

**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**  

---

**SINGAPORE**

**CZ4042  
NEURAL NETWORK AND DEEP LEARNING  
PROJECT 1**

<b>NAME</b>	<b>MATRIC NUMBER</b>
BRYAN LEOW XUAN ZHEN	U1721837L
WANG XIAOYU	U1721992H

## Table of Contents

<b>Part A: Classification Problem</b> .....	3
<b>Introduction</b> .....	3
<b>Method</b> .....	3
Train-test split .....	3
5-fold cross validation .....	4
Input feature scaling.....	4
Model building.....	4
<b>Experiments and Results</b> .....	5
Question 1 .....	5
Question 2 .....	7
Question 3 .....	13
Question 4 .....	17
Question 5 .....	23
<b>Conclusion</b> .....	25
Answers to the queries.....	25
Summary of findings .....	25
Experience of experiments.....	25
<b>Part B: Regression Problem</b> .....	26
<b>Introduction</b> .....	26
<b>Method</b> .....	26
Train-test split .....	26
Input feature scaling.....	27
Model building.....	27
<b>Experiments and Results</b> .....	28
Question 1 .....	28
Question 2 .....	29
Question 3 .....	30
Question 4 .....	34
<b>Conclusion</b> .....	35
Answers to the queries.....	35
Summary of findings .....	36

Experience of experiments.....36

# Part A: Classification Problem

## Introduction

In this project, we are required to build a 3-layer feedforward neural network consisting of an input layer, a hidden layer with ReLu activation function and an output SoftMax layer. The neural network built will then be used for the classification of the Cardiotocography dataset on 2126 cardiotocograms. The aim is to accurately predict the NSP labels of the test dataset after training the neural network on the training dataset.

## Method

### Train-test split

To conduct model selection and true error estimation at the same time, we have to make use of Three-Way Data Splits method to split the data into 3 subsets: training set, validation set and test set. The training set is used to train the model and learn the parameters; the validation set is used to compare the error in order to select the best model; the test set is used to estimate the true error and assess the performance of the model objectively.

To do so, we first shuffle the original data, because patterns of the same class are likely clustered together in the dataset. After that, we take the first 70% of the data as the training set, and the rest as the test set.

```
6 # Define the size of training set and data set
7 dataset_size = len(dataX)
8 train_size = int(0.7 * dataset_size)
9 test_size = dataset_size - train_size
10
11 # Shuffle the data before splitting to train test set
12 # Data of same class are usually clustered together in the dataset, we want to shuffle them
13 idx = np.arange(dataset_size)
14 np.random.shuffle(idx)
15 dataX = dataX[idx]
16 data_Y = data_Y[idx]
17 dataY = np.zeros((data_Y.shape[0], NUM_CLASSES))
18 dataY[np.arange(data_Y.shape[0]), data_Y-1] = 1 #one hot matrix
19
20 # Split the training and testing set
21 trainX = dataX[:train_size]
22 trainY = dataY[:train_size]
23
24 testX = dataX[train_size:]
25 testY = dataY[train_size:]
```

### *Extra analysis*

The above method implemented is **random subsampling**, whereby we randomly sample data for training and testing. To improve the performance of the model, we may consider **stratified sampling** method, which ensures that patterns from different classes are represented in a similar distribution as the entire population. This could avoid the situation when random sampling results in data of one particular class to be under-represented (or in the worst case, not present at all) in the training set. As such, test error may be improved with stratified sampling.

## 5-fold cross validation

We employed the 5-fold cross validation for the training set. If we consider from the point of view of the Three-Way Data Splits, this ensures that all the examples in the training dataset are eventually used for both training and validation. The folds for training in cross validation will then serve as examples for learning to find the “optimal” weights with the gradient descent rule, while the fold for validation will allow us to determine the error of different models. By comparing the accuracy of the 5-fold cross validation of the different models (in this case different feedforward neural networks), we can select the optimal model by choosing the one with the highest cross-validation accuracy. The selected model hyper-parameters will then be used to train the entire training set (folds for training + fold for validation) to find the new “optimal” weights and predict on the test set for evaluation.

## Input feature scaling

We employed feature-wise Min-Max Normalisation to scale the input features. This is to avoid the situation where one (or a few) feature has a dominant effect on the learning, while weights for other features are not well-learned by the network.

## Model building

We started with a 3-layer feedforward neural network using the TensorFlow package in Python. The network consists of an input layer of 21 nodes, a hidden layer of 10 neurons, and an output layer of 1 neuron.

The input layer takes in a batch of vectors of dimension 21, because the number of input features (`NUM_FEATURES`) for each pattern is 21. The hidden layer consists of  $n$  neurons, where the value of  $n$  (`hidden_units`) is initiated to 10. The output layer returns a batch of vectors of dimension 3 within range [0, 1], indicating that the probability of this pattern falling in the 3 categories (N, S, P) respectively. The output factor determines the classification result - the class with the greatest probability among the 3 values will be the predicted class by the model.

As this is a classification problem, the cost function is defined as the sum of the multi-class cross entropy. and the accuracy is defined by the percentage correct prediction. The updating of parameters is handled by the in-built tensorflow function `tf.train.GradientDescentOptimizer` to minimise the cost function.

The parameters in the model are initialised to the following:

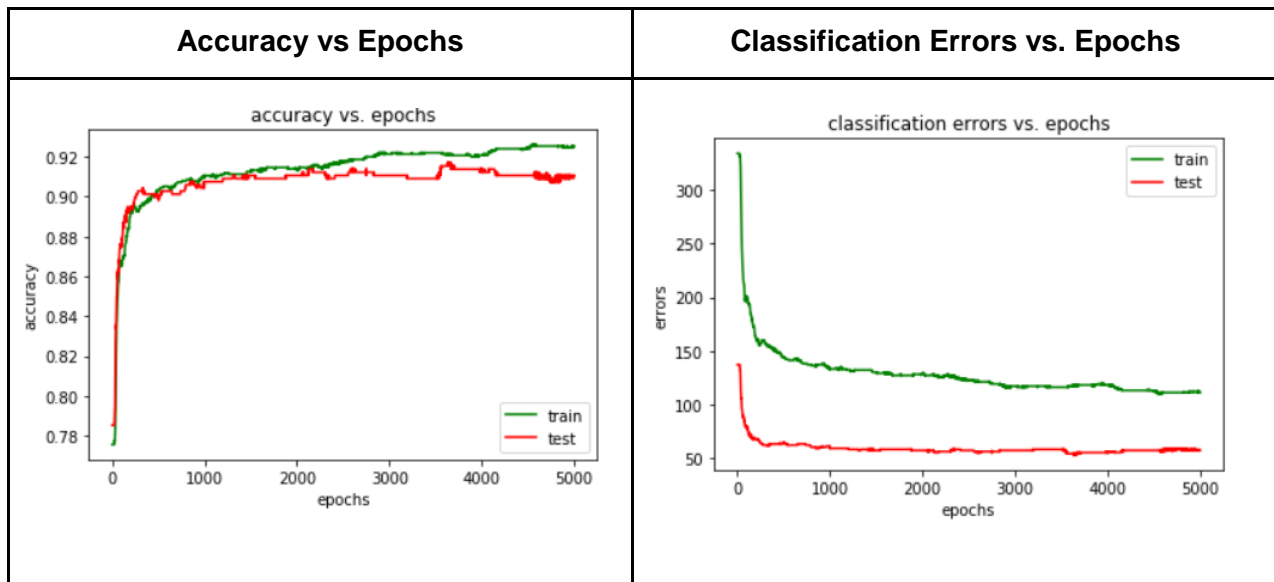
- `NUM_FEATURES = 21`
- `NUM_CLASSES = 3`
- `hidden_units = 10`
- `learning_rate = 0.01`
- `beta = 10**-6`
- `batch_size = 32`
- `epochs = 5000`

To assess the performance during the training procedure, we plot the training accuracy, test accuracy, training errors and test errors.

## Experiments and Results

### Question 1

- a) Use the training dataset to train the model and plot accuracies on training and testing data against epochs.**
- b) State the approximate number of epochs where the test error converges**



As shown above, the graph on the left shows the training and testing accuracies against epochs, while the graph on the right shows the training and testing errors against epochs.

From the graph on the right, it is difficult to assess where the test error converges, as such a log is used to keep track of the training and testing accuracies and errors for every 1000 epochs.

```

iter 1: train accuracy 0.7755376100540161
iter 1: test accuracy 0.7852664589881897
iter 1: train error 334
iter 1: test error 137

iter 1000: train accuracy 0.9106183052062988
iter 1000: test accuracy 0.907523512840271
iter 1000: train error 133
iter 1000: test error 59

iter 2000: train accuracy 0.9139785170555115
iter 2000: test accuracy 0.9106582999229431
iter 2000: train error 128
iter 2000: test error 57

iter 3000: train accuracy 0.9213709831237793
iter 3000: test accuracy 0.9106582999229431
iter 3000: train error 117
iter 3000: test error 57

iter 4000: train accuracy 0.9206989407539368
iter 4000: test accuracy 0.9122257232666016
iter 4000: train error 118
iter 4000: test error 56

iter 5000: train accuracy 0.9254032373428345
iter 5000: test accuracy 0.9106582999229431
iter 5000: train error 111
iter 5000: test error 57

```

We can see that at iteration 4000, the test error is lower than that of iteration 1, 1000, 2000, 3000 or 5000. Hence, the approximate number of epochs where the test error converges will be **4000**.

## Question 2

- a) **Plot cross-validation accuracies against the number of epochs for different batch sizes. Limit search space to batch sizes {4, 8, 16, 32, 64}. Plot the time taken to train the network for one epoch against different batch sizes**

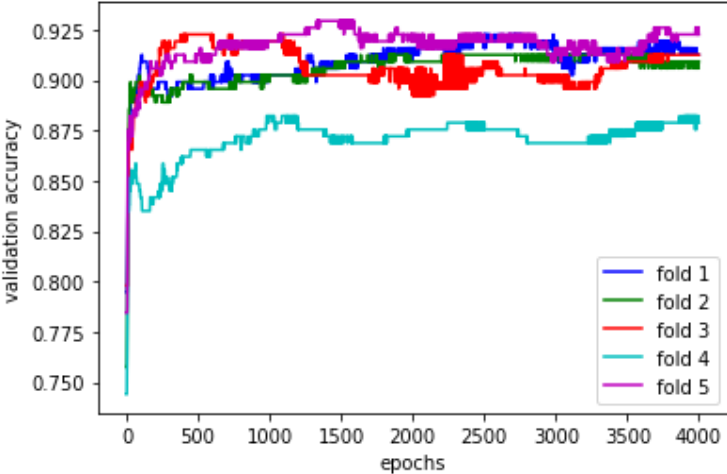
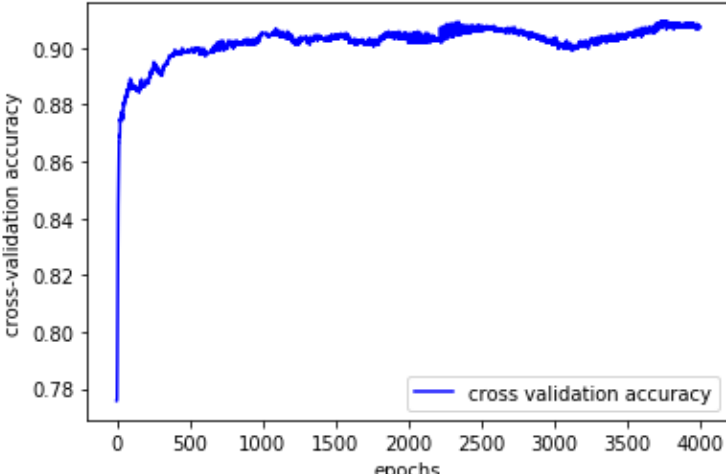
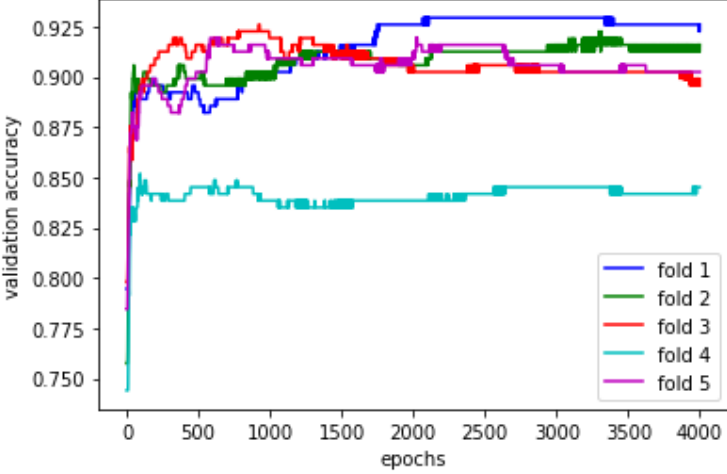
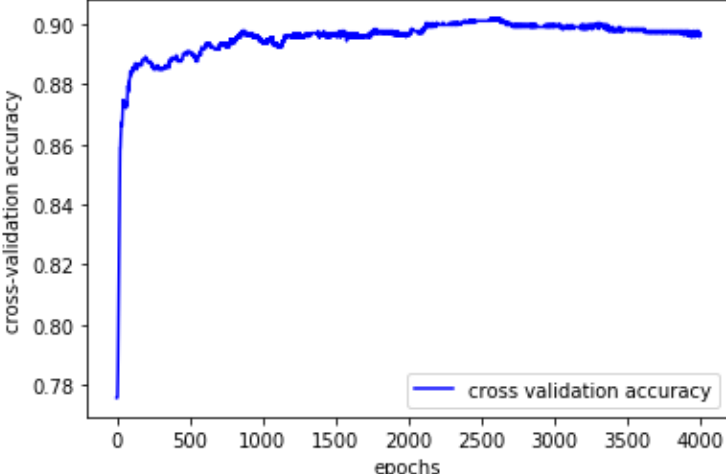
### *Extra analysis*

As shown below, in our experiment, we have plotted the validation accuracy for each fold against epochs on the left and the cross-validation accuracy against epochs on the right. This is done for each of the batch size.

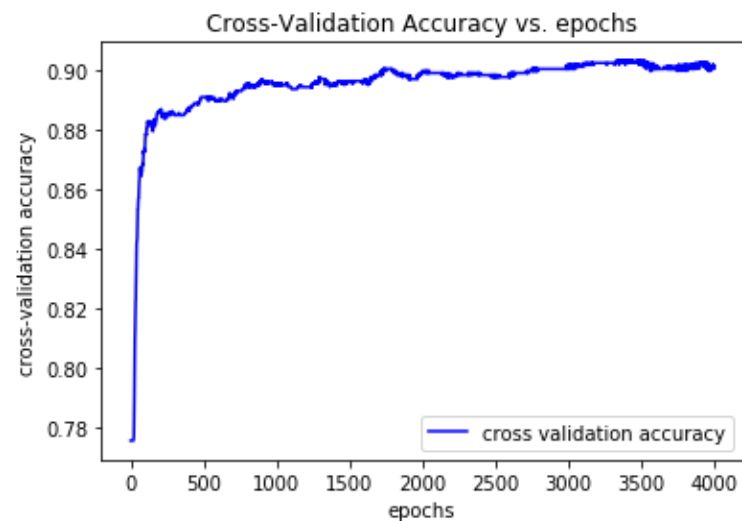
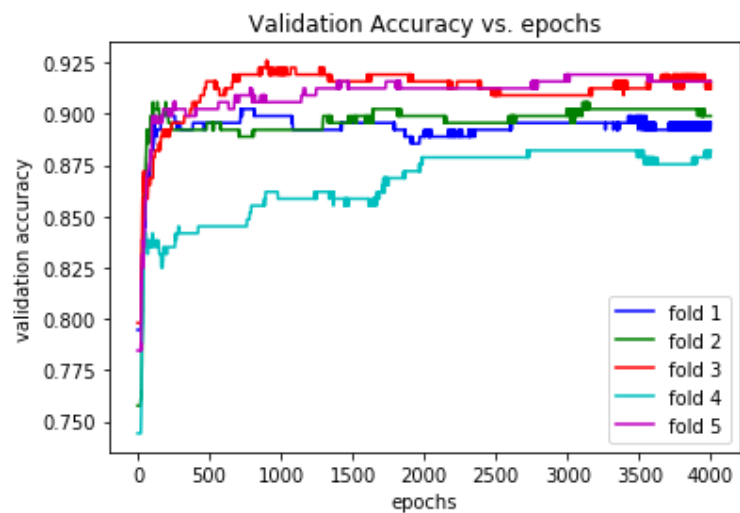
The graph on the left allows us to gain some insights on the data. For example, we see that fold 4 has a lower validation accuracy compared to other folds, perhaps due to reasons such as biases in data in the 4th fold, whereas the data in the other folds are more balanced.

The graph on the right can be derived by the graph from the left since cross-validation accuracy is by definition, an average of the validation accuracy for each of the folds.

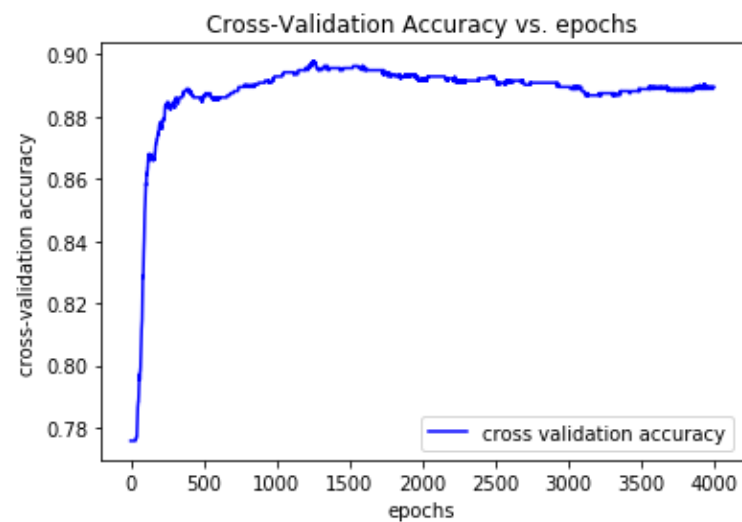
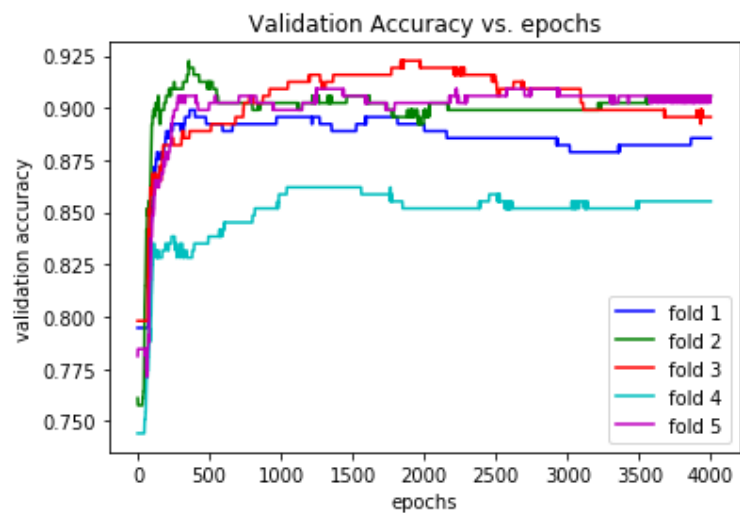


Batch Size	Validation Accuracy vs. Epochs	Cross-Validation Accuracy vs. Epochs
4	<p data-bbox="625 284 982 311">Validation Accuracy vs. epochs</p> 	<p data-bbox="1348 284 1780 311">Cross-Validation Accuracy vs. epochs</p> 
8	<p data-bbox="625 852 982 880">Validation Accuracy vs. epochs</p> 	<p data-bbox="1348 852 1780 880">Cross-Validation Accuracy vs. epochs</p> 

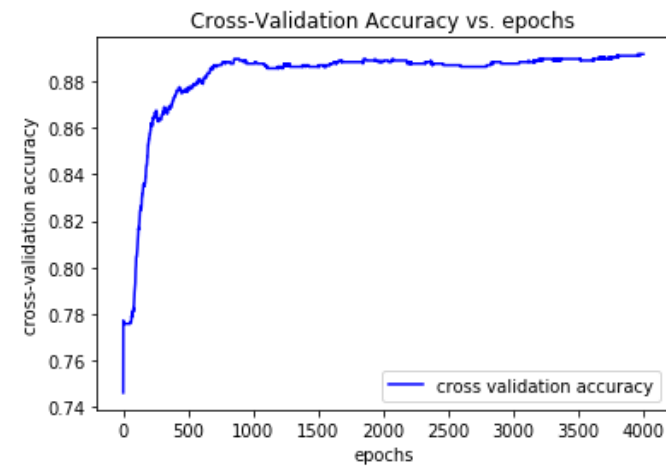
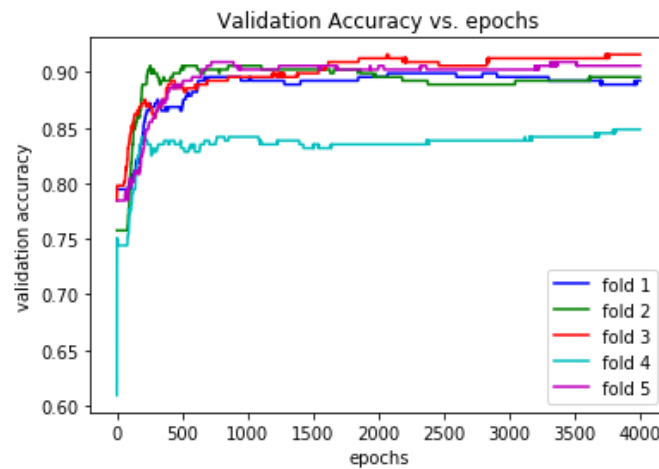
16



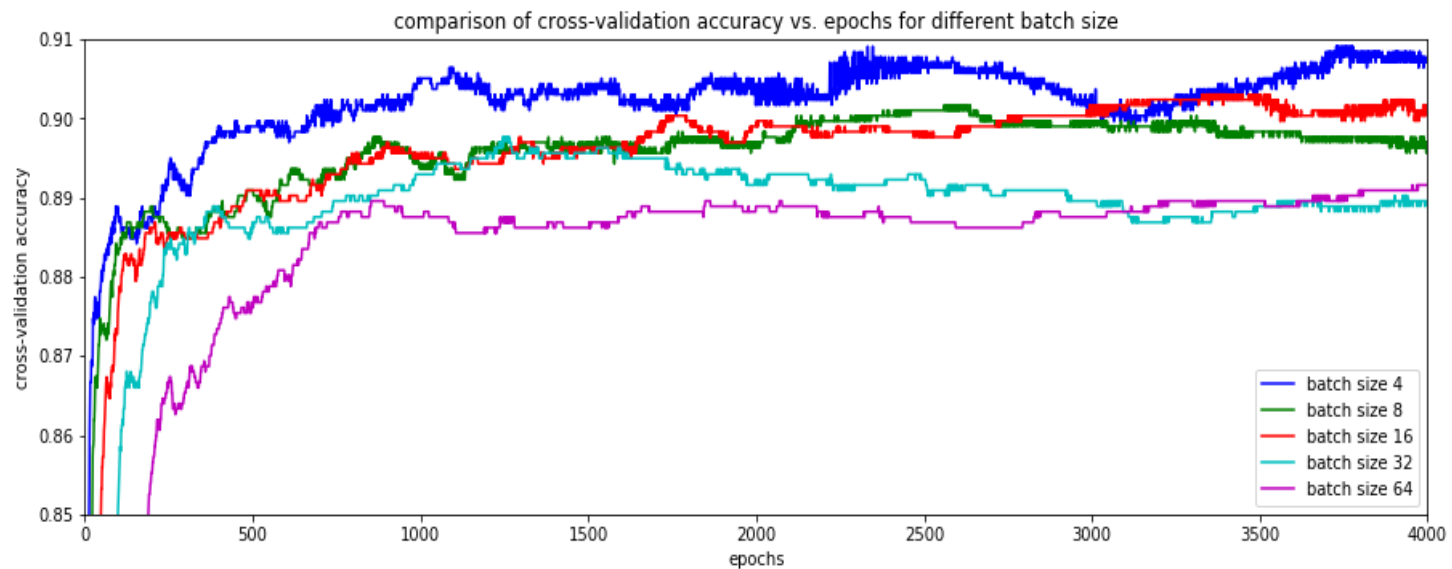
32



64



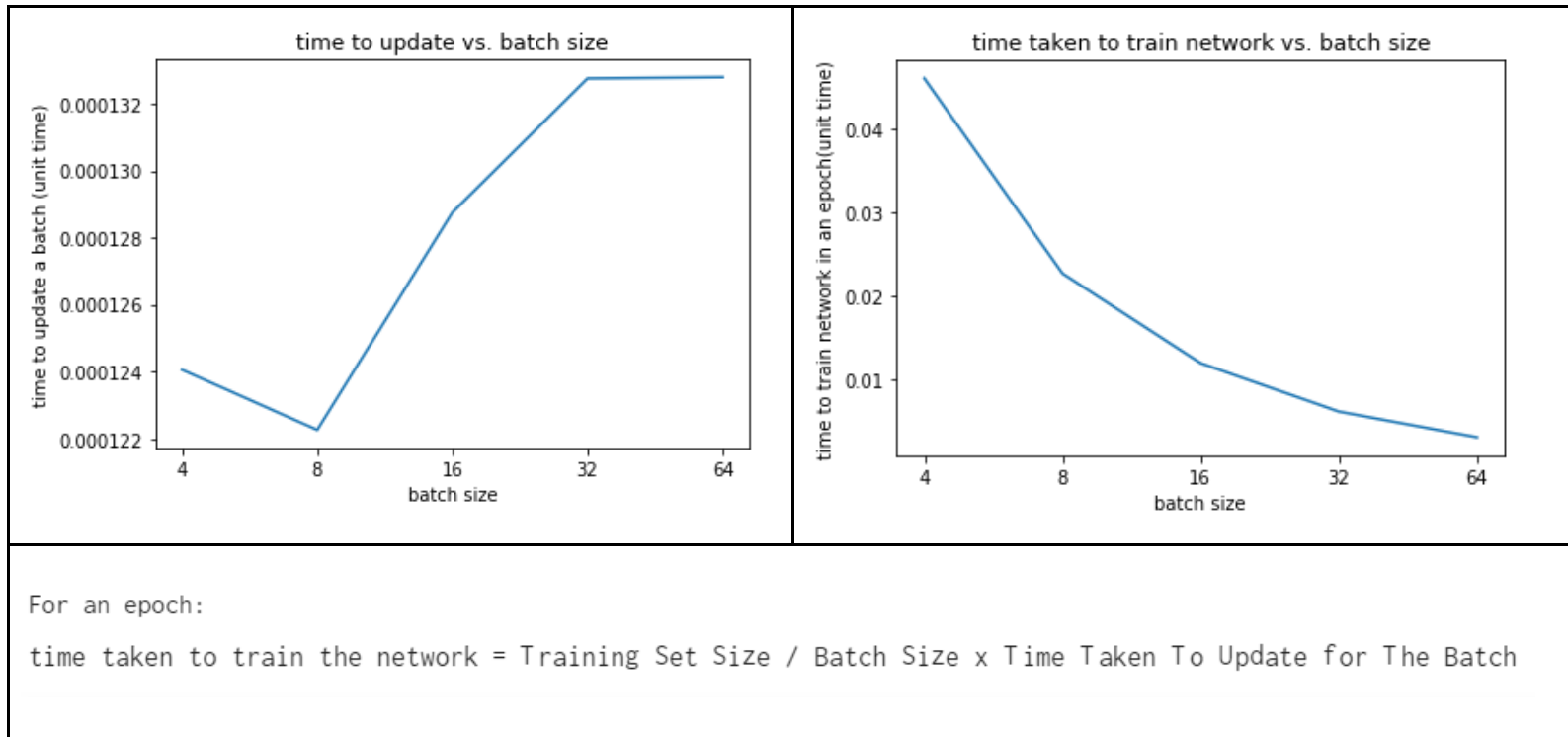
To compare the performance of for each batch size, we plot and compare the ***cross-validation accuracy against the epochs*** for different ***batch sizes***.



We can see that batch size 4 generally has the highest cross validation accuracy against epoch. This is followed by batch size 8 and 16 being tied for epochs below 2750. However, for epoch more than 2750, we can see that batch size 16 has higher cross-validation accuracy. Batch size 32 has a lower cross validation accuracy compared to batch size 8 and 16. Batch size 64 performs the worst.

In our experiment, we also find the time taken to update one batch for each batch sizes. From there, we are able to find the time taken to train the network for one epoch against different batch sizes. We proceed to plot 2 graphs titled:

- *time to update vs. batch size*
- *time taken to train network vs batch size*



From the graph on the right, we observe that the time taken to train the network in an epoch decreases at a decreasing rate as the batch size increases.

*Extra analysis*

From the graph on the left, by intuition, the time to update a smaller batch should take a shorter time than a larger batch. While this is generally true, we can see an exception here. Here, the time taken to update a batch with size 4 is larger than a batch of size 8. There could be many reasons due to it. One possible reason is that it could depend on the cache size of the processing unit.

**b) Select the optimal batch size and state reasons for your selection.**

From the above plots titled:

**comparison of cross-validation accuracy vs. epochs for different batch size**

We can see that batch size 4 generally has the highest cross validation accuracy against epoch. This is followed by batch size 8 and 16 being tied for epochs below 2750. However, for epoch more than 2750, we can see that batch size 16 has higher cross-validation accuracy. Batch size 32 has a lower cross validation accuracy compared to batch size 8 and 16. Batch size 64 performs the worst.

**time to train network in an epoch (unit time)**

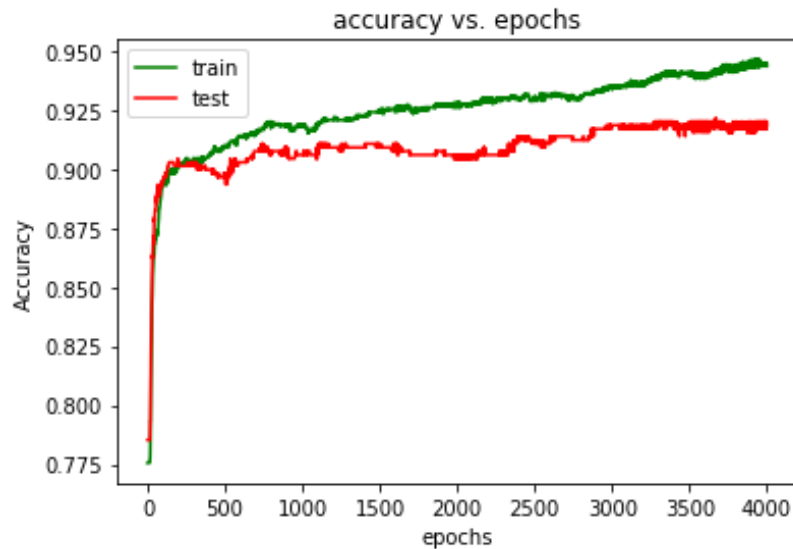
We can see that the time taken to train the network in an epoch decreases at a decreasing rate as the batch size increases.

Keeping these 2 observations in mind, we decide to keep a balance between the cross-validation accuracy and the time to train the network. We thus decided to **select batch size 16** as the optimal batch size.

**c) Plot the train and test accuracies against epochs for the optimal batch size.**

Shown below are:

- The plot of train/test error versus epochs, and
- The training logs



```

iter 1: train accuracy 0.7755376100540161
iter 1: test accuracy 0.7852664589881897

iter 1000: train accuracy 0.9180107712745667
iter 1000: test accuracy 0.9059560894966125

iter 2000: train accuracy 0.926075279712677
iter 2000: test accuracy 0.9059560894966125

iter 3000: train accuracy 0.9334677457809448
iter 3000: test accuracy 0.9184952974319458

iter 4000: train accuracy 0.9435483813285828
iter 4000: test accuracy 0.9184952974319458

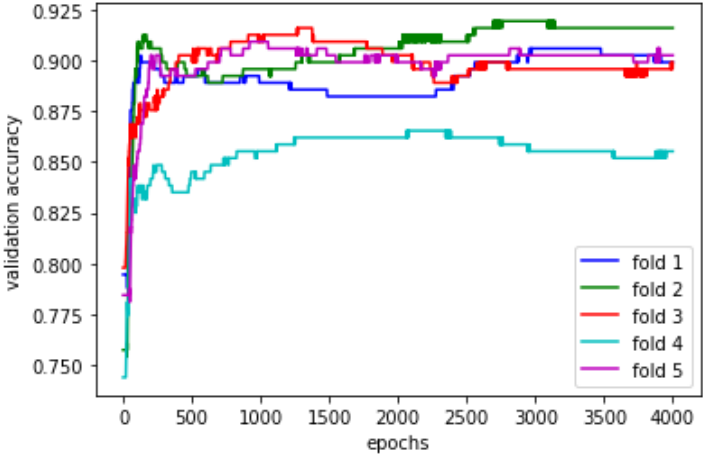
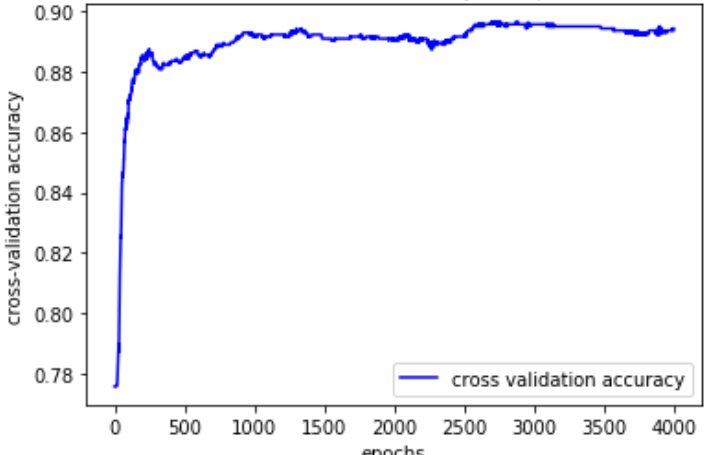
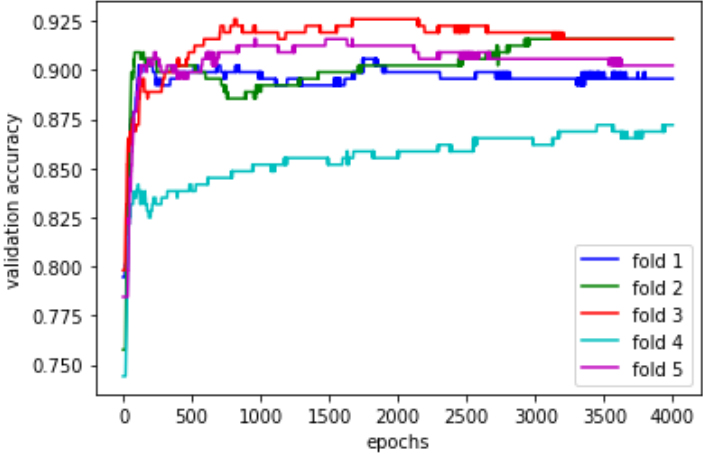
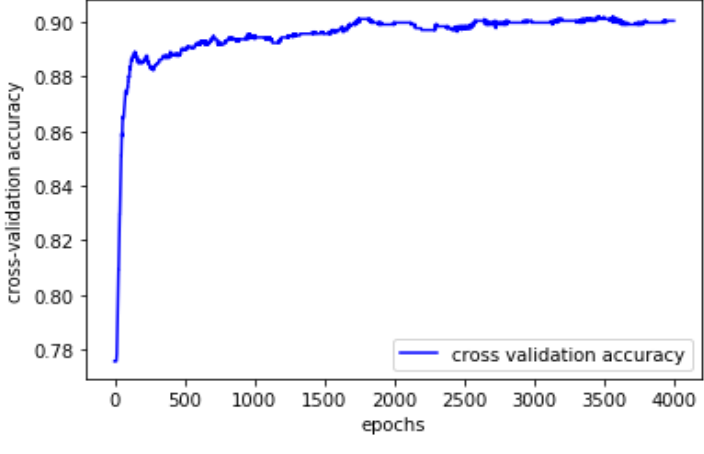
```

### Question 3

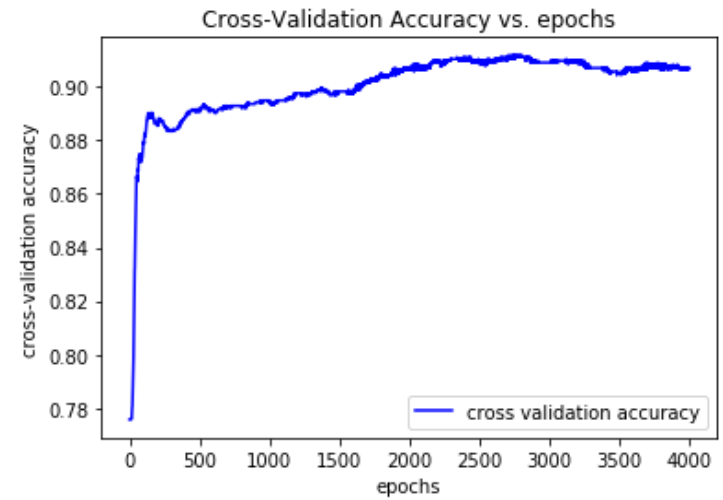
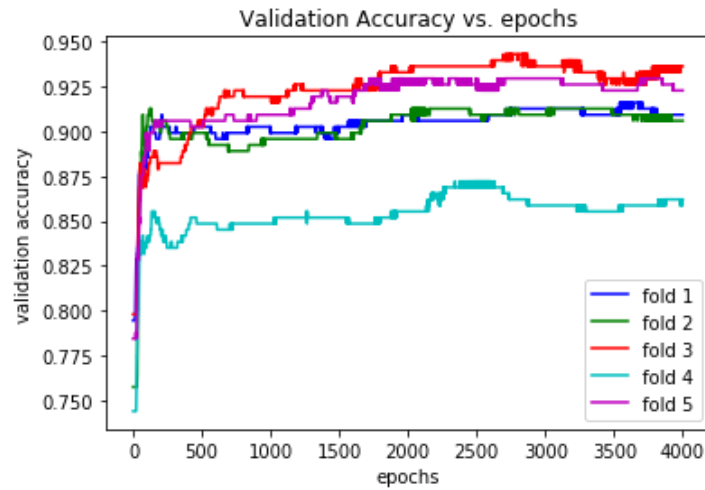
- a) **Plot the cross-validation accuracies against the number of epochs for different number of hidden-layer neurons. Limit the search space of number of neurons to {5,10,15,20,25}.**

From question 2, the optimal batch size selected is 16. In this question, we then build the neural network with a batch size of 16.

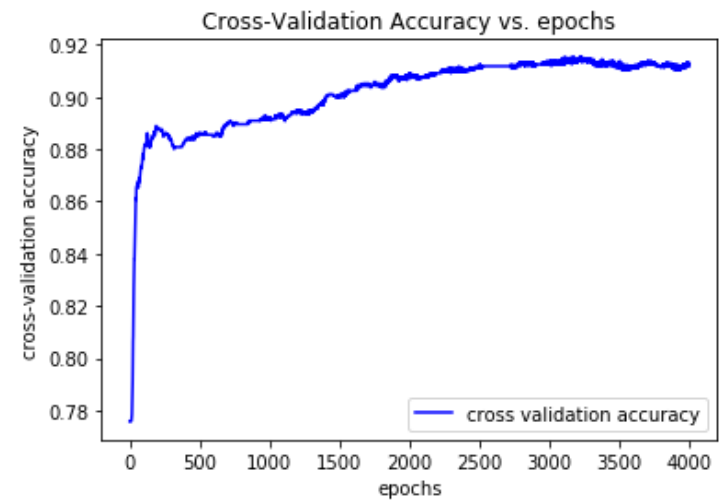
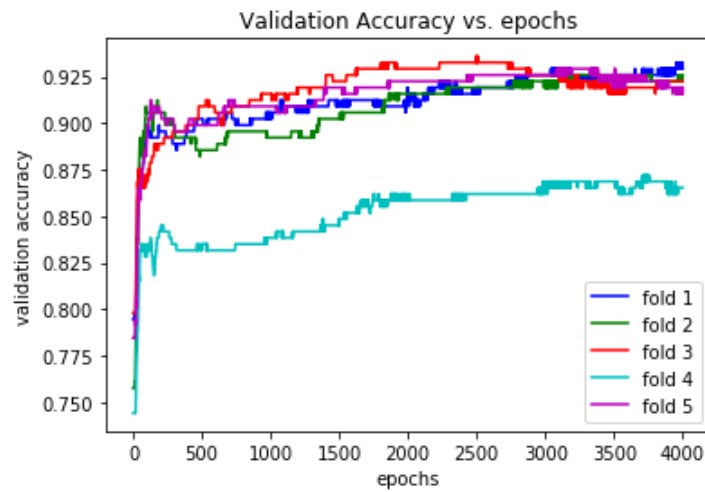
Similar as before, we plotted the validation accuracy for each fold against epochs on the left and the cross-validation accuracy against epochs on the right. Now, instead of doing it for different batch sizes, we are doing it for different number of hidden neurons.

No of hidden neurons	Validation Accuracy vs. epochs	Cross-Validation Accuracy vs. Epochs
5	<p data-bbox="646 326 989 354">Validation Accuracy vs. epochs</p> 	<p data-bbox="1354 326 1772 354">Cross-Validation Accuracy vs. epochs</p> 
10	<p data-bbox="646 873 989 901">Validation Accuracy vs. epochs</p> 	<p data-bbox="1354 873 1772 901">Cross-Validation Accuracy vs. epochs</p> 

15

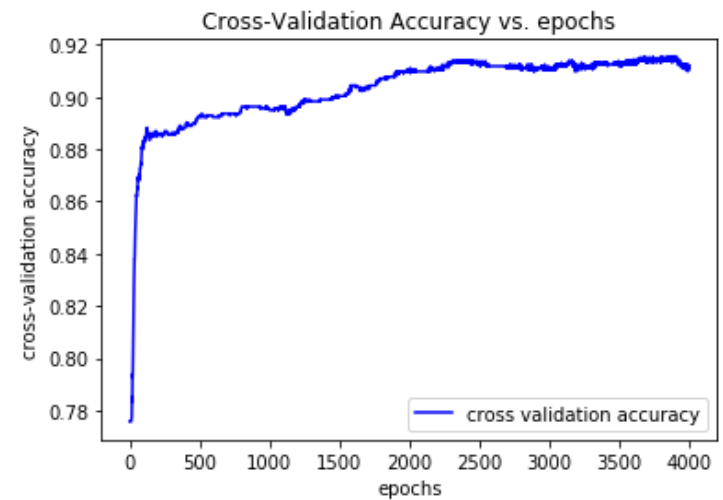
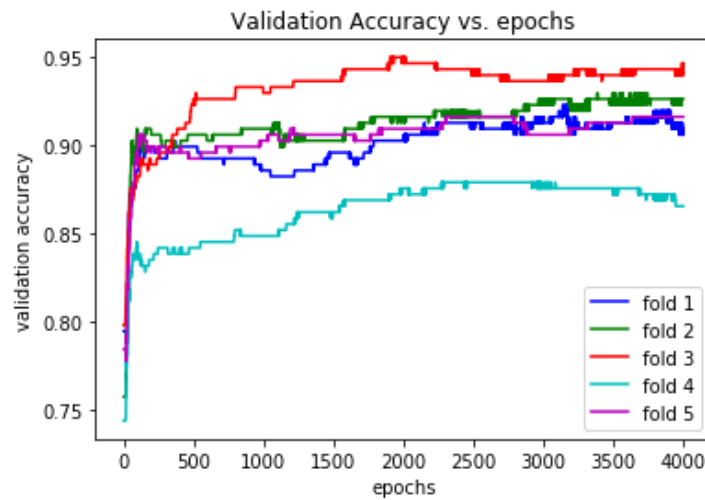


20

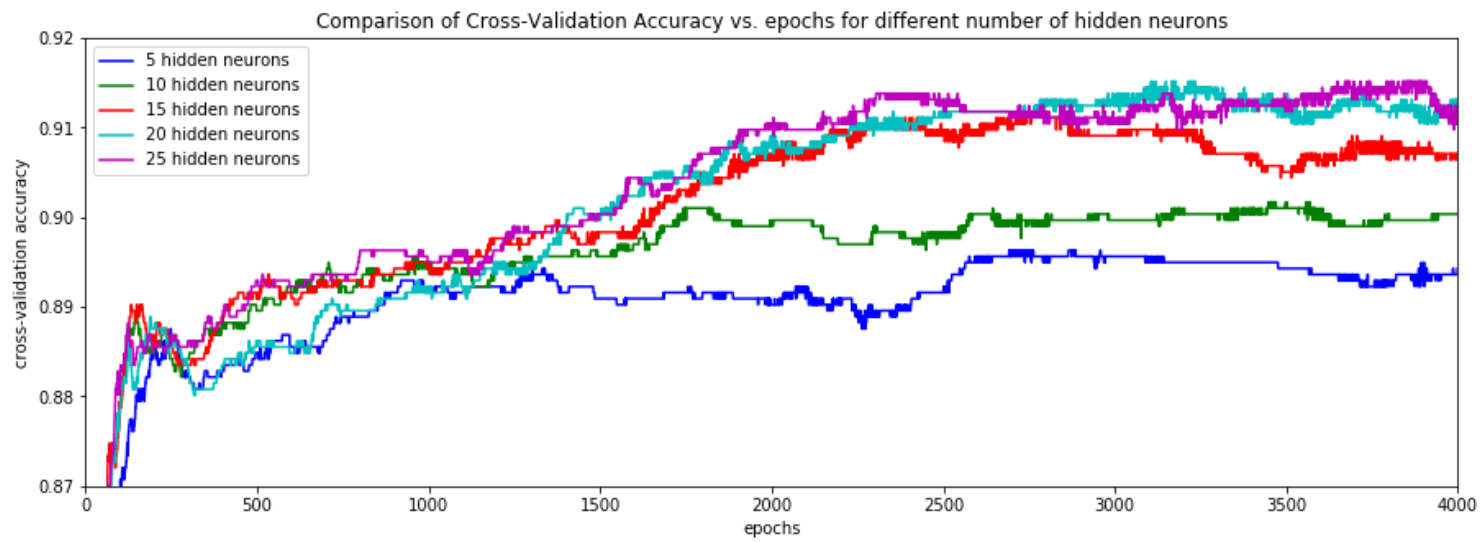




25



To compare the performance of for each batch size, we plot and compare the ***cross-validation accuracy against the epochs for different hidden neurons.***



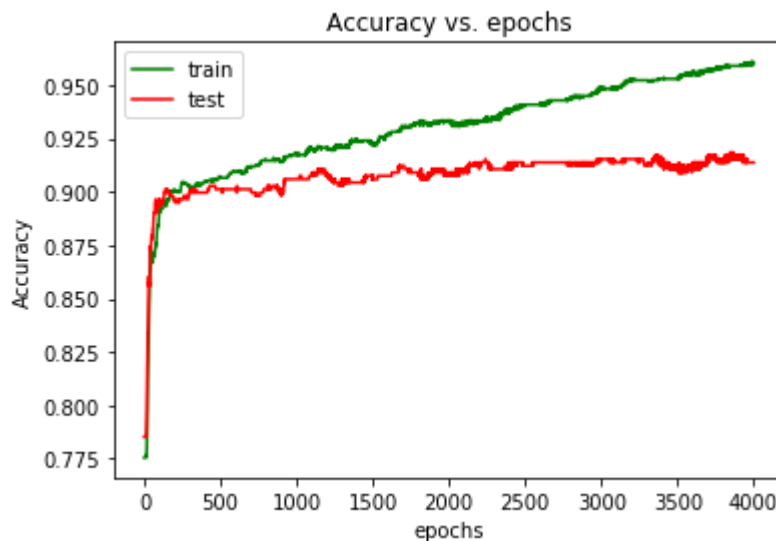
**b) Select the optimal number of neurons for the hidden layer. State the rationale for your selection.**

From the plot titled comparison of cross-validation accuracy vs. epochs for different batch size, we can see that the neural network leveraging on 25 neurons have the best cross-validation accuracy overall, therefore, the optimal number of neurons will be **25**.

**c) Plot the train and test accuracies against epochs with the optimal number of neurons.**

Shown below are:

- The plot of train/test error versus epochs, and
- The training logs



```
iter 1: train accuracy 0.7755376100540161
iter 1: test accuracy 0.7852664589881897

iter 1000: train accuracy 0.9173387289047241
iter 1000: test accuracy 0.9059560894966125

iter 2000: train accuracy 0.9334677457809448
iter 2000: test accuracy 0.9090909361839294

iter 3000: train accuracy 0.948924720287323
iter 3000: test accuracy 0.9153605103492737

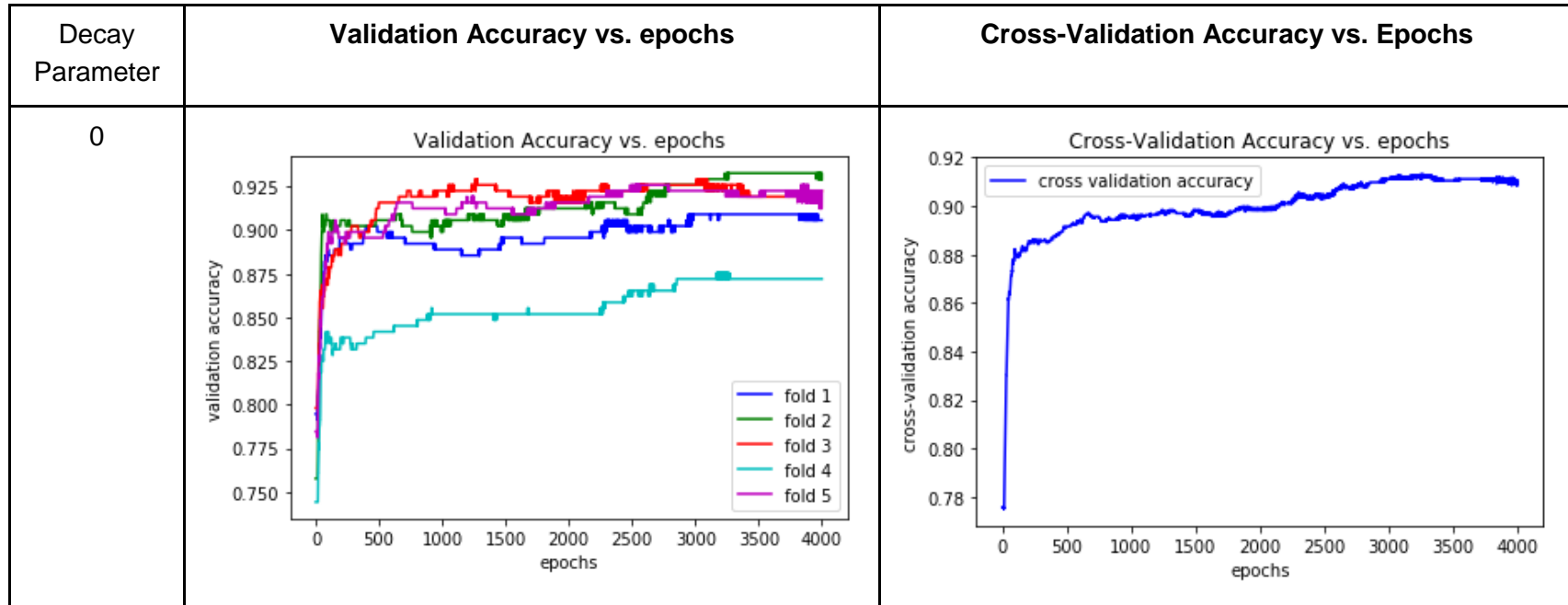
iter 4000: train accuracy 0.960349440574646
iter 4000: test accuracy 0.9137930870056152
```

#### Question 4

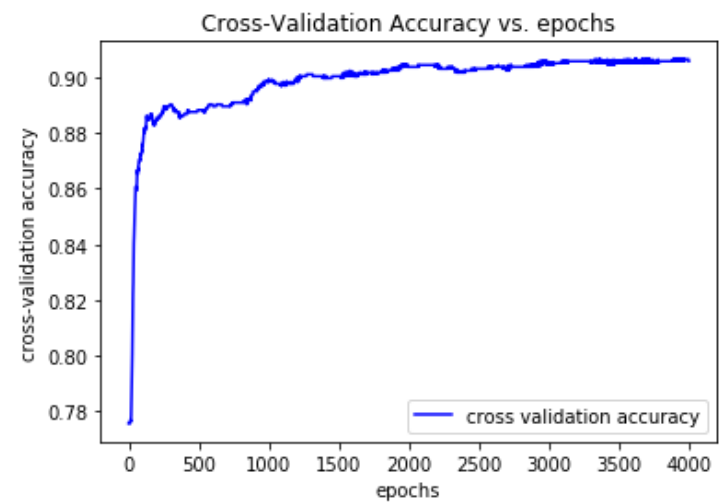
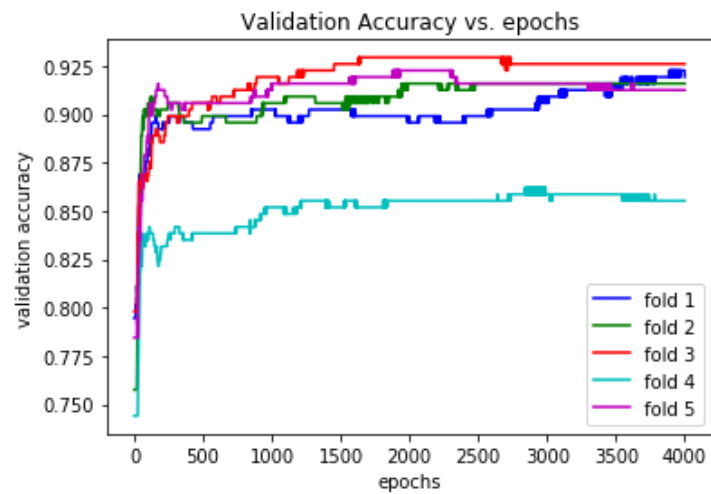
**a) Plot cross-validation accuracies against the number of epochs for the 3-layer network for different values of decay parameters. Limit the search space of decay parameters to  $\{0, 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}\}$ .**

From question 2 and question 3, the optimal batch size selected is 16 and the optimal number of neurons is 25. In this question, we will build the neural network with a batch size of 16 with 25 neurons in the hidden layer.

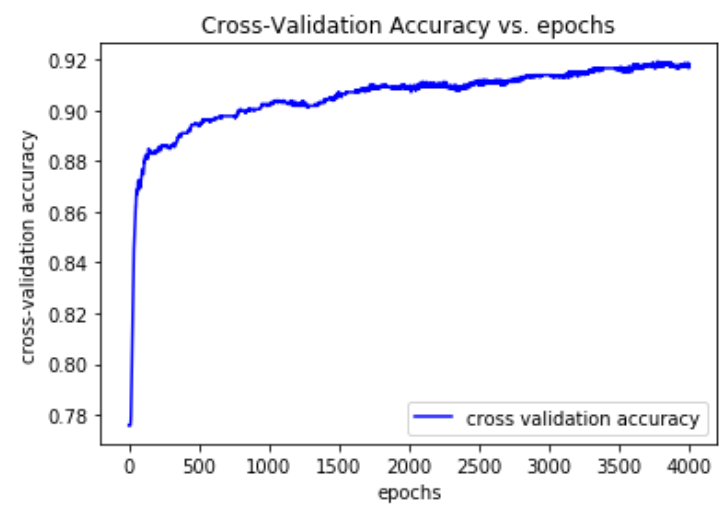
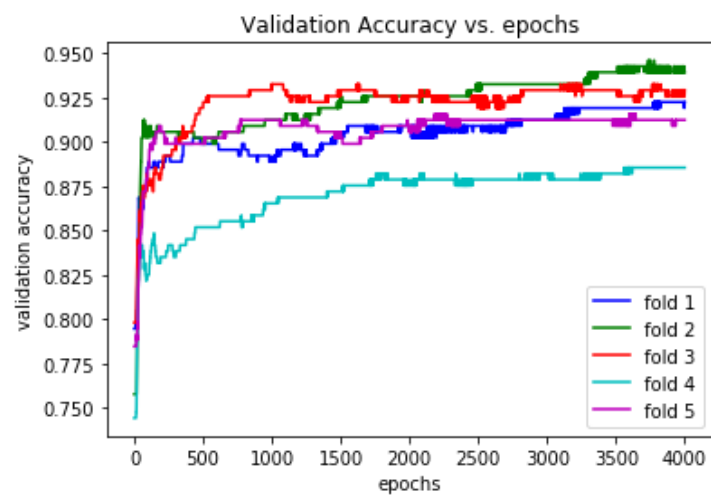
Similar as before, we plotted the validation accuracy for each fold against epochs on the left and the cross-validation accuracy against epochs on the right. Now, instead of doing it for different batch sizes, we are doing it for different decay parameters.

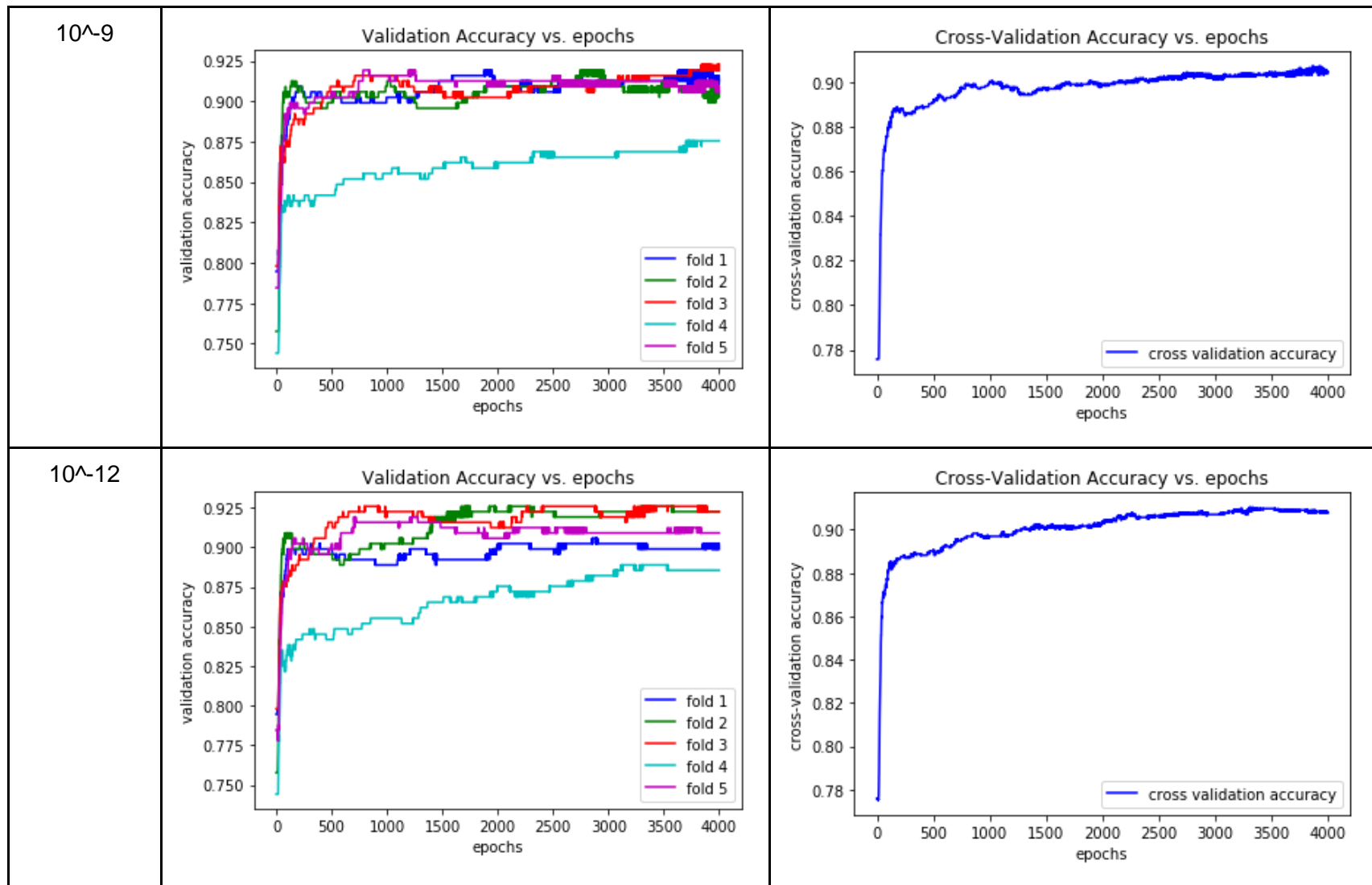


$10^{-3}$

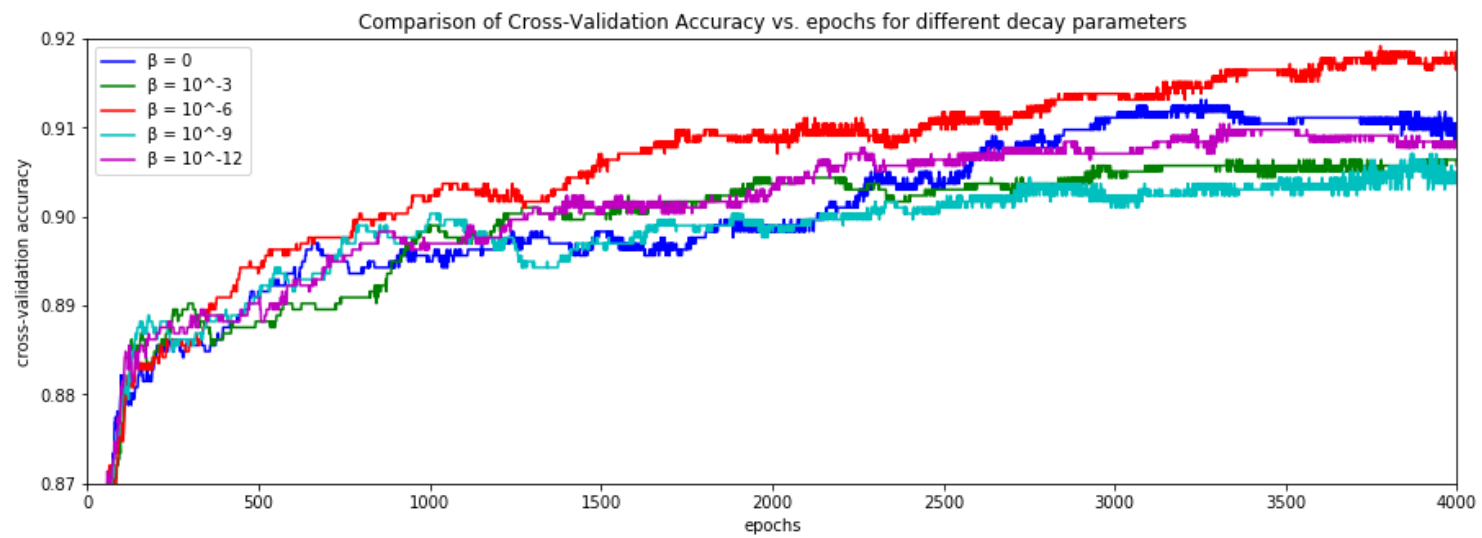


$10^{-6}$





To compare the performance of for each batch size, we plot and compare the ***cross-validation accuracy against the epochs for different decay parameters.***



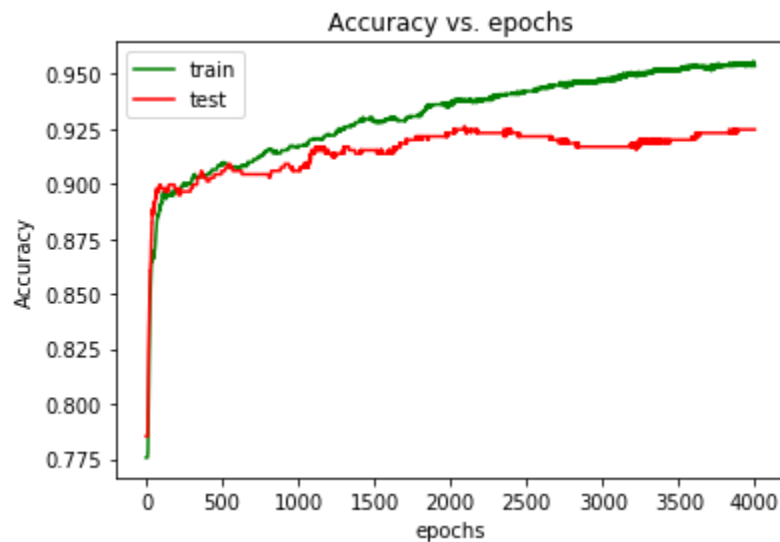
- b) **Select the optimal number of neurons for the hidden layer. State the rationale for your selection.**

From the plot titled Comparison of Cross-Validation Accuracy vs. epochs for different decay parameters, we can see that best cross-validation accuracy overall is achieved when the decay parameter is  $10^{-6}$ . Hence, it will be selected as the optimal epoch.

- c) **Plot the train and test accuracies against epochs with the optimal number of neurons.**

Shown below are:

- The plot of train/test error versus epochs, and
- The training logs



```
iter 1: train accuracy 0.7755376100540161
iter 1: test accuracy 0.7852664589881897

iter 1000: train accuracy 0.9166666865348816
iter 1000: test accuracy 0.9059560894966125

iter 2000: train accuracy 0.9354838728904724
iter 2000: test accuracy 0.9231975078582764

iter 3000: train accuracy 0.9469085931777954
iter 3000: test accuracy 0.9169278740882874

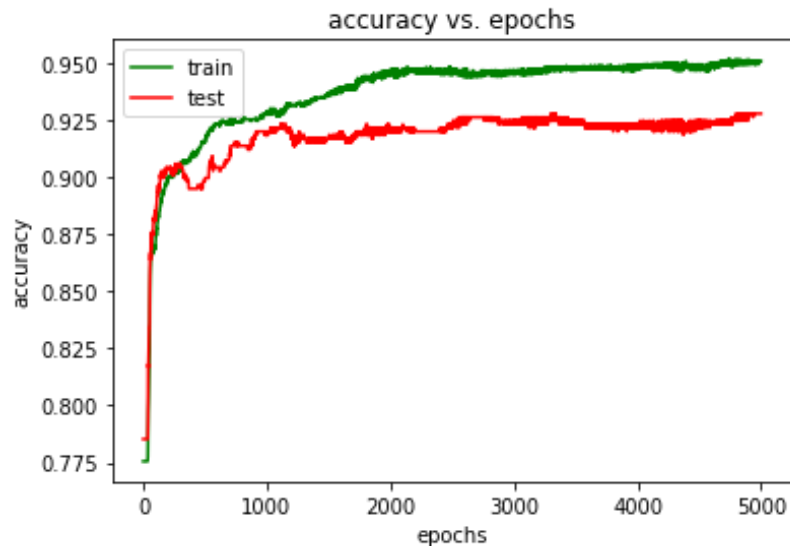
iter 4000: train accuracy 0.9536290168762207
iter 4000: test accuracy 0.92476487159729
```

## Question 5

### a) Plot the train and test accuracy of the 4-layer network

Shown below are:

- The plot of train/test error versus epochs, and
- The training logs



```
iter 1: train accuracy 0.7755376100540161
iter 1: test accuracy 0.7852664589881897

iter 1000: train accuracy 0.9280914068222046
iter 1000: test accuracy 0.9184952974319458

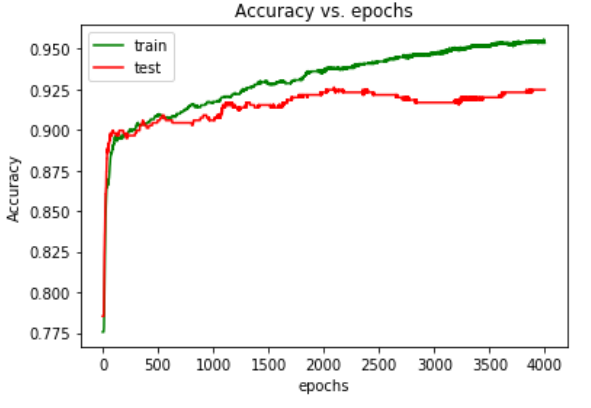
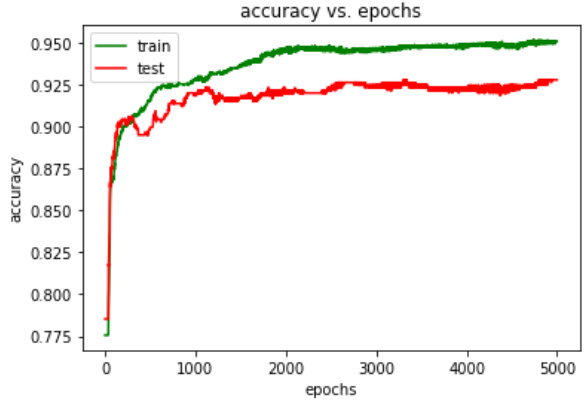
iter 2000: train accuracy 0.9455645084381104
iter 2000: test accuracy 0.9216300845146179

iter 3000: train accuracy 0.9455645084381104
iter 3000: test accuracy 0.92476487159729

iter 4000: train accuracy 0.9482526779174805
iter 4000: test accuracy 0.9231975078582764
```

### b) Compare and comment on the performances of the optimal 3-layer and 4-layer networks.



Optimal 3-layer networks	4-layer network
	
<pre> iter 1: train accuracy 0.7755376100540161 iter 1: test accuracy 0.7852664589881897  iter 1000: train accuracy 0.9166666865348816 iter 1000: test accuracy 0.9059560894966125  iter 2000: train accuracy 0.9354838728904724 iter 2000: test accuracy 0.9231975078582764  iter 3000: train accuracy 0.9469085931777954 iter 3000: test accuracy 0.9169278740882874  iter 4000: train accuracy 0.9536290168762207 iter 4000: test accuracy 0.92476487159729 </pre>	<pre> iter 1: train accuracy 0.7755376100540161 iter 1: test accuracy 0.7852664589881897  iter 1000: train accuracy 0.9280914068222046 iter 1000: test accuracy 0.9184952974319458  iter 2000: train accuracy 0.9455645084381104 iter 2000: test accuracy 0.9216300845146179  iter 3000: train accuracy 0.9455645084381104 iter 3000: test accuracy 0.92476487159729  iter 4000: train accuracy 0.9482526779174805 iter 4000: test accuracy 0.9231975078582764 </pre>

Amongst the iterations printed out, we observed that the best test accuracy for both the optimal 3 layer optimal network and 4 layer network are both the same at 0.92476487159729. The difference however, lies in when that test accuracy is achieved. For the optimal 3-layer network, it is observed at iteration 4000. However, for the 4-layer network, it is observed at iteration 3000, which is 1000 iterations faster than the 3-layer networks. We can therefore conclude that the test error converges faster for the 4-layer network than the 3-layer network.

#### *Extra analysis*

A faster convergence of test accuracy, however also could also implied that it will overfit faster too. For example, at iteration 4000, the 4-layer network has its accuracy decreased from when it is at iteration 3000. However, we cannot be sure whether the decrease is really due to an overfitting or due to noise. For example, for the optimal 3-layer network, its accuracy at iteration 2000 is higher than its accuracy at iteration 3000. A deeper analysis will be needed to differentiate the noise.

# Conclusion

## Answers to the queries

Answers are elaborated in the above Experiments and Results section.

## Summary of findings

- Tradeoff exists in the optimisation of the neural network. For example, even though batch size of 4 will give the most accurate result, we did not select it as the optimal batch size after considering the time taken to train the neural network. This explains why stochastic gradient descent while giving a better result is rarely used in industry with people choosing to use mini batch gradient descent instead.
- Sometimes we may not be able to fully explain why some observations appear. For example, in Question 2 above, batch size 4 has a longer time to update a batch than batch size 8. This seems to contradict our intuition that it takes a shorter time to compute a smaller matrix. However, there might be some complex hardware issues related to the occurrence of these results.

## Experience of experiments

During the experiments, we had a taste of the experience of training a neural network for classification from scratch on real datasets. Specifically, we found some interesting observations:

- The selection of optimal hyper-parameters for training a neural network is largely an empirical process. Even though we can use some theory and/or experience learnt from other people to help us limit the search space, we still have to search through a set of possible values, because the optimal hyper-parameters are very dependent on the dataset itself.
- The importance of hardware resources is highlighted during our project. In the project, we have tried to use the lab desktop and Google Colaboratory to train the model for Question 2. While Google Colaboratory took a total of 3 hours to perform 5 fold cross validation for each batch size, the school desktop took a total of 7 hours. Through this experience, we have discovered the importance of GPU in training neural networks.

# Part B: Regression Problem

## Introduction

In this project, we are required to build a 3-layer feedforward neural network consisting of an input layer, a hidden layer with ReLu activation function and a linear output layer. The neural network built will then be used for the regression of Graduate Admissions Prediction dataset. The aim is to accurately predict the chance of getting an admit (ranging from 0 to 1) of the test dataset after training the neural network on the training dataset.

## Method

### Train-test split

To conduct model selection, we have to split the data into 2 subsets: training set and test set. The training set is used to train the model and learn the parameters; the test set is used to estimate the true error and select the model with the best performance.

To do so, we first shuffle the original data in order to perform random sampling. After that, we take the first 70% of the data as the training set, and the rest as the test set.

```
8 # Define the size of training set and data set
9 dataset_size = len(dataX)
10 train_size = int(0.7 * dataset_size)
11 test_size = dataset_size - train_size
12
13 # Shuffle the data before splitting to training and testing set
14 idx = np.arange(dataset_size)
15 np.random.shuffle(idx)
16 dataX = dataX[idx]
17 dataY = dataY[idx]
18
19 # Split the training and testing set
20 trainX = dataX[:train_size]
21 trainY = dataY[:train_size]
22
23 testX = dataX[train_size:]
24 testY = dataY[train_size:]
```

*Extra analysis*

The above method implemented is **random subsampling**, whereby we randomly sample data for training and testing. To improve the performance of the model, we may consider performing **cross validation** during training. In this way, we can make full use of the limited data because all patterns in the dataset can be used for both training and testing.

### Input feature scaling

We employed feature-wise Z-Score Standardisation to normalise the input features. This is to avoid the situation where one (or a few) feature has a dominant effect on the learning, while weights for other features are not well-learnt by the network.

### Model building

We started with a 3-layer feedforward neural network using the TensorFlow package in Python. The network consists of an input layer of 7 nodes, a hidden layer of 10 neurons, and an output layer of 1 neuron.

The input layer takes in a batch of vectors of dimension 7, because the number of input features (*NUM\_FEATURES*) for each pattern is 7. The hidden layer consists of  $n$  neurons, where the value of  $n$  (*hidden\_units*) is initiated to 10. The output layer returns a batch of vectors of dimension 1, indicating the chance of getting an admit.

As this is a regression problem, the cost function is defined as the mean square error with L2 regularisation of weights, i.e. a regularised cost function. The updating of parameters is handled by the in-built tensorflow function `tf.train.GradientDescentOptimizer` to minimise the cost function.

The parameters in the model are initialized to the following:

- `NUM_FEATURES = 7`
- `hidden_neurons = 10`
- `batch_size = 8`
- `beta = 10**-3`
- `learning_rate = 10**-3`
- `epochs = 150000` (It is intentionally set to be large enough to observe over-fitting.)

To assess the performance during the training procedure, we plot the training errors and test errors.

# Experiments and Results

## Question 1

- a) Use the train dataset to train the model and plot both the train and test errors against epochs.

Shown below are:

- The training log, and
- The plot of train/test error versus epochs.

```
iter 40000: train error  0.004058993887156248
iter 40000: test error  0.004340378567576408

iter 45000: train error  0.003974443767219782
iter 45000: test error  0.0042689042165875435

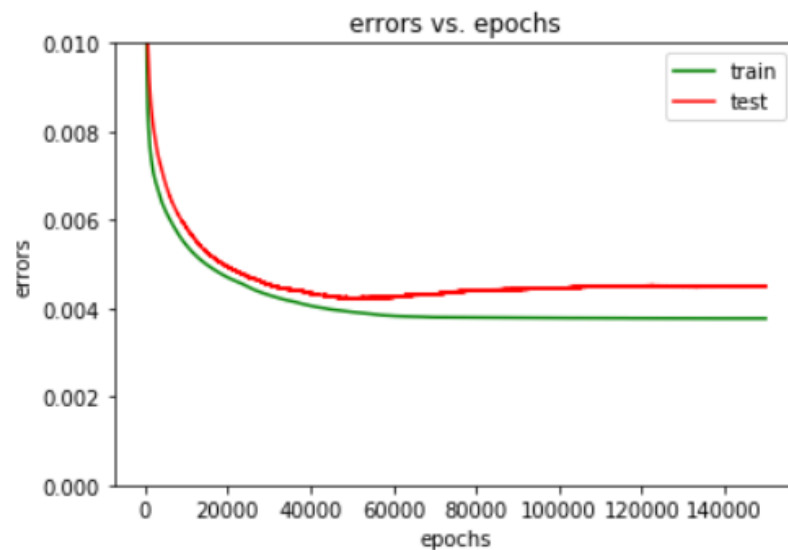
iter 50000: train error  0.003917704336345196
iter 50000: test error  0.004231365397572517

iter 55000: train error  0.0038688243366777897
iter 55000: test error  0.004244870971888304

iter 60000: train error  0.003829037304967642
iter 60000: test error  0.004251451231539249

iter 65000: train error  0.0038122795522212982
iter 65000: test error  0.004300627391785383

iter 70000: train error  0.003797971410676837
iter 70000: test error  0.004320033825933933
```

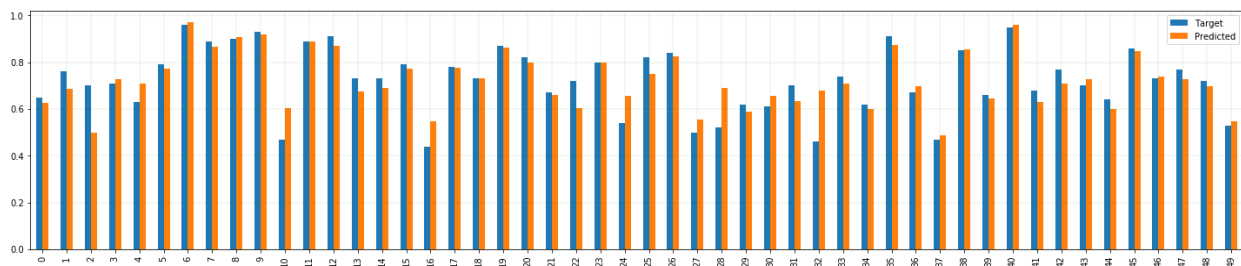


- b) State the approximate number of epochs where the test error is minimum and use it to stop training.

From the training log and the plot above, we can see that the test error starts to increase at about 50,000 iterations. To confirm this observation, we also found the minimum test error in the training process to be 0.004224059 and the optimal epoch is at 50,648.

- c) Plot the predicted values and target values for any 50 test samples.

With the optimal epoch determined above, we re-train the model until epoch 50,648. We then use the new model to predict on test set and plot a bar chart showing the target and predicted values for 50 test patterns. The plot below shows that the predicted values are quite close to the target values.



(Refer to notebook output for a plot of higher resolution.)

## Question 2

We utilised the in-built function from Pandas library to compute the correlation matrix. To do so, we first convert the training dataset, which is a numpy array, to a pandas dataframe. Then, we compute and plot the correlation matrix as shown below.

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1	0.8353	0.658681	0.612688	0.55787	0.819244	0.582038	0.807911
TOEFL Score	0.8353	1	0.677912	0.630784	0.538535	0.823203	0.474201	0.794497
University Rating	0.658681	0.677912	1	0.740724	0.644755	0.736585	0.440473	0.68931
SOP	0.612688	0.630784	0.740724	1	0.724421	0.735287	0.459717	0.692015
LOR	0.55787	0.538535	0.644755	0.724421	1	0.672362	0.436861	0.663962
CGPA	0.819244	0.823203	0.736585	0.735287	0.672362	1	0.519085	0.864054
Research	0.582038	0.474201	0.440473	0.459717	0.436861	0.519085	1	0.565128
Chance of Admit	0.807911	0.794497	0.68931	0.692015	0.663962	0.864054	0.565128	1

### Extra Analysis

The diagonal entries are 1 because a variable is always perfectly and positively correlated with itself. However, this information is typically not insightful in statistics/data analytics.

**a) Which features are most correlated to each other? Is it justifiable?**

The top 3 pairs of most correlated features are listed below:

Features	Correlation
GRE Score, TOEFL Score	0.8353
TOEFL Score, CGPA	0.8232
GRE Score, CGPA	0.8192

These are justifiable because candidates who have the aptitude to do well in examinations would usually be able to perform well in **different kinds of academic assessments** - GRE, TOEFL and school examinations which contribute to CGPA. Hence, candidates with high scores in one of the three examinations above would likely to also have high scores for the other two, therefore the high correlation.

**b) What features have the highest correlations with the chances of admit?**

The 3 features below have the highest correlation with the chance of admit.

Features	Correlation with chance of admit
CGPA	0.864054
GRE Score	0.807911
TOEFL Score	0.794497

### Question 3

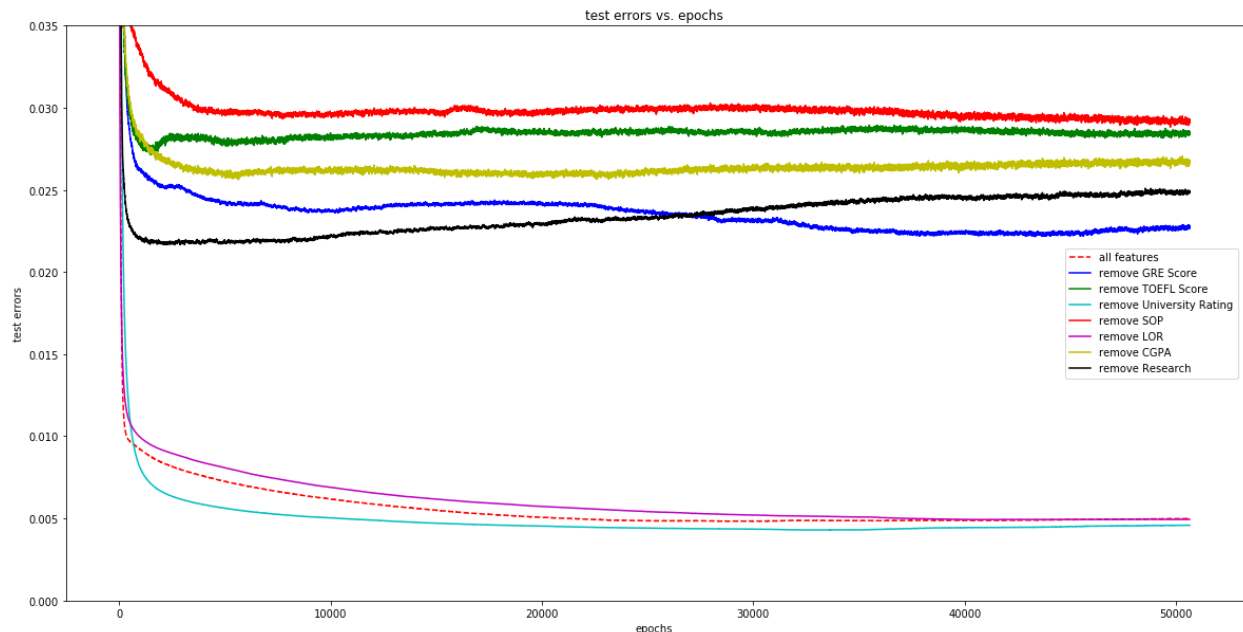
#### Procedure

Recursive feature elimination (RFE) requires us to remove one input feature at a time. The removed input feature should cause the minimum drop, or the maximum improvement, in performance.

- We first try to remove one feature, which would cause the number of input features to decrease from 7 to 6. In order to decide which feature to remove, we examine the performance of the model after each of the features is removed.

For example, we remove the first feature (GRE Score), and then re-train the model to obtain the test errors after removal. These errors are recorded. We repeat this process for 7 iterations because there are 7 input features available.

The test errors after removal of one feature is shown below:



(Refer to notebook output for a plot of higher resolution.)

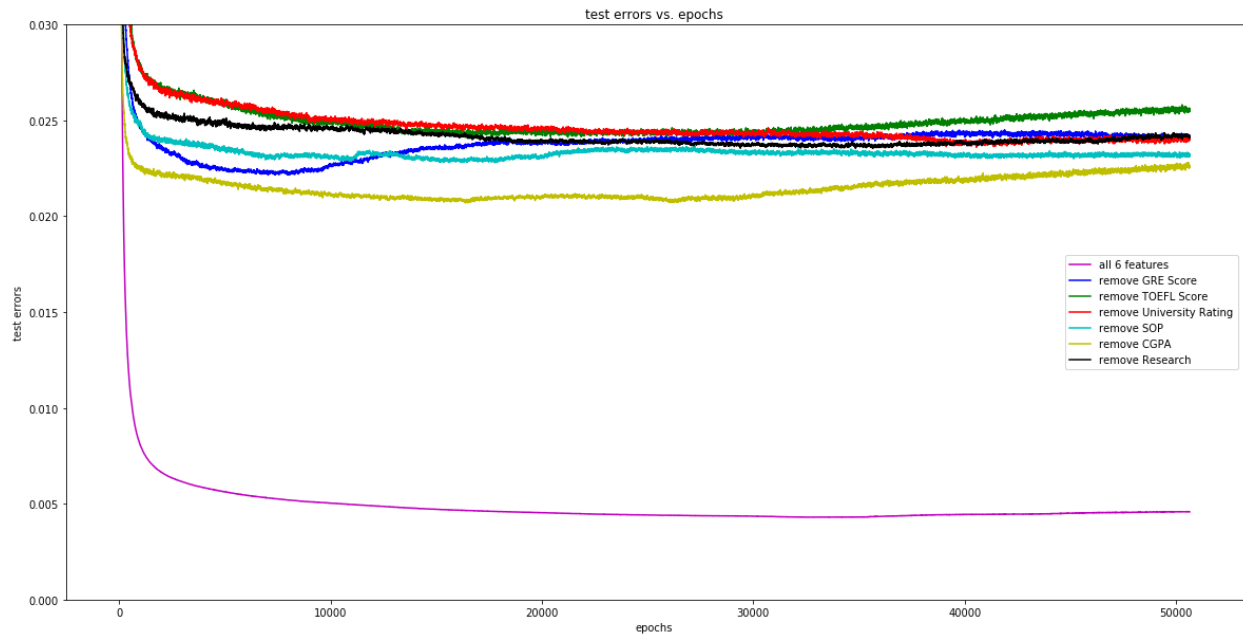
From the plot above, we can see that by removing the input feature University Rating, the test errors actually decrease (as shown by the lowest curve in the plot).

Therefore, in the first round of removing input features, we decide to remove the input feature University Rating. The 6 input features left are: GRE Score, TOEFL Score, SOP, LOR, CGPA, and Research.

- b) We then try to remove another feature, which would cause the number of input features to decrease further from 6 to 5. In order to decide which feature to remove, we again examine the performance of the model after each of the features from part (a) is removed.

The test errors after removal of one feature is shown below:





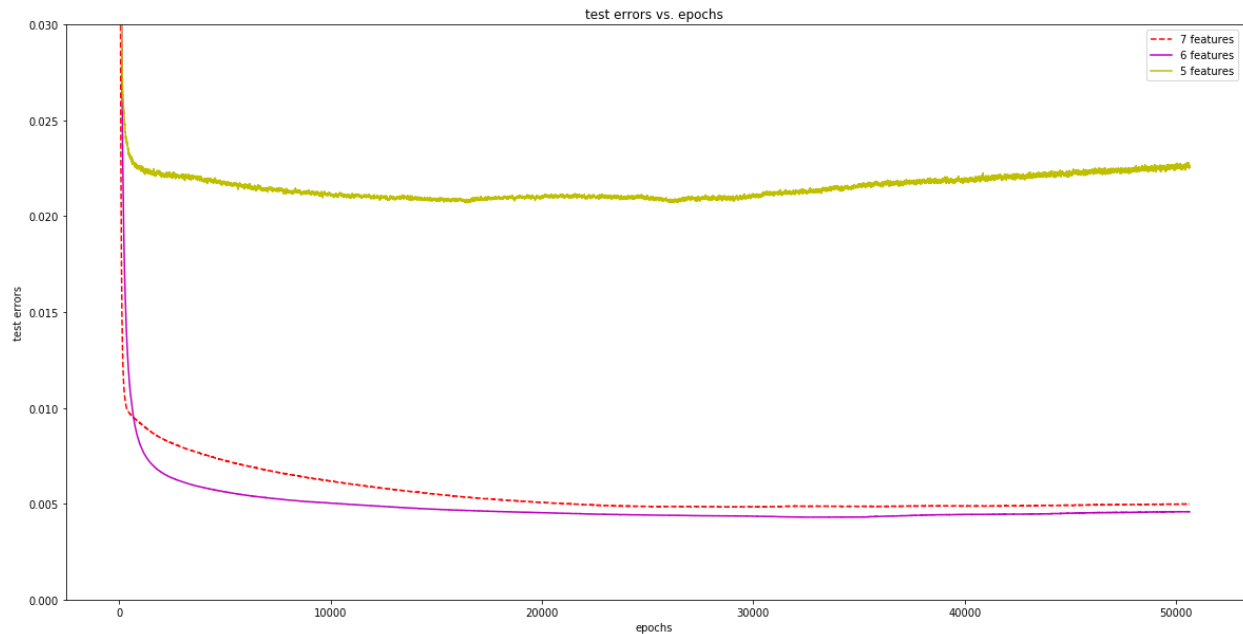
(Refer to notebook output for a plot of higher resolution.)

From the plot above, we can see that by removing the input feature CGPA, the test errors increase by the smallest amount (as shown by the second lowest curve in the plot).

Therefore, in the second round of removing input features, we decide to remove the input feature CGPA. The 5 input features left are: GRE Score, TOEFL Score, SOP, LOR and Research.

### Comparison

The test errors for models with 7, 6, and 5 input features respectively are shown below:



#### 7 input feature vs 6 input features:

By removing the input feature University Rating, the test errors actually improve. This implies that the model can perform better without information about University Rating, which creates more noise in training and prediction.

This might be related to the fact that University Rating and Chance of Admit have a low correlation coefficient of 0.68931, which could mean that there is little relationship between the two variables. As such, by removing the 'irrelevant' feature University Rating, the model can focus on learning the 'relevant' features and produce better results.

#### 7 input feature vs 5 input features:

By removing the input features University Rating and CGPA, the test errors increase to about 4 times of the original. This implies that the model can perform worse without information about CGPA, and CGPA is actually a good predictor of the Chance of Admit. However, from procedure (b), we decided to remove CGPA because removing any of the other features would cause test errors to increase even more. Thus, we can infer that CGPA is a good predictor, but other features are better predictors of the Chance of Admit.

This might be related to the fact that CGPA is highly correlated with GRE Score and TOEFL Score. This means that these variables carry similar information about the pattern and hence, if one of them is present in the model, the other two might be redundant. In other words, there might be multi-collinearity in the data when one of the three variables above can be linearly predicted by the other two. As such, by removing the 'repeated' feature CGPA, the model can focus on learning the 'unique' features and produce better results.

Additionally, the model with 5 input features converges faster at about 25,000 epochs, and the test error starts to increase afterwards. On the other hand, the model with 7 input features only converges at about 50,000 epochs (as shown in Question 1).

This could be a natural result from feature reduction. As the number of features becomes smaller, the network tends to remember the training data at earlier epochs, which results in over-fitting at earlier epochs.

#### *Extra analysis*

The RFE method above is able to produce the most optimal input feature set based on empirical experiments and a heuristic approach. It can automate the process of feature selection, especially when there is limited domain knowledge and we humans are not able to tell which features are important.

However, one issue with RFE is that it can be expensive to run, i.e. its execution time can be long. For example, in procedure (b) above, it takes us 84 minutes to complete the model training and determine the best input feature to be removed. In practice, feature selection can be done in a more efficient way (at least for initial stages of feature removal).

Removing correlated features is one of the statistical methods that can be employed here<sup>1</sup>. For example, from Question 2, we see that GRE Score, TOEFL Score and CGPA are highly correlated. We could consider removing one or two of these because they carry very similar information about the data. Another statistical method is Principal Component Analysis, which gives us a transformation of the features that carries the largest variation in the data.

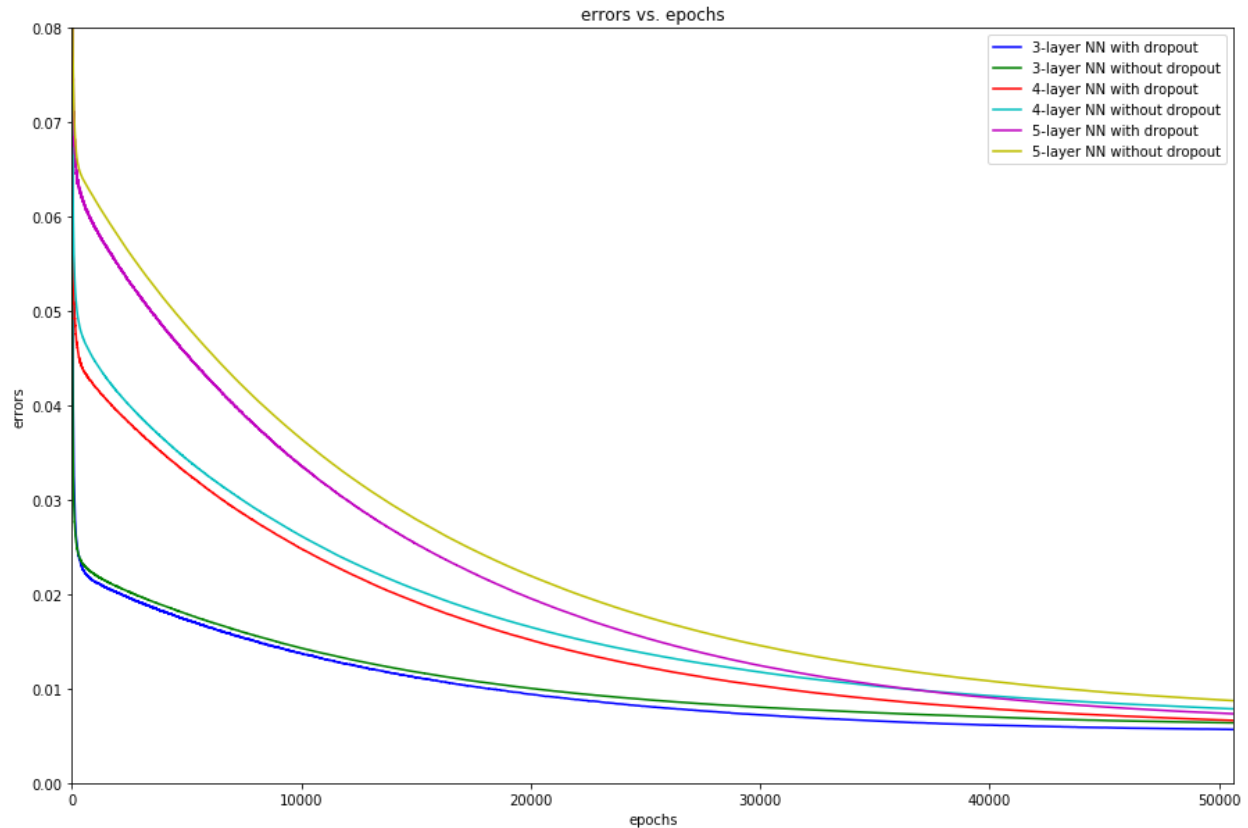
After we remove some input features by either statistical approach or domain knowledge, we can then proceed with RFE. In this way, RFE will be more efficient due to the smaller input feature size, because with one less input feature, we can essentially save one iteration of the model building process.

### Question 4

**Design a four-layer neural network and a five-layer neural network, with the hidden layers having 50 neurons each. Use a learning rate of 10-3 for all layers and optimal feature set selected in part (3). Introduce dropouts (with a keep probability of 0.8) to the layers and report the accuracies. Compare the performances of all the networks (with and without dropouts) with each other and with the 3-layer network.**

---

<sup>1</sup> <https://towardsdatascience.com/feature-selection-in-python-recursive-feature-elimination-19f1c39b8d15>



The graph above compares the performances of all the networks which are of interest to us. There are some interesting observations:

- The testing errors for a neural network with dropout are lower than a neural network of the same number of layers without dropout.
- Neural networks with fewer layers have lower errors compared to those with more layers within epochs of 50000.
- As the number of epochs increases, the differences in errors between neural network with different layers decreases. They will all converge to the same point.

## Conclusion

### Answers to the queries

Answers are elaborated in the above Experiments and Results section.

## Summary of findings

- Applying dropout can help to reduce overfitting which helps in generalisation of the model on unseen data.
- One would have thought that a neural network with more layers will be able to predict much better than neural network with fewer layers. From this experiment, we have discovered that this is not true. In fact, the gap is only reduced with more epoch. This implies a more complex neural network will take a longer time to properly learn the patterns in a dataset.

## Experience of experiments

During the experiments, we had a taste of the experience of training a neural network for regression from scratch on real datasets. Specifically, we found some interesting observations:

- Initialisation of parameters is important in model convergence. Even with the same code and model architecture, different weight initialisation can result in very different learning curves and minimum test error. As such, if time and resources permit, it may be more preferred to train the same model several times, and take the average test error as the basis for model selection.