

[Emotions Data](#h.ujszbsqfifpt)

[Preparing data](#h.16i9abukwdp1)

[10 Fold Cross Validation](#h.wjdxlk9z1ohb)

[Comparing to Human Performance](#h.ez52nsex22dc)

[Reasoning and Challenges](#h.ufq5fm7r1vnp)

[Challenges](#h.bh1iao3k8m4b)

[References](#h.f6l2sbd574cy)

[Appendix A: Tools, Sources and Scripts](#h.vzo8vv5ryyef)

[Materials](#h.q2bzackmle3b)

[Matlab R2014a](#h.mo7m45nuqvja)

[Artificial Neural Networks GUI tool (nntool)](#h.mo7m45nuqvja)

[Data from provided emotions\_data.mat file](#h.lrlonrs6fer3)

[Data preparation script(convertNum.m)](#h.vyz49qeyz8og)

[Output conversion script(convert1D.m)](#h.asawko4u0de6)

[Appendix B : nntool procedures](#h.47k9b1qmod9i)

[Creating and training the Network using Matlab nntool](#h.gm19iinhlkj3)

[Appendix C - Human Test Script](#h.lj2ld7d9x76g)

# Emotions Data

Data provided in emotions\_data.mat file contains

1. A matrix x of dimensions 612×136 , (There are 612 pieces of emotion data and 136 is the dimensionality of the feature vector computed by concatenating the x and y coordinates of 68 facial points).
2. A vector y of dimensions 612×1, containing the emotion labels of the corresponding examples. These labels are numbered from 1 to 6 and correspond to an emotion follows:

1=anger, 2=disgust, 3=fear, 4=happiness, 5=sadness and 6=surprise

# Preparing data

Before we can use the y label vector, we need to convert it to a matrix of rows of ‘1’s and ‘0’s corresponding to the value of the original vector cell. This was done by developing a matlab function, convertNum(inputMatrix)(see appendix A)

Feeding y into this function: ymat = convertNum(y); the output and x are transposed so that the number of rows in ymat corresponds to the number of columns in the new matrix.

# 10 Fold Cross Validation

We make use of N-fold splitting, and as suggested by the lab manual, divide data into 10-folds in order to avoid overfitting.

Using crossvalind(), data is split into 10 randomly generated indices [1]. In each iteration, one of the folds is used for testing and the other 9 are used for training.

The maximum value of the output of each sample is the predicted label and it can then be used to calculate the confusion matrix and the true/false positives and negatives using a library found on MATLAB Exchange confusionmatStats.m [2].

# Comparing to Human Performance

After initial tests using the ANN, and getting validation accuracies of approximately 50%, it was decided to compare the performance of the algorithm to what a human is capable of. A program was written that displays the data points in a scatter graph and asks the user to guess which emotion the data represented. Even before completing the first test, it was apparent that accurate emotion recognition from a set of feature points was not a simple task. The average accuracy obtained from human input was 45%. See appendix C for the human test MATLAB code.

# Reasoning and Challenges

We decided to use 34 nodes to strike a balance between input nodes of 136 and output nodes of 6.

The difference between splitting 34 nodes in separate layers are as below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of hidden layers** | **nodes per layer** | **execution time** | **accuracy mean** | **accuracy over all folds** | **Report files** |
| 4 | 8 to 9 | 66.495s | 58.9325% | 53.5948% | ANNresults(4 hidden layers).mat  Time Profile Summary 4 hidden layers.pdf |
| 2 | 17 | 140.746 s | 62.037% | 54.6841% | ANNresults(2 hidden layers).mat  Time Profile Summary 2 hidden layers.pdf |
| 1 | 34 | 352.651 s | 63.4532% | 57.8976% | ANNresults(1 hidden layer).mat  Time Profile Summary 1 hidden layers.pdf |

More layers give better execution time, while lumping all the nodes in a single layer gives better accuracy.

Levenberg-Marquardt backpropagation, trainlm, is described as one of the fastest back propagation training functions in the matlab library [3]. Trainlm function provided us with the best accuracy rate, even though trainrp had the fastest execution time.

Gradient descent with momentum weight/bias learning function, learngdm, is an incremental learning algorithm. It can avoid getting stuck in a local minima by setting its momentum and learning rate to appropriate variables [4].

Learning rate adjusts the weights in the network by the factor that it is set to. A small learning rate can slow down the process, while a larger one can have an adverse effect on learning by training the network to a local minimum solution. Learning rate is more appropriate for sequential processing [5]. Since we used trainlm function, we did not need to adjust the learning rate.

Time parameter needs to be set to a reasonable number in seconds. This value limits the training time so that in case an operation runs too long(or does not stop), there is a timeout to it. This cannot be set too low either as we need to ensure successful operations are allowed time to run. We initially chose 60, but later increased it to 100.

Each epoch is a single iteration of the inputs being fed into the system and the calculation of the weights. Increasing the number of epochs for a data can help us obtain more appropriate weights, but it can also cause overfitting. Small epoch size can cause the misadjustment of the weights. Our epoch is set to 100 because it provided the best accuracy rate.

Also the size of the data affects the output performance. The more training data we have, the better performance can be expected. This has been clearly noted when doing the training using the 448 compared to the 612 samples.

Overfitting was avoided by using the previously mentioned k fold algorithm. Randomly shuffling the data before each test ensures that the dividing lines are a general representation of the boundaries between the distinct types of data. Data shuffling also avoided any artifacts that may have been caused by the fact that the initial data was organised into distinct groups by emotion.

Below is a table showing the f-measure analysis of our folds for each label.

**4 layers, 8 to 9 nodes per layer:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| **Fold 1** | 0.5714 | 0.2667 | 0.5455 | 0.7333 | 0.6667 | 0.6 |
| **Fold 2** | 0.5714 | 0.7059 | 0.6364 | 0.88 | 0.4444 | 0.9286 |
| **Fold 3** | 0.2222 | 0.6316 | 0.6364 | 0.8 | 0.6667 | 0.6667 |
| **Fold 4** | 0.8 | 0.7143 | 0.7 | 0.8276 | 0.8 | 0.8276 |
| **Fold 5** | 0 | 0.5556 | 0.5217 | 0.7097 | 0.4706 | 0.6667 |
| **Fold 6** | 0.4444 | 0.6667 | 0.6667 | 0.8276 | 0.5 | 0.7097 |
| **Fold 7** | 0.5455 | 0.6316 | 0.5556 | 0.9032 | 0.4444 | 0.72 |
| **Fold 8** | 0 | 0.5714 | 0.4211 | 0.88 | 0.6 | 0.6857 |
| **Fold 9** | 0.4706 | 0 | 0.7059 | 0.8667 | 0.6154 | 0.6111 |
| **Fold 10** | 0.3636 | 0.4444 | 0.4167 | 0.8462 | 0.5882 | 0.7143 |
| **Average** | 0.39891 | 0.51882 | 0.5806 | 0.82743 | 0.57964 | 0.71304 |
| **overall** | 0.563 | 0.6623 | 0.7032 | 0.9286 | 0.807 | 0.8955 |

**1 Layer, 34 nodes:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| **Fold 1** | 0.3333 | 0.6316 | 0.7273 | 0.9286 | 0.7059 | 0.75 |
| **Fold 2** | 0.6667 | 0.75 | 0.6667 | 0.8667 | 0.625 | 0.6923 |
| **Fold 3** | 0.6667 | 0.1538 | 0.8421 | 0.7742 | 0.6957 | 0.7826 |
| **Fold 4** | 0.5333 | 0.56 | 0.4444 | 0.8333 | 0.8235 | 0.7407 |
| **Fold 5** | 0.4 | 0.5882 | 0.6667 | 0.8148 | 0.8 | 0.8571 |
| **Fold 6** | 0.8 | 0.5714 | 0.7826 | 0.9286 | 0.7692 | 0.8148 |
| **Fold 7** | 0.5882 | 0.6154 | 0.625 | 0.8571 | 0.5333 | 0.8485 |
| **Fold 8** | 0.3636 | 0.6 | 0.4286 | 0.6875 | 0.7368 | 0.6923 |
| **Fold 9** | 0.5714 | 0.7778 | 0.7368 | 0.8148 | 0.5556 | 0.9231 |
| **Fold 10** | 0.7059 | 0.6154 | 0.7273 | 0.9375 | 0.7143 | 0.6923 |
| **Average** | 0.56291 | 0.58636 | 0.66475 | 0.84431 | 0.69593 | 0.77937 |
| **overall** | 0.8101 | 0.8875 | 0.8202 | 0.947 | 0.8834 | 0.8865 |

**Optimal Parameter Output**

The best results we got for one hidden layer with 34 neurons and other parameters unchanged as default.

(attached output messages file : ‘output screen messages for 1 hidden layer 34 nodes.txt’)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 |
| Fold 1 | Recall | **28.57%** | **66.67%** | **88.89%** | **92.86%** | **66.67%** | **69.23%** |
|  | Precession | **40.00%** | **60.00%** | **61.54%** | **92.86%** | **75.00%** | **81.82%** |
| Fold 2 | Recall | **87.50%** | **66.67%** | **55.56%** | **92.86%** | **62.50%** | **64.29%** |
|  | Precession | **53.85%** | **85.71%** | **83.33%** | **81.25%** | **62.50%** | **75.00%** |
| Fold 3 | Recall | **62.50%** | **12.50%** | **88.89%** | **85.71%** | **88.89%** | **64.29%** |
|  | Precession | **71.43%** | **20.00%** | **80.00%** | **70.59%** | **57.14%** | **100.00%** |
| Fold 4 | Recall | **50.00%** | **77.78%** | **44.44%** | **71.43%** | **77.78%** | **71.43%** |
|  | Precession | **57.14%** | **43.75%** | **44.44%** | **100.00%** | **87.50%** | **76.92%** |
| Fold 4 | Recall | **28.57%** | **62.50%** | **77.78%** | **78.57%** | **75.00%** | **92.31%** |
|  | Precession | **66.67%** | **55.56%** | **58.33%** | **84.62%** | **85.71%** | **80.00%** |
| Fold 6 | Recall | **85.71%** | **50.00%** | **100.00%** | **86.67%** | **62.50%** | **84.62%** |
|  | Precession | **75.00%** | **66.67%** | **64.29%** | **100.00%** | **100.00%** | **78.57%** |
| Fold 7 | Recall | **71.43%** | **50.00%** | **55.56%** | **80.00%** | **50.00%** | **100.00%** |
|  | Precession | **50.00%** | **80.00%** | **71.43%** | **92.31%** | **57.14%** | **73.68%** |
| Fold 8 | Recall | **28.57%** | **75.00%** | **33.33%** | **78.57%** | **77.78%** | **64.29%** |
|  | Precession | **50.00%** | **50.00%** | **60.00%** | **61.11%** | **70.00%** | **75.00%** |
| Fold 9 | Recall | **50.00%** | **87.50%** | **77.78%** | **78.57%** | **62.50%** | **85.71%** |
|  | Precession | **66.67%** | **70.00%** | **70.00%** | **84.62%** | **50.00%** | **100.00%** |
| Fold 10 | Recall | **85.71%** | **44.44%** | **88.89%** | **100.00%** | **62.50%** | **64.29%** |
|  | Precession | **60.00%** | **100.00%** | **61.54%** | **88.24%** | **83.33%** | **75.00%** |
| Average | Recall | **55.61%** | **56.65%** | **67.94%** | **77.83%** | **69.01%** | **75.75%** |
|  | Precession | **56.48%** | **70.12%** | **69.00%** | **83.56%** | **69.88%** | **82.77%** |

Aggregation of the confusion matrices of the folds:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 43 | 12 | 5 | 1 | 12 | 1 |
| 12 | 50 | 7 | 6 | 4 | 5 |
| 3 | 4 | 64 | 8 | 2 | 9 |
| 1 | 5 | 7 | 121 | 2 | 7 |
| 10 | 6 | 4 | 3 | 58 | 3 |
| 4 | 7 | 12 | 5 | 5 | 104 |

# Challenges

One of the challenges in this activity is the balance required to strike between having good accuracy, fast training, and avoiding overfitting. When using the GUI tool, the samples were randomly divided by the tool using a default ratio. This gave an accuracy of about 67%. Once the 10-fold cross validation was introduced, accuracy came down to between 52% to 57%. A higher accuracy may not mean a better network, as the objective is to be able to generalize.

# References

1. Matlab. 2015. crossvalind. [ONLINE] Available at: <http://uk.mathworks.com/help/bioinfo/ref/crossvalind.html>. [Accessed 20 February 15].
2. Matlab. 2015. Confusion matrix, Accuracy, Precision, Specificity, Sensitivity, Recall, F-score. [ONLINE] Available at: <http://uk.mathworks.com/matlabcentral/fileexchange/46035-confusion-matrix--accuracy--precision--specificity--sensitivity--recall--f-score>. [Accessed 20 February 15].
3. Matlab. 2015. trainlm. [ONLINE] Available at: <http://uk.mathworks.com/help/nnet/ref/trainlm.html>. [Accessed 20 February 15].
4. Matlab. 2015. learngdm. [ONLINE] Available at: <http://uk.mathworks.com/help/nnet/ref/learngdm.html>. [Accessed 20 February 15].
5. Nikolay Nikolaev. 2015. Practical Aspects of Backpropagation. [ONLINE] Available at:<http://homepages.gold.ac.uk/nikolaev/311practbp.htm>. [Accessed 20 February 15].

# Appendix A: Tools, Sources and Scripts

## Materials

## 

## Matlab R2014a

## Artificial Neural Networks GUI tool (nntool)

## Data from provided emotions\_data.mat file

## 

## Data preparation script(convertNum.m)

this function simply converts the number to a row of 6 binary numbers with the column corresponding to the number set to ‘1’ leaving the other columns at 0. E.g 1=100000,2=010000,3=001000…

function OUT = convertNum (inMat)

[m,n] = size(inMat);

outMat = zeros(m,6);

for j = 1:m

OUTA = [0 0 0 0 0 0];

OUTA(inMat(j)) = 1;

outMat(j,1:6) = OUTA;

end

OUT = outMat;

end

## Output conversion script(convert1D.m)

function OUT = convert1D (inMat)

[m,n] = size(inMat);

outMat = zeros(n,1);

for i=1:n

[val,ind] = max(inMat(:,i));

outMat(i) = ind;

end

OUT = outMat;

end

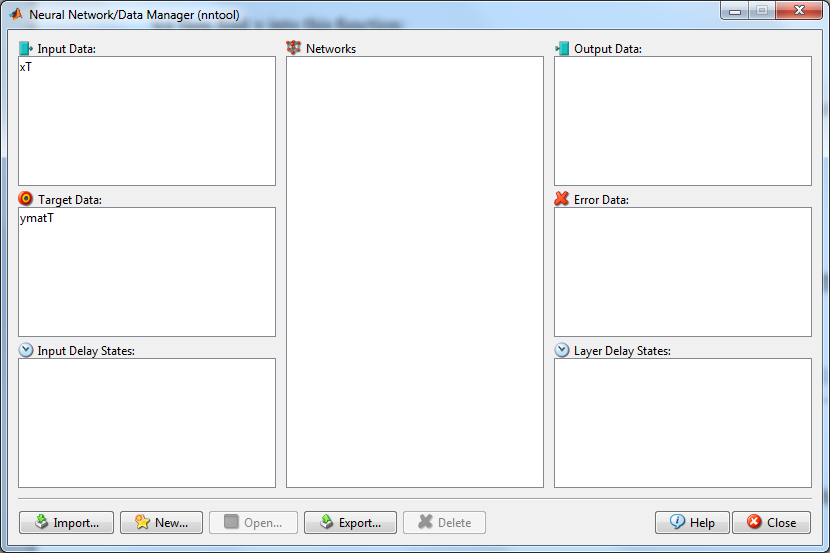
Overall training and simulation script(ANNkfoldver3b.m) attached in zip file.

# Appendix B : nntool procedures

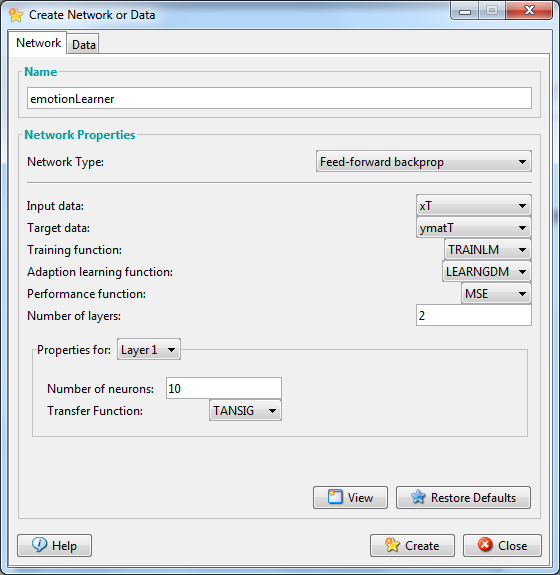
## Creating and training the Network using Matlab nntool

We used emotions\_data.mat comprised of 448 samples for testing nntool functions.

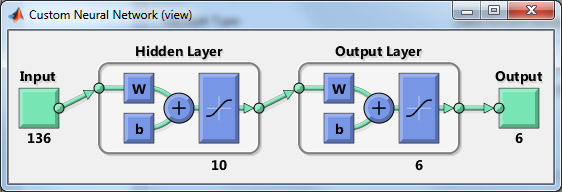
Using nntool, xT was fed as input data while ymatT was fed as the target data as seen in figure below:

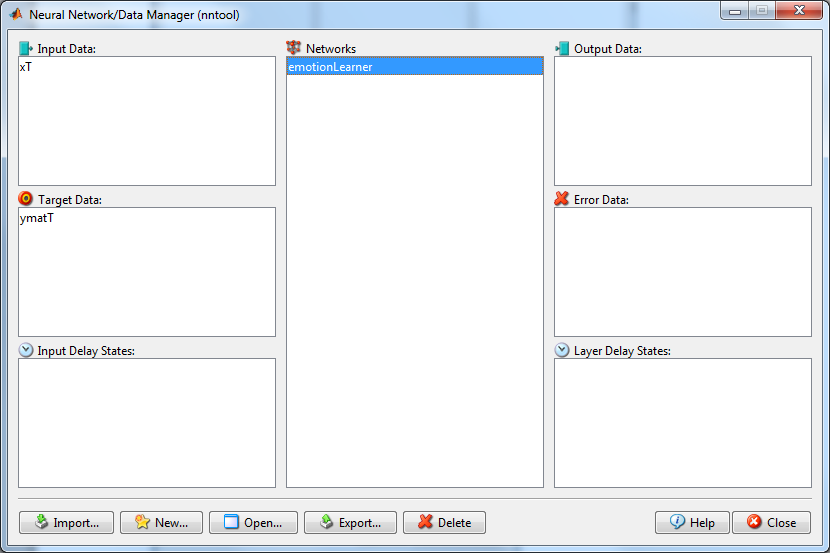


A new neural network learner was created using this data:

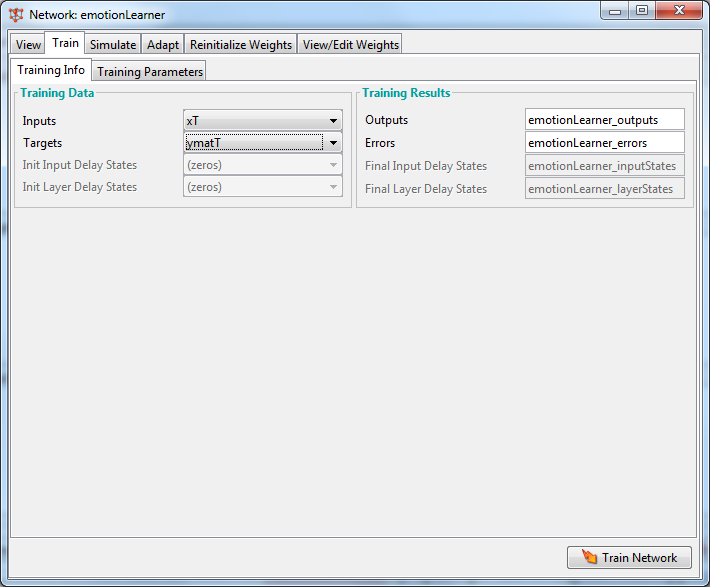


This creates a 2 layer network(one hidden and one output) with 136 row inputs according to xT with 6 rows of target outputs according to ymatT. Adjusting the number of layers would create additional hidden layers. This can be viewed as below:

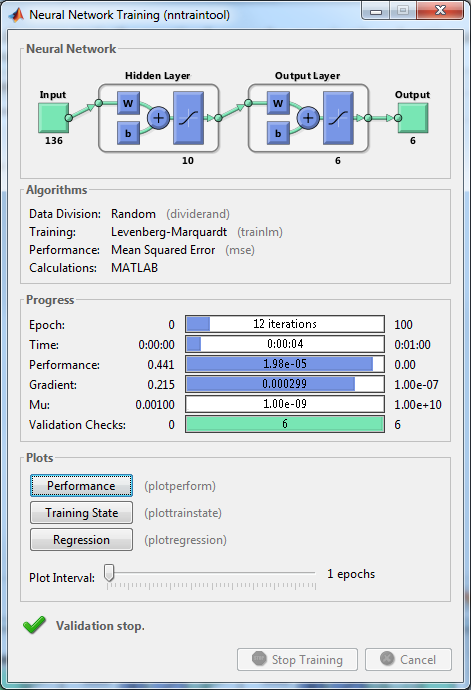


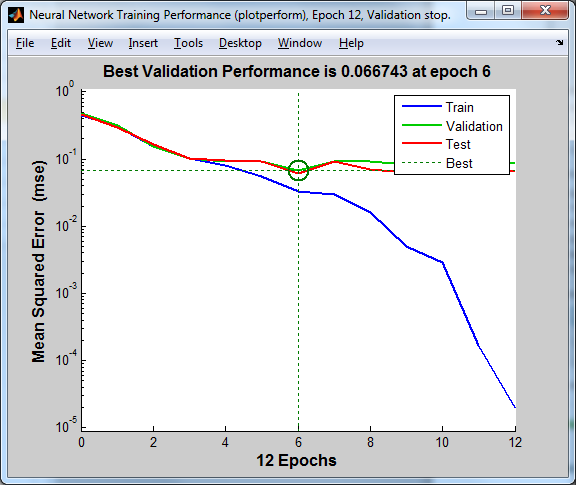


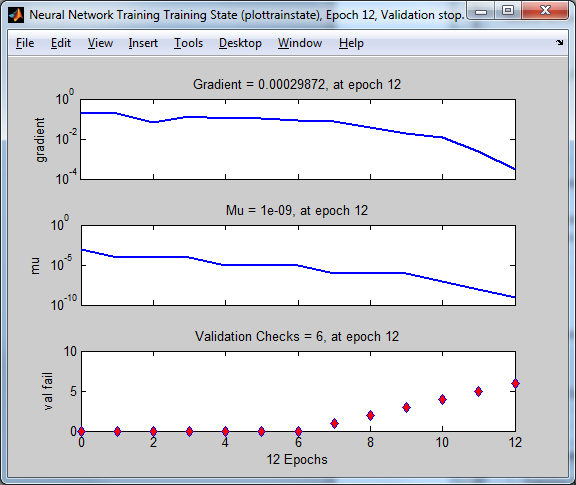
We then can configure the emotionLearner network by opening it and setting our training info and parameters:

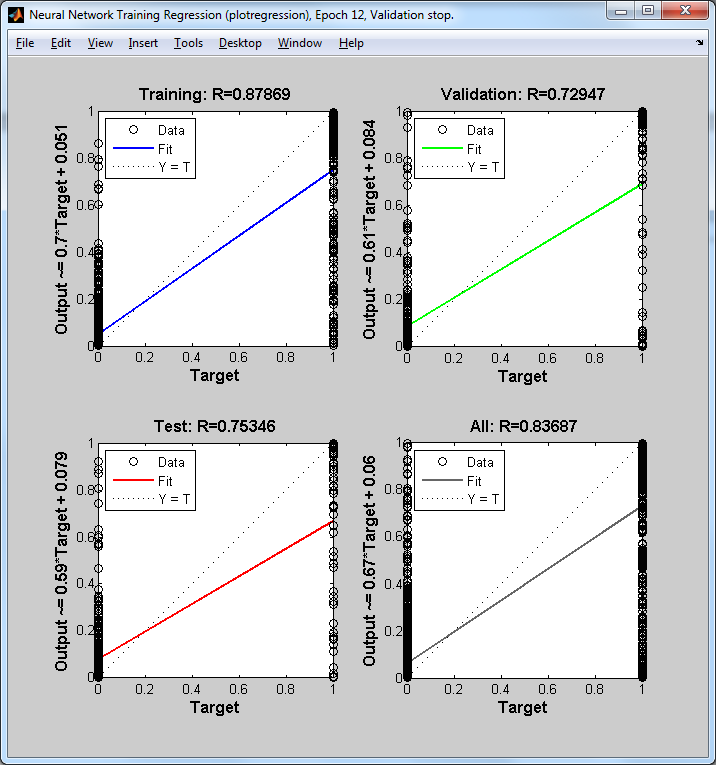


We begin with a 2 layer setting(1 hidden + 1 output), 100 epochs (learning rate) showing at a rate of 5 epochs with a time of 60.



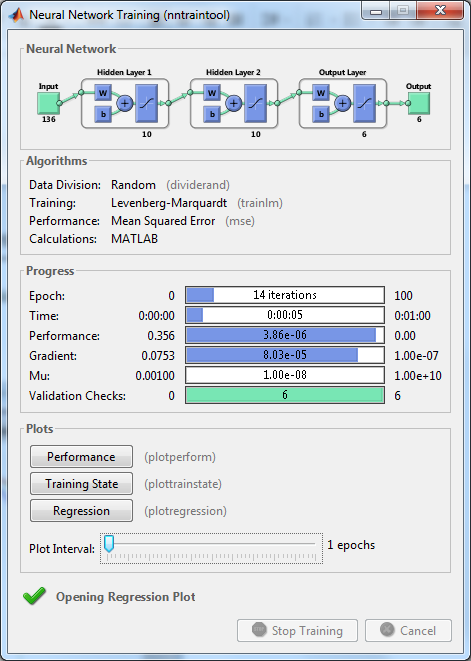


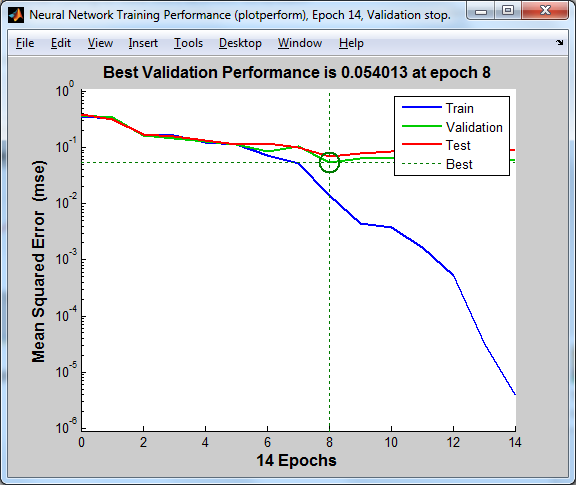




Regression plots here would not help as the data is meant for a classification activity.

Using 2 hidden layers gives a different best performance in validation:





# Appendix C - Human Test Script

function [humanAccuracy] = humanTest(dataX, dataY)

[m,n] = size(dataX);

currentLine = 1;

humanGuess = zeros(1);

while currentLine <= m

xVis = dataX(currentLine,1:n/2);

yVis = dataX(currentLine,(n/2)+1:end);

clf;

hold off;

scatter(-xVis,-yVis);

disp('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

disp(strcat('Current guess: ', int2str(currentLine), '-out-of-' , int2str(m)));

disp('select emotion: from the following:');

disp('1 = anger');

disp('2 = disgust');

disp('3 = fear');

disp('4 = happiness');

disp('5 = sadness');

disp('6 = surprise');

humanGuess(currentLine) = input('?...')';

disp(' ');

disp('your guess was: ');

if dataY(currentLine) == humanGuess(currentLine)

disp('correct');

else

disp('not correct');

end

disp(' ');

currentLine = currentLine + 1;

end

humanAccuracy = calcAccuracy(humanGuess, dataY);

disp('your guesses: ');

disp(humanGuess);

disp('Ground truth: ');

disp(dataY');

end

function accuracy = calcAccuracy (in1, in2)

binaryMatrix = zeros(1, length(in1));

for x = 1:length(in1)

if in1(x) == in2(x)

binaryMatrix(x) = 1;

end

end

accuracy = (sum(binaryMatrix)/length(in1)) \* 100;

end