NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE/CZ 4042 NEURAL NETWORKS

Assignment 1

Date of Last Update: 19 October 2019

Group Members: Ang Qi Xuan (U1621610H)
Lim Zheng Min (U1622040D)

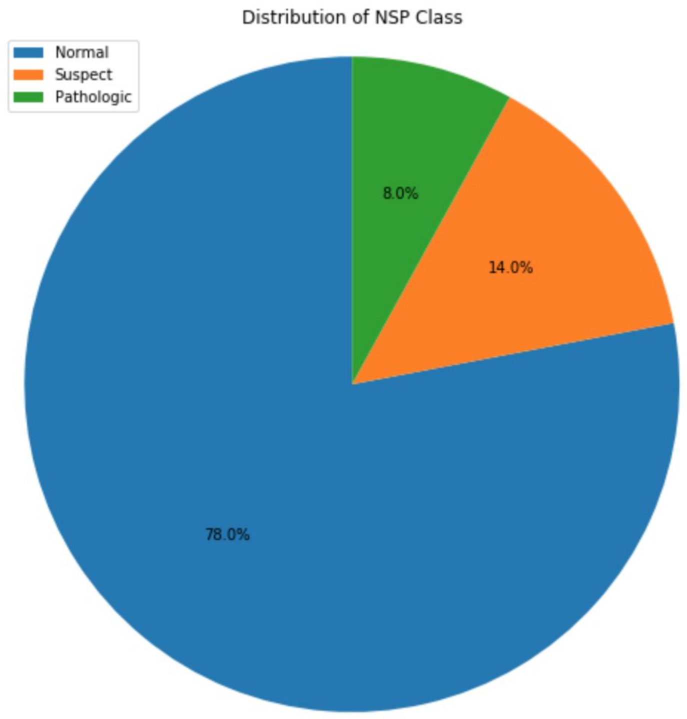
1 Classification Problem

1.1 Introduction

We are given a Cardiotocography dataset containing the measurements of fetal heart rate (FTR) and uterine contractions (UC) features on 2126 Cardiotocograms classified by expert obstetricians. Given this dataset, our objective is to build a neural network architecture that can predict the N,S,P class labels in the dataset

1.1.2 Exploring Dataset

The dataset consist of 2126 samples of data and 21 different features. When we explore the distribution of class labels in the given dataset, we can see that 'Normal' class is the majority, making up 78% of the data.



This might pose a problem of an imbalanced dataset later on when we are training our Neural Network architecture to predict the NSP classes. For now we will ignore the nature of the imbalanced dataset.

1.2 Methods

For this question, we can see that the process of designing a model with optimal hyperparameters involves building a new model iteratively with different values of the hyperparameters in order to cross validate with our validation set to determine the optimal value to choose from.

Hence, we have built separate functions that compute certain part of the model building, to make sure that the respective computation are abstracted away, creating a modular architecture.

1.2.1 Functions

- **datagen() :** Function that read in the csv, separate into input and output. The function will also split the dataset into train/test set of 70:30 ratio.
- **scale() :** Standardise the value of the feature set, so that our gradient descent algorithm can converge faster
- **ffn3() :** Function that calculate the logit for a 3-layer architecture based on the number of neurons provided
- **training() :** Function that calculates the the validation accuracy for each hyperparameter value range
- **nn3() :** Return the test accuracy of a 3-layer model with the chosen hyperparameters
- **ffn4() :** Function that calculate the logit for a 4-layer architecture based on the number of neurons provided
- **nn4() :** Return the test accuracy of a 4-layer model with the chosen hyperparameters
-

1.2.2 Multi-Processing

Because of the need to build the tensorflow computation graph iteratively across the various given hyperparameters values to choose the optimal value, it will be time consuming to run it in series with a for-loop. Thus, to increase the computation time we used a Python library *Multiprocessing* that enable us to leverage on the multi-core processor to run the various computations in parallel. This allows us to better utilize the CPU processing capacity and at the same time, reduce our computation time if we are computing large number of epochs.

1.2.3 Regularisation

To prevent our Neural Network from overfitting, we will need regularisation. The intuition of regularisation is to penalise large weights given by the network and keep the weights small, so that the output of the network doesn't depend too heavily on certain features. Hence, preventing our model from overfitting, giving a simpler model. Although our training accuracy will drop with a simpler model as a result of regularisation, it will help to reduce our variance, making our model more capable of generalising to unseen new data.

Final Loss = Loss + Regularisation term

1.2.4 K-Fold Validation & Data Shuffling

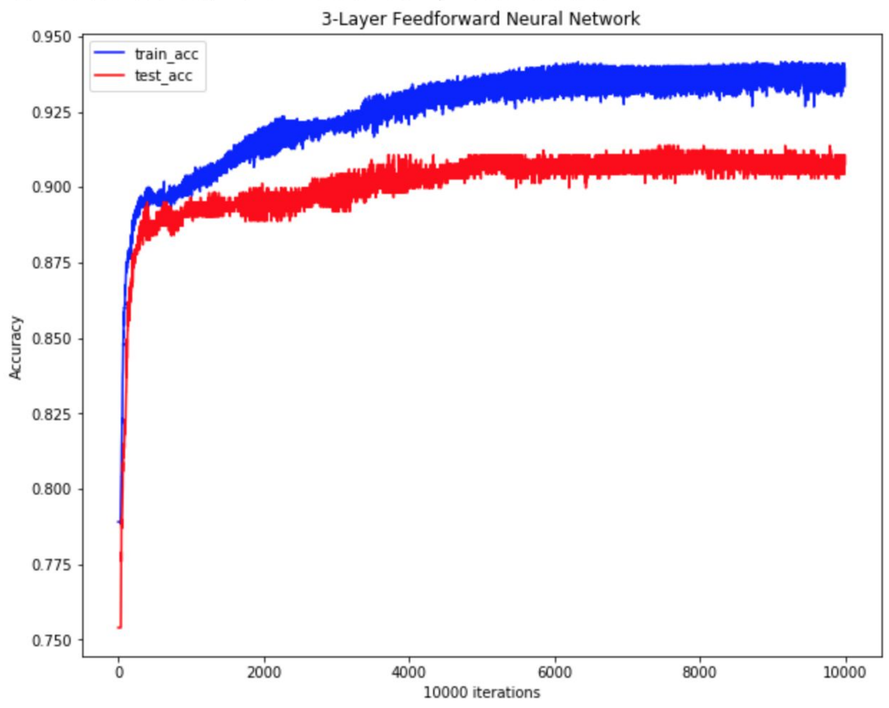
In this assignment, we need to plot cross validation accuracy. Our approach will be to use the **Three-Way Data Split** method:

1. Split dataset into train:test with 70:30 ratio. Hold out our test set during validation
2. Using the train set, we use K-fold split for implementing cross validation to find the optimal hyperparameter value.
3. Train the final model with the entire train set and estimates its accuracy with our test set

By holding out the test set and using k-fold to do cross-validation, we can be sure that our model will not be bias as it is not used to select the hyperparameters during validation. The testset will be able to give a good estimate of the model.

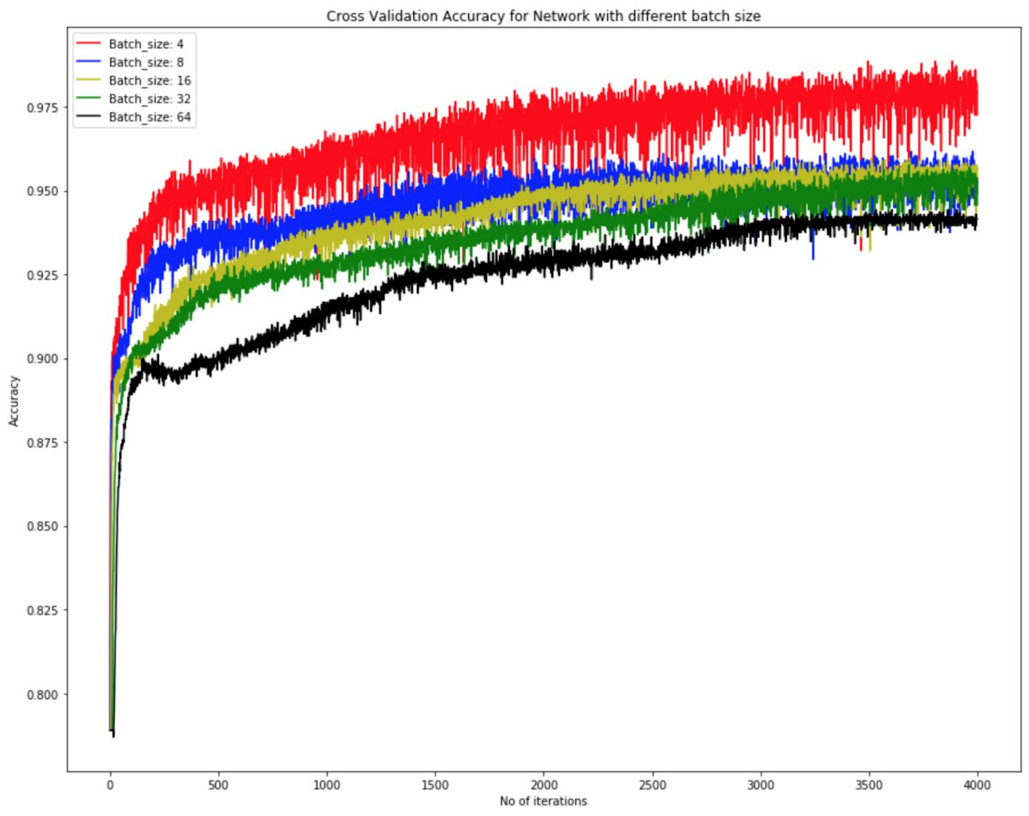
1.3 Experiments & Results

1. (a) 3-Layer Network



(b) From the graph above, we can see that the test error of the 3-layer network converges after 4000 epochs. We will use this value as the training epoch for the subsequent sections

2. (a) Cross validation accuracies for different batch size



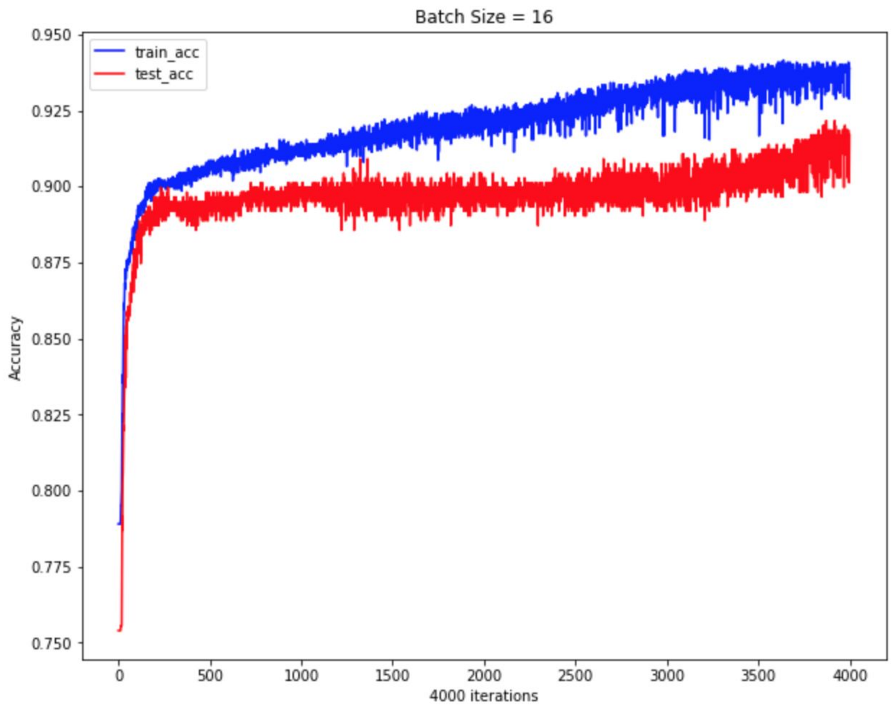
Time taken to train network for one epoch against different batch size

```
batch size: 4
13.373472929000854
batch size: 8
12.288278818130493
batch size: 16
12.018205642700195
batch size: 32
11.726881504058838
batch size: 64
11.144592523574829
```

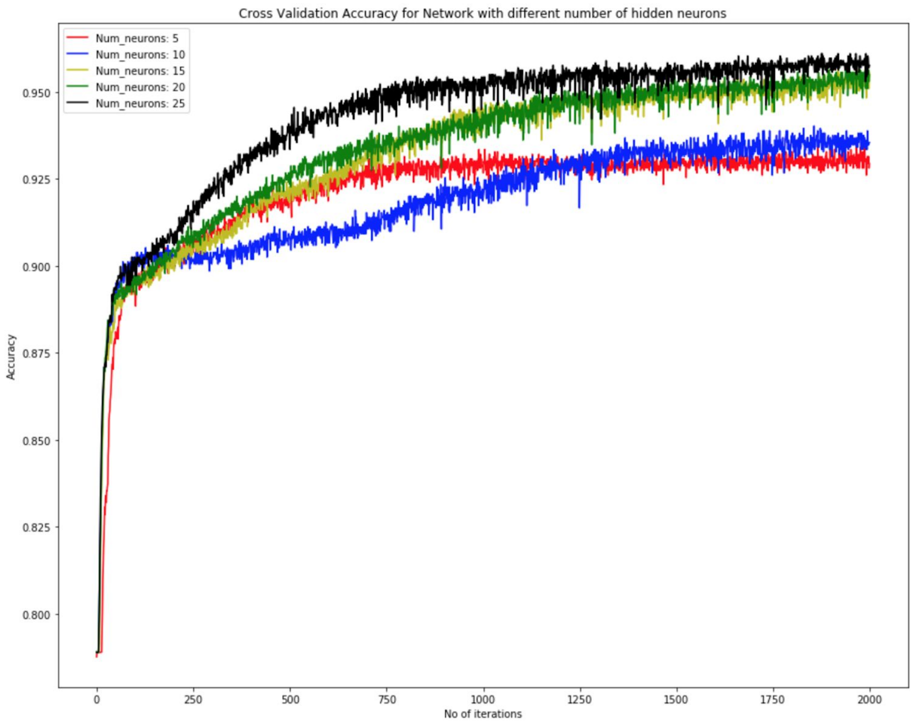
(b) Batch size is important for the gradient descent algorithm to achieve effective learning. In our case, we can see that batch size = 4 will give the highest validation accuracy. However, there is a lot of noise in the accuracy and it also takes the longest time for a single epoch since the weights are updated after training on every 4 examples.. What we would like is to achieve a balance in terms of fast convergence and minimal time for training each epoch.

Hence, we chose our batch size = 16 as it takes relatively less time per epoch and the validation accuracy is also amongst the highest other than batch size 4

(c) Train vs Test accuracy against epoch for optimal batch size = 32



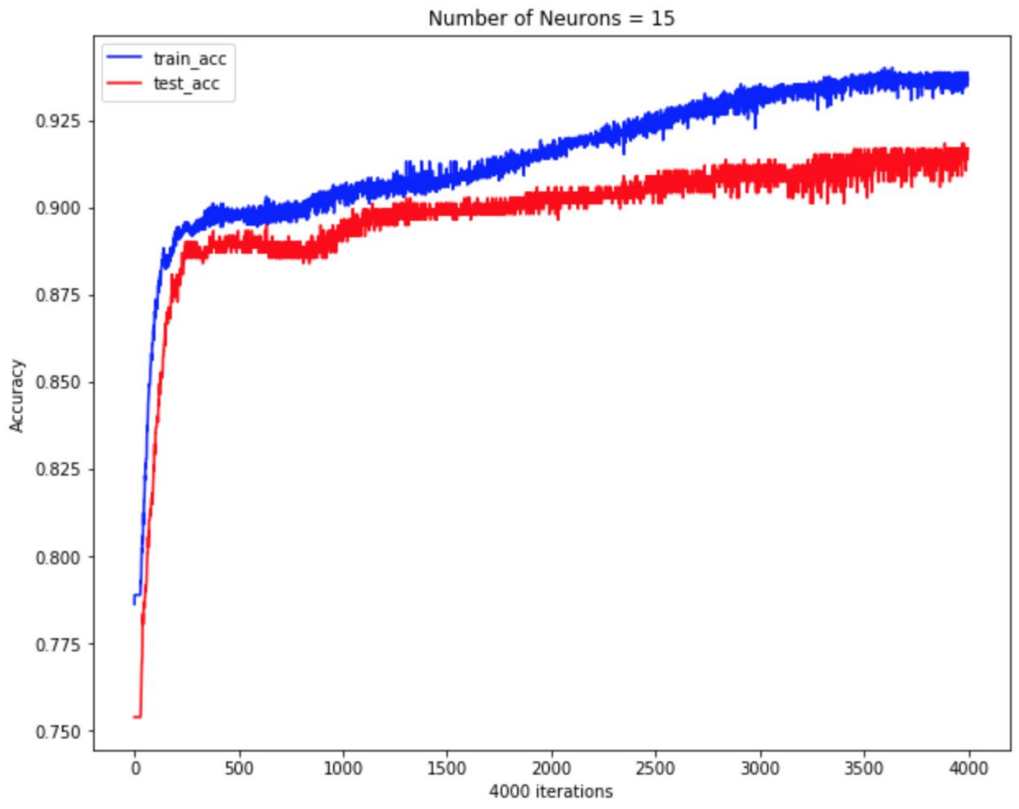
3. (a) Cross validation accuracy for different number of neurons



(b) The general pattern we can observe is that the validation accuracy of the model increases with the number of neurons in the hidden layer of the network. This is because with more neurons in the network, the model is able to gain a deeper knowledge on the data distribution and hence able to give a better estimate.

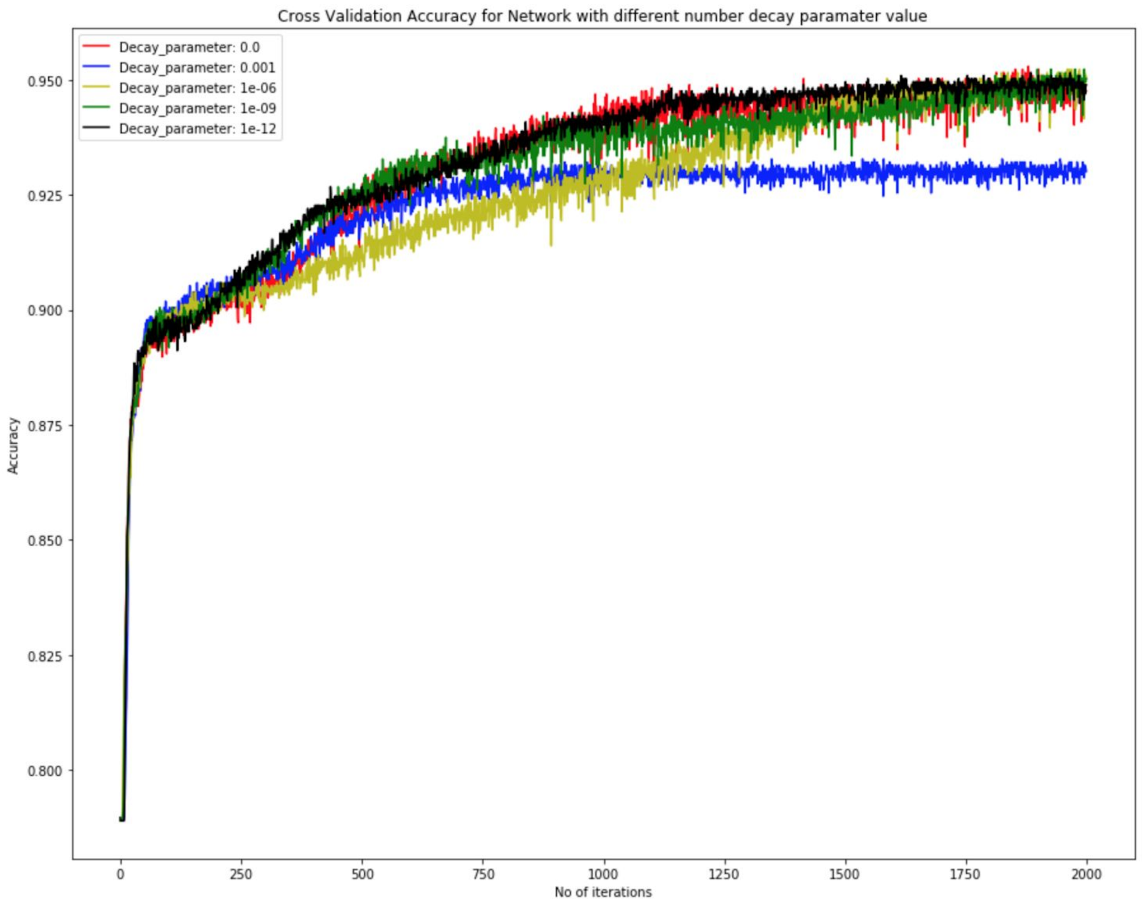
In choosing the optimal number of neurons,we will try to find a balance between having a decent training accuracy, and also a model that is not too complex such that it will overtfit. Since the validation accuracy the network with 5 and 10 neurons are much lower, we will select amongst 15,20 and 25. We will choose number of neurons= 15 as our optimal hyperparameters because not only did it achieve a high validation accuracy, it is also less complex compared to having 20 and 25 neurons. Hence, more robust to overfitting and able to generalise well to new unseen data.

(c) Train vs Test accuracy against epoch for optimal number of neurons = 15



From the train-test plot, we observe that the difference in accuracy between the train set and test set is < 2%, showing that the model is not being overfitted and is able to generalise well to unseen test set with an accuracy of ~90%.

4. (a) Cross validation accuracy for different decay parameter values

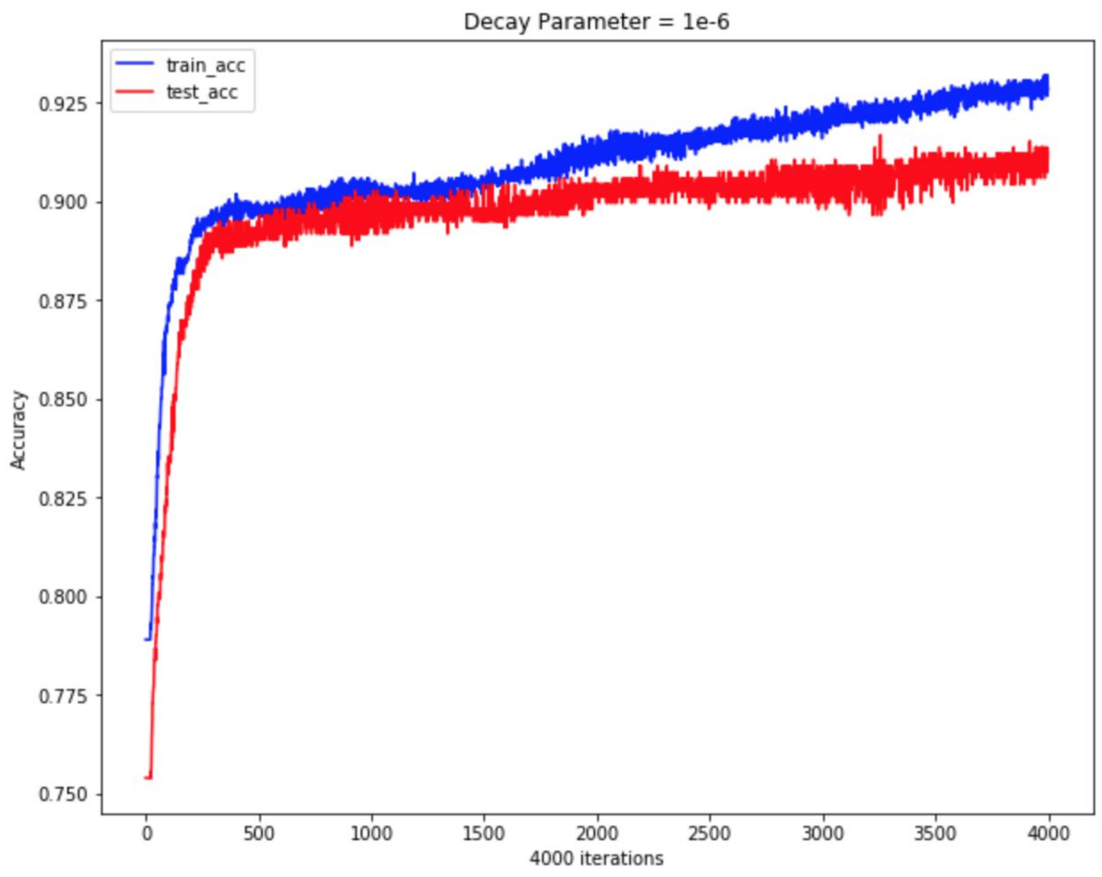


(b) Decay parameter will determine how much regularisation is applied to the model. A huge decay parameter will reduce the weights significantly, which will compromise the accuracy of the model. While too little regularisation will cause the model to overfit.

Hence in this case, we will need to find a balance between bias and variance when choosing our decay parameter. We can see that the highest value, 1e-3, will affect the performance significantly, with a much lower validation accuracy. When regularisation is applied, we can see that the validation accuracy is one of the highest, where the model could be overfitted to the training data since every example is seen by the model during training with 5-fold validation.

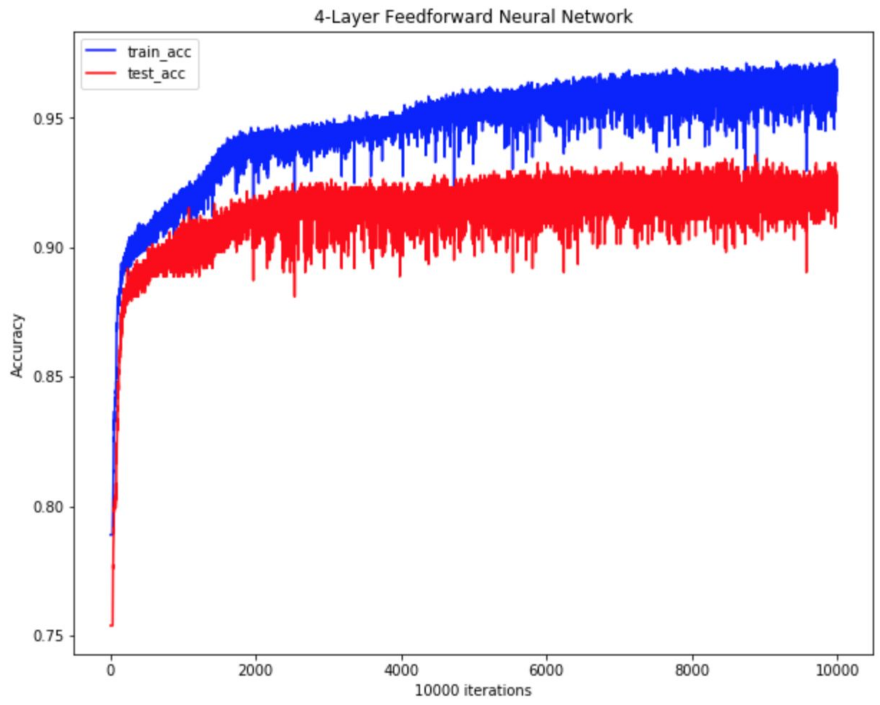
Since other values of decay parameters gives relatively similar validation accuracy, we will choose the one with the greatest value, which is **1e-6**, to ensure that the high validation accuracy is not due to overfitting by the model.

(c) Train vs Test accuracy against epoch for optimal decay parameter = 1e-6

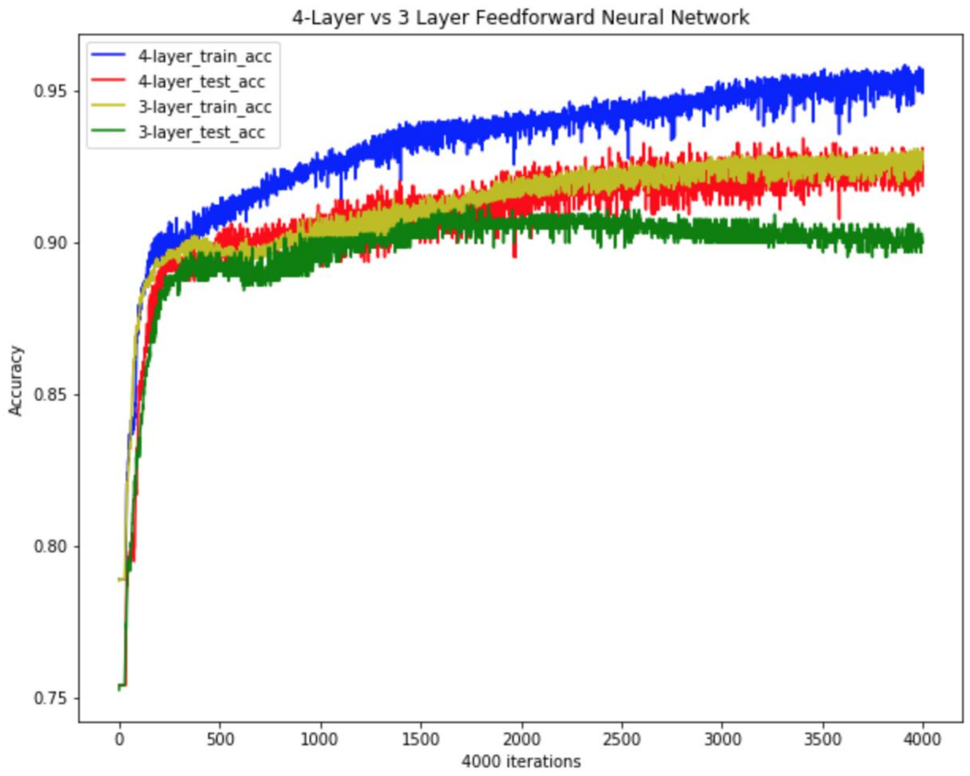


We can see that the difference in train accuracy and test accuracy is < 2%, hence, this decay parameter is optimal in generating a model which is highly accurate and capable of generalising well to new unseen data.

5. (a) Plot train and test accuracy of a 4-layer network



(b) Compare and comment on 3-layer vs 4-layer Neural network



Overall, both train and test accuracy of a 4-layer network is higher than the respective accuracy of the optimal 3-layer network. One reasoning could be because with more hidden layers in the network, the network is able to gain deeper knowledge of the data distribution and forms complex function to model the distribution better. Not only that, with more hidden layers, there will be more parameters to train, and more learning takes place with backpropagation. The increased learning helps the model to gain higher accuracy in terms of predicting the NSP classes.

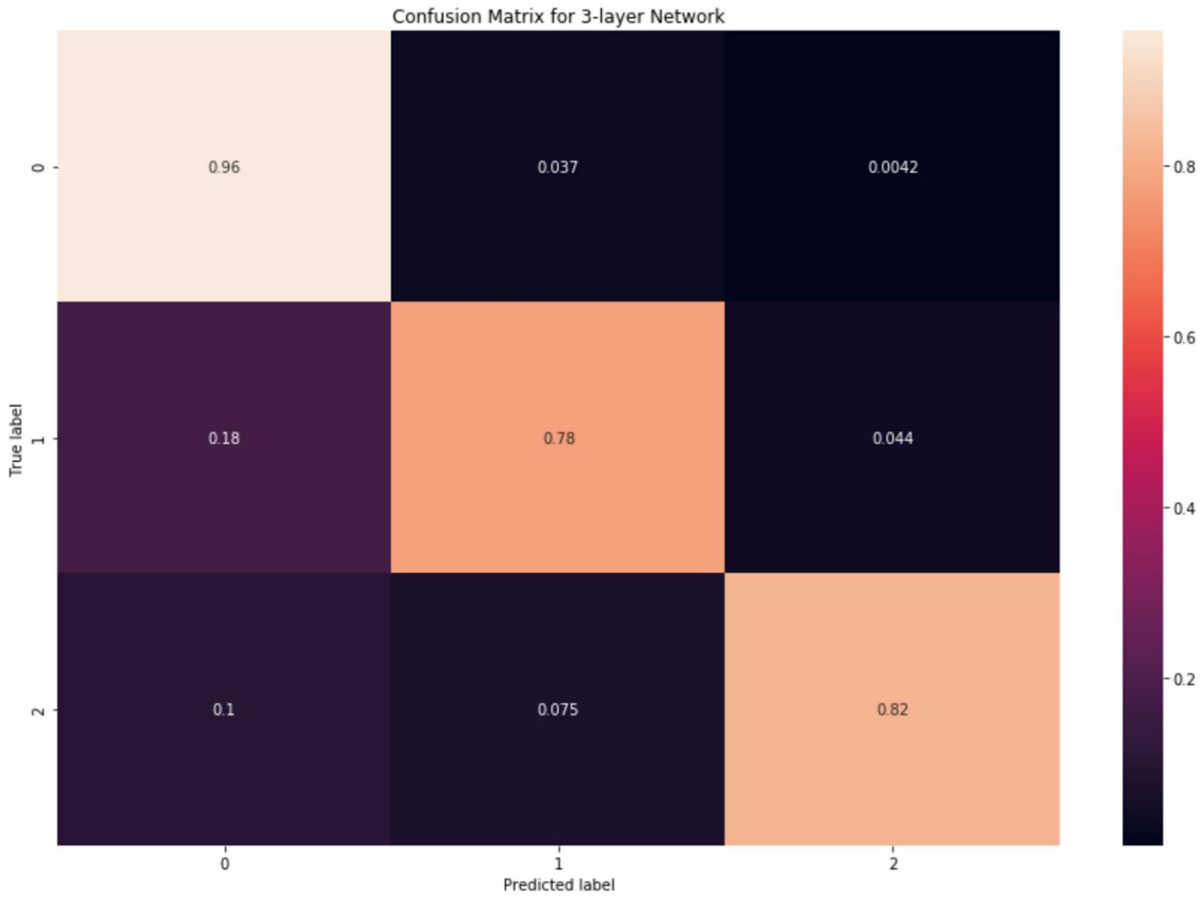
We can also see that with more hidden layers, there is more variance in the accuracy, due to the increased complexity of the model.

1.4 Conclusion

Conclusion
In conclusion, from our validation we have concluded with our final optimal 3-layer network: Batch size of 16, Number of neurons = 15, Decay parameter = 1e-6. And from the results, we are able to attain a model with high test accuracy of ~90%. With the addition of regularisation, the model we have trained is robust to overfitting, balancing well when it comes to the Bias Variance trade-off.
A 4-layer network also performs better than our optimal 3-layer network because of the additional hidden layer, which helps to generate a much more complex model that can predict the right NSP classes more accurately.

Experience
Based on my experience with this question, i realised the importance of choosing the right batch size when doing neural network. Given this small dataset, the effect is not significant. However, when we are dealing with big data, time is a huge factor as the batch size will affect the computation time significantly. With deep neural networks, we are able to solve many complex problems, especially if we have access to huge amount of data. However, it will also be harder for us to interpret the model behaviour.
Ultimately, the choice on the hyperparameters and the type of network architecture to choose from can vary based on the resources available, the timeline and also the objectives of the problem we are trying to solve.

Analysis on Imbalanced dataset
In our introduction, we have mentioned about the imbalanced distribution of classes in the dataset, which may affect our model as a whole. Using the optimal 3-layer network, we are able to come up with the following confusion matrix by predicting the classes against our test set.



We can see that the model is able to predict up Class 0 ('N' class) with up to 96% accuracy, while the prediction accuracy for 'S' and 'P' classes are much lower at 78% and 82% respectively. This is mainly because up to 78% of the data comes from class 0, which is 'N'. The model will be exposed to more examples of class 'N' during training. With more class 'N' training examples, the model will be able to extract more information about class 'N', resulting in higher accuracy. On the other hand, because both class 'S' and 'P' only takes up 22% of the data, the model may not be well trained to predict these classes.

Therefore, suggested solutions to better improve our methodology and model performance could be the following:
1. Get more data from other fetals belonging to class 'S' and 'P' to make the distribution of classes in the dataset more even.

2. Apply oversampling to our minority classes to increase their numbers in the dataset

3. Apply SMOTE to artificially synthesize these minority classes to achieve a more balanced dataset.

2 Regression Problem

2.1 Introduction

We are given a Graduate Admissions Predication dataset containing 7 variables that affects admission rate. The 7 variables are GRE score, TOEFL score, university rating, strengths of Statement of Purpose and Letter of Recommendation, undergraduate CGPA and research experience. Given this dataset, our objective is to build a neural network for regression that can predict the admission rate based on the 7 input variables given.

2.2 Methods:

The input variables are normalized before putting into the model for training. Normalization ensures that the features in the dataset are of the same scale. This ensures that all features are represented equally during training.

2.2.1 Functions:

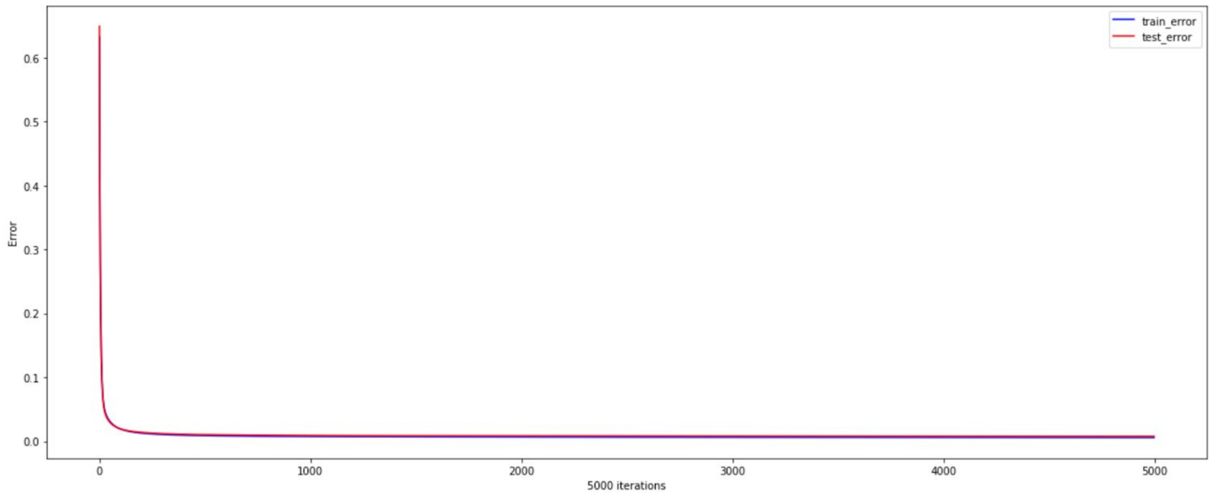
To ensure better viewability and reusability, functions are utilised.

- **datagen() :** Function that reads in the csv file, separate the dataset to X_data, Y_data that corresponds to the input variables and output. The function will reshapes the Y_data, splits the dataset into training and testing dataset of 70:30 ratio before normalizing the values.
- **ffn3(x, num_neuron):** Function that calculate the output for a 3-layer feedforward neural network based on the number of neurons provided
- **datagenRFE(X_datas, Y_datas, index):** Operated in the same manner as **datagen()** , with the additional parameter *index*, which determines which column to drop for the implementation of Recursive Feature Elimination
- **training (batch_size, num_neurons, beta):** Function that calculates the test and train error for each hyperparameter value range
- **trainingRFE (batch_size, num_neurons, beta, index,X_data, Y_data, num_features):** Function that calculates the test and train error for Recursive Feature Elimination
- **ffn3RFE(x, num_neuron, num_features):** Function that operates in the same manner as ffn3(), with the additional parameter of taking in the number of features
- **datagen_removal(index):** Operated in the same manner as **datagen()** , with the additional parameter *index*, which determines which column to drop
- **datagen_6features():** Operates in the same manner as **datagen()**, with the addition task of removing the column “Research”

- **ffn4() :** Function that calculate the output for a 4-layer feedforward neural network based on the number of neurons provided. A boolean parameter *dropout* determines whether dropout would be introduced with a keep probability of *keep_prob*
- **ffn5() :** Function that calculate the output for a 5-layer feedforward neural network based on the number of neurons provided. A boolean parameter *dropout* determines whether dropout would be introduced with a keep probability of *keep_prob*

2.3 Experiments and Results

1(a)



A 3-layer feedforward neural network was implemented with a hidden layer of 10 neurons having ReLU activation functions, using mini-batch gradient descent with a batch size of 8. L2 regularization was also implemented at weight decay parameter of 10-3 and learning rate of 10-3.

The train and test error of the aforementioned model can be seen from the figure shown above. It can be observed that both the train and test error converges very early on during the training process with 10,000 epochs.

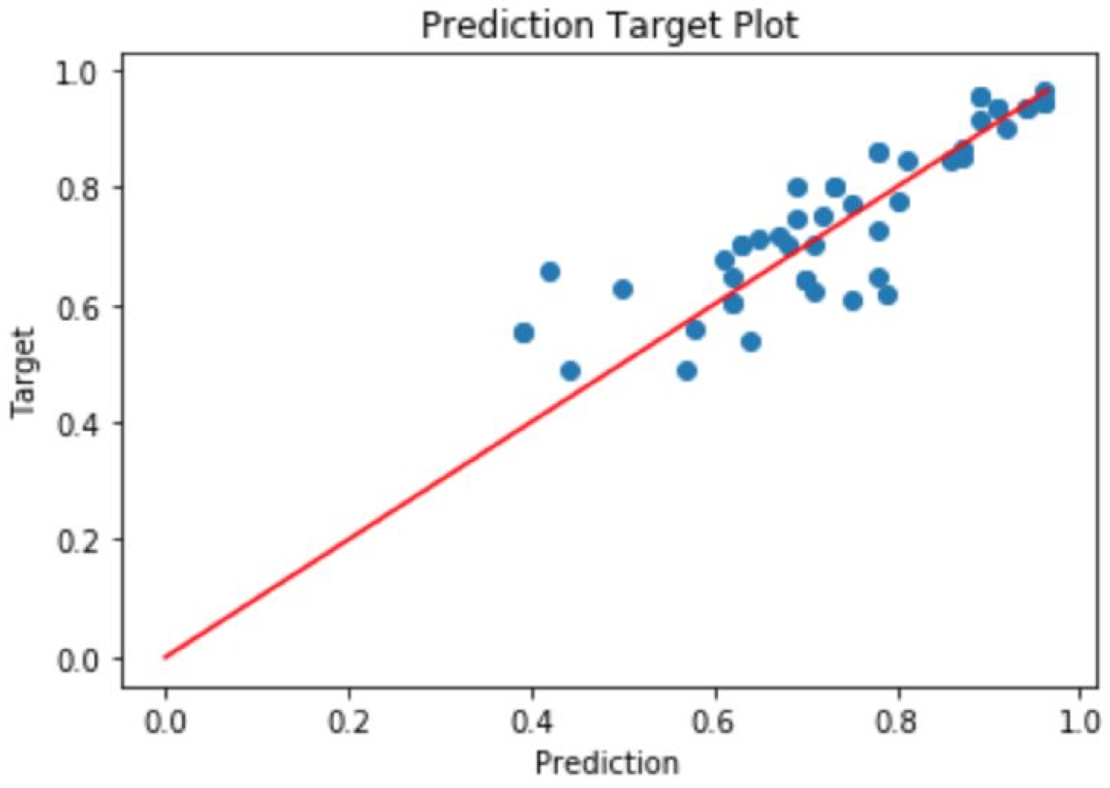
1b)

Patience: number of epochs that produced the monitored quantity with no improvement after which training will be stopped.

The challenge in 1(b) is to train the model long enough to ensure that it is able to generalise well and not so long that it overfits the training data. Hence, we decided to implement EarlyStopping where we determine the conditions whereby the model will stop training. Since we were training with 10,000 epochs, we decided to set the **patience** to be 50 epochs. It is considered an improvement if the *test_loss* registered for an epoch is less than 90% of the lowest recorded *best_test_loss* up till that point, and the *best_test_loss* would be replaced with the lower *test_loss*. The training would stop only when there is no recorded improvement in the *best_test_loss* after 50 epochs.

161 epochs is the number of epoch whereby no further improvement is projected to occur. This coincides with the graph plotted in question 1a, whereby the model do not show any significant improvement in its test and train error very early on during training.

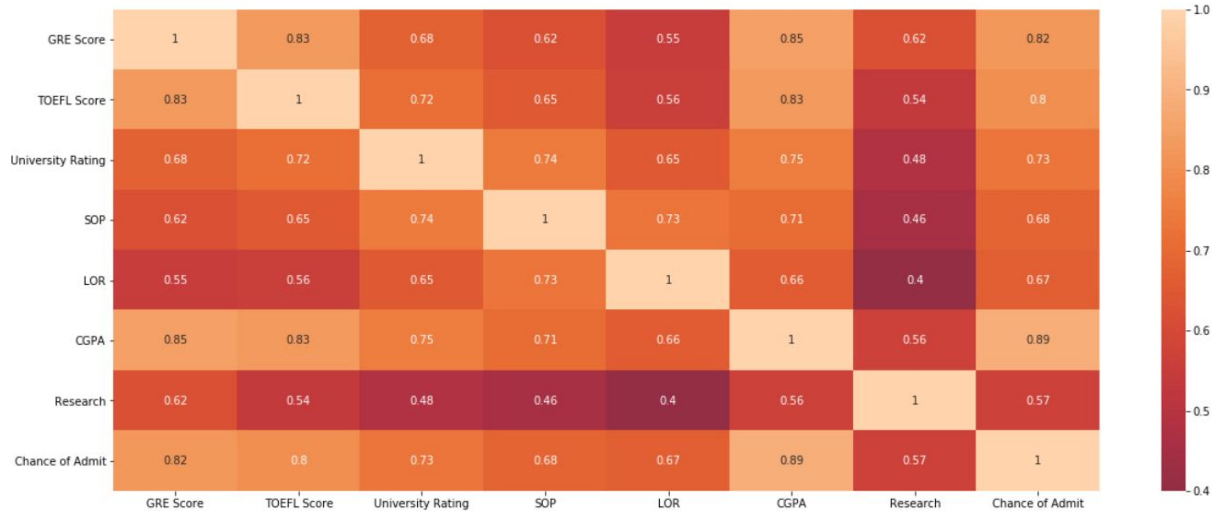
1c)



The Figure shown above shows the predicted values and target values for any 50 test samples when using the feedforward 3-layer neural network implemented in question 1a.

2a)

The figure below shows the 8x8 correlation matrix between the feature scores and corresponding “chance of admit”



	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1	0.83	0.68	0.62	0.55	0.85	0.62	0.82
TOEFL Score	0.83	1	0.72	0.65	0.56	0.83	0.54	0.8
University Rating	0.68	0.72	1	0.74	0.65	0.75	0.48	0.73
SOP	0.62	0.65	0.74	1	0.73	0.71	0.46	0.68
LOR	0.55	0.56	0.65	0.73	1	0.66	0.4	0.67
CGPA	0.85	0.83	0.75	0.71	0.66	1	0.56	0.89
Research	0.62	0.54	0.48	0.46	0.4	0.56	1	0.57
Chance of Admit	0.82	0.8	0.73	0.68	0.67	0.89	0.57	1

According to the Figure shown above, the features most correlated to each other are as follows:

GRE Score & TOEFL Score = 0.83

GRE Score & CGPA = 0.85

TOEFL Score & CGPA = 0.83

TOEFL test is a highly respected English-language proficiency test. **GRE** is a standardized test that is an admission requirement for many graduate schools in the United States and Canada. **GRE** test mainly measures verbal reasoning, quantitative reasoning, analytical writing and critical thinking skills that consists of algebra, geometric vocabulary questions. To obtain a high score on the **GRE**, it is presumed that the English proficiency of exam-takers are of decent level as **GRE** has harder readings and writing sections compared to **TOEFL.** As such, a high correlation between **GRE** and **TOEFL** scores are justificable.

CGPA is the cumulative grade point average in college, which is calculated as an average of the **GPA** obtained from all completed semesters. The examinations in college is structured to test the students’ understanding of core concepts. This is largely similar to what the **GRE** is testing, hence the high correlation between the two variables are justifiable.

The assumption is made that the dataset is retrieved from students who studies in English-speaking colleges. The high correlation between **TOEFL** scores and **CGPA** is therefore justifiable because to score well in **CGPA,** a decent level of understanding of the questions(written in English) needs to be established.

2b)

CGPA, **GRE** and **TOEFL** scores have a relatively high correlation with “chances of admit” with all of them scoring above 0.8.

3)

According to the correlation matrix, we would assume that “Research” be dropped, before “SOP” or “LOR” due to its low correlation score with “chance of admit”. When implementing the Recursive Feature Elimination, result shows that 6 input features produces the lowest error and the input variable to be removed is “Research”. This substantiates our initial hypothesis of features with low correlation with the target being less relevant when used as feature in training a neural network.

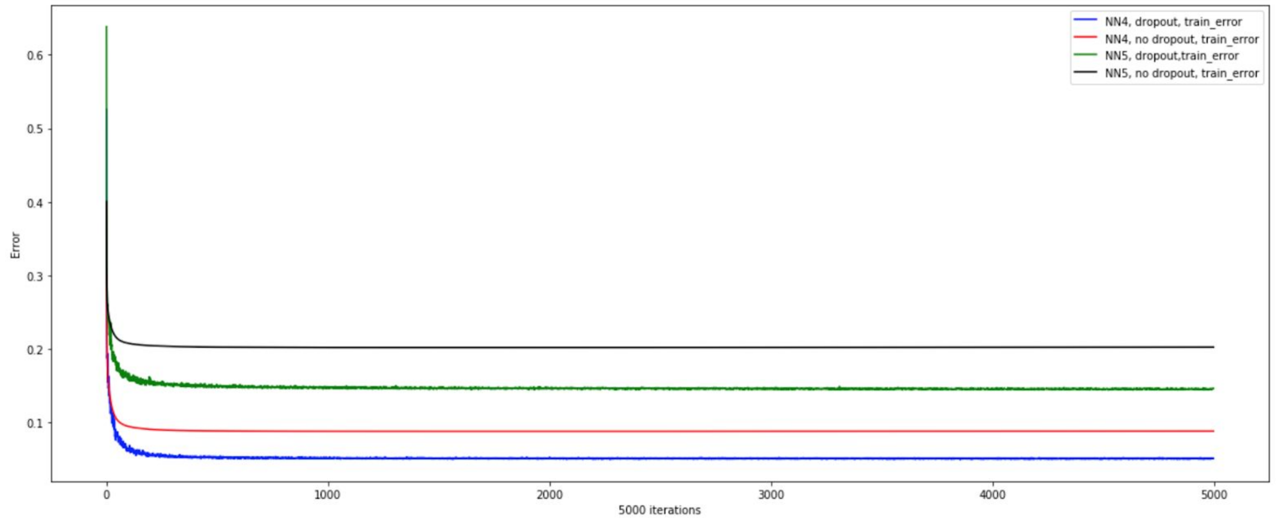
Mean Squared Error (MSE)		
7 Features	**6 Features**	**5 Features**
0.00751284	0.00952884	0.0122634

What we can gather from the results is that the neural network performs better when there are more features to extract information from. Although some of the features might not generate a lot of useful information, it is still improving the model’s performance, although not significant. Hence, the idea of feature engineering may not be significant in the context of deep learning, albeit the importance it has in the area of machine learning.

However, the problem with using all the features will be the extra numbers of parameters to train for each additional features that are not relevant to the model, which leads to inefficient and irrelevant computations.

4)

Mean Squared Error (MSE)				
3-layer	**4-layer**	**4-layer (dropout)**	**5-layer**	**5-layer (dropout)**
0.0079	**0.0879**	**0.05**	**0.202**	**0.144**



The Figure above shows the 4-layer neural network and 5-layer neural network, with and without dropouts implemented.

Dropout freezes random neurons during training, forcing the other neurons within the layer to learn the inputs better. Dropout ensures that situations whereby the network layers co-adapt to correct mistakes from previous layers does not occur, making the model more robust. From the figure displayed above, it can be observed that those models with dropout of *keep_prob* = 0.8 implemented display a lower error rate than those without.

From the table shown above, the increase in the number of hidden layers in a neural network results in a higher mean-squared error. One possible explanation for such occurrence may be due to the small dataset involved in training, leading to overfitting of training data.

2.4 Conclusion

In conclusion, we observed that 161 epoch is the approximate number of epochs where the test error is minimum. From the plotting of the correlation matrix between the 7 input features and output, ‘chance of admit’, we came up with the hypothesis that “Research”, followed by either “SOP” or “LOR” feature should be removed due to their poor correlation with “chance of admit”. The implementation of Recursive Feature Elimination, further substantiates our conjecture and we removed “Research” feature, resulting in 6 input features, which is the optimal number of features that returns the minimum error during training.

Finally, we observed that the introduction of dropout helps in reducing the mean-squared error compared to models without dropout. Furthermore, it is noticed that more hidden layers does not corresponds to a lower error rate, during the implementation in this question. This might be due to the fact that a small dataset is involved, resulting in overfitting.

2.5 Experience

Through this project, we realised that the number of input features is proportionate with the accuracy of a model. However, this is contrary to what we were taught. One explanation for this occurrence could be due to the small dataset (400 rows) used. Finally, through this project, we managed to implement EarlyStopping and Recursive Feature Elimination, which aids in our understanding of these functions.