NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CE/CZ4042
Neural Networks and Deep Learning
Assignment 1 Report

SCSE Yew Wei Chee
U1820962G

Assignment 1 Report

CE/CZ4042 Neural Networks and Deep Learning

Yew Wei Chee

INTRODUCTION

This assignment is to help us better understand the neural networks and its components. In part A, a classification model is built to predict the fetal state based on 21 input features related to the fetal cardiotocograms (CTGs). Several experiments are carried out on the hyperparameters of the model such as the batch size, the number of neurons, the number of layers and the regularization weight decay parameters, to see how different hyperparameters may affect the training result of neural networks. In part B, a regression model is built to predict the chance of getting admit. We also look at how different combination of input features and the introduction of dropout layers may affect the training result of neural networks.

1. Methods

Several method that are used in this assignment are explained below.

1.1 Dataset Preparation and Preprocessing

In this assignment, the *'train_test_split'* method from *'scikit-learn'* [1] is used to split the dataset into train dataset and test dataset. The ratio of test dataset used is 0.3.

Besides, to improve on the neural network stability, the inputs to the model are scaled using *'MinMaxScaler'* from *'scikit-learn'* to the scale of 0 to 1. It is important to scale all the input features to the same scale to prevent problems that may arise, for instance, the models only learn features with large values, leading to exploding gradients and unstable weight updates. Scaling input features can also help models to treat all input features equally and speed up the learning process.

1.2 K-Fold Cross Validation

K-Fold cross validation is a technique used to assess the performance of a model. Using K-Fold cross validation with k=5, the dataset is partitioned into 5 folds. When training, 5 similar models are created, each of them is trained by choosing one of the different 5 folds as the validation dataset and the remaining 4 folds as the training dataset. The performance of the model is then assessed using the average loss and accuracy of the 5 models.

To perform K-Fold cross validation, the *'KFold'* class in *'scikit-learn'* is used.

1.3 Optimizer and Loss Function

In this assignment, SGD optimizer is used. The SGD optimizer in Tensorflow [2] uses mini-batch gradient descent algorithm to update the model weights. The loss function used for part A is the sparse categorical cross entropy whereas for part B is mean squared error.

1.4 Activation Function

In all hidden layers, the ReLU activation function is used. For the classification model in part A, softmax activation function is used on the final layer to model the probability distribution over the predicted output classes (N, P and S). In part B, the activation function used on the final layer of the regression model is the sigmoid activation function, which is to fit the output within the range of 0 to 1.

1.5 L2 Regularization

In this assignment, we will also experiment on the weight decay parameter of the L2 regularizer. L2 regularization is used to prevent overfitting by adding a penalizing term onto the loss function as shown below:

$$J(w) = Loss + \frac{\lambda}{2}\|w\|^2$$

λ is the weight decay parameter, which is a hyperparameter that is to be optimized for better training results.

2. Part A: Classification Problem

A classification model is built to classify the fetal state based on the features of cardiotocograms. The dataset used is the Cardiotocography dataset [3] available from UCI Machine Learning Repository which consists of 2126 cardiotocogram features and their corresponding fetal state. We will experiment on how different batch sizes, number of neurons, weight decay parameters and number of neural network layers affect the training results.

Firstly, a feedforward neural network is created, which consists of one hidden layer of 10 neurons with ReLU activation function and an output softmax layer. A learning rate $\alpha = 0.01$, weight decay parameter $\beta = 10^{-6}$ and batch size of 32 is used. The accuracies and loss on training and testing data are shown in Figure 2(a) and Figure 2(b).

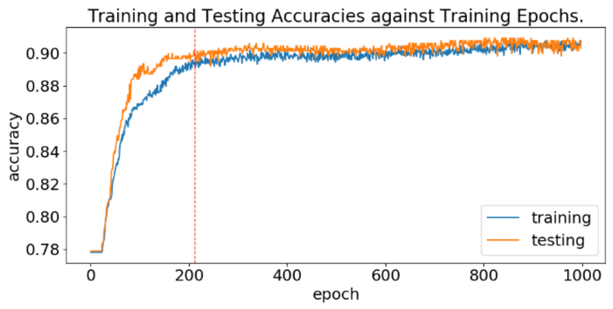


Figure 2(a): Training and testing accuracies against epochs.

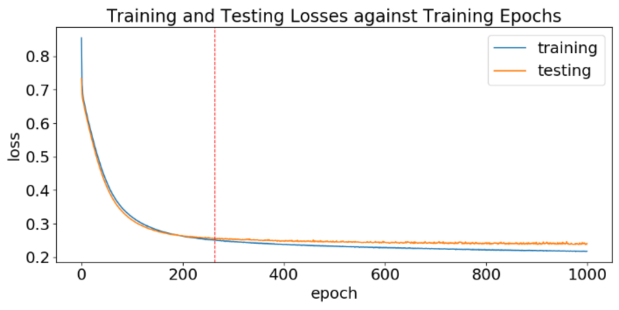


Figure 2(b): Training and testing losses against epoch.

From Figure 2(a) and Figure 2(b), we can see that the test accuracy started to converge around 200 epochs and the test loss started to converge around 250 epochs.

2.1 Batch Size

Next, we compare the performance of the model over different batch sizes {4, 8, 16, 32, 64} using K-Fold cross validation. The cross validation accuracies for different batch sizes are shown in Figure 2.1(a).

We can observe that as the batch size increases, the number of epochs to converge increases, whereas the fluctuations of cross validation accuracy over epochs decreases.

The timing information per epoch is given in Figure 2.1(b) and the final cross validation accuracy for different batch sizes is given in Figure 2.1(c).

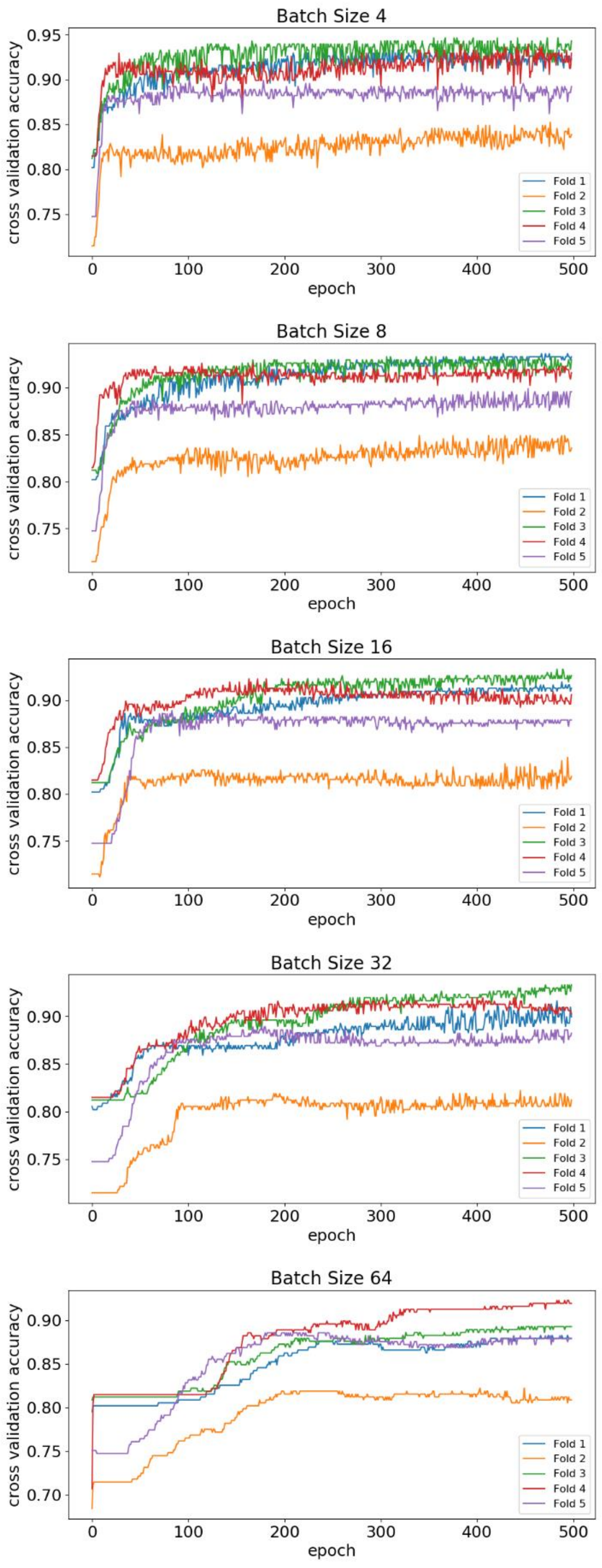


Figure 2.1(a): Cross validation accuracy for different batch sizes.

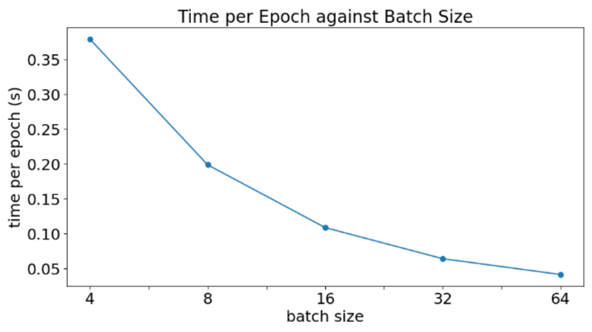


Figure 2.1(b): Time per epoch for different batch sizes.

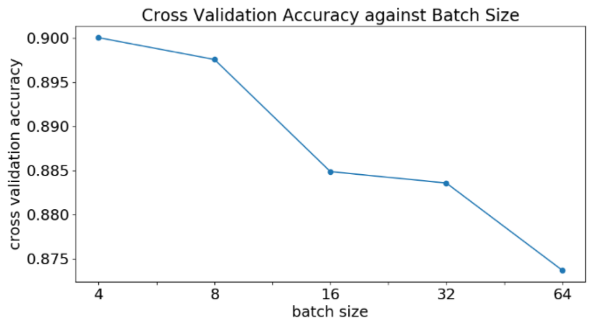


Figure 2.1(c): Cross validation accuracy for different batch sizes.

From Figure 2.1(b), we can see that the time taken per epoch decreases as batch size increases. From Figure 2.1(c), we can also see that the cross validation accuracy decreases as batch size increases. This means, choosing an optimal batch size is the trade off between training time and accuracy.

From the graphs above, I have chosen my optimal batch size as 8 due to the following reasons.

- The cross validation accuracy is high enough, just slightly lower than batch size 4, and much higher than other batch sizes.
- The training time reduced significantly (halved) compared to batch size 4 and reduce gradually less for batch size greater than 8.
- The fluctuations of cross validation accuracies is good enough, not as vigorous as batch size 4 and almost similar to batch size 16.

The train and test accuracies for the optimal batch size 8 is shown in Figure 2.1(d).

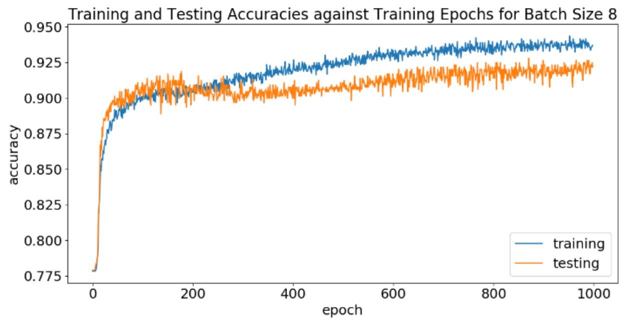


Figure 2.1(d): Train and test accuracies for optimal batch size 8 over 1000 epochs.

2.2 Number of Neurons

Using the optimal batch size obtained previously, we continue to experiment on the effect of number of neurons in the hidden layer on the training results. The number of neurons that has been tried out are {5, 10, 15, 20, 25}. The cross validation accuracies for different number of neurons over each training iteration are shown in Figure 2.2(b). The curve is smoothed for better visualization.

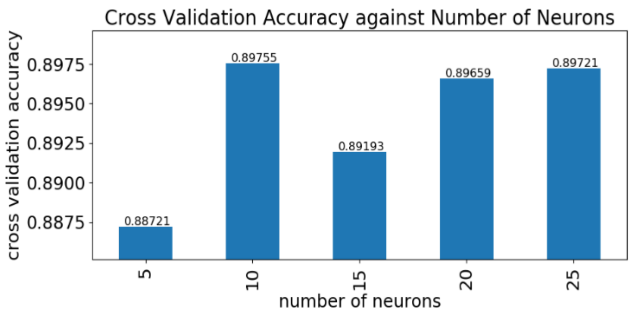
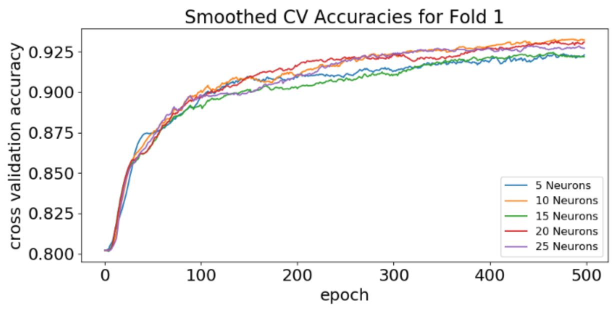


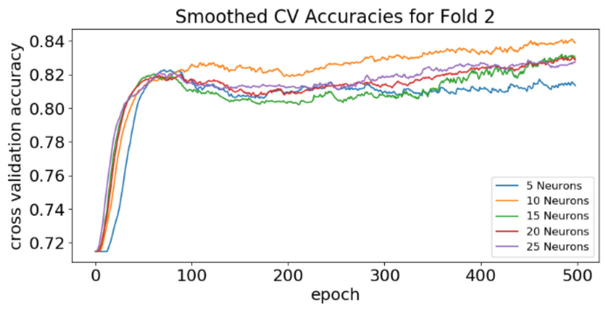
Figure 2.2(a): Cross validation accuracies for different number of neurons.

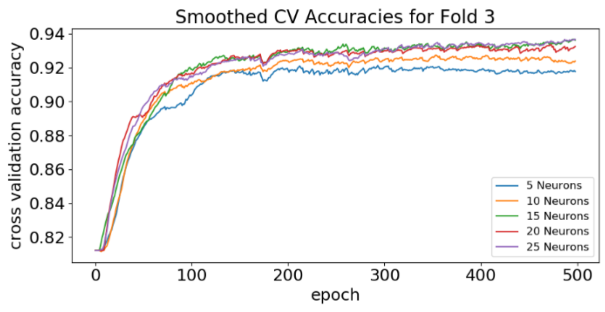
Number of Neurons	Cross Validation Accuracy
5	0.88721
10	**0.89755**
15	0.89193
20	0.89659
25	0.89721

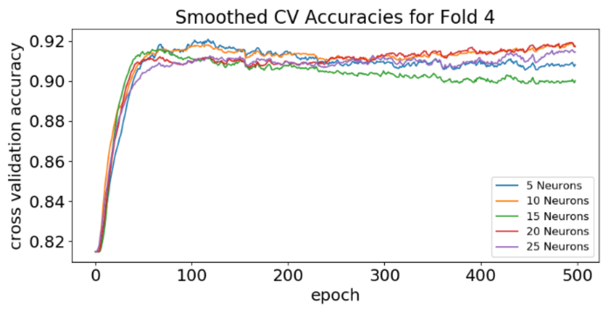
Table 2.2(a): Cross validation accuracies for different number of neurons.

From Table 2.2(a), the number of neurons that yields the highest cross validation accuracy is 10. Therefore, 10 is chosen as the optimal number of neurons for the hidden layer.









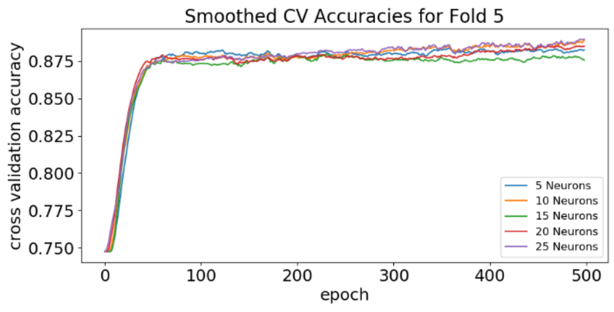


Figure 2.2(b): Smoothed cross validation accuracies for different number of neurons over 5 training iterations.

The train and test accuracies for the optimal number of neurons (10) is shown in Figure 2.2(c).

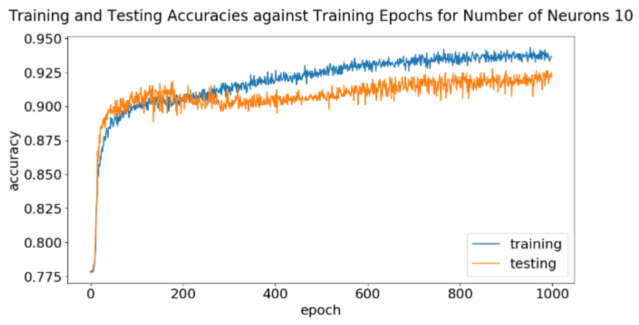


Figure 2.2(c): Train and test accuracies for optimal number of neurons 10 over 1000 epochs.

2.3 Weight Decay

Using the optimal batch size and optimal number of neurons obtained previously, next, we will find the optimal weight decay parameter from {0, 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}}. The cross validation accuracies for different weight decay parameters over each training iteration are shown in Figure 2.3(b). The curve is also smoothed for better visualization.

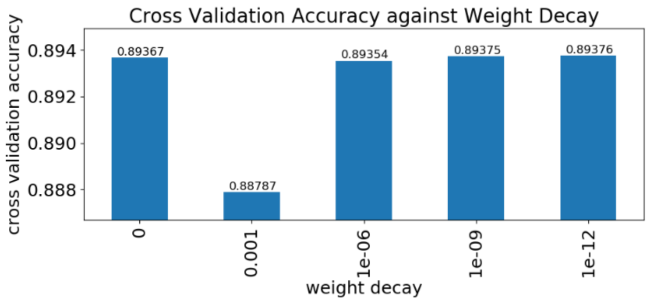


Figure 2.3(a): Cross validation accuracies for different weight decay parameters.

Weight Decay	**Cross Validation Accuracy**
0	0.89367
10^{-3}	0.88787
10^{-6}	0.89354
10^{-9}	0.89375
10^{-12}	**0.89376**

Table 2.3(a): Cross validation accuracies for different number of neurons.

From Table 2.3(a), we can see that the cross validation accuracy for weight decay 10^{-9} and 10^{-12} are almost the same, and also the highest among all. The model trained using weight decay parameters 10^{-3} and 10^{-6} are considered slightly underfitting as the performance is slightly lower than the model trained without using L2 regularization. The optimal weight decay chosen is 10^{-12}.

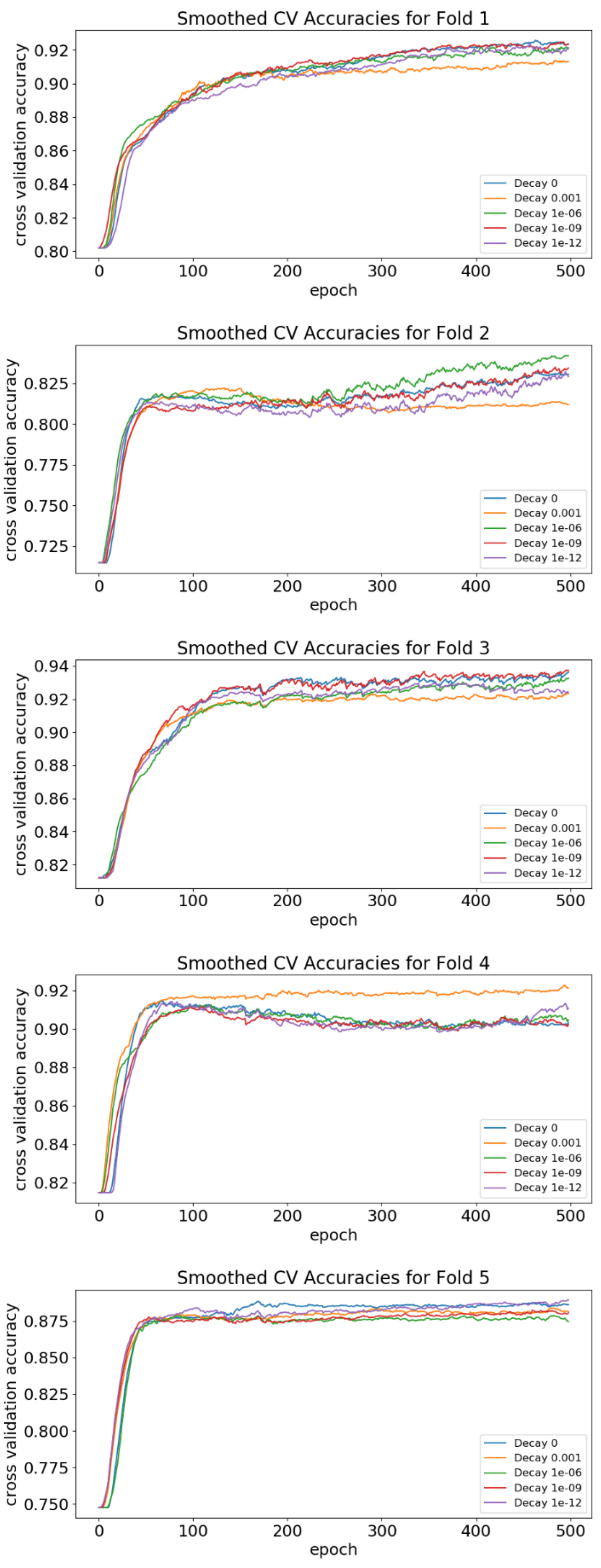


Figure 2.3(b): Smoothed cross validation accuracies for different weight decay parameters over 5 training iterations.

From the bar chart in Figure 2.3(a), we can notice that the cross validation accuracy is almost similar for weight decay 10^{-6} and below and the weight decay of 10^{-3}, this implies that there is not much regularization needed for this 3 layer network. This may be due to the shallowness of the neural network we used. We are only using a 3-layer neural network with 10 neurons in the hidden layer, which is a very simple neural network, therefore, there is not much overfitting despite the small dataset.

Using the optimal weight decay, the train and test accuracies are shown in Figure 2.3(c).

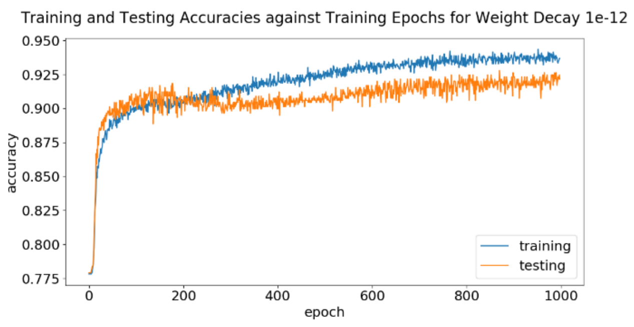


Figure 2.3(c): Train and test accuracies for optimal weight decay parameter 10^{-12} over 1000 epochs.

2.4 4-Layer Network

A 4-layer network is created with 2 hidden layers, each consisting of 10 neurons and trained with a batch size of 32 and a weight decay parameter of 10^{-6}.

	3-Layer Network	4-Layer Network
Number of neurons in each hidden layer	10	10
Batch size	8	32
Weight decay	10^{-12}	10^{-6}

Table 2.4(a): The comparison of 3-layer and 4-layer network.

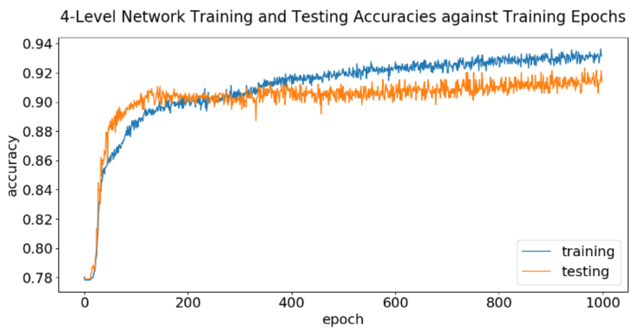


Figure 2.4(a): Train and test accuracies for 4-layer network over 1000 epochs.

From Figure 2.4(b), the optimal 3-layer network and 4-layer network have similar test accuracies around 400 epochs. After that, the optimal 3-layer network slightly outperforms the 4-layer network with a slightly higher accuracy than the 4-layer network.

There can be 2 contrasting reasons that causes the performance of the 4-layer neural network to be lower than the 3-layer neural network.

Firstly, this might be due to the overfitting of the 4-layer neural network. As the dataset size that we use only consist of 2126 data samples, it is a considerably small dataset. A 4-layer neural network in this case might be slightly complex in modelling this small dataset. If this is the case, then we can try to increase the regularization weight decay or introduce dropout to better generalize the 4-layer neural network.

Secondly, this might be because of the regularization weight decay is slightly too high, which causes the 4-layer model to be underfitting. If this is the case, we can reduce the regularization weight decay to prevent underfitting.

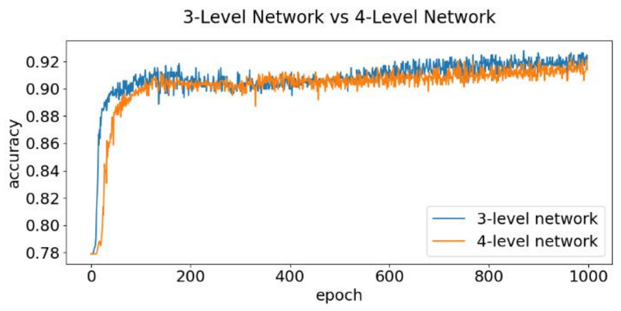


Figure 2.4(b): The test accuracy for the optimal 3-layer network and 4-layer network.

	3-Layer Network	**4-Layer Network**
Test Loss	**0.22496**	0.22917
Test Accuracy	**0.91844**	0.91161

Table 2.4(b): The performance of the optimal 3-layer network and 4-layer network.

To test out what is the reason that causes the 4-layer network to perform worse than the 3-layer network, a new 4-layer model is created and trained with the same batch size and weight decay as the optimal 3-layer network, the results are shown in Figure 2.4(c) and Table 2.4(c).

We can see that after reducing the weight decay to 10^{-12}, the accuracy of the modified 4-layer network increases. Therefore, we can conclude that the previous 4-layer neural network is underfitting.

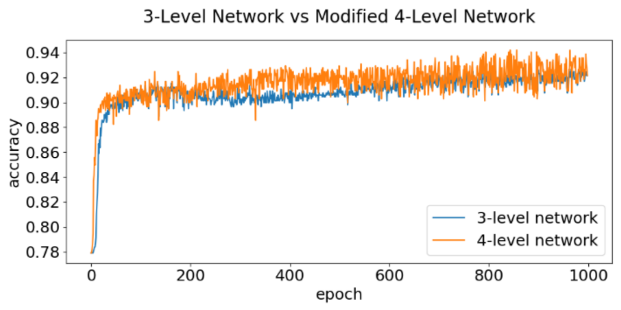


Figure 2.4(c): The test accuracy for the optimal 3-layer network and the modified 4-layer network

	3-Layer Network	**Modified 4-Layer Network**
Test Loss	**0.22496**	0.23403
Test Accuracy	0.91844	**0.92587**

Table 2.4(c): The performance of the optimal 3-layer network and the modified 4-layer network.

We can also notice that the test loss increases together with test accuracy, this is because the model manages to find possible outliers of the data sample and fits itself towards the non-outliers of the data sample. Therefore, the accuracy increases because more non-outliers are predicted correctly and the loss increases because the prediction on the outliers deviates away more.

3. Part B: Regression Problem

A regression model is built to predict the probability of getting an admit using data from the Graduate Admissions Predication [4] which consists of only 400 data samples. The features used to predict the chance of admit includes GRE score, TOEFL score, university rating, SOP, LOR, CGPA and research. The heatmap of correlation of variables are shown in Figure 3(a).

From the heatmap in Figure 3(a), we can observe that the correlation between CGPA, GRE score and TOEFL score is quite high (greater than 0.8). Whereas the correlation between CGPA, university rating, SOP and LOR are moderately high (0.65 – 0.75). The input feature 'research' is less correlated to other input features (less than 0.6).

Firstly, a feedforward neural network is created, which consists of one hidden layer of 10 neurons with ReLU activation function and an output sigmoid layer. A learning rate $\alpha = 10^{-3}$, weight decay parameter $\beta = 10^{-3}$ and batch size of 8 is used. The mean squared error on training and testing data are shown in Figure 3(b).

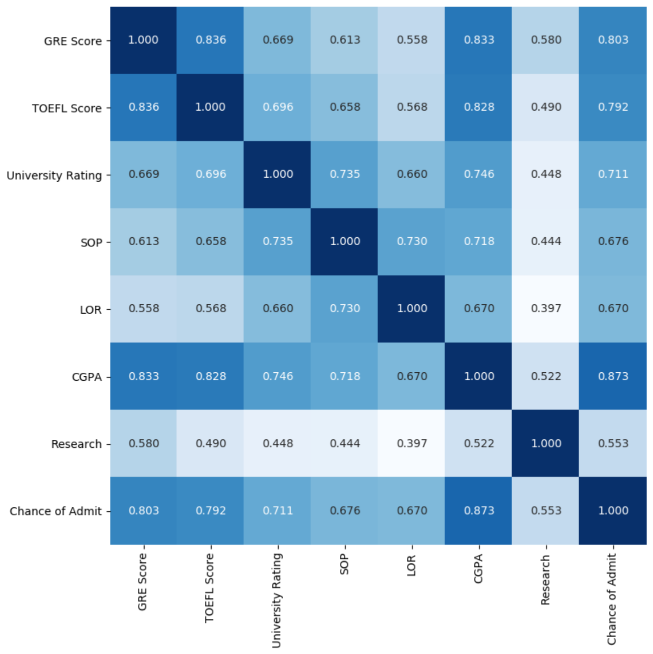


Figure 3(a): Correlation of variables in the Graduate Admissions Predication dataset.

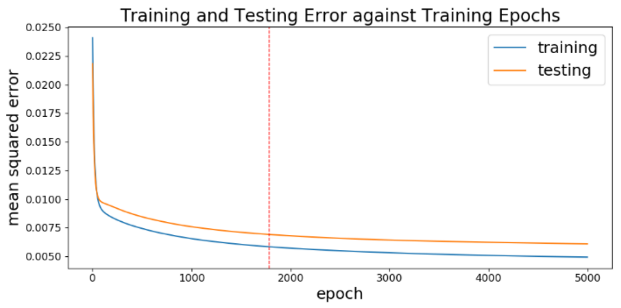


Figure 3(b): Train and test mean squared error against epochs.

From the graph in Figure 3(b), we can see that the test mean squared error started to converge around 1800 epochs. The plot of predicted values and target values for 50 test samples is shown in Figure 3(c).

The red dotted line in Figure 3(c) is the line where the predicted values are equal to the real values. The closer the data point is to the line, the lower the mean squared error of the data point. We can see that most of the data points lies close to the line, with an error of around 0.1 from the real values. From the graph in Figure 3(b), the test mean squared error converged to around 0.006, therefore the root mean squared error is around 0.077, which infers the observation from Figure 3(c).

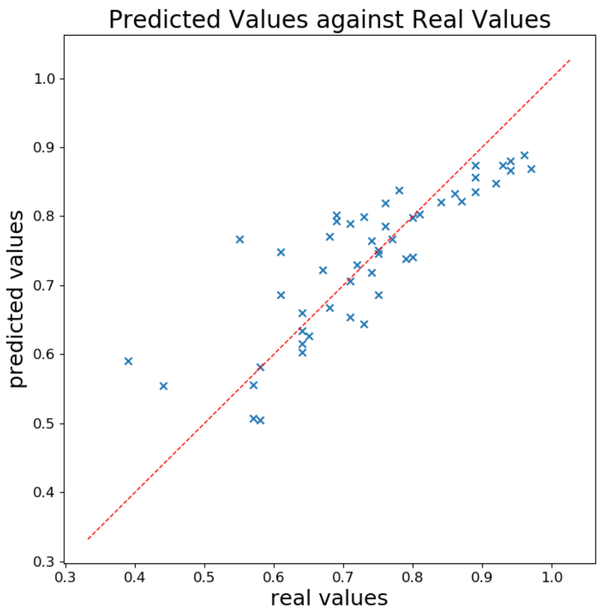


Figure 3(c): Plot of predicted values and real values for 50 test samples.

Next, we will use recursive feature elimination (RFE) to choose the optimal feature set, and then compare the model performance of models with different number of hidden layers and with or without dropout layers.

3.1 Recursive Feature Elimination (RFE)

Recursive feature elimination is used to select the number of desired features in the dataset which are most relevant in predicting the target variable. The algorithm works as follows:

1. Remove one feature at a time, then retrain the model and observe the test mean squared error after removing that feature. The least important feature can be obtained by choosing the feature that yields the lowest test mean squared error after removing it.
2. Remove the feature from the set of features, then repeat step 1 using the reduced set of features. This process continues until the desired number of features is obtained.

There are 7 input features in the original dataset. Firstly, we remove 1 feature at a time and retrain the model. The mean squared error of test dataset for each of the model is plotted in Figure 3.1(a). The set of optimal 6 input features that yields the lowest mean squared error is {GRE score, SOP, LOR, university rating, CGPA and research}. Therefore, TOEFL score is removed.

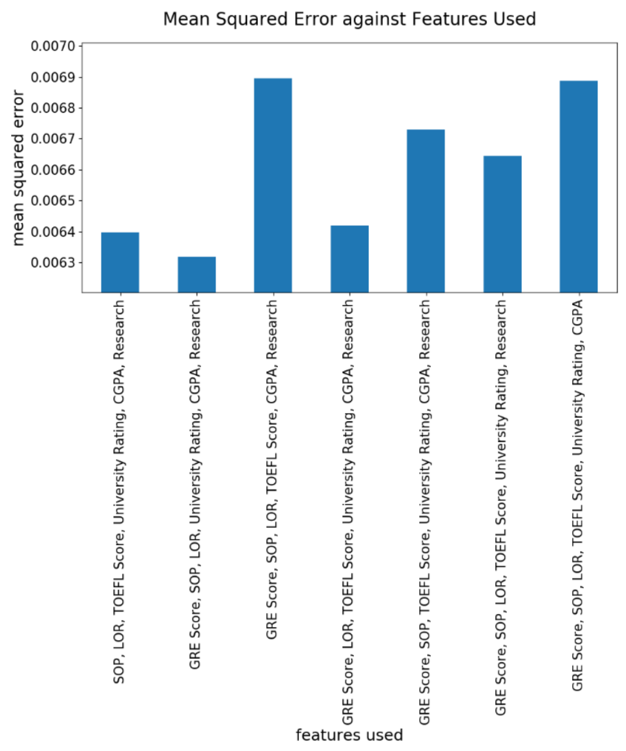


Figure 3.1(a): The test mean squared error of model using 6 input features

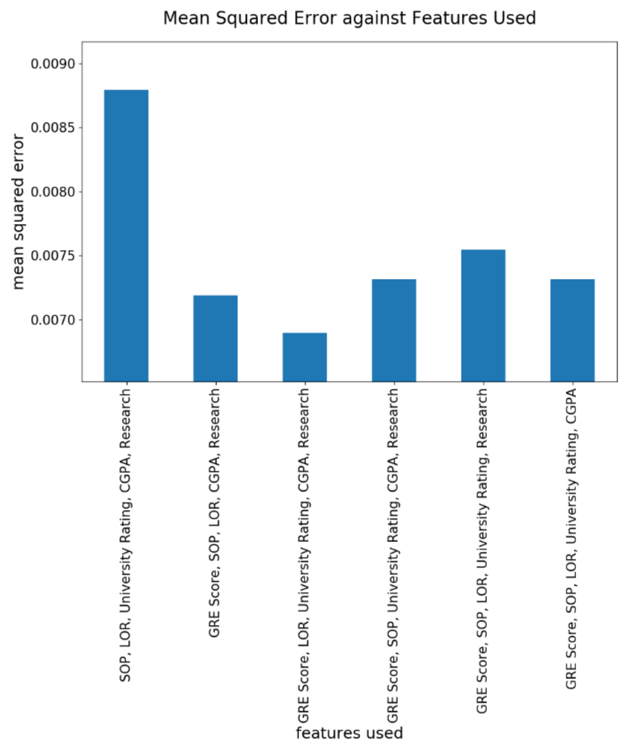


Figure 3.1(b): The test mean squared error of model using 5 input features.

Using the set of optimal 6 input features above, we continue to remove 1 feature at a time and retrain the model. The mean squared error of test dataset for each of the model trained using only 5 input features are plotted in Figure 3.1(b). The set of optimal 5 input features that yields the lowest mean squared error is {GRE score, LOR, university rating, CGPA and research}. Therefore, SOP is removed.

We can notice that even though from Figure 3(a), the correlation between 'TOEFL score' and 'chance of admit' is quite high, which is around 0.792, however, it is chosen as the first feature to be removed. This might be due to the multicollinearity of TOEFL score, GRE score and CGPA. These three features are considered highly correlated to each other with correlation coefficient of around 0.8. Therefore, we can say that these 3 features are representing the similar things.

There are 2 effects of multicollinearity on neural network. Firstly, on updating the weights, the gradient is calculated by assuming independence of weights among each other. However, since these 3 features are correlated, the weights applied onto these 3 features are no longer independent of each other. Therefore, the variance of the weight increases, which causes bigger fluctuations in weight updating and leads to instability and slower convergence during training. Secondly, since these 3 features are representing similar things, the model may be biased towards learning more about these 3 features and learning less about other features, causes lower performance despite having other useful features as well.

The comparison of the performance of model trained using all input features, 6 optimal input features and 5 optimal input features are plotted in Figure 3.1(c).

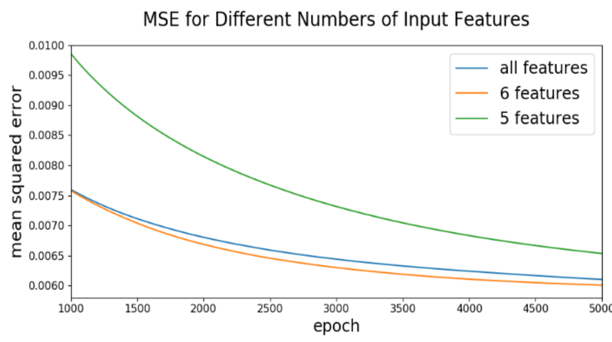


Figure 3.1(c): Mean squared error against epoch for models trained with all features, 6 optimal features and 5 optimal features.

From the graph in Figure 3.1(c), the model trained with 6 optimal input features yields the lowest mean squared error whereas the performance of the model trained with 5 optimal input features is worse. This implies that the feature TOEFL score does not help to improve the performance of neural network whereas the feature SOP should not be removed as removing it yields a worse model performance. Therefore, the optimal feature set is {GRE score, SOP, LOR, university rating, CGPA and research}.

3.2 4-Layer Network, 5-Layer Network and Dropout

Two 4-layer neural networks, with and without dropout and two 5-layer neural networks, with and without dropout is created, with 50 neurons in all hidden layers. The dropout rate used is 0.2. Using the optimal feature set found previously to train these models, the performance of these models together with the previous 3-layer network is plotted in Figure 3.2(a).

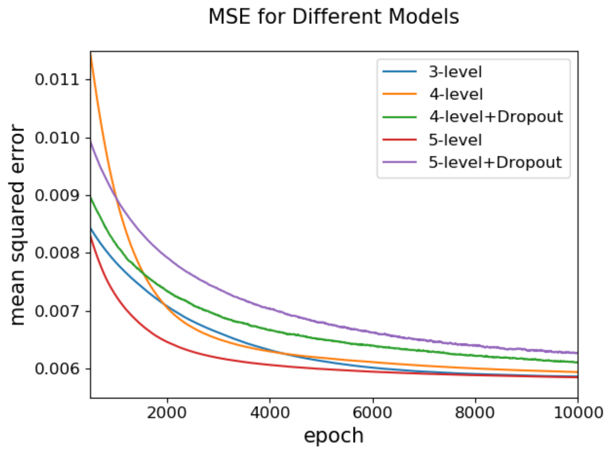


Figure 3.2(a): Mean squared error for different models with regularization over 10000 epochs.

Neural Network With Regularization	Mean Squared Error
3-Layer	0.006038
4-Layer	0.006230
4-Layer + Dropout	0.006387
5-Layer	**0.005987**
5-Layer + Dropout	0.006654

Table 3.2(a): Summary of mean squared error for the 5 different models with regularization.

From Figure 3.2(a), the 5-layer neural network without dropout converges the fastest and yields the lowest mean squared error. The 4-layer neural network yields a slightly higher mean squared error compared to the 3-layer and the 5-layer neural network. Introducing dropouts to the 4-layer and 5-layer network make the convergence much slower, both the neural networks does not converge fully after 10000 epochs. As the number of dropout layers in the 5-layer neural network is more than the number of dropout layers in the 4-layer neural network, the 5-layer neural network with dropout converges slower than the 4-layer neural network with dropout.

In another words, introducing dropout layers to neural networks in this case does not help in preventing overfitting, but also lead to underfitting. This might be due to the regularization applied onto these neural networks are good enough to overcome the overfitting problem. Additional dropout layers applied onto these neural networks started to cause underfitting. To test on this hypothesis, the models are retrained without regularization.

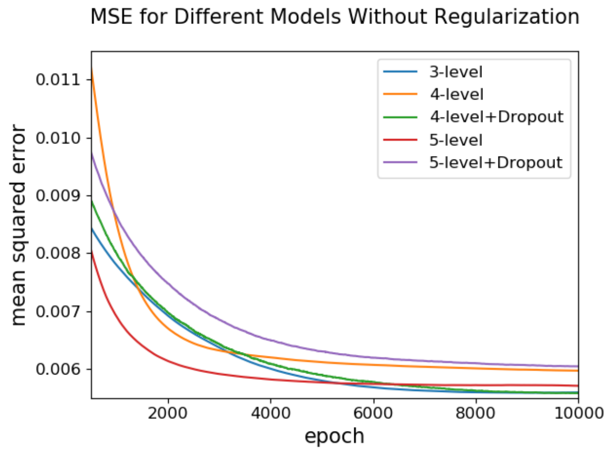


Figure 3.2(b): Mean squared error for different models without regularization over 10000 epochs.

Mean Squared Error	3-Layer	4-Layer	5-Layer
Without Regularization and Dropout	**0.005731**	0.006177	0.006173
With Only Regularization	0.006038	0.006230	0.005987
With Only Dropout	-	0.005799	0.006283
With Regularization and Dropout	-	0.006387	0.006654

Table 3.2(b): Summary of results of all models.

From Table 3.2(b) we can see that the models trained with either regularization or dropout have better performance than the models trained with both regularization and dropout.

For 4-layer network, dropout layers generalized the model better than regularization whereas for 5-layer network, regularization generalized the model better than dropout layers. For 3-layer and 4-layer networks, regularization makes the model perform worse, but for 5-layer network, regularization improves the model performance. This might be due to the complexity of 5-layer neural network which makes it overfit easily onto the small dataset.

CONCLUSION

For part A, the initial 3-layer model converges around 200 to 250 epochs. After hyperparameter tuning, we can conclude that our final optimal 3-layer network performs best when trained using a batch size of 8, 10 neurons in its hidden layer and a weight decay parameter of 10^{-12}. A 4-layer network can perform better than our 3-layer network when trained using the same hyperparameter as the optimal 3-layer network.

For part B, the initial 3-layer model converges around 1800 to 2000 epochs. After implementing recursive feature elimination, 'TOEFL score' is the first feature to be removed, followed by 'SOP'. The model trained with 6 optimal input features performed the best, after removing TOEFL score from the input features. Lastly, using dropout layers together with L2 regularization in this case does not help improve the model performance. In this case, smaller networks perform better using dropout layers only whereas larger network performs better using L2 regularization only. The model that yields the lowest mean squared error is the 3-layer network trained without regularization and dropout.

Tuning hyperparameters when training neural network model is very important as it helps to create better model performance and speed up the training process. Through this assignment, I have learnt to isolate and tune each hyperparameter accordingly and make decisions based on the validation and testing results of the model.

REFERENCES

[1] *scikit-learn. [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 10-Oct-2020].*

[2] *"TensorFlow Core v2.2.0," TensorFlow. [Online]. Available: https://www.tensorflow.org/versions/r2.2/api_docs/python/tf. [Accessed: 10-Oct-2020].*

[3] *UCI Machine Learning Repository: Cardiotocography Data Set. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Cardiotocography. [Accessed: 10-Oct-2020].*

[4] *M. S. Acharya, "Graduate Admission 2," Kaggle, 28-Dec-2018. [Online]. Available: https://www.kaggle.com/mohansacharya/graduate-admissions. [Accessed: 10-Oct-2020].*