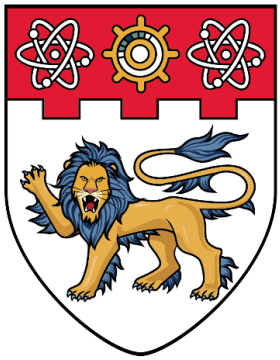


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE4042 Neural Networks and Deep Learning

Project Report 1

by

Lai Qing Hui, U1622339G
Ron Ng Jian Ying, U1622393B

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

2019

Contents

1 Introduction

The project will be exploring the usage of the off the shelves Deep Learning framework to solve classification and regression problems.

2 Methods

For the classification problem, neural networks will be built to classify the Cardiotocography dataset containing measurements of fatal heart rate (FHR) and uterine contraction (UC) features on 2126 cardiotocograms classified by expert obstetricians. Various learning parameters of the neural network such as the number of neurons, the batch size use and the weight decay value will be experimented to find out the ideal value.

The regression problem uses a Kaggle dataset containing 8 pieces of data – GRE SCore, TOFEL Score, University Rating, Statement of Purpose (SOP), Letter of Recommendation (LOR), undergraduate GPA (CGPA) and research experience and the Chance of Admission. SOP and LOR are assigned scores up to 5 and research experience is simply a 1 or 0 indicating whether the student had research experience. The goal of the regression problem is to use that to predict the chance of admission based on the first seven columns of data (serial number is ignored).

3 Experiments and Results

3.1 Classification Problem

Follow is the experiments procedure and the results gathered from the classification problem.

3.1.1 3-Layers Neural Network

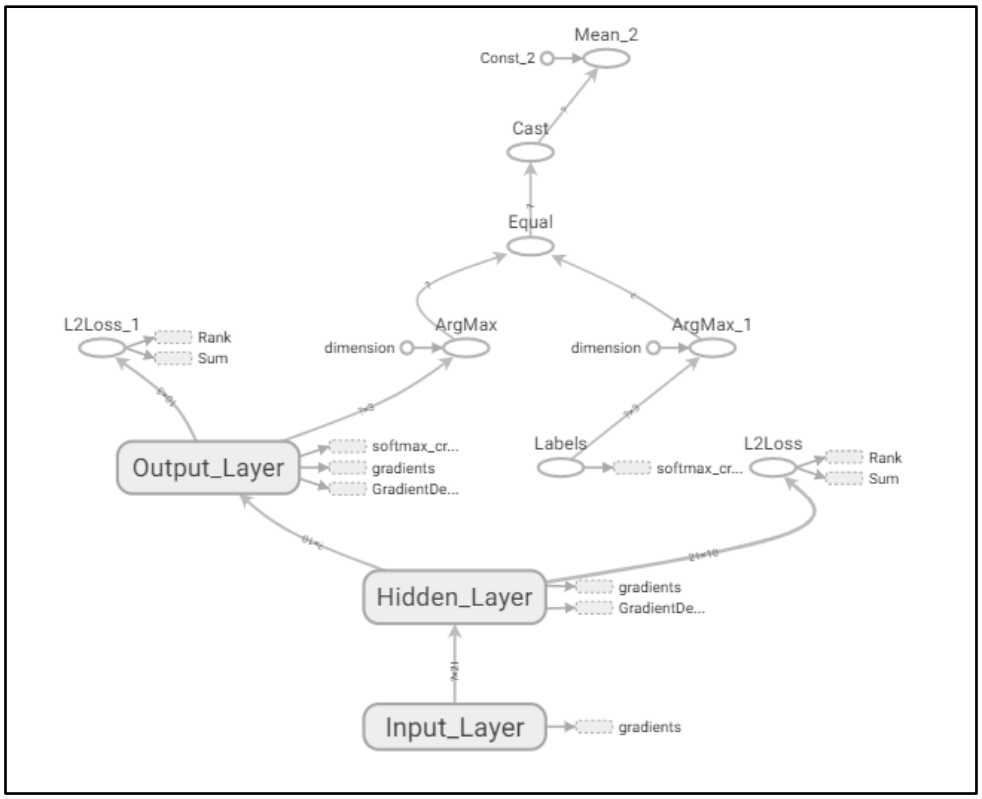


Figure 1: Computational graph of 3-layer neural network

Figure 1 show the overall concise view of the 3-layer neural network, where some of the operations are segregated into sub-graphs for easier view.

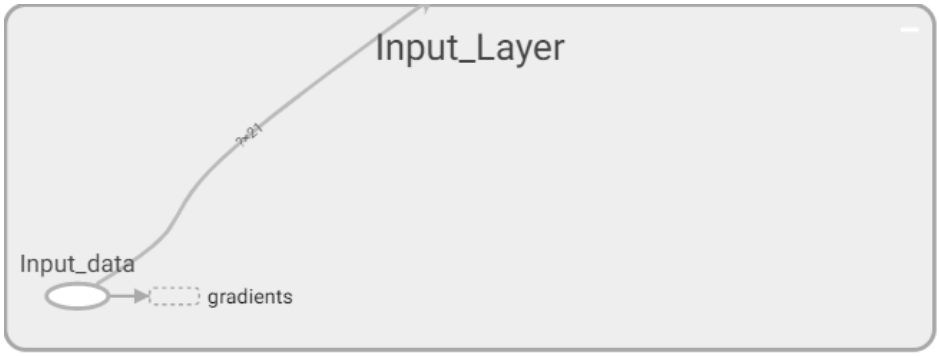


Figure 2: Place-holder in input layer

The input layer consists of a placeholder for the input data which will be execute through the computational graph.

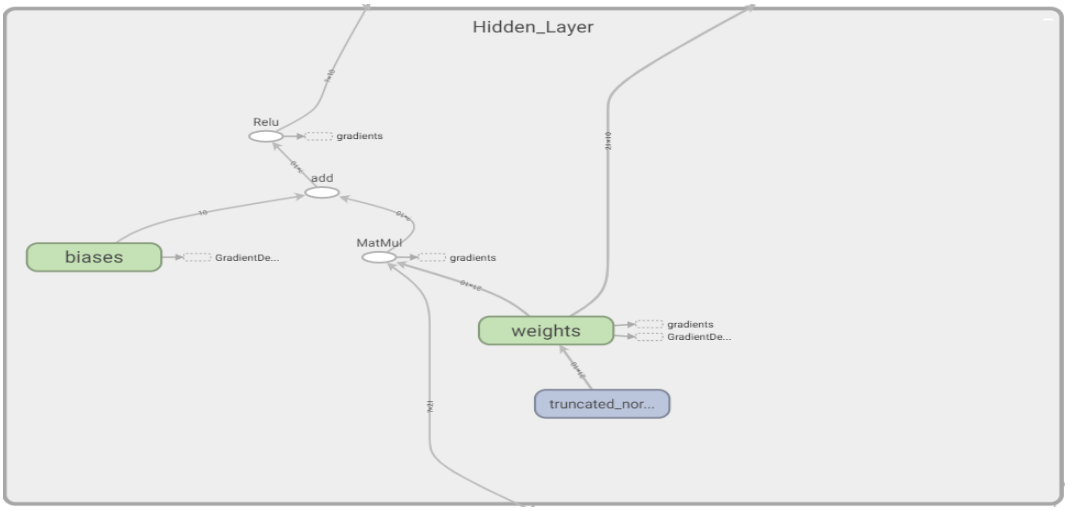


Figure 3: Computational graph of hidden layer

```
with tf.name_scope('Hidden_Layer'):
    weights = tf.Variable(
      tf.truncated_normal([no_features, hidden1_units],
                          stddev=1.0 / np.sqrt(float(no_features))),
        name='weights')
    biases = tf.Variable(tf.zeros([hidden1_units]),
                         name='biases')
    hidden1 = tf.nn.relu(tf.matmul(x, weights) + biases)
```

Figure 3 shows the computational graph of the hidden layer which is generated according to the TensorFlow operations as shown in the code snippet above. The hidden layer is consisting of 10 neurons with *ReLU* as its activation function, and a *Softmax* operation at its output.

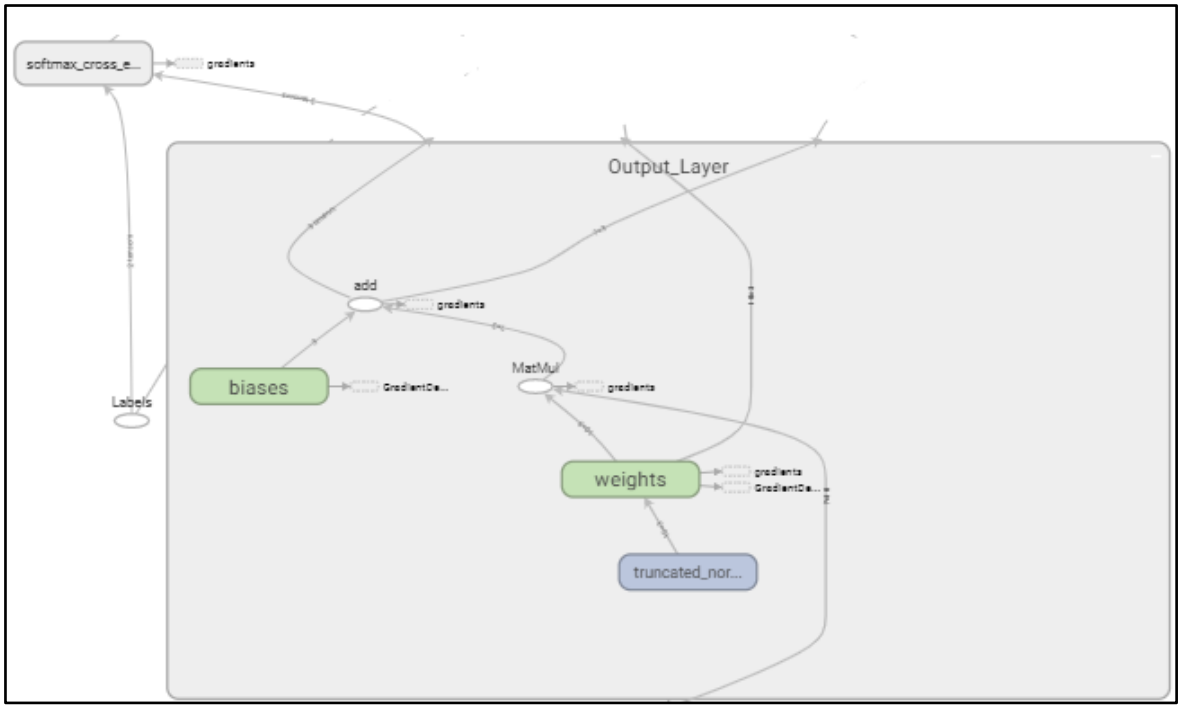


Figure 4: Output-layer computational graph

```
with tf.name_scope('Output_Layer'):
  weights = tf.Variable(
      tf.truncated_normal([hidden1_units, no_labels],
                          stddev=1.0 / np.sqrt(float(hidden1_units))),
      name='weights')
  biases = tf.Variable(tf.zeros([no_labels]),
                       name='biases')
  logits = tf.matmul(hidden1, weights) + biases
```

Figure 4 shows the computational graph of the output layer which is generated according to the TensorFlow operations as shown in the code snippet. The output layer is a *Softmax* output layer.

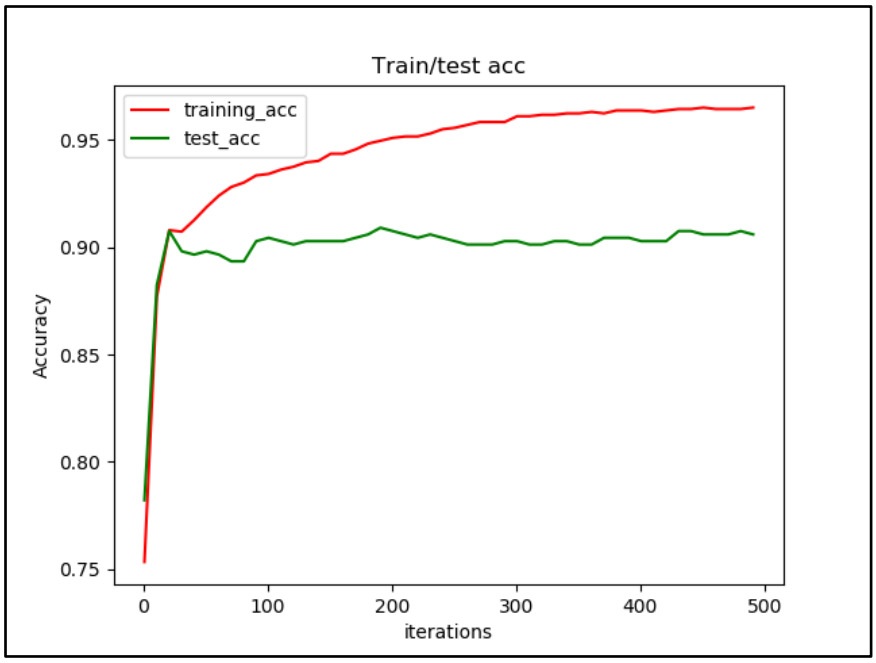


Figure 5: Train/Test accuracy after 500 iterations

Figure xxx shows the accuracy of model after 500 iterations. It is observed that the model converges after around 200 iterations.

3.1.2 Optimal Batch size

The approach taken for this experiment is to split the data into train and test set with a ratio of 70:30. For each batch size, 500 iterations were performed. In each iteration, the train data is split into 5 folds and each fold is used to conduct training for that iteration. The average accuracies for the 5 folds will then be recorded for that iteration to generate the plot of cross-validation accuracies against number of epochs. The result of the experiment is shown in the figure below.

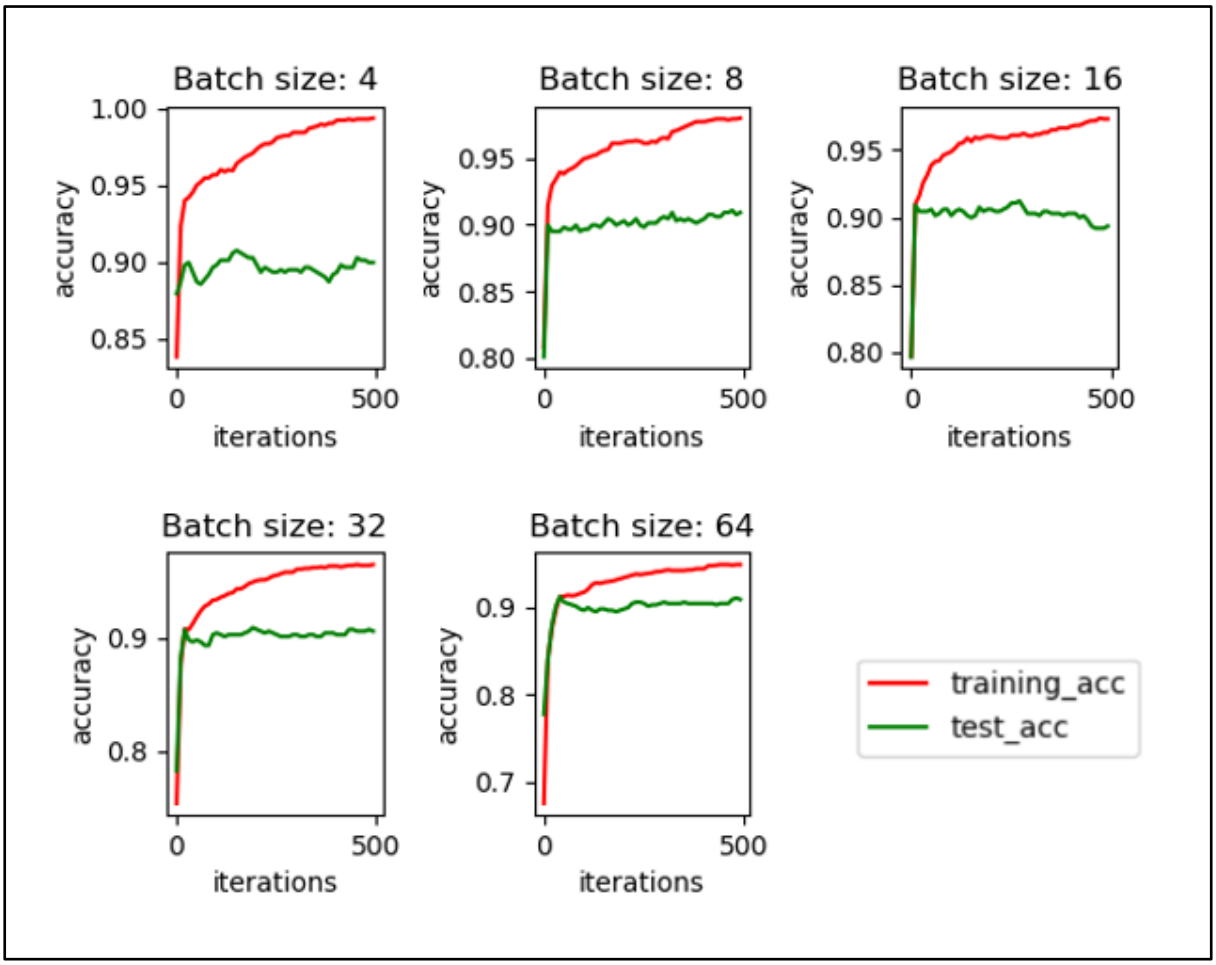


Figure 6: Cross-validation training and test accuracy for different batch sizes

The batch size of 64 has the least fluctuation between each epoch before it came to a convergence. Therefore, it will be chosen as the ideal batch size for the rest of the experiment.

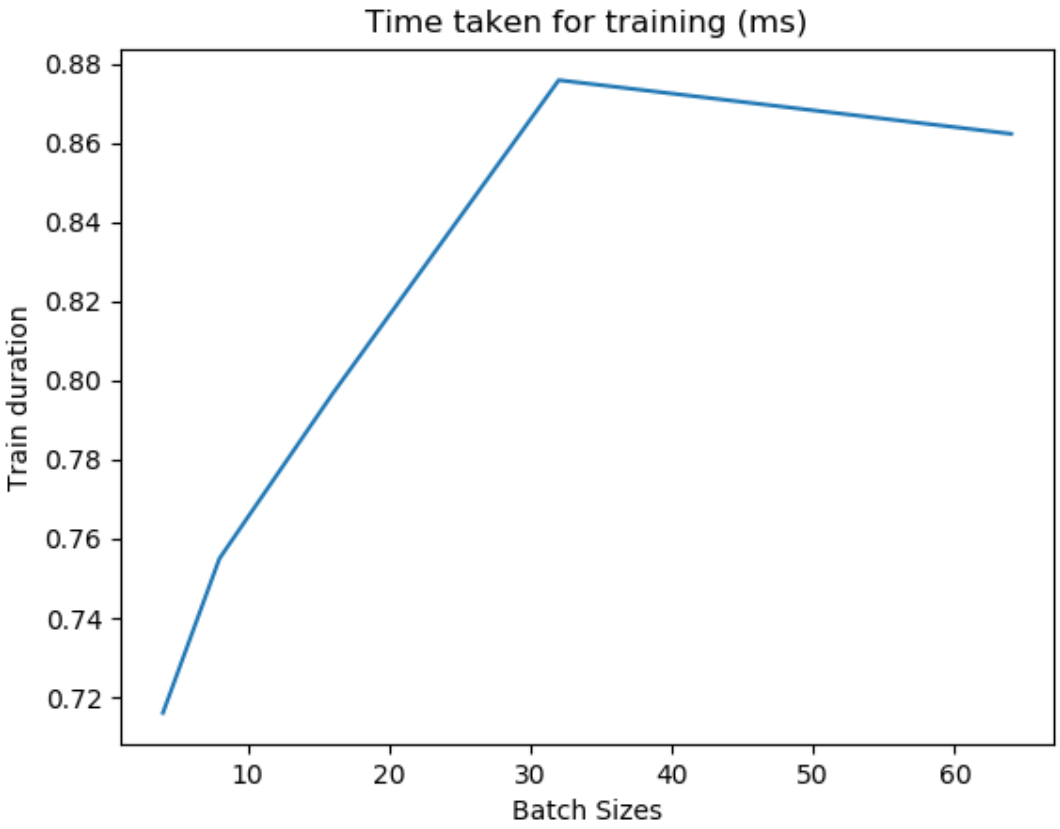


Figure 7: Training duration VS batch sizes

From figure 7, it can be observed that the average training duration per epoch increases steadily as the batch size increases. However, interestingly the duration starts to decrease after an epoch of 32.

3.1.3 Optimal Number of Hidden Neurons

For the next experiment, the model is evaluated with differing number of neurons in the hidden layer. Similar to the batch size experiment. The evaluation of the model is done iteratively with increasing number of neurons from 5 to 25 in steps of 5 each over 500 iterations. For each evaluation, the average 5-fold cross-validation accuracies and the test accuracies are plotted against the number of epochs. The result of the experiment is shown in the figure below.

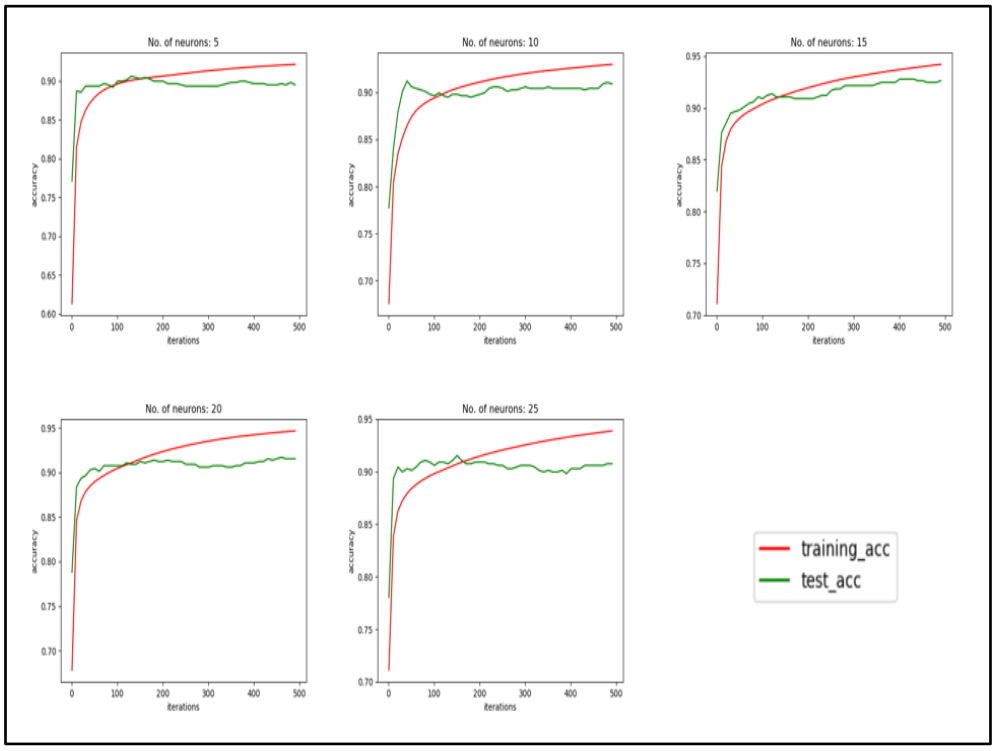


Figure 8: Cross-Validation training accuracy and test accuracies for different no. of neurons

3.1.4 Optimal Decay Parameter

During training, some weights variable will attain large values to reduce training error, thus jeopardizing its ability to generalizing. In order to avoid this, a decay parameter (regularization term) is added to the cost function. To find out the impact to the value of penalty term to the performance of the model, decay parameter of value 0, 10^{-3} , 10^{-6} , 10^{-9} and 10^{-12} was used to train and evaluate the model performance. The evaluation of the model is done iteratively with the set of decay parameter each over 500 iterations. The result of the experiment is shown in the figure below.

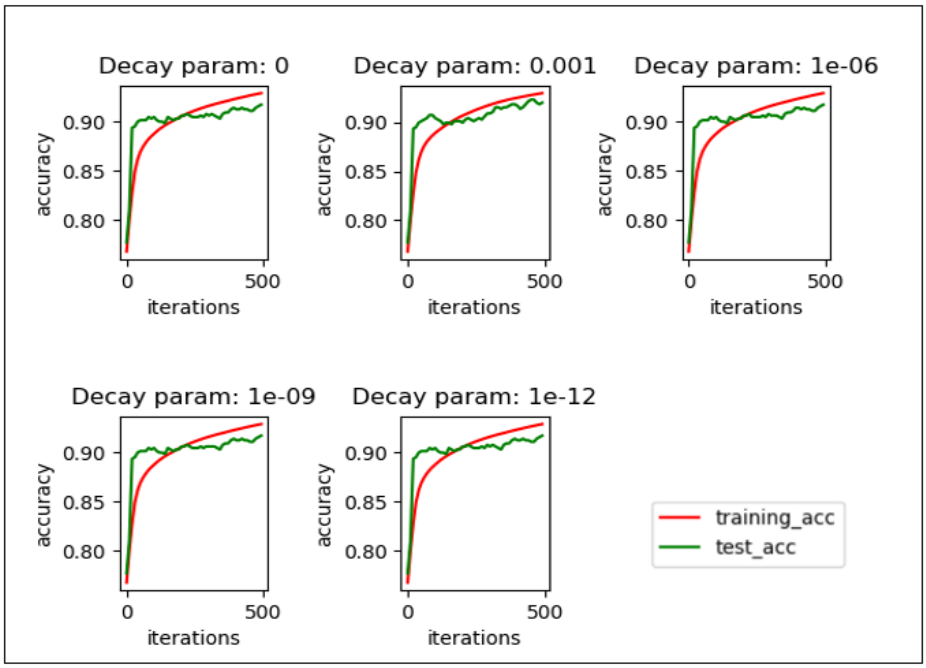


Figure 9: Cross-Validation training accuracy and test accuracies varying decay parameter

The results show that the optimum value of the decay parameter is 10^{-3} as the test accuracy converges closest to its train accuracy counterpart. This shows that this model is the least likely to have its test accuracy diverging from the train accuracy which will results in overfitting of the train dataset.

3.1 5-Layers Neural Network

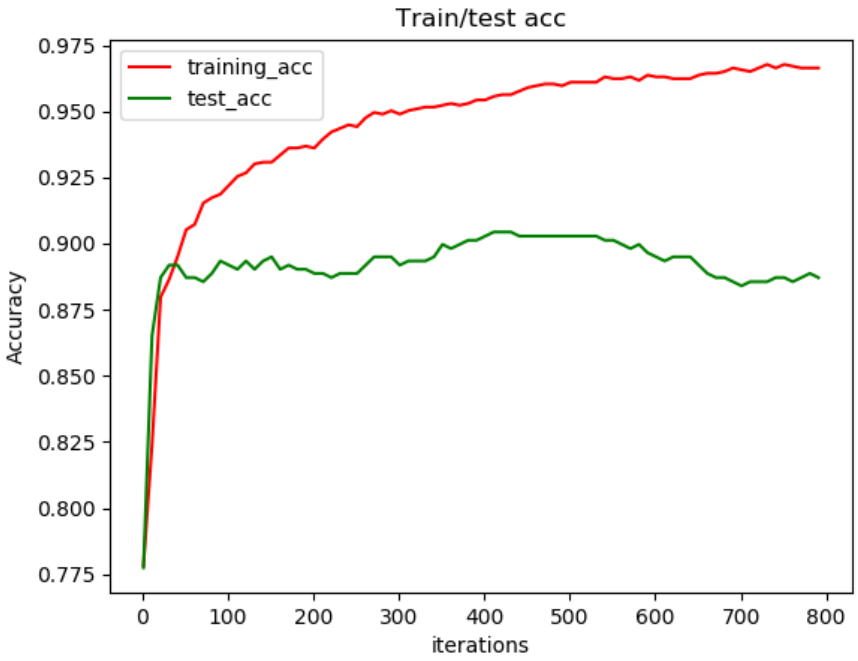


Figure 10: Train/test accuracies against no. of epoch

Figure 9 shows the training and test accuracies of the 5-layer neural network after 800 iterations. It can observed that the 5-layer neural network achieve its maximum test accuracies at approximately 450 iterations but is seen to diverge as the iteration increases to 550 iterations. The 5-layer neural network is seen to be less stable compared to its 3-layer neural network counterpart.

3.2 Regression Problem

Follow is the experiments procedure and the results gathered from the classification problem.

3.2.1 3-Layers Neural Network

A 3-layer feedforward neural network was built which consisted of an input layer, a hidden-layer of 10 neurons with ReLU activation functions, and an output layer with a linear activation function.

Mini-batch gradient descent was used with batch size of 8. L_2 regularization of 10^{-3} is used to reduce overfitting issues on our loss function. A learning rate of 10^{-3} is used.

The weights and biases of the hidden layer and output layer are defined as follows:

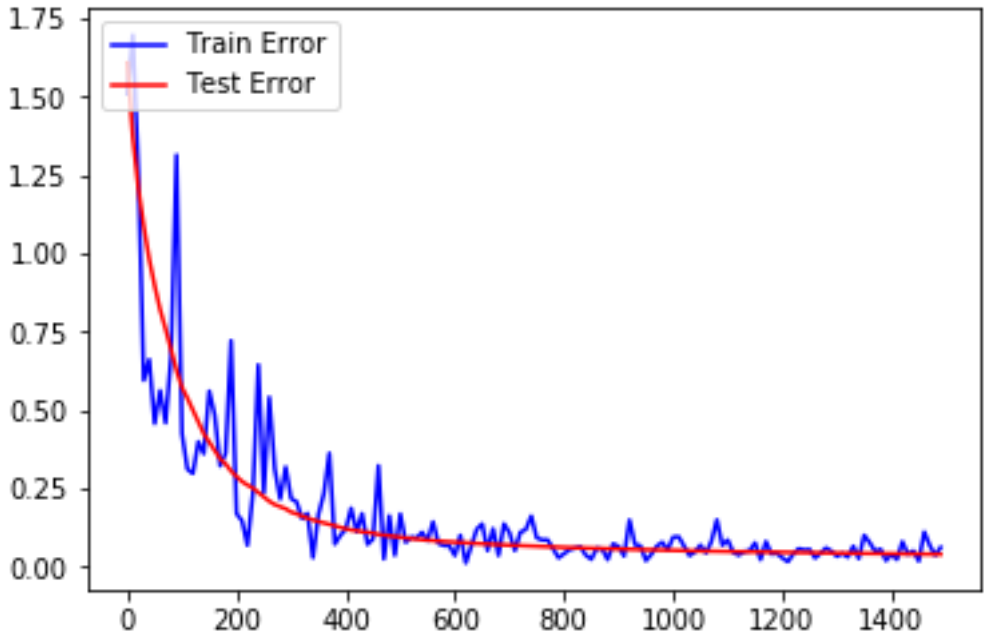
```
weights = {
    'h1': tf.Variable(tf.random_normal([NUM_FEATURES, NUM_NEURON])),
    'out': tf.Variable(tf.random_normal([NUM_NEURON, 1]))# 1 ouput label
}

biases = {
    'b1': tf.Variable(tf.random_normal([10])),
    'out': tf.Variable(tf.random_normal([1]))
}
```

And the neural net is defined as follows:

```
def neural_net(x):
    #hidden layer 1
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1) #activation function
```

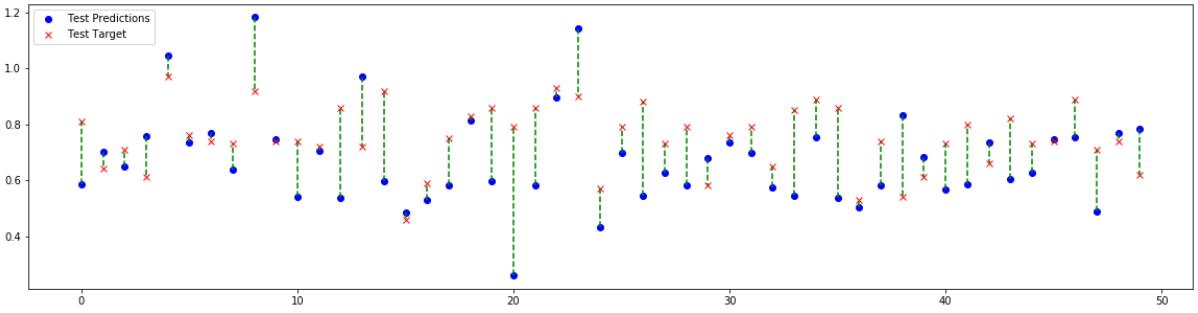
```
    out_layer = tf.matmul(layer_1, weights['out']) + biases['out']
    return (out_layer)
```

Figure 11: Results with all 7 features



The figure above is a plot between the test and train error over the number of epochs. The approximate epoch at which test error is minimum is around the 1200th epoch.

Figure 12: Predicted vs target



At the 1500^{h} epoch, the train error and test error are as follows:

```
train error 0.0340372, test error 0.040109
```

3.2.2 Correlation matrix

The table below shows the 8x8 correlation matrix. The diagonal values are all 1 because it is the value of correlation with itself which we can ignore.

	GRE Score	**TOEFL Score**	**University Rating**	**SOP**	**LOR**	**CGPA**	**Research**	**Chance of Admit**
GRE Score	1	0.835977	0.668976	0.612831	0.557555	0.83306	0.580391	0.80261
TOEFL Score	0.835977	1	0.69559	0.657981	0.567721	0.828417	0.489858	0.791594
University Rating	0.668976	0.69559	1	0.734523	0.660123	0.746479	0.447783	0.71125
SOP	0.612831	0.657981	0.734523	1	0.729593	0.718144	0.444029	0.675732
LOR	0.557555	0.567721	0.660123	0.729593	1	0.670211	0.396859	0.669889
CGPA	0.83306	0.828417	0.746479	0.718144	0.670211	1	0.521654	0.873289
Research	0.580391	0.489858	0.447783	0.444029	0.396859	0.521654	1	0.553202
Chance of Admit	0.80261	0.791594	0.71125	0.675732	0.669889	0.873289	0.553202	1

By rearranging the data, we can get a better understanding of the correlations of features. In the table below, we can see that CGPA, GRE Score and TOEFL score have the highest correlations with the Chance of Admit.

	1st	2nd	3rd	4th	1st_Val	2nd_Val	3rd_Val	4th_Val
GRE Score	TOEFL Score	CGPA	Chance of Admit	University Rating	0.835977	0.83306	0.80261	0.668976
TOEFL Score	GRE Score	CGPA	Chance of Admit	University Rating	0.835977	0.828417	0.791594	0.69559
University Rating	CGPA	SOP	Chance of Admit	TOEFL Score	0.746479	0.734523	0.71125	0.69559
SOP	University Rating	LOR	CGPA	Chance of Admit	0.734523	0.729593	0.718144	0.675732
LOR	SOP	CGPA	Chance of Admit	University Rating	0.729593	0.670211	0.669889	0.660123
CGPA	Chance of Admit	GRE Score	TOEFL Score	University Rating	0.873289	0.83306	0.828417	0.746479
Research	GRE Score	Chance of Admit	CGPA	TOEFL Score	0.580391	0.553202	0.521654	0.489858
Chance of Admit	CGPA	GRE Score	TOEFL Score	University Rating	0.873289	0.80261	0.791594	0.71125

This is justifiable since graduate admissions tend to be based on undergraduate GPA, GRE score and TOEFL score all of which are metrics that allow admission officers to gauge your prior educational performance.

3.2.3 Recursive Feature Elimination (RFE)

The Sklearn *feature_selection* library was used for implementing Recursive Feature Elimination (RFE). The code excerpt below is the implementation of it in Python.

```
#Stores feature ranking from rfe.
rfe_ranking = []
for n in range(5, 7):
```

```
    X_train, X_test, y_train, y_test = train_test_split(features,y,
test_size = 0.3, random_state = 42)
    model = LinearRegression()
    rfe = RFE(model,n) #second arg is number of features to select
    X_train_rfe = rfe.fit_transform(X_train,y_train)
    X_test_rfe = rfe.transform(X_test)
    model.fit(X_train_rfe,y_train)
    score = model.score(X_test_rfe,y_test)
```

The code runs twice, the first run selects five features while the second run selects six features. The results are appended in the rfe_ranking list and are as follows:

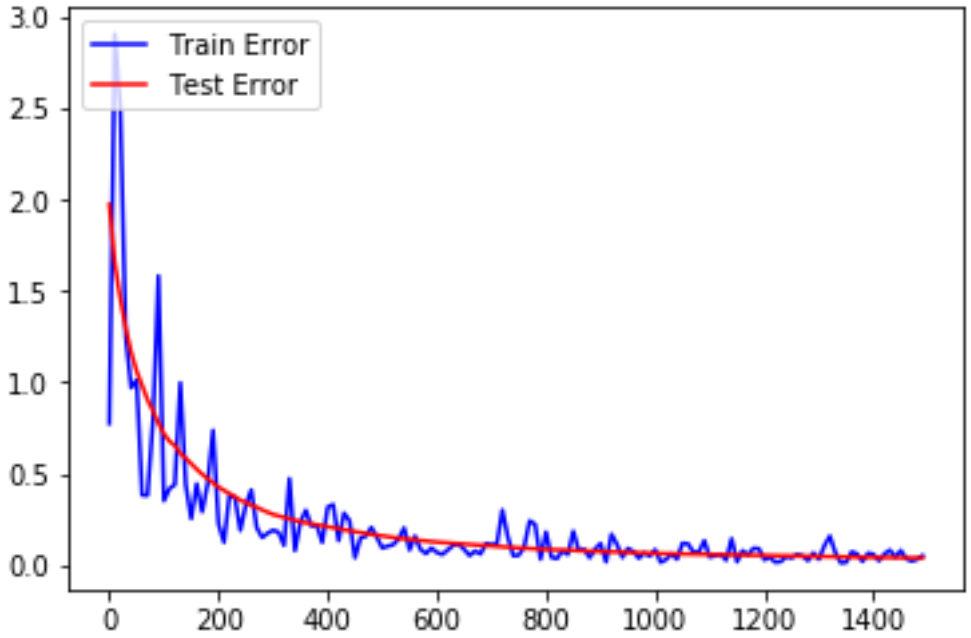
```
Rfe_ranking[0] = [2 1 1 3 1 1 1]

Rfe_ranking[1] = [1 1 1 2 1 1 1]
```

The numbers ranging from 1 to 3 denote the importance of the feature. The index represents the order of features in the dataset – the element in the first index corresponds to GRE Score, followed by TOEFL score so on and so forth.

The values in Rfe_ranking[0] tells us that the GRE Score and Statement of Purpose were not as important as the other features.

By running the neural network with GRE Score and SOP features removed, we get the following results:

Figure 13: RFE 5 features



If however 6 features were selected instead, only the feature SOP is dropped. Figure 14 shows the results. The training error for 6 features is noisier compared to the 5 features model at the beginning but settles at around the 800th epoch.

Figure 14 : RFE 6 features

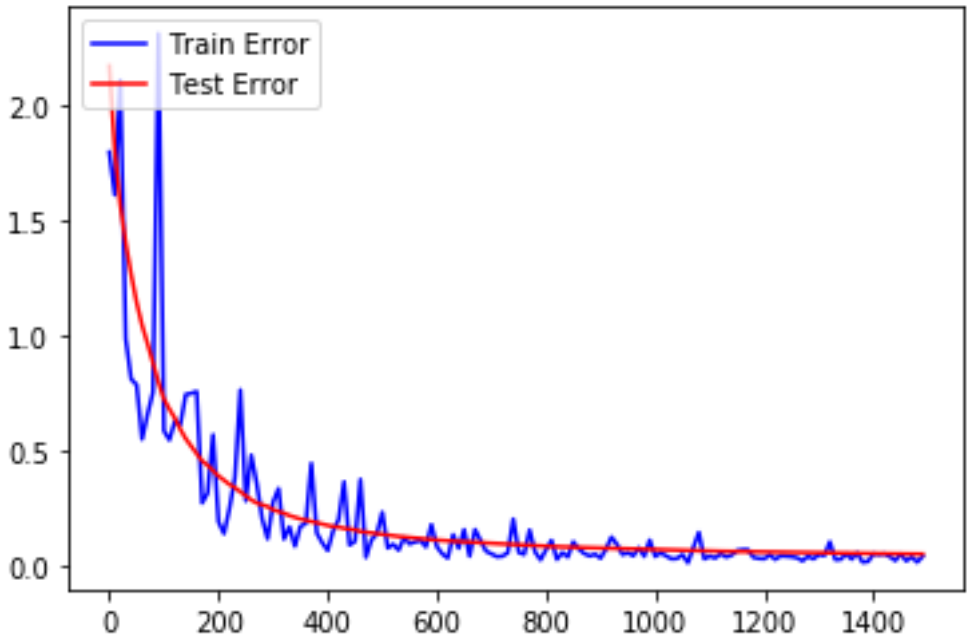
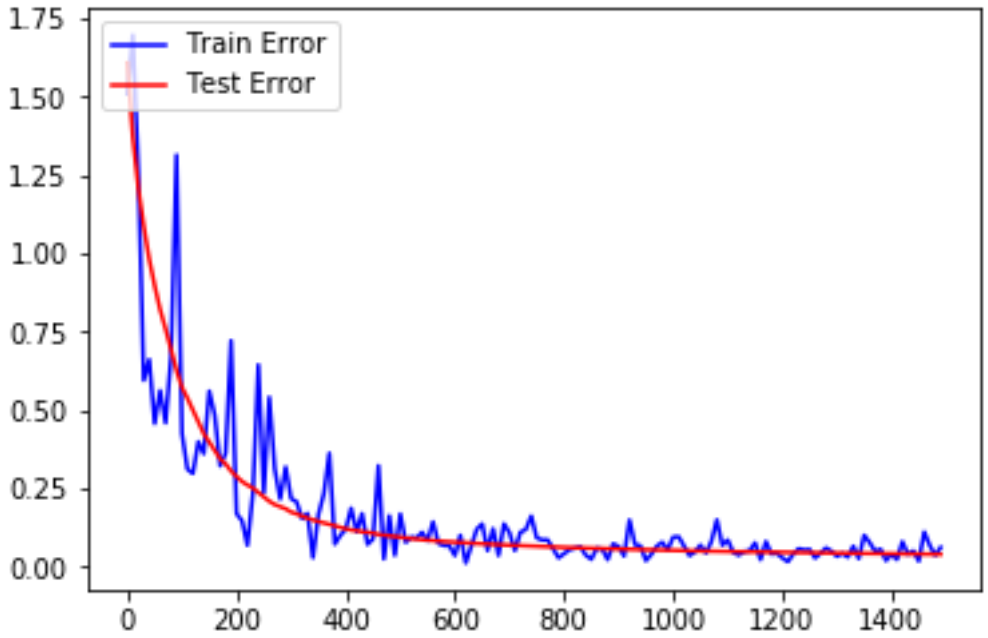


Figure 15: 7 features



Compared with using all 7 features however, using 6 features is significantly noisier for training in later epochs.

3.2.3.1 Accuracy

Figure 16: Predicted vs target (5 Features)

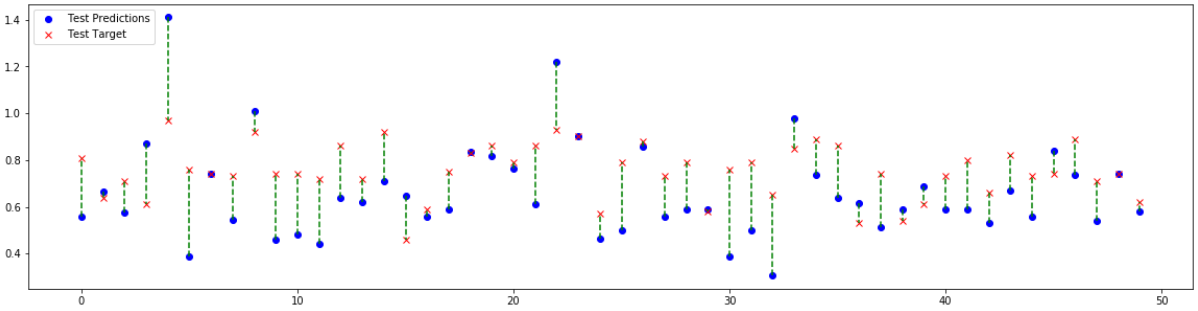


Figure 17: Predicted vs target (6 Features)

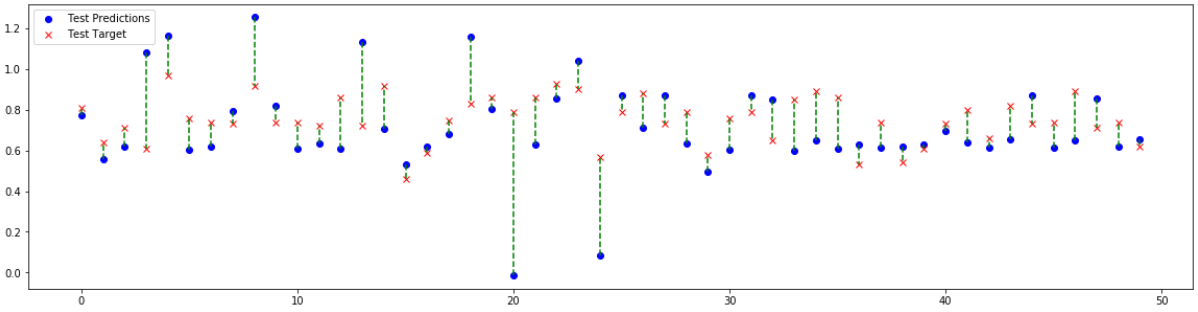
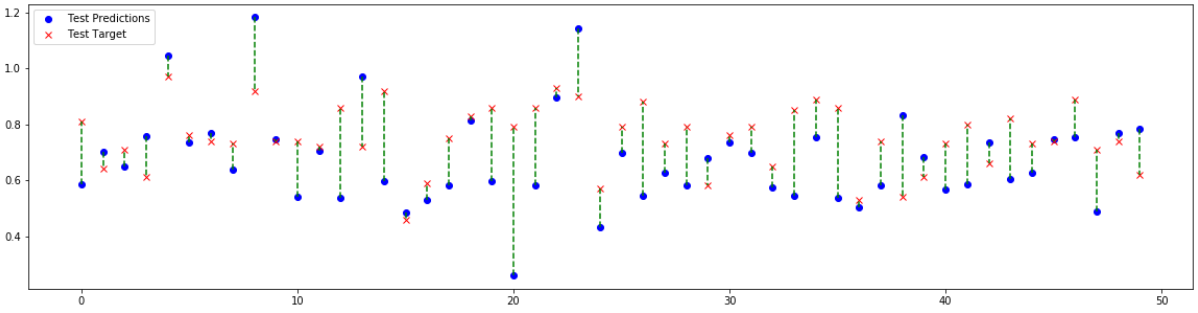


Figure 18: Predicted vs target (7 Features)



Comparing the accuracy, we can see that the model trained from using 5 features is the most inaccurate (Figure 16) compared to Figure 17 and Figure 18. The model trained using only 6 features is more accurate than the one with 7 features albeit only slightly.

3.2.4 4 & 5 Layers Neural Network of 50 Neurons Each

This question requires the implementation of a four and five-layer neural network with the hidden layers having 50 neurons each. The learning rate remains the same at 10^{-3}. Dropout at a 0.2 probability rate is introduced to prevent overfitting as we have a lot of neurons in the hidden layers. Note that we are **using the optimal feature set** selected in Question 3 (6 features, SOP dropped).

Figure 19: 3 layer, 10 neurons

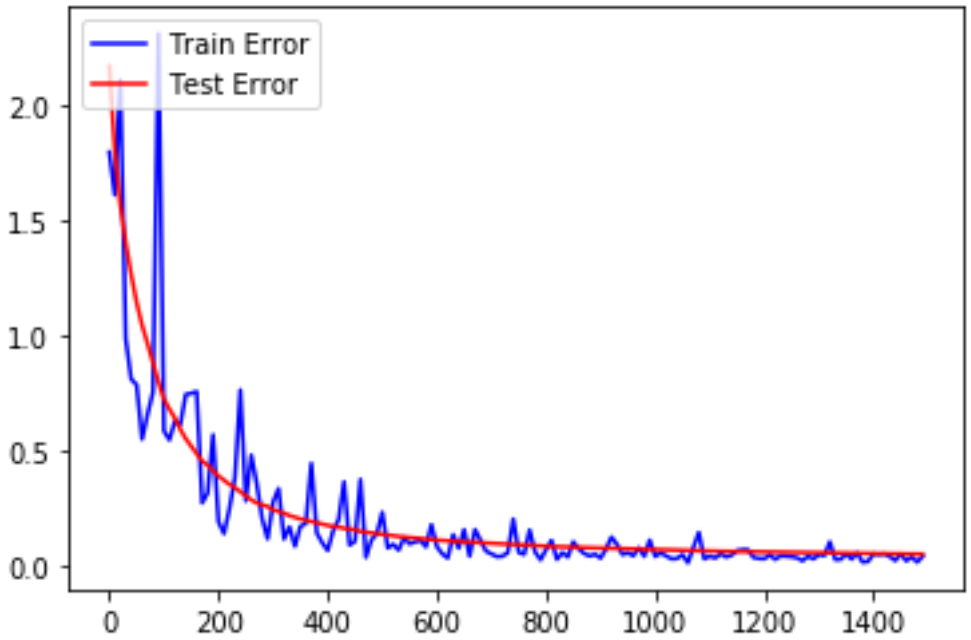


Figure 20: 4 layer, 50 neurons

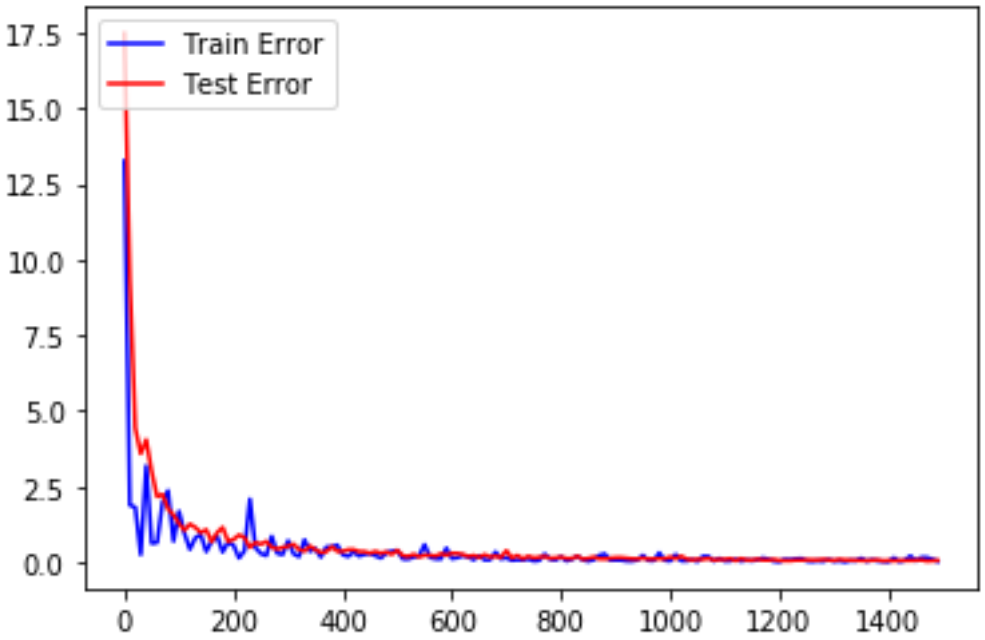
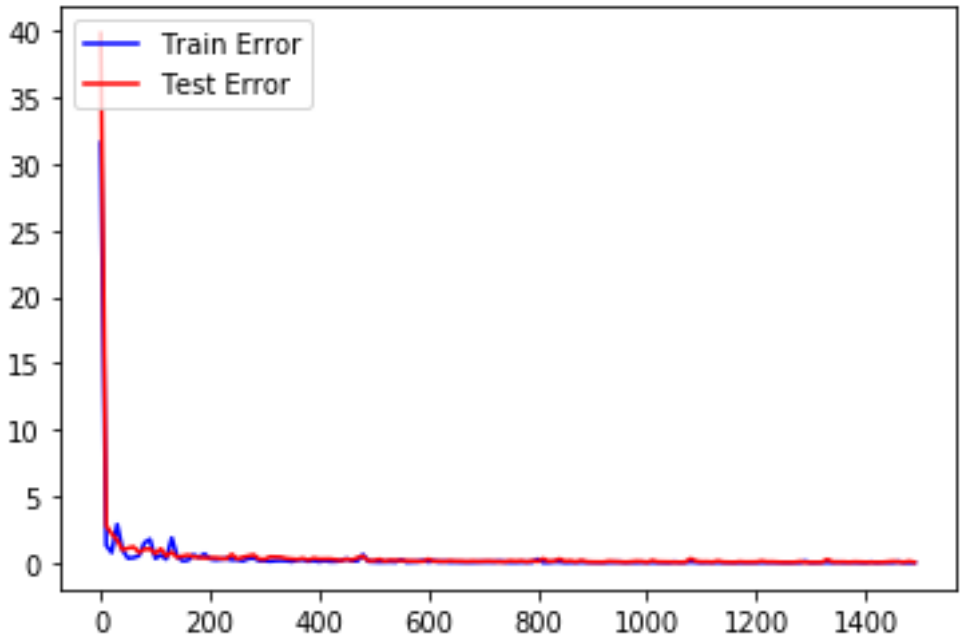


Figure 21: 5 layer, 50 neurons



The three models being compared are as follows:

- 3-layer, 10 neurons in hidden-layer
- 4-layer, 50 neurons in hidden-layer
- 5-layer, 50 neurons in hidden-layer

Moving from the 3-layer model to the 4-layer model, the immediate observation is that the 3-layer model train error is nosier and settles slower than the 4-layer model. While both the 3-layer and 4-layer model test errors converges to roughly the same value, the 4-layer model converges must faster.

The 5-layer network is better than the 3-layer and 4-layer models in terms of train error and test error as it converges at around the first few epochs.

Figure 22: 3-layer, 10 neurons predictions

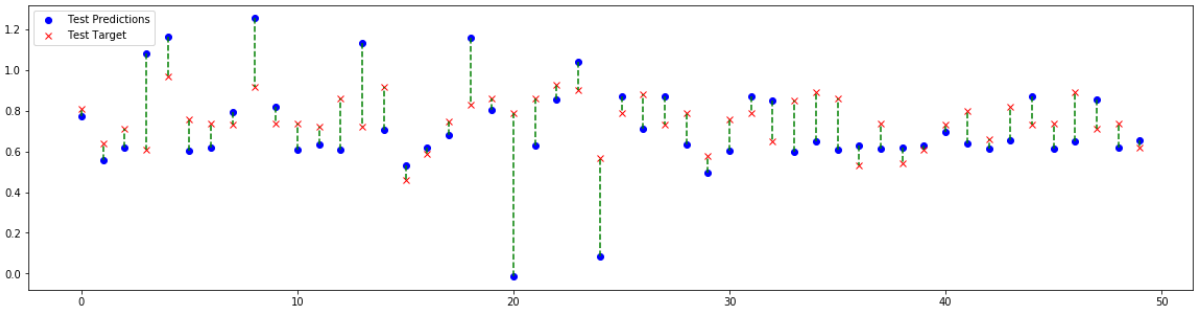


Figure 23: 4-layer, 50 neurons predictions

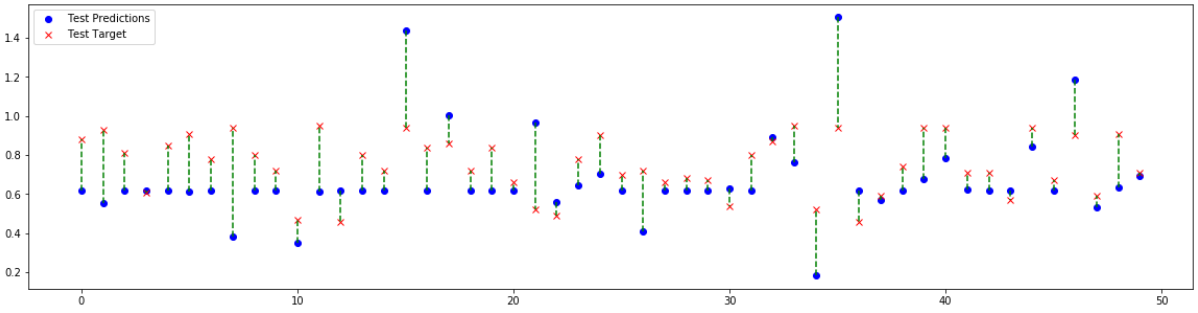
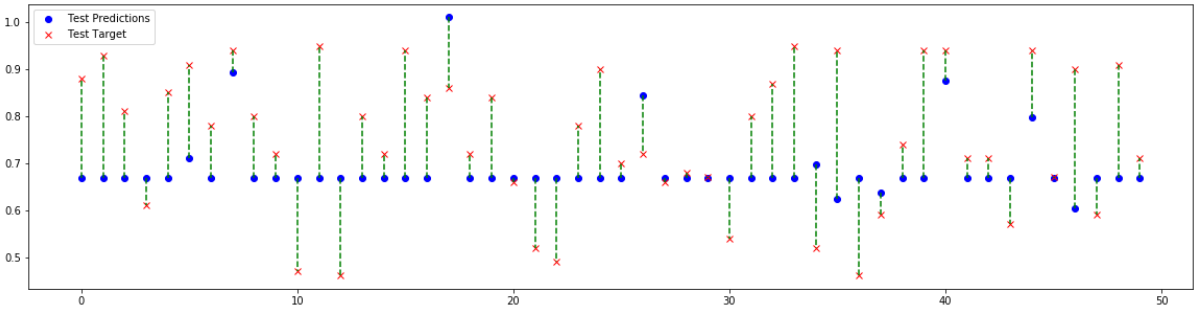


Figure 24: 5-layer, 50 neuron predictions



The predictions however appear to be better for the 3-layer, 10 neuron model. This might be because of overfitting. More hidden layers and neurons make it easier for the model to memorize the training dataset and hence it wasn't making predictions but rather giving out the actual target value corresponding to the input.