  
  
  
  
  
  
  
  
National University of Singapore  
  
EE4305 Fuzzy Logic and Neural Networks  
  
**Neural Network Assignment**  
  
  
  
Lee Kai Yi A0122357L  
  
 

**1 Introduction**  
  
Neural networks are computer systems made up of a number of simple, highly connected processing elements. Such networks are known to be useful for overcoming the limitations of machines, by being capable to learn, and fault tolerant, and have a wide variety of real life applications in different types of problems.

Classification Problems

Classification problems are problems that aim to split a given dataset into a known, fixed output.  
In terms of classification problems, Feed-forward neural networks have been applied extensively in the field of image recognition in medical devices, consumer devices, and even in robots and self driving cars. Recent developments in especially self driving cars have been fast and impressive, with big companies like Google and Tesla having their own self-driving car projects, and Uber coming in trying to have a driverless fleet. Image recognition in consumer devices has also taken great leaps forward, with natural language processing in phones now enabling such consumer devices to process handwritten text in real time.

Classification MLPs Apart from in image recognition, classification MLPs could also classify data based on a dataset apart from an image, used in pattern recognition in other domains. Applications of these can be found used in search engines and biology, being used to predict trends in these fields. The most familiar application of this is notably in search engines, where almost everyone relies on search giants like Google to classify and return search results based on entered text.

Regression Problems  
  
Regression problems are used when a user wants to find a continuous output dependent variable from a series of fixed input independent variables. Regression problems are useful to predict the effects of how some variables affect others, though caution has to be taken as the correlation of an input variable to an output result might not imply causation.

Regression analysis could take place over a huge data set, using probabilistic models to predict the result of the problem. A recent breakthrough took the form of AlphaGo’s victory over the current reigning world champion of Go just this year, which was traditionally a game that artificial intelligence could not beat human players due to the complexity and sheer number of combinations in the game.

Other common applications of regression include helping to detect weather trends, and other trends that have a source of data, and you would want to detect the relationship between the data and the results you have obtained, like in the economy using economic devices to predict stock market trends.

**2 Classification Problem**  
  
In this problem, we are given 8000 samples of 561 features of HAPT (Human Activity and Posture Treatment) that were extracted from the raw inertial signals of smartphone sensors, and we are expected to categorize them into 12 target classes of activities, using the Neural Network Pattern and Recognition functions from the standard MATLAB Neural Network Toolbox.

**2.1 Configuration of the neural network**

**2.1.1 Default configurations**

Type of activation function

The default activation function is tansig for hidden layers, and softmax for the final layer. We will not be changing this in our network as it will not greatly affect our network performance.

Normalization

Normalization is crucial to ensure an increased accuracy in neural networks. Input attributes should be normalized between -1 and 1. In this case, since the data set is already normalized, it is okay as an input, and we will continue using this throughout our experiments.

Training functions

The default training function is scaled conjugate gradient (trainscg). There are other training algorithms like resilience backpropogation (trainrp) and Levenbery-Marquardt backpropagation (trainlm), but we will not be modifying them during this experiment.

Type of performance function

We will be using the confusion matrix as our main form of performance measurement in this experiment

**2.1.2 Variable configurations**

Number of neuron and layers

A quick literature survey would tell us that it is ideal to have up to 2 hidden layers, and a number of neurons in between the output classes and input attributes. Hence, I decided to stick with a default value of 100 neurons over 1 layer, and vary my inputs later.

Division of data

The data is divided into training ratio, validation ratio and testing ratio, in the ratio of 70-15-15. A quick literature survey tells us that a training ratio of 60-80% is optimal, and we will vary it to see how it affects the performance of our MLP.

Stop condition

Setting a limit to the stop condition is a way for the neural network to prevent overfitting to the training data set, as a way to obtain a good generalization of the data. We are currently using the default value of 6 as our stop condition, and will vary this in our experiments.

**2.2 Measuring performance of classification MLP**

**2.2.1 Observation on input data**

The input data given is skewed to the classes of 1-6, as seen in the histogram below in Figure 1.

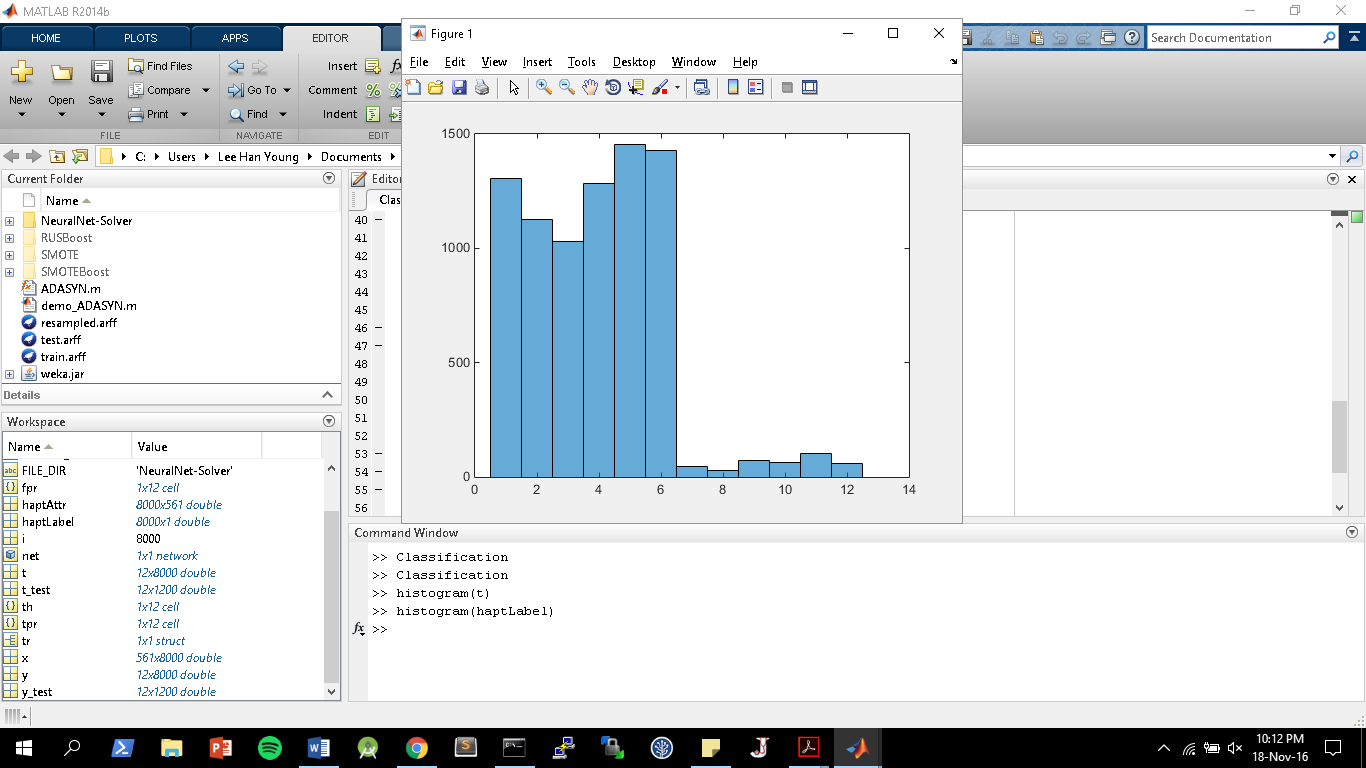


Figure 1: Histogram of values in classes for input

As such, purely using a neural network is potentially not the most reliable method to train our network due to data unbalancing issues, leading to the classifier to tend to classify the minority inputs as the majority inputs, which has a high accuracy when testing with the test data due to the spread of the test data being the same as the dataset given, but performing badly otherwise. I will discuss methods to overcome this, beyond the scope of this classification network, in section 2.4 below.

**2.2.2 Confusion matrix**

The confusion matrix is the breakdown showing correct and incorrect predictions made by the MLP classifier. MATLAB automatically generates 4 sets of confusion matrices, one each for the training, validation and test sets, and an overall confusion matrix. We will be mainly making use of the test confusion matrix in measuring the performance of each MLP design, as it measures the performance of the MLP as applied to a previously unseen dataset.  
  
**2.3 Performance of network**

**2.3.1 Default Performance**

For our default experiment, we made use of the default configurations as mentioned above. As seen from the resulting confusion matrix, we have obtained a high final accuracy of 96.6%, as seen in Figure 2.

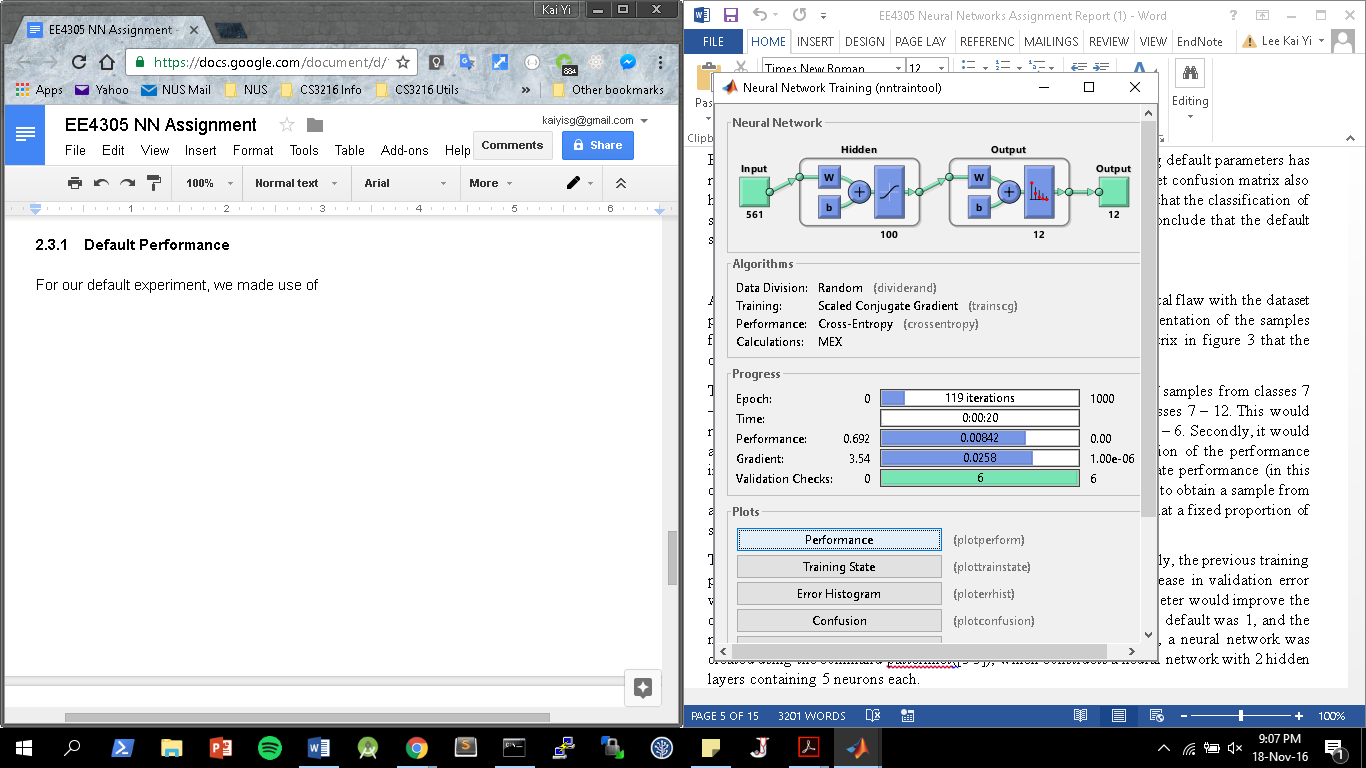


Figure 2: Default Configuration of MLP

However, the classification of data from classes 7-12 have lower accuracy than that of 1-6, averaging an accuracy of 50%, while the classes from 1-6 have an accuracy of near 100%, as seen in Figure 3 below.

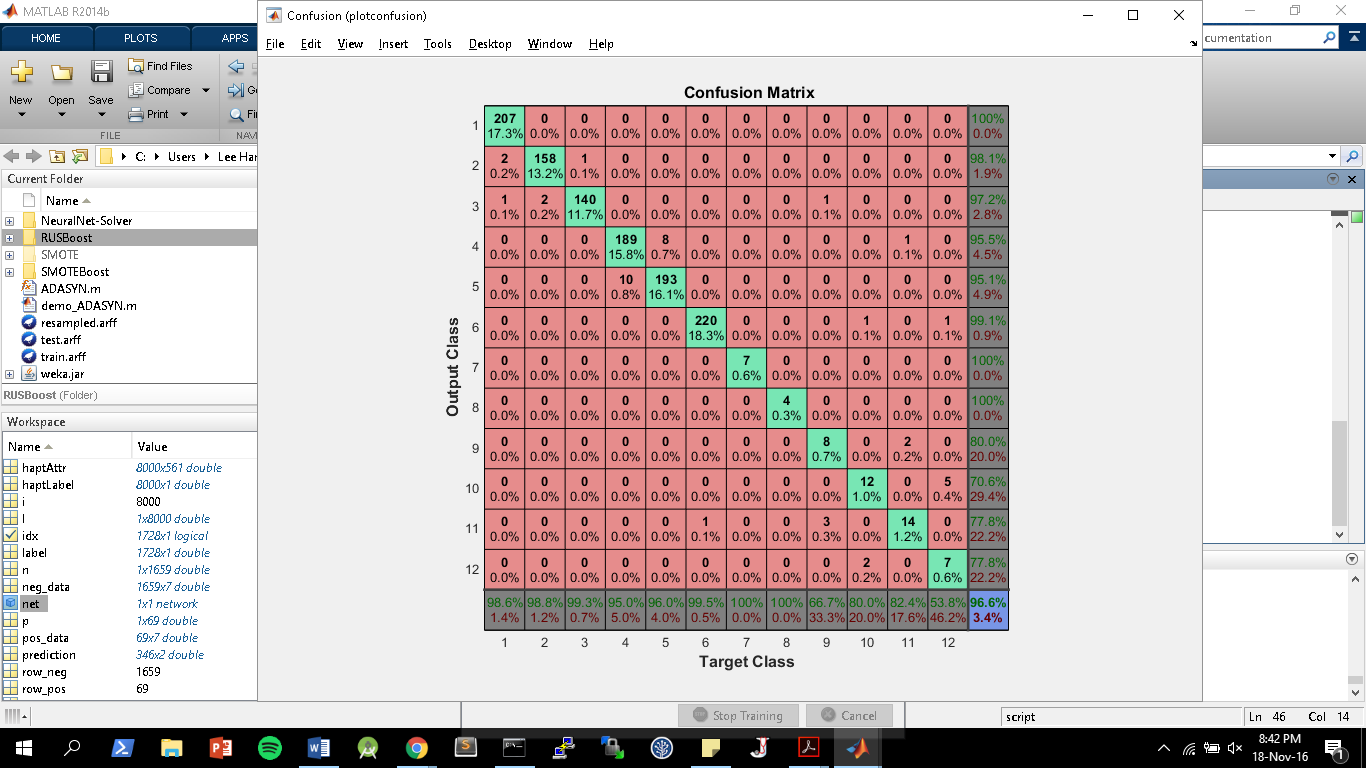


Figure 3: Default test confusion matrix

A simple 1-layer default MLP has performed rather well classifying our output. However, I would now explore how changing certain parameters has changed my confusion matrix, and what that means for the output.

**2.3.2 Changing number of layers and neurons**

We are using a MLP architecture now with 100 neurons in 1 layer.

The below table details the effects of different architectures on the MLP performance, with the performance metric of the percentage accuracy in the confusion matrix that we have previously mentioned.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Network / Layers | Trial 1 / % | Trial 2 / % | Trial 3 / % | Trial 4 / % | Average / % |
| 10 | 94.8 | 94.5 | 95.1 | 94.6 | 94.9 |
| [10 10] | 94.1 | 94.3 | 94.5 | 94.3 | 94.3 |
| 100 | 95.9 | 96.3 | 95.9 | 96.0 | 96.1 |
| [100 100] | 95.5 | 95.3 | 95.7 | 95.5 | 95.5 |
| 200 | 95.7 | 95.3 | 96.0 | 95.6 | 95.7 |

As can be seen from the observed performance, the 1 layer 100 neural network MLP configuration performs the best among the 5 other tests that I carried out. Furthermore, as discussed previously, the data is skewed towards the classes 1-6, hence we also have to look at how accurately the tests are classified in the 7-12 range of classes to measure the performance of the neural network. As this is an inherent limitation of the data set, we cannot improve that performance here, hence the best network configuration is the 100 neurons 1 layer network, that we chose above.

**2.3.3 Changing division of data**

We are using the default 70-15-15 division of data now for the neural network between the training, validation and test data sets.

The below table details the effects of different division of the data sets on the MLP performance, with all else kept constant.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Division of data (Training / Validation / Test) | Trial 1 / % | Trial 2 / % | Trial 3 / % | Trial 4 / % | Average / % |
| 70 - 15 - 15 | 95.9 | 96.3 | 95.9 | 96.0 | 96.1 |
| 50 - 25 - 25 | 94.0 | 94.3 | 93.7 | 93.9 | 94.0 |
| 90 - 5 - 5 | 96.0 | 95.7 | 96.5 | 96.4 | 96.3 |

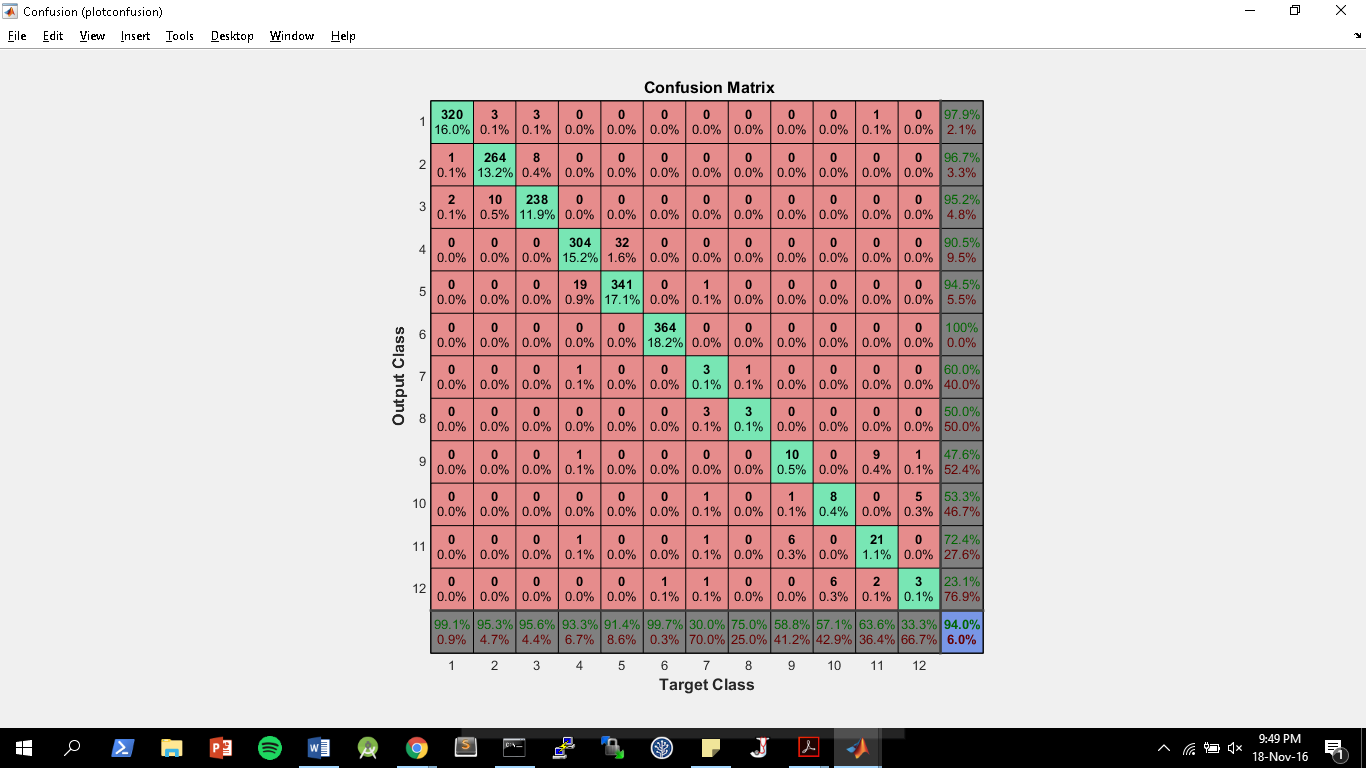


Figure 4: Confusion matrix for 50-25-25 data configuration

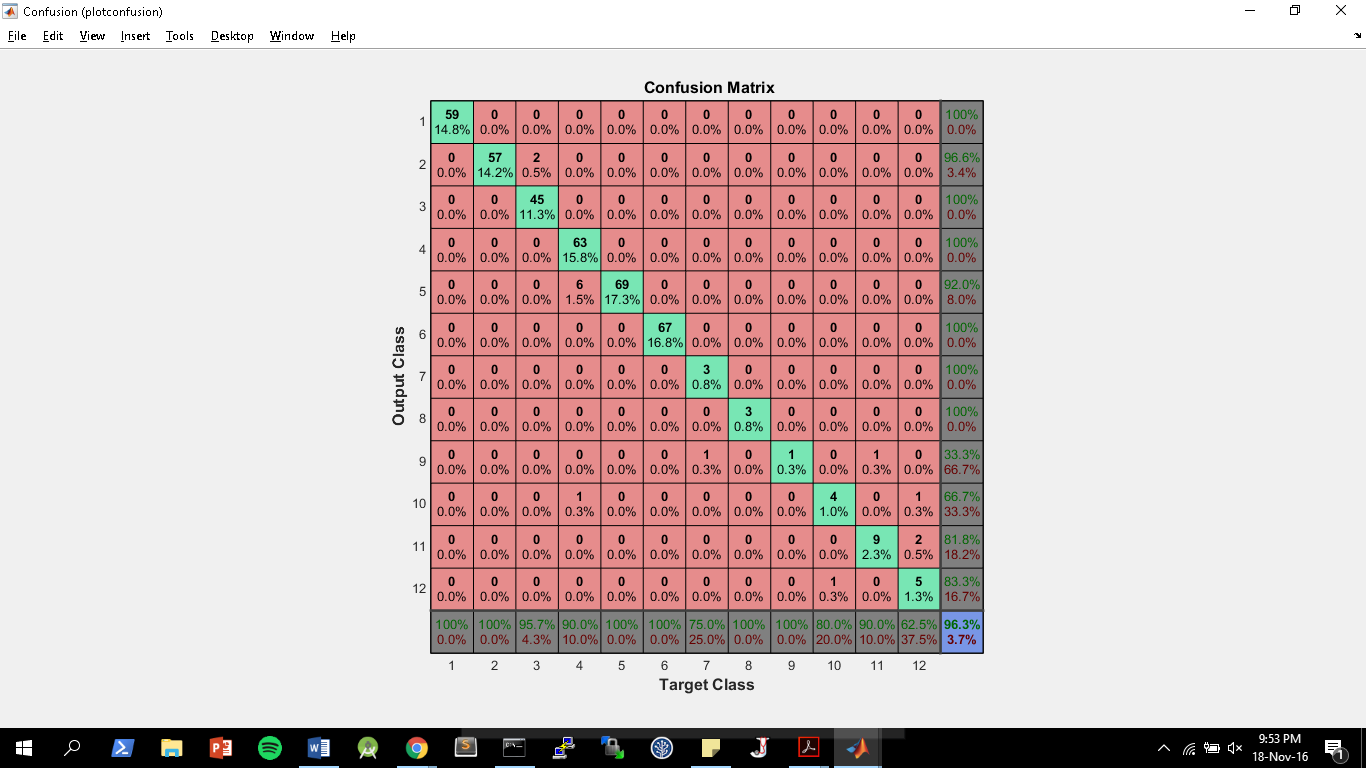


Figure 5: Confusion matrix for 90-10-10 data configuration

As can be seen from Figure 4 and Figure 5, as we run more training trials, we get a higher average test percentage of accuracy. However, when running MLP, accuracy is not the only metric we should look out for. In order to devote so much percentage of the data to the training set, we require a large pool of total data to play around with. With only 8000 data points, we may not have enough data to set a 90-5-5 configuration to only have 5% of the data for both validation and testing, hence it might be more favorable to use more data for testing to ensure that the system we have designed is robust. Hence, the trade-off of 70-15-15 as provided in the default settings is robust enough for our purposes.

**2.3.3 Changing stop condition**

Our default stop conditions stops our dataset after 6 trials. However, we would want to change this around to see if a larger or smaller stop condition would give us a more general curve, that is less overfitted.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Stop Condition | Trial 1 / % | Trial 2 / % | Trial 3 / % | Trial 4 / % | Average / % |
| 6 | 95.9 | 96.3 | 95.9 | 96.0 | 96.1 |
| 10 | 97.1 | 96.9 | 97.3 | 96.5 | 96.9 |
| 2 | 93.9 | 93.6 | 93.9 | 93.5 | 93.8 |

The overall accuracy generally increases as the stop condition increases, as the MLP is trained more within this time frame. However, care has to be taken not to overtrain the data as previously mentioned, as this might lead to overtraining, which leads to a too ungeneralized curve which would perform badly against data points not on the data set. Hence, the value of 6 as chosen in the default values is ideal.

**2.4 Summary & Further Improvements**

Through attempting to vary the different parameters that we have identified as crucial a classification neural network like the architecture of the network in terms of the number of hidden layers and neurons, the stop condition of the classifier to prevent overfitting to the training data set, and the division of data between the training, test and validation sets, we are able to see that the network with the best performance is actually the one which we originally identified.

In terms of improvements, the dataset that we obtained is actually skewed to 1-6, as mentioned in the analysis of the data set above. Hence, accuracy might sometimes not be a good indicator, as it only indicates how the test data is classified, but may perform worse with data that has a balanced range between 1-12. Hence, other methods could be used, like oversampling the minority data or undersampling the majority dataset.

**3. Regression Problem**

In this problem, we are given 350 samples of 30 features of students, and we are expected to use regression to predict their final score. We are also given 2 extra results, which we will use as additional features for the students, to predict their scores. We will do this using the Neural Network regression functions from the standard MATLAB Neural Network Toolbox.

**3.1 Configuration of the neural network**

**3.1.1 Default configurations**

Type of activation function

The default activation function is purelin for for the final layer. We will not be changing this in our network as it will not greatly affect our network performance.

Normalization

Normalization can be used to ensure an increased accuracy in neural networks, by normalizing input attributes between -1 and 1. However, we will not be doing normalization in this case as it will not vastly affect our results.

Type of performance function

We will be using the mean-squared error (mse) method to measure the performance of our regression system.

**3.1.2 Variable configurations**

Number of neuron and layers

Similar to the first part (as it is still a neural network MLP), it is ideal to have up to 2 hidden layers. However, we will vary this in our experiments below. We will have a default number of neuron and layers as 1 layer with 10 neurons.

Training functions

The default training function now is Levenbery-Marquardt backpropagation (trainlm). We will be experimenting with using different training functions for different parts of our program.

**3.2 Measuring performance of regression MLP**

**3.2.1 Error histogram**

The error histogram is a measure of how much error is present at the end when feeding the data into the trained MLP. For very accurate networks it can be hard to distinguish their performance using this, but as we will see later, if a network performs badly, especially in regression problems whereby the inputs have no relationship with the outputs at all, this can be a quick way to identify this bad performance.

**3.2.2 Regression Plot**

In attempting to fit the input data to a certain linear prediction, a regression plot is useful in determining how well the data fits the input data. In this case, only the test dataset regression plot is useful for us to determine the performance of the MLP on the data set. In each regression plot, there will be an R value to determine how well the data fits with the regression plot line, and this value is between 0 to 1. A value of R=1 suggests a good fitting, while a value of R=0 suggests no relationship between the data to the predicted regression line.

**3.3 Performance of regression model without G1 & G2 in predicting G3**

**3.3.1 Default Performance**

For our default experiment, we made use of the default configurations as mentioned above, and given these parameters, the program ran for a few cycles and stopped. As seen from the resulting error histogram in Figure 6, there is a lot of error between the target and actual output in all 3 partitioned sets of data, from the training, to the validation and test.

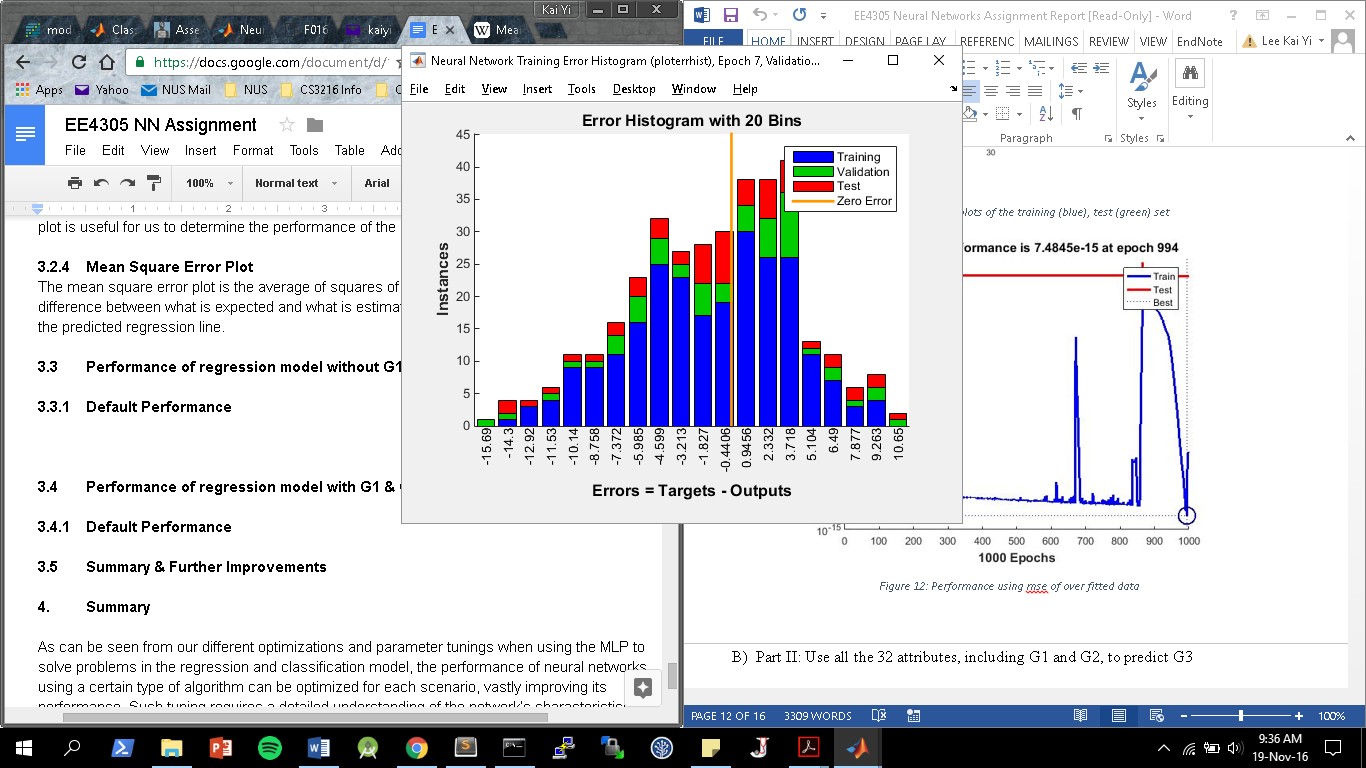


Figure 6: Error histogram of default MLP without G1 & G2

This bad performance is confirmed when looking at the regression plot in Figure 7, where the regression line drawn is predicted to have a R value of -0.076514, indicating no relationship between the inputs and the outputs.

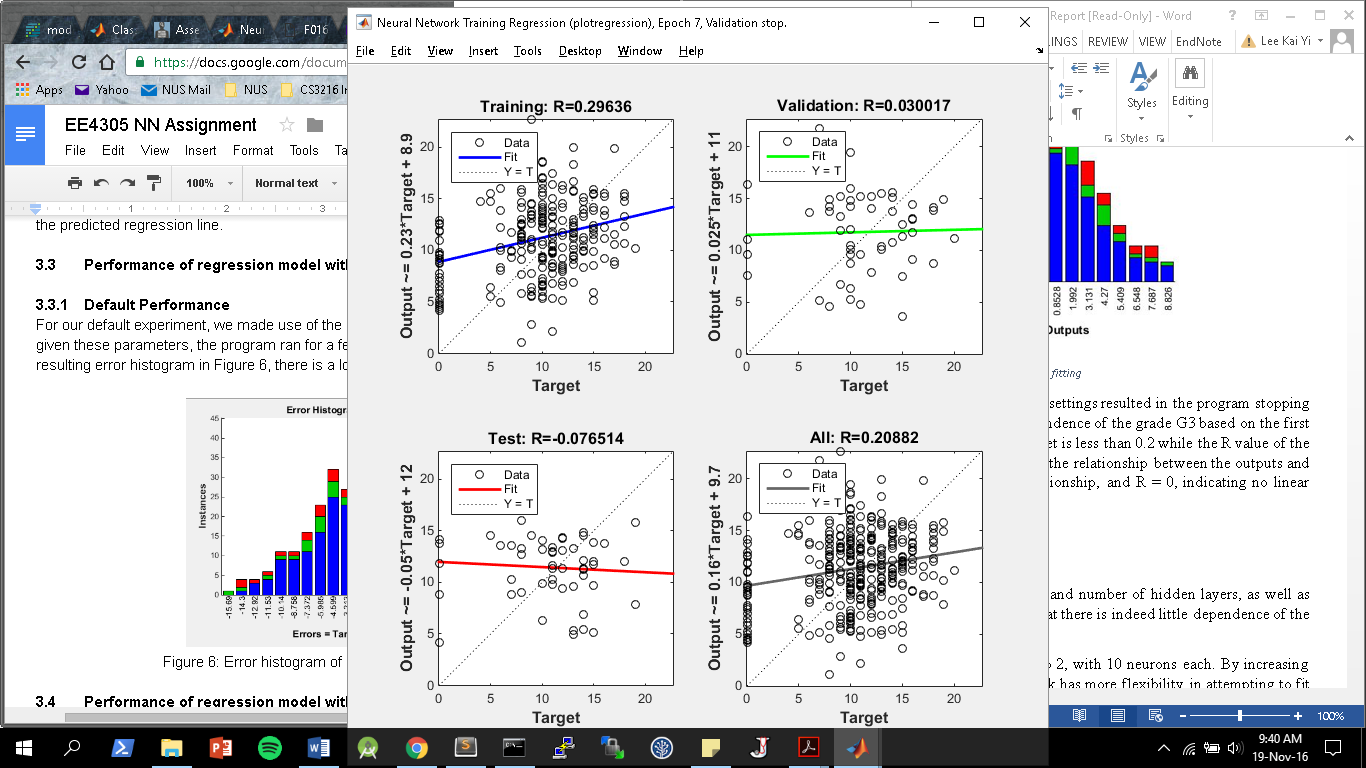


Figure 7: Regression plot of default MLP without G1 & G2

Given the high error and the low correlation as suggested through checking the regression plot and error histogram, it would not be viable to check other performance metrics like the mean square error plot, and we should try to optimize our model before proceeding on to checking its performance.

**3.3.2 Changing training function**

We are using a MLP training function of trainlm, and we will be attempting to change this to trainbr in this section.

Unlike in trainlm which has early stopping when there is too much error detected, trainbr has no such function. Hence, when no good regression fit was detected, due to no early stopping, trainbr ran until it reached the maximum epochs set, which was 1000. However, from the histogram in Figure 8 below, we still cannot manage to find any correlation between the input set and the results G3.

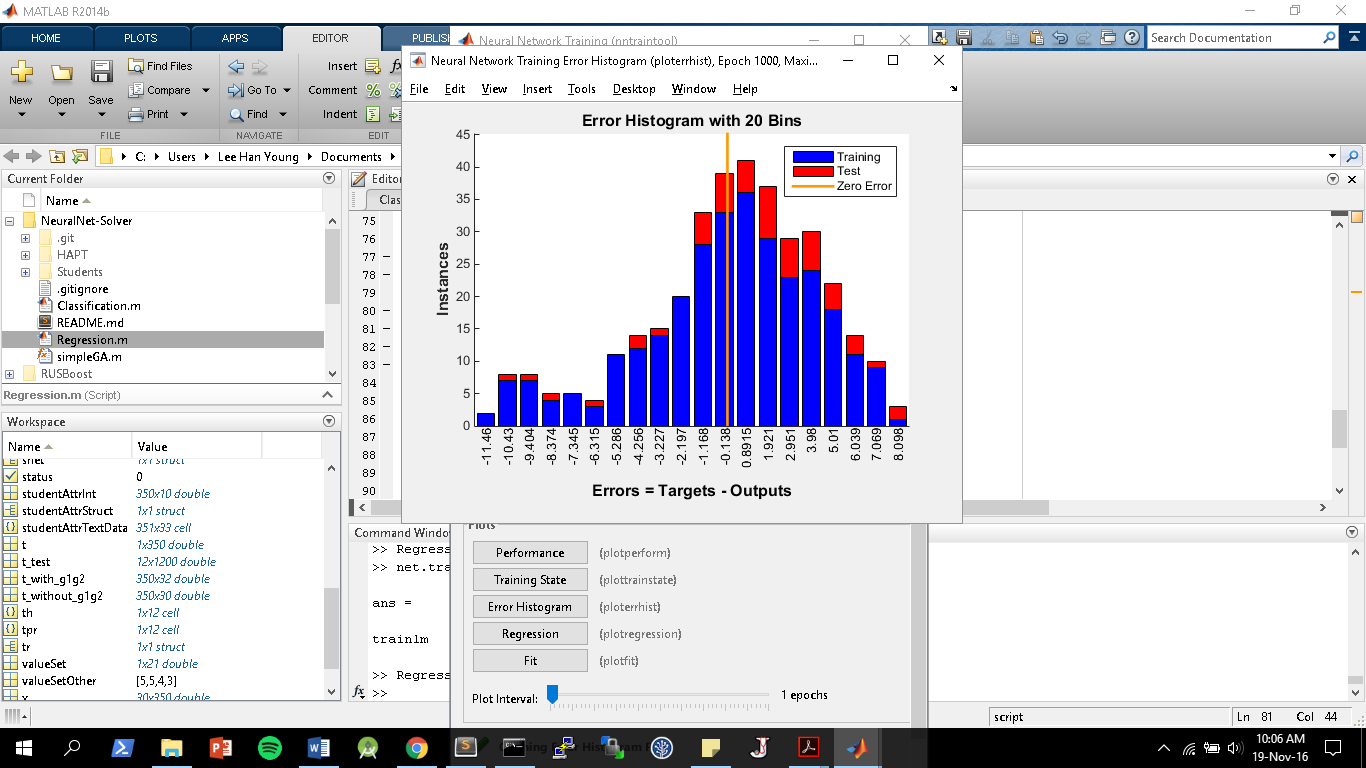


Figure 8: Error histogram of trainbr MLP without G1 & G2

**3.3.3 Changing number of neurons and layers**

We are using a MLP training number of neuron and layers as 1 layer with 10 neurons, and we will be changing it to a 2 layer 10 neuron each network.

Our first observation is that when using the trainlm function, this new network runs for more iterations as compared to the single layer, 10 neuron each MLP.

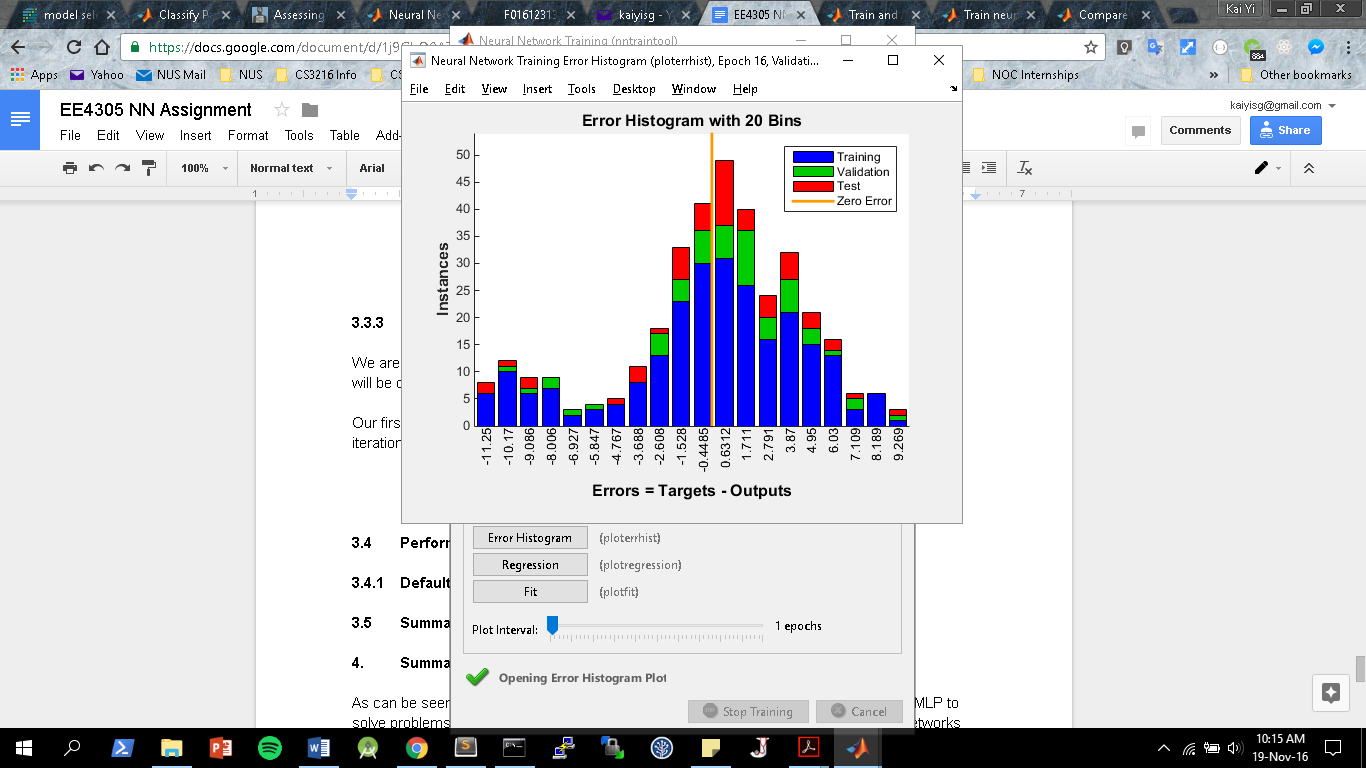


Figure 9: Error histogram of [10,10] MLP without G1 & G2

As can be seen from Figure 9 above, regardless of how fast the network has stopped training, through varying the MLP architecture in terms of the number of neurons in the hidden layers, it does little to vary the huge presence of error in the error histogram.

**3.3.4 Conclusion of regression model without G1 & G2 in predicting G3**

Given that we have changed all the possible parameters to tune that affects the MLP in this setup, with no visible change to the fact that the error histogram still indicates a lot of error between what is expected and the actual output of the regression model built, we can conclude that the dataset has no correlation with predicting G3 of students.

**3.4 Performance of regression model with G1 & G2 in predicting G3**

**3.4.1 Default Performance**

For our default experiment, we made use of the default configurations as mentioned above. As seen from the resulting error histogram in Figure 10, there is now less error as compared to when compared to the first part, suggesting a better correlation between the data and G3 with the addition of the additional inputs G1 and G2.

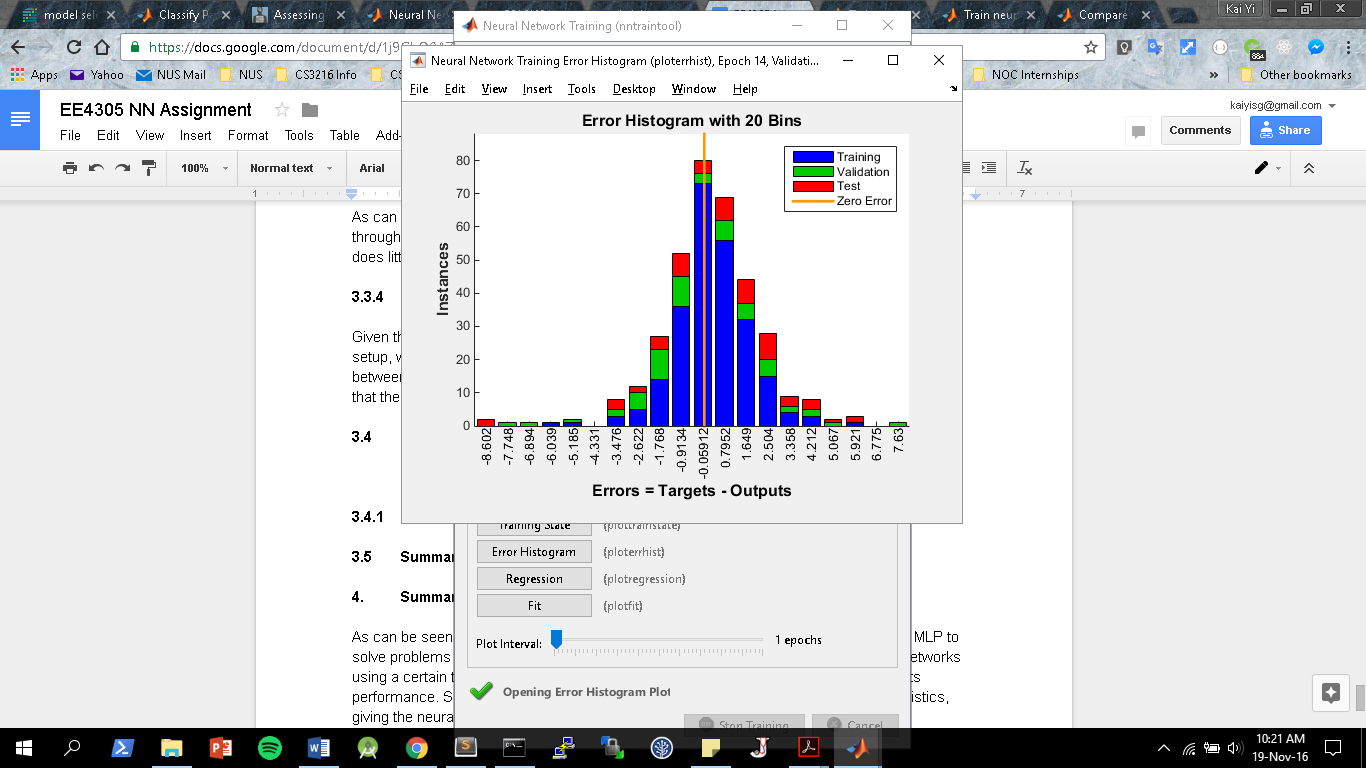


Figure 10: Error histogram of default MLP with G1 & G2

Furthermore, this is further confirmed in Figure 11 where we see a higher R value when testing the test set data against the regression line plotted, with a R value now of 0.80, which is close to 1, suggesting a good correlation.

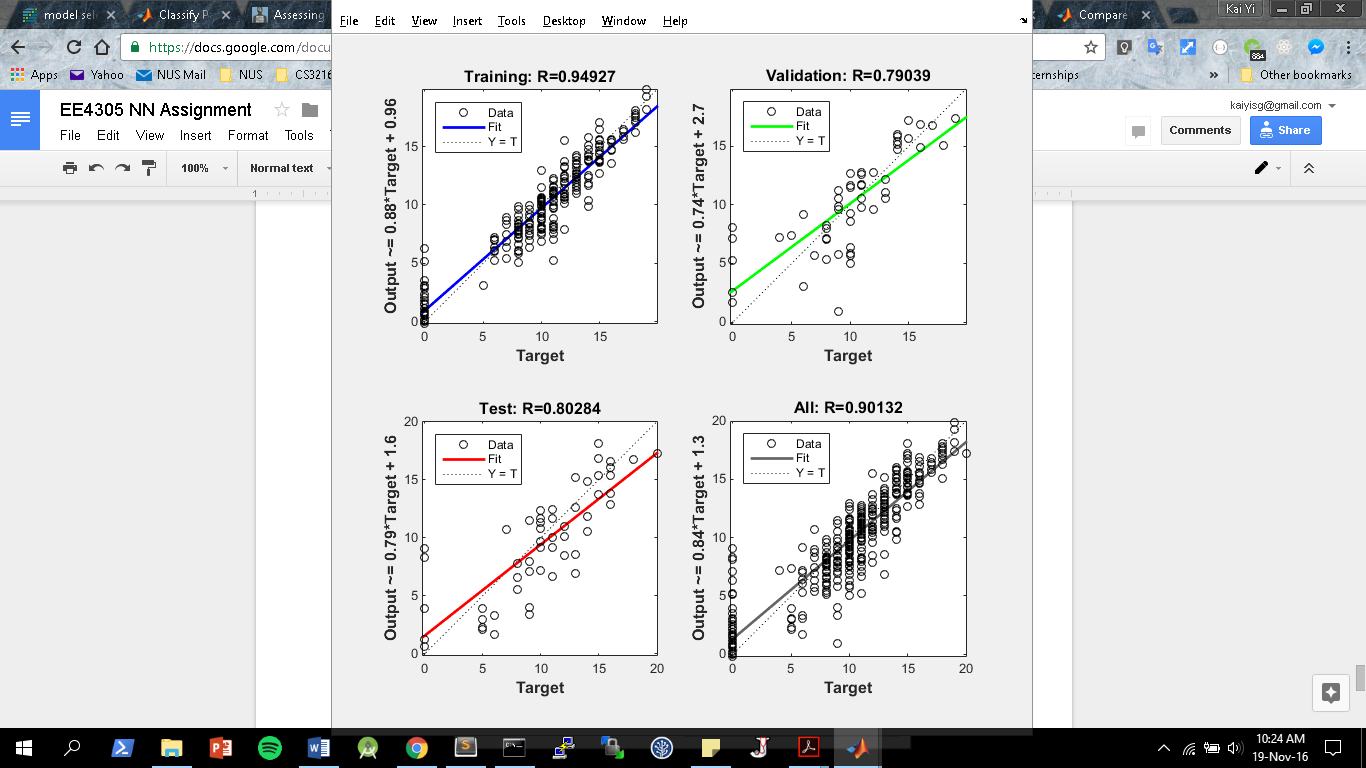


Figure 11: Regression plot of default MLP with G1 & G2

We will now proceed to tune our system to check for a good fit for the parameters to our system, to get a better regression line.

**3.4.2 Changing training function**

We have changed the training function now from trainlm to trainbr, which requires more epochs and time to train our network (around 351 passes needed). Trainbr also has a worse performance at generalizing, having a worse regression score of 0.782 with the test data as compared to 0.802 of the trainlm default setting.

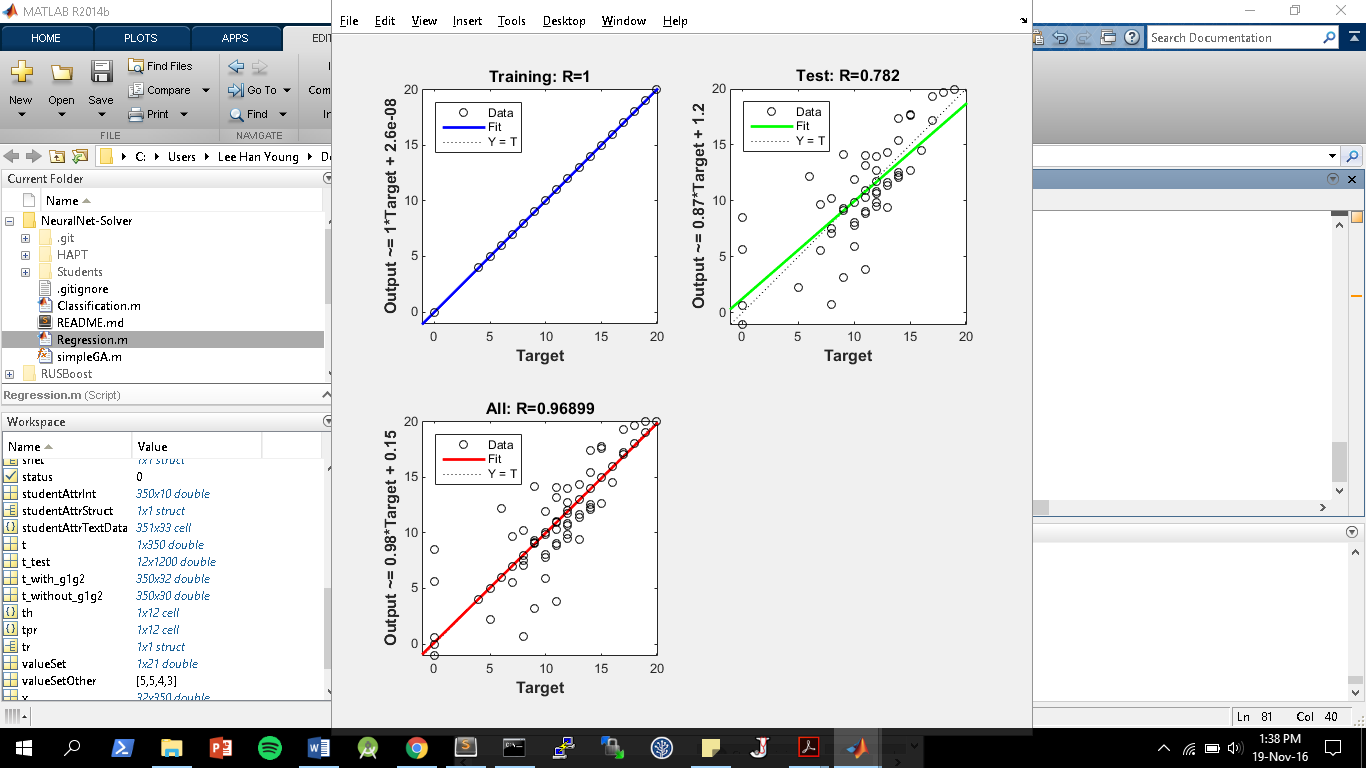


Figure 13: Regression plot using trainbr with G1 & G2

Hence, the training method of trainlm was a more effective method in this case as it both runs faster with less epochs, and gives a MLP that has a higher regression coefficient with test data sets.

**3.4.3 Changing neural net architecture**

Keeping our network using the trainlm training function, we now proceed to vary the number of layers in the network. The table below details the results of changing these parameters, on the regression R value of the test data:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Network / Layers | Trial 1 | Trial 2 | Trial 3 | Trial 4 | Average |
| 1 | 0.86 | 0.88 | 0.85 | 0.89 | 0.86 |
| [1 1] | 0.88 | 0.93 | 0.91 | 0.87 | 0.90 |
| 10 | 0.88 | 0.84 | 0.85 | 0.87 | 0.85 |
| [10 10] | 0.80 | 0.76 | 0.70 | 0.69 | 0.72 |

Based on the above results, we can conclude that a 2 hidden layer network, of 1 neuron each, is sufficient in our case to train the network.

**3.4.4 Conclusion of regression model without G1 & G2 in predicting G3**

Since the data sets that include G1 & G2 can give MLPs that perform with high regression line values when compared to a test data set, we can conclude that the student’s G1 and G2 scores are correlated to the G3 scores, but the rest of the attributes that were provided were not correlated.

The performance of the network is optimized by using the trainlm function, using 2 layers of 1 neurons each in the network.

**4. Summary**

As can be seen from our different optimizations and parameter tunings when using the MLP to solve problems in the regression and classification model, the performance of neural networks using a certain type of algorithm can be optimized for each scenario, vastly improving its performance. Such tuning requires a detailed understanding of the network’s characteristics, giving the neural network field a lot of potential.

In general, regardless of the type of problem being encountered or the type of algorithm employed to solve such problems, the neural networks, and the field of artificial learning, has a lot of potential in developing, and is still a field that is growing rapidly today.