NEURAL NETWORKS & DEEP LEARNING

ASSIGNMENT 1

Team Members / Emails:

Ang Yang / U1522821C / yang025@e.ntu.edu.sg

Ng Tze Yang/ U2520987B / tng016@e.ntu.edu.sg

1. Introduction

The goal of project 1 is to help us better understand the structure of a neural network and how the different components of a model can affect the results. Based on these results, we can then select a optimal design for the model.

Part A of project 1 deals with choosing optimal batch sizes, number of hidden neurons, decay parameter and number of layers for a classification problem based on returned results from trying out different variations of each component.

Part B of project 1 deals with choosing optimal learning rate, number of hidden neurons and layers for a regression problem based on the validation results using 5-fold cross validation.

2. Methods

There are many methods adopted in this project, the following will be a discussion of the methods used.

2.1 L2 regularization

To prevent overfitting, we use L2 regularization, which penalizes the weight weight matrices of the nodes in a neural network by using the following equation.

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

Figure 1: Cost function for L2 regularization

Lamba here is the regularization parameter which is the parameter which is optimized for better results. By applying L2 regularization on our cost function, penalise the model for having high weight values and prevents the model from overfitting the training data.

2.2 Batch Size

In both questions, we utilised a mini-batch gradient descent methodology when training the network. The training data set is first randomly shuffled at the start of every epoch and broken down into batches of the desired batch size. The model was then given batches sequentially to be trained and the whole process was repeated for the next epoch.

2.3 Softmax layer

A typical logistic regression produces a decimal between 0 and 1.0, representing the percentage chance of a result matching a label. The softmax function extends this concept into the multi-class world by assigning decimal probabilities to each class in a multi-class problem, where the result adds up to 1. This additional constraint helps the training to converge quicker than it originally would. The softmax layer is usually implemented just before the output layer.

2.4 K-fold cross-validation

K-fold cross-validation is a resampling procedure used to evaluate the skill of a machine learning model on unseen data. It uses a limited sample to estimate how the model is expected to perform in general when used to make predictions on data not used during the actual training of the model. This can thus improve neural network accuracy.

2.5 Feature Scaling

To improve the accuracy of the model, we scaled the inputs to the model using the max-min normalisation method in Part A and with the standardisation method in Part B.

2.6 Weights and Bias Initialisation

Weights were initialised using the Truncated Normal method and biases were initialised to zero for both parts.

2.7 Activation Function

In part A, we used the sigmoid function as the activation function to the perceptron. In part B, we used the ReLu function instead as instructed.

3. Experiments and results

The following is a simple presentation of our experiment and results for each question and its parts.

3.1 Part A Overview

In this problem, we are given a dataset containing multispectral values of pixels in a 3x3 neighbourhoods in satellite images and are tasked to train a neural net to classify images into 7 different classes.

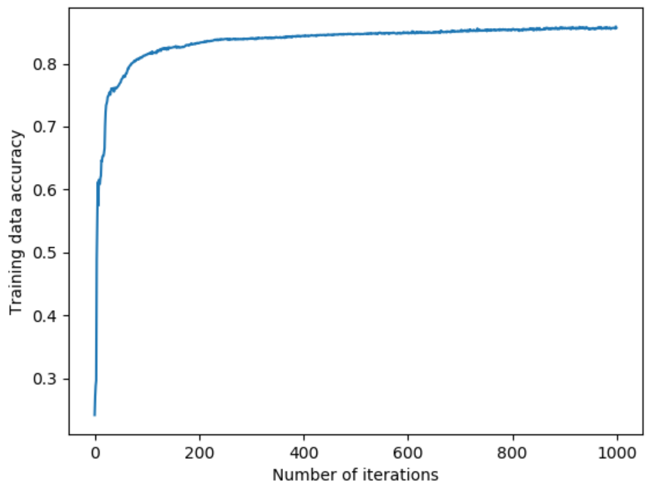
The first step of the program deals with preprocessing the data which includes extracting the data from a text file and modifying the classification column of the data into a one-hot matrix to be used in the model.

Next, we initialize the weights using the truncated normal method as explained previously and set the biases to zero. For the activation function, we used the sigmoid function as we wanted the hidden layer to be a perceptron layer. We calculated the loss based on a softmax cross entropy with L2 regularisation.

Finally, we trained the model with the full data set consisting of 4435 training samples and 2000 test samples using a mini-batch gradient descent.

3.2 Part A Question 1

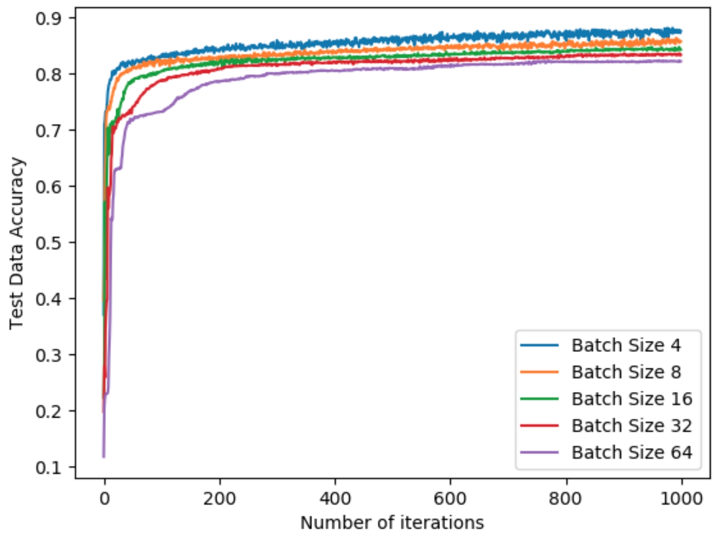
We designed a 3-layer feedforward neural network with the instructed parameters and trained the model. With it, we achieved a training accuracy rate of **85.11%.**



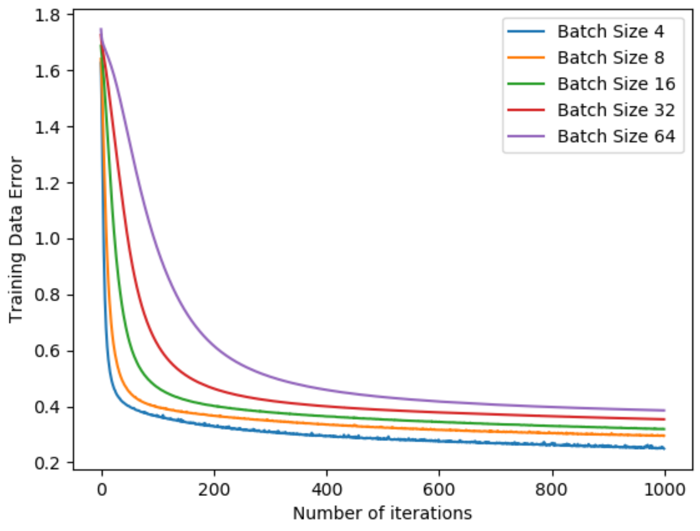
Training accuracy for part A question 1

3.3 Part A Question 2

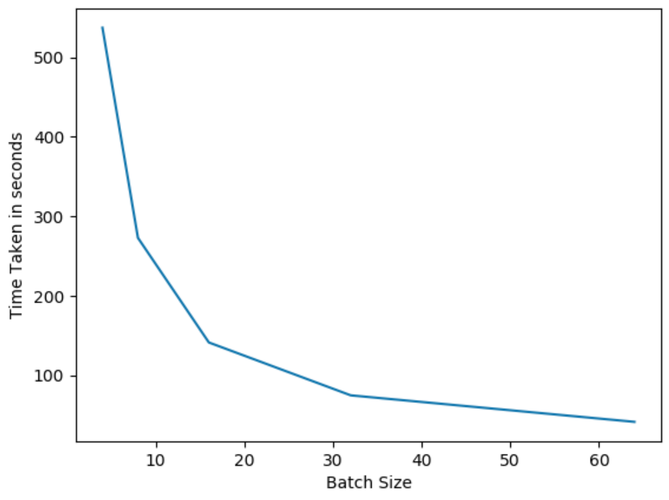
Next, we experimented with the batch size parameter. By altering the batch size, we measured the training errors and test accuracies against the number of epochs in figure 3 and 4. We also measured the time taken for the program to complete training for each batch size and plotted it in figure 5. The summary of finalised errors, accuracies and time taken is summarized in table 1.



Test accuracy for varying batch sizes



Training error for varying batch sizes



Time taken for training for varying batch sizes

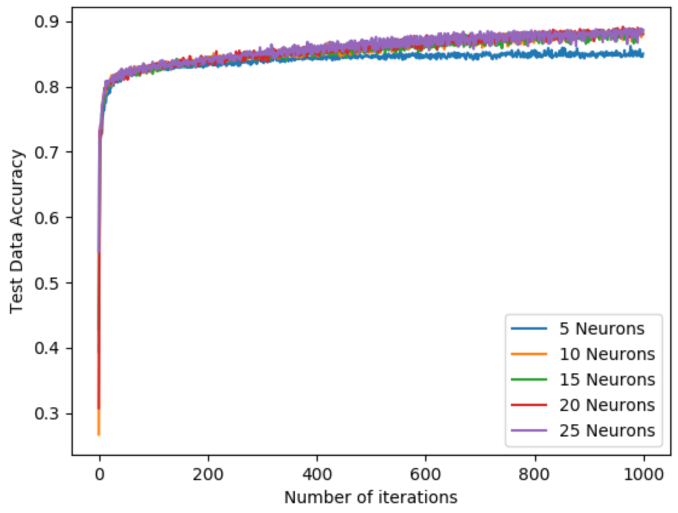
Batch Size	**Final Test Accuracy**	**Final train error**	**Time Taken (s)**
4	0.8765	0.249198	510
8	0.858	0.295312	267
16	0.8435	0.319084	135
32	0.834	0.353962	81
64	0.8225	0.3857	33

Based on the findings, we concluded that a batch size of 4 would be the optimal size for the training of the data set. This is a batch size of 4 yields the highest accuracy in comparison with all other sizes. Although it is the most costly in terms of timing, the total time taken to train the network on this data set is still within a reasonable timing of approximately 8 minutes. This also takes into consideration the fact that the training was not done using a GPU which would have significantly improved performance.

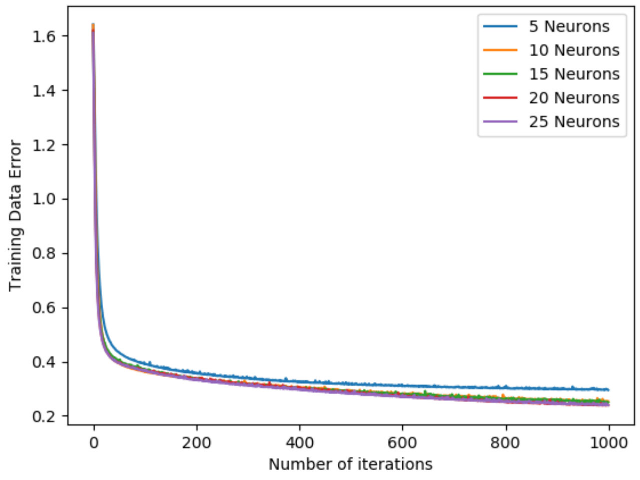
Optimal batch size = 4

3.4 Part A Question 3

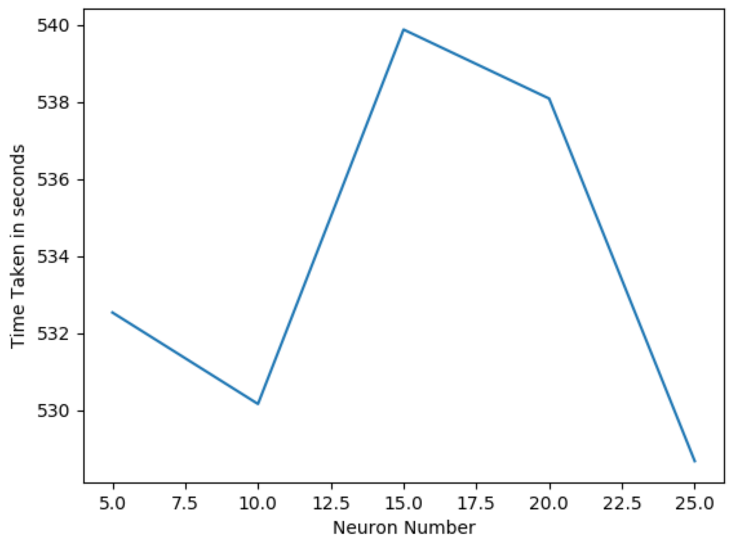
With batch size set to 4, we next experimented with the number of neurons in the hidden layer. Similar to question 2, we measured the training errors and test accuracies against the number of epochs in figure 6 and 7. We also measured the time taken for the program to complete for each batch size and plotted it in figure 8. The summary of finalised errors, accuracies and time taken is summarized in table 2.



Test accuracy against varying number of neurons in hidden layer



Training error against varying number of neurons in hidden layer



Time taken for training for varying batch sizes

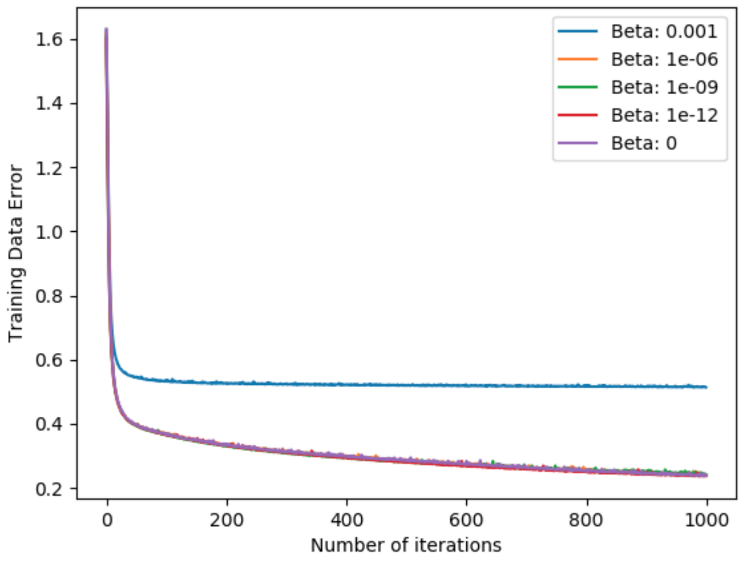
Num of Neurons	Final Test Accuracy	Final Train Error	Time Taken(s)
5	0.8505	0.294578	532.535
10	0.8825	0.253068	530.163
15	0.886	0.249201	539.876
20	0.88	0.239082	538.086
25	0.8865	0.238922	528.68

Based on the results, all runs yielded similar results in terms of time taken and final test accuracy. Therefore, we chose 25 to be optimal number of neurons as it yielded the highest accuracy among all other runs.

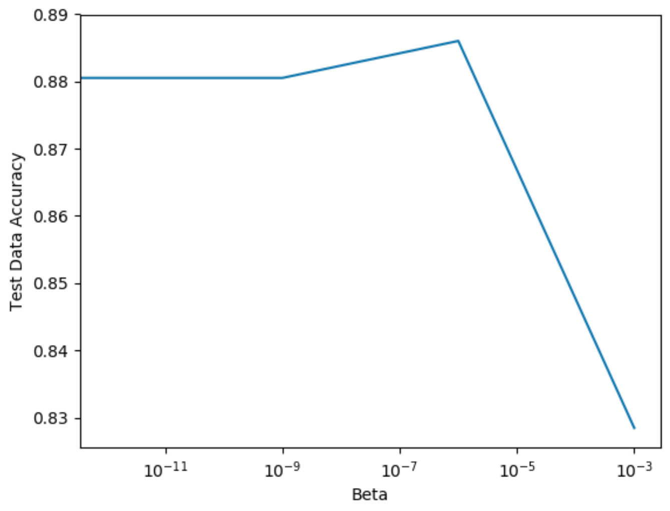
Optimal number of neurons = 25

3.5 Part A Question 4

After setting number of neurons to 25, we next altered the lambda for L2 regularisation. Similar to previous questions, we measured the training errors and test accuracies against the number of epochs in figure 9 and 10. The summary of finalised errors, accuracies and time taken is summarized in table 3.



Test accuracy against varying number of neurons in hidden layer



Training error against varying number of neurons in hidden layer

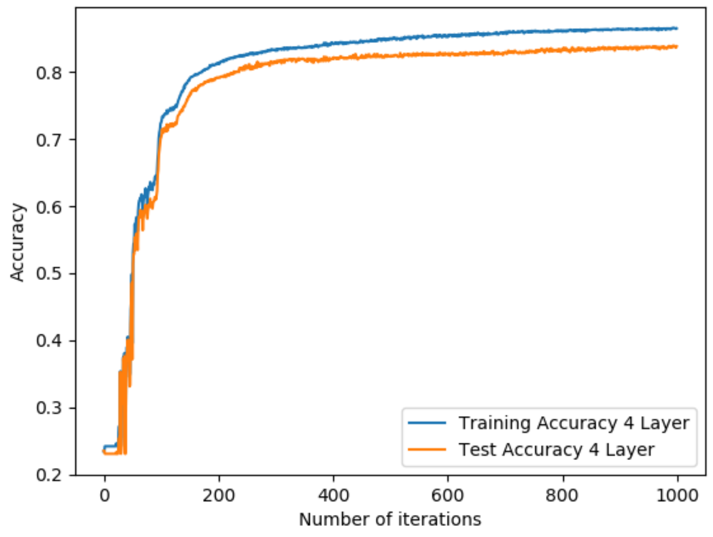
Lambda	**final test accuracy**	**final train error**
0	0.887	0.238747
10e-12	0.8805	0.237863
10e-9	0.8805	0.24255
10e-6	0.886	0.239803
10e-3	0.8285	0.513777

Based on the findings, we concluded that using a lambda of 10e-6 was the most optimal choice as it provided the best accuracy and lowest training error.

Optimal lambda = 10e-6

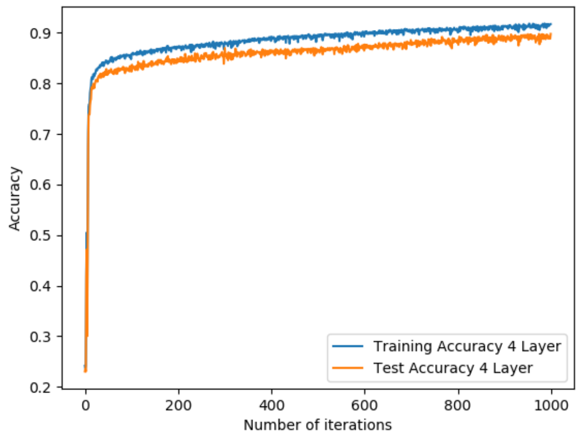
3.6 Part A Question 5

In this question, we designed a 4-layer network with two hidden perceptron layers with 10 neurons each. This network was trained with a batch size of 32 and decay parameter 10e-6. In this experiment, we achieved a final test accuracy of 0.8385 and training accuracy of 0.865163.



Test and Training error for 4 layer neural network

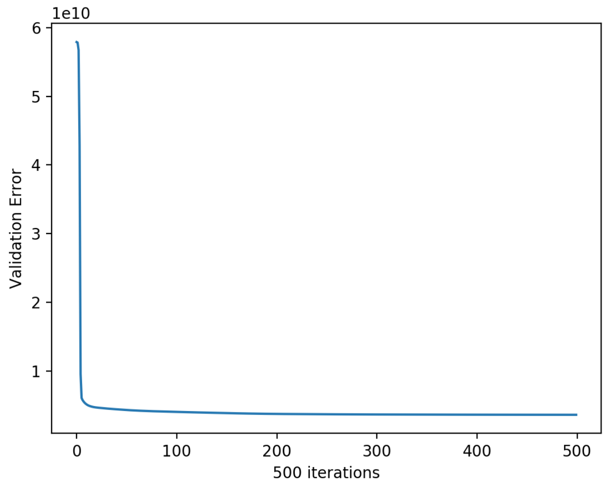
We should not that the test accuracy of the 4 layer neural network is slightly less accurate than that of the 3 layer network we managed to find in Question 4. This is because the training parameters such as number of neurons, value of lambda and batch size have not been optimised yet. To show this, we conducted another experiment with a 4 layer network using the optimal values we have found in previous questions and achieved a test accuracy of 0.898 and training accuracy of 0.91745.



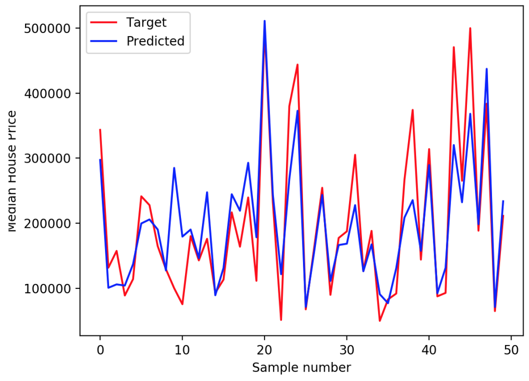
Test and Training error for 4 layer neural network with optimised parameters

3.7 Part B Question 1

For Question 1, we designed a 3 layer feedforward neural network with the specified parameters and received the following results.



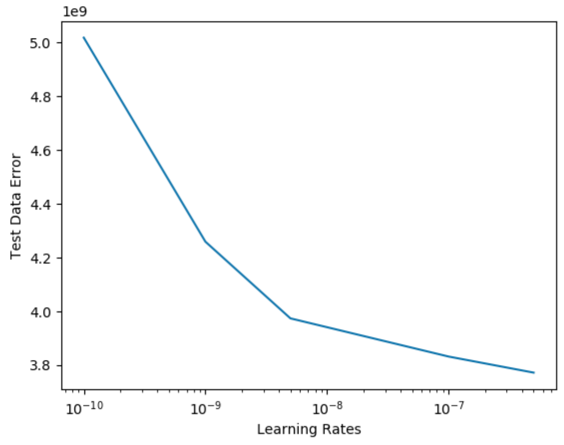
Plot of validation errors against epochs



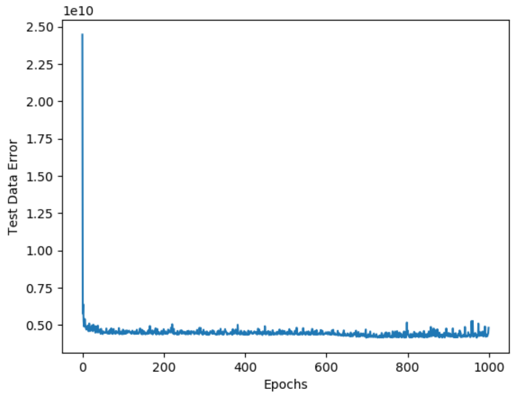
Plot of predicted values and target values for 50 test samples

3.8 Part B Question 2

For question 2, using 5-fold cross validation on validation data, we searched for the optimal learning rate for our designed neural network. The results and conclusion are obtained below.



Plot of cross-validation errors achieved on different learning rates



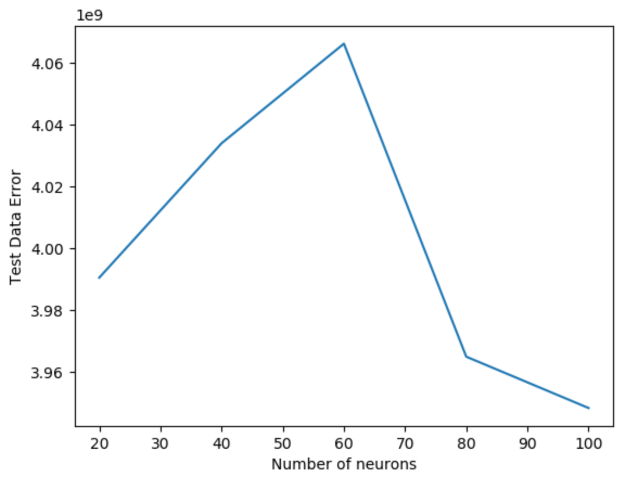
Plot of test errors against training epochs for chosen learning rate

As seen from the first graph, the test data errors decrease as the learning rate increases. We thus selected the learning rate of 0.5x10e-6 as our optimal learning rate.

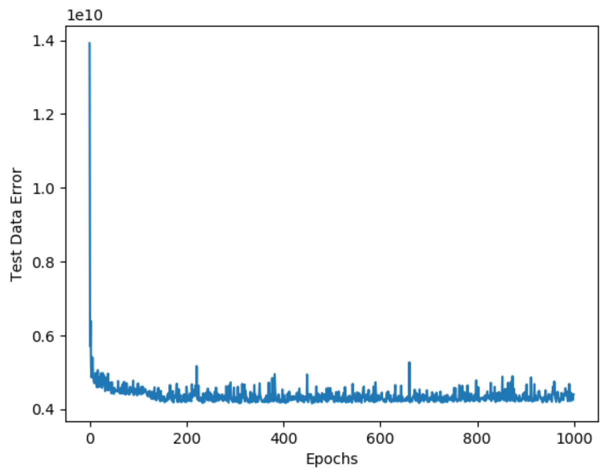
Optimal Learning Rate: 0.5x10e-6

3.9 Part B Question 3

For this question, we were tasked with finding the optimal number of hidden neurons for our designed neural network. Results are as follows.



Plot of cross-validation errors against number of neurons



Plot of test errors against number of epochs

Based on the above graphs, 100 neurons for the hidden layer achieved the lowest test data error. We thus selected 100 neurons as the optimal number of neurons for our hidden layer. However, it is also good to note that having a higher number of hidden neurons may not always be the best choice as it may result in overfitting.

Optimal number of neurons for hidden layer: 100

3.10 Part B Question 4

For the last question in part B, we designed neural networks with the specifications as defined by the question. However due to data overflow we had to scale the error output by a factor of 10^-6 (i.e. Each Y value scaled down by 1000).

The data we obtained for each neural network is shown in the table below.

Number of Layers (with/without dropout)	Validation Error
4 Layers	4149.88
4 Layers without dropout	3360.34
5 Layers	3804.76
5 Layer without dropout	2950.42
3 Layers	4414.52

As we can see from the above table, networks with more layers generally performed better than networks with less layers. Although not always the case, more layers gives the model more room to abstract more complex functions, thus improving the accuracy of the model.

We may also see that for both 4 layer and 5 layer networks, networks with dropouts performed better than networks without dropouts included.

4. Conclusions

The ability of a Neural Network model to perform well is heavily dependant on the hyperparameters as it controls the behaviour of the training algorithm. With validation methods and testing we can thus tune the hyperparameters to achieve a better overall performance.

Through this project we thus learnt the importance of hyperparameters and observing how they affected the model by isolating and tuning each hyperparameter accordingly. Through these methods and plotting the results out on graphs for better visualisation thus aided us in choosing appropriate hyperparameters for our model.