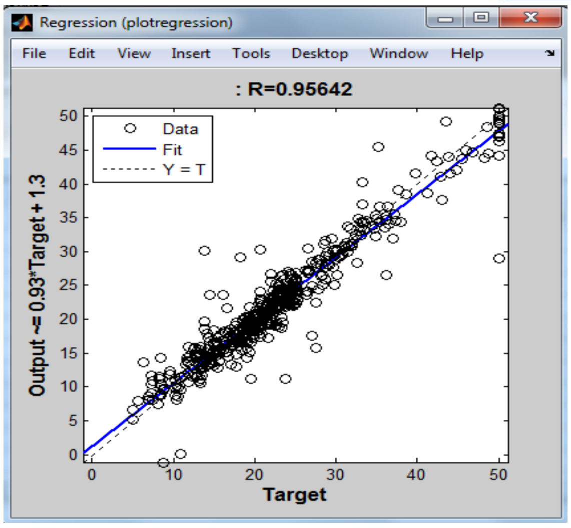
```
testP = P (:, tr.testInd);
testT = T (:, tr.testInd);
testA = net ( testP);
perf = mse (net, testT, testA);
perf =
   19.3315
```

II. **Regression plot**.

- 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 ***(correlation between actual network outputs and the target values).***
- To display the **regression plot, we can use** either**:**
  - **"Regression"** button in the training tool.
  - Call **plotregression** function:

```
A = net (P);
plotregression (T, A);
```



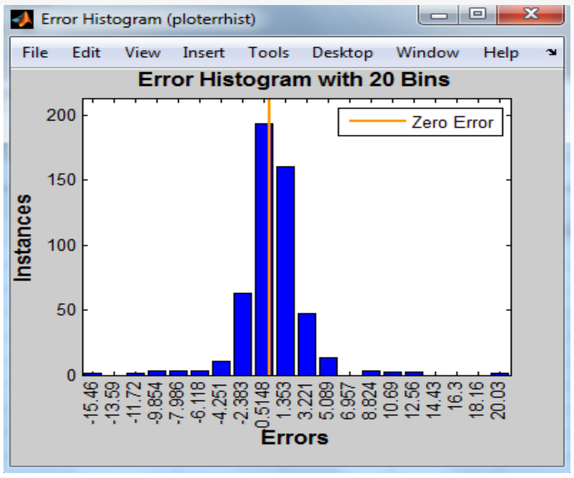


III. **Error histogram** is used to show how the error sizes are distributed. Typically most errors are near zero, with very few errors far from that.

♠ To display the **Error histogram, we can use** either**:**

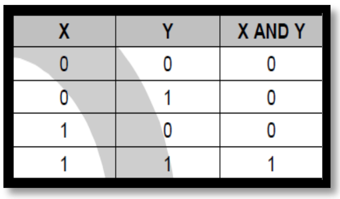
- ➢ **"Error histogram"** button in the training tool.
- ➢ Call **ploterrhist** function:

```
E = T - A;
ploterrhist (E);
```



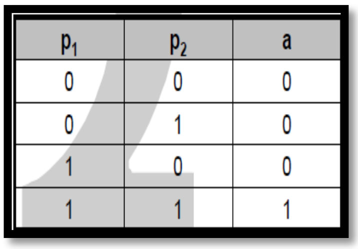
2) Modeling Logical AND Function (linear separable classification)

- Truth table:

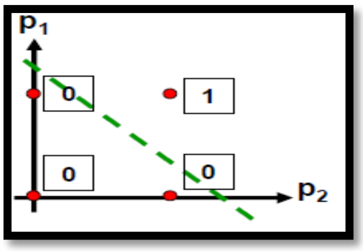


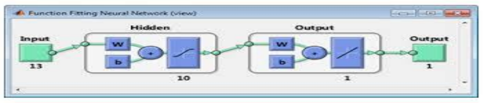
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

- The problem is **linearly-separable**, try to build a one neuron perceptron network with following inputs and output:



p_1	p_2	a
0	0	0
0	1	0
1	0	0
1	1	1



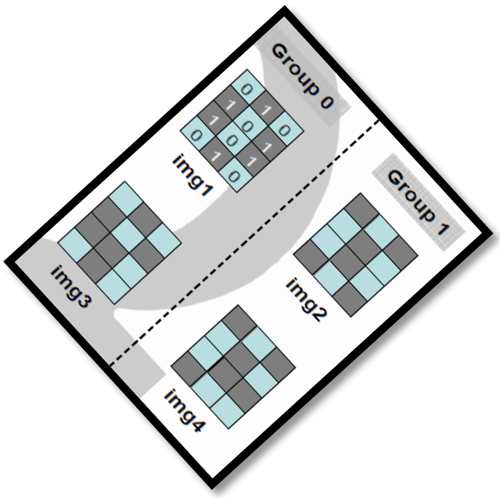


Solution:

P = [0 0 1 1; 0 1 0 1]; % training inputs, p = [p1; p2]
T = [0 0 0 1]; % targets
net = newp ([0 1; 0 1], 1);
net = train (net, P, T);
a = sim (net, P);
a =
0 0 0 1
T =
0 0 0 1

3) Pattern Classification

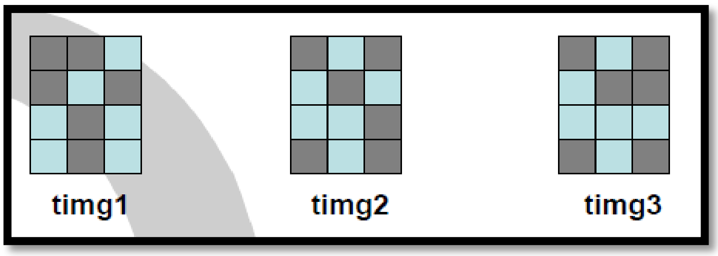
- Build a perceptron that can differentiate between two groups of images:



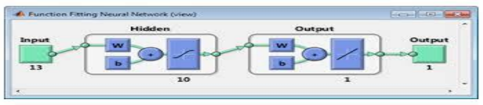
- Use Boolean values 1's and 0's to represent the image.

Solution:

image1 = [0 1 0 1 0 1 1 0 1 0 1 0]';%group 0
image2 = [1 0 1 0 1 0 0 1 0 1 0 1]';%group 1
image3 = [0 1 0 1 0 1 1 1 0 0 1 0]';%group 0
image4 = [1 0 1 0 1 0 0 1 1 1 0 0]';%group 1
%Try testing the trained perceptron on following images:



timg1 = [1 1 0 1 0 1 0 1 0 0 1 0]';
timg3 = [1 0 1 0 1 0 0 0 1 1 0 1]';
timg2 = [1 0 1 0 1 1 0 0 0 1 0 1]';
P = [image1 image2 image3 image4];
T = [0 1 0 1];
net = newp (minmax(P), 1);



```
net = train (net, P, T);
a = sim (net, P)
% Load the test images and ask the perceptron to classify it
PTest = [timg1 timg3 timg2];
testALL = sim (net, PTest) ;
test1 = sim (net, timg1) % to do similarly for other test images
```