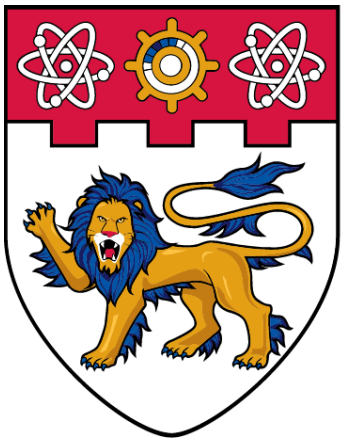

NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

CZ4042 Neural Networks and Deep Learning

Individual Assignment 1 Part B

Zhang Yuhan

U1823060F

Table of Contents

Part B: Regression Problem **3**

Introduction 3

Methods 3

Experiments and Results 5

Question 1: 5

Question 2: 8

Question 3: 9

Conclusion 11

Part B: Regression Problem

Introduction

This problem aims at finding the optimal feature(s) in a dataset that gives the highest accuracy. The dataset used in this problem is the Graduate Admissions Prediction from Kaggle, which contains data on 7 attributes that help determine the probability of admission for graduates. The attributes include: GRE score, TOEFEL score, University rating, Strength of recommendations, GPA, and research experience.

I will be using mean square error (MSE) as the loss function for models generated for this question. This allows me to determine when to stop training in question 1. Then, I will be implementing the recursive feature elimination (RFE) method to remove unnecessary features that negatively affect the accuracy of the models. Finally, I will be implementing a 4-layer and 5-layer network with a dropout rate of 20% in order to analyse the effect of dropout rate on the accuracy of the models.

All figures and codes provided in this report is present in the corresponding jupyter notebook files for each question.

Methods

```
def fit_model(train_X, train_Y, test_X, test_Y, epochs, batch_size, model, isCallback):
    if isCallback:
        # stops training when there is no observed improvement in mse for 5 epochs
        callback = tf.keras.callbacks.EarlyStopping(monitor='val_mse', min_delta = 0.0001,patience=5)
    else:
        callback = timeCallback()
    history = model.fit(train_X, train_Y,
                        epochs=epochs,
                        callbacks = [callback],
                        verbose = 2,
                        batch_size=batch_size,
                        validation_data = (test_X, test_Y),
                        shuffle=True)
    #timeTaken = np.mean(timeTaken.logs)
    return history, model
```

Figure 18: Implementation of early stopping

Intuitively, when a model reaches its lowest test error, it should stop training in order to prevent over fitting. Thus, the fit_model() function from the previous question is modified to take in a Boolean value ‘isCallback’ which determines whether the model will stop training once its lowest validation MSE is reached.

```
def recursive_drop(original_mse, train_X, test_X, train_Y, test_Y, keepTrack):
    mse_log = []
    test_mse_log = []
    improved_mse = []
    hasImproved = False

    # Delete one feature from original dataset
    for i in range(len(train_X[0])):
        print("Dropping feature at index: ",i)
        dropped_trainX = np.delete(train_X,0,1)
        dropped_testX = np.delete(test_X,0,1)

        model = get_model()
        history, model = fit_model(dropped_trainX, train_Y, dropped_testX, test_Y, epochs, batch_size, model, False)
        mse_log.append(history.history['mse'])
        test_mse_log.append(history.history['val_mse'])

    for i in range(len(mse_log)):
        print('Checking new mse against original: ', i)
        if str(mse_log[i][-1]) < str(original_mse[-1]):
            improved_mse.append(mse_log[i][-1])
            hasImproved = True
            print("Original mse: \t",original_mse[-1])
            print("Improved mse: \t",mse_log[i][-1])
        else:
            improved_mse.append(100)

    if hasImproved:
        drop_index = improved_mse.index(min(improved_mse))
        new_trainX = np.delete(train_X,drop_index,1)
        new_testX = np.delete(test_X,drop_index,1)
        print("\nDropped feature at index: ", drop_index)
        if keepTrack:
            order.append(drop_index)
        return recursive_drop(mse_log[drop_index], new_trainX, new_testX, train_Y, test_Y, keepTrack)
    elif len(train_X[0]) == 2:
        return original_mse
    else:
        return original_mse
```

Figure 19: Recursive Elimination Method

The recursive feature elimination method will eliminate one feature from the original dataset and compare the training results against the results without any feature drops. Then, it will drop the feature that results in the lowest MSE after being eliminated. This new MSE will then be passed back into the recursive function as the original MSE. Once there is no observed improvement in the MSE compared to the original, the function will return the train and test MSE values for the optimal model as well as the order the features were dropped.

```
def get_model_4(isDropout):
    if isDropout:
        model = keras.Sequential([
            keras.layers.Dense(hidden_layer, use_bias = True, activation='relu',
                               kernel_regularizer = tf.keras.regularizers.l2(decay_beta)),
            keras.layers.Dropout(rate=0.2, seed=seed),
            keras.layers.Dense(hidden_layer, use_bias = True, activation='relu',
                               kernel_regularizer = tf.keras.regularizers.l2(decay_beta)),
            keras.layers.Dropout(rate=0.2, seed=seed),
            keras.layers.Dense(1)
        ])
    else:
        model = keras.Sequential([
            keras.layers.Dense(hidden_layer, use_bias = True, activation='relu',
                               kernel_regularizer = tf.keras.regularizers.l2(decay_beta)),
            keras.layers.Dense(hidden_layer, use_bias = True, activation='relu',
                               kernel_regularizer = tf.keras.regularizers.l2(decay_beta)),
            keras.layers.Dense(1)
        ])

    SGD_opt = keras.optimizers.SGD(learning_rate=learning_rate)

    model.compile(optimizer=SGD_opt,
                  loss=keras.losses.MeanSquaredError(),
                  metrics=['mse'])
    return model
```

Figure 19: Modified get_model() for optional dropout

In order to introduce dropouts, the extra parameter keras.layers.Dropout() was used. The differences between the original model and the model with dropouts can be observed in Figure 19 above.

Experiments and Results

Question 1:

Design a 3-layer feedforward neural network consists of an input layer, a hidden-layer of 10 neurons having ReLU activation functions, and a linear output layer. Use mini-batch gradient descent with a batch size = 8, $L2$ regularization at weight decay parameter $\beta = 10{-}3$ and a learning rate $\alpha = 10{-}3$ to train the network

Part A)

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

In order to observe where the minimum MSE occurs, a larger epoch was used. The hyper parameters used in the question are shown below:

```
NUM_CLASSES = 7

epochs = 1000
batch_size = 8

decay_beta = 10**-3
learning_rate = 10**-3
hidden_layer = 10

seed = 10
```

The results of the model without early stopping is shown below:

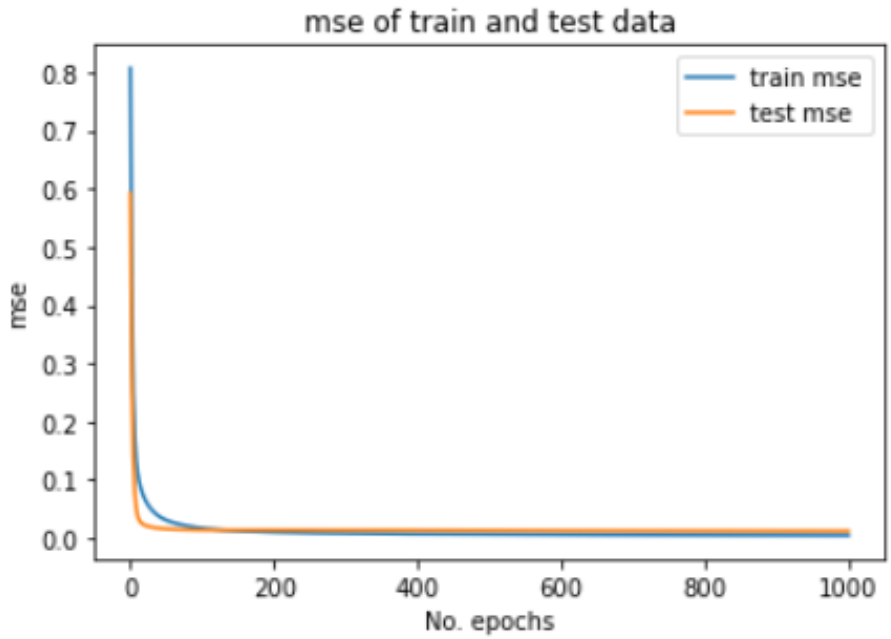

Figure 20: MSE of train and test data without early stopping

Part B)

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

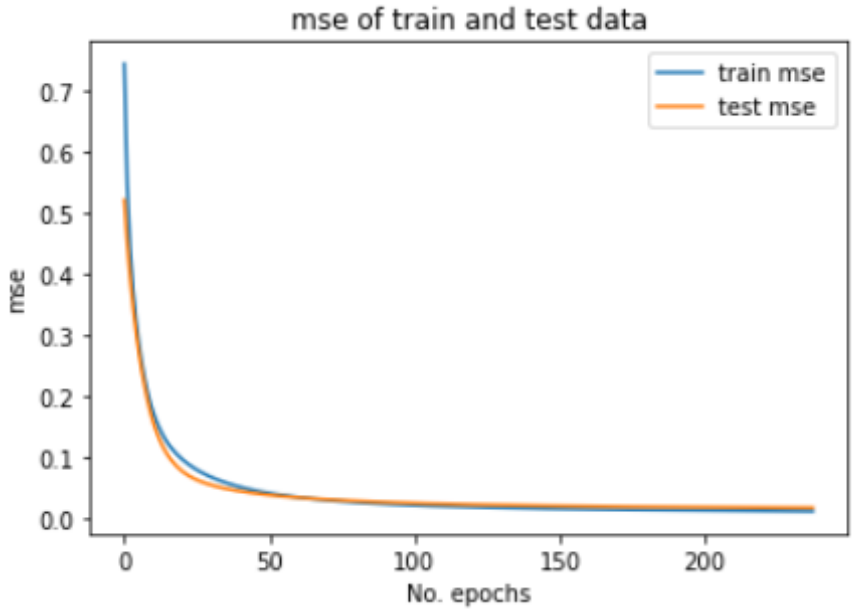

Figure 21: MSE of train and test data with early stopping

As can be seen from Figure 20, it is quite hard to observe when the MSE of validation data stops decreasing. Thus, I used the keras early stopping function to monitor the val_mse. As seen in Figure 21, the minimum MSE of the test data occurs at around 250 to 300 epochs.

Part C)

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

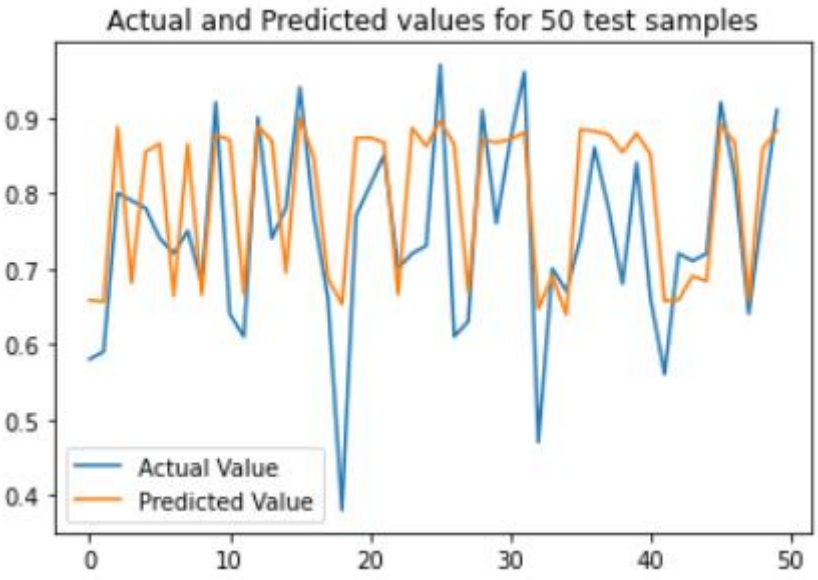

Figure 22: Actual and predicted values for 50 test samples

Question 2:

- Recursive feature elimination (RFE) is a feature selection method that removes unnecessary features from the inputs. Start by removing one input feature that causes the minimum drop (or maximum improvement) in performance. Repeat the procedure recursively on the reduced input set until the optimal number of input features is reached. Remove the features one at a time. Compare the accuracy of the model with all input features, with models using 6 input features and 5 input features selected using RFE. Comment on the observations.

In order to find which features hinder the minimum MSE of the original model, the recursive drop function in Figure 19 was used. The results are shown in Figure 23 below, and the features that were dropped were at indices [2,3], which corresponds to University and SOP features in the original dataset.

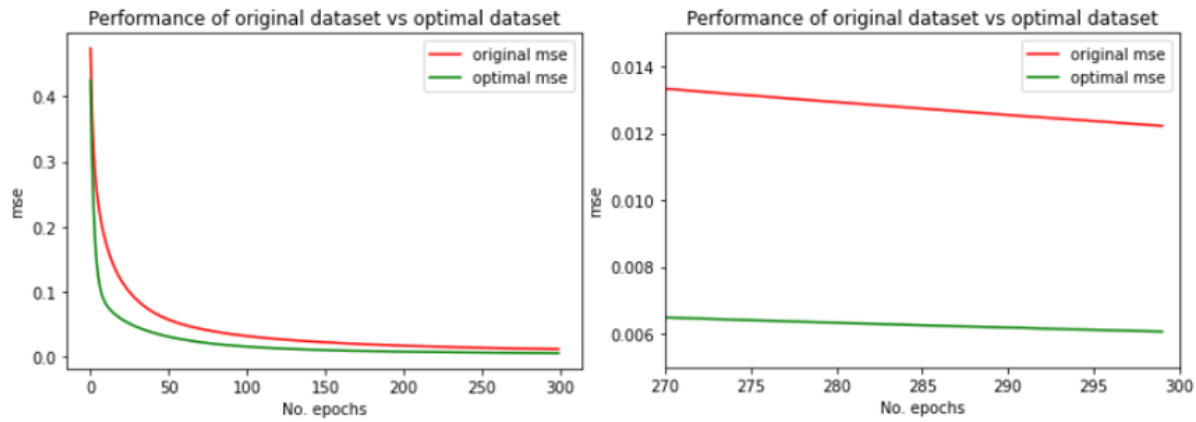

Figure 23: Performance of original dataset vs optimal dataset

With this information, I manually dropped the features at index 2 and 3 one by one, the results are shown below:

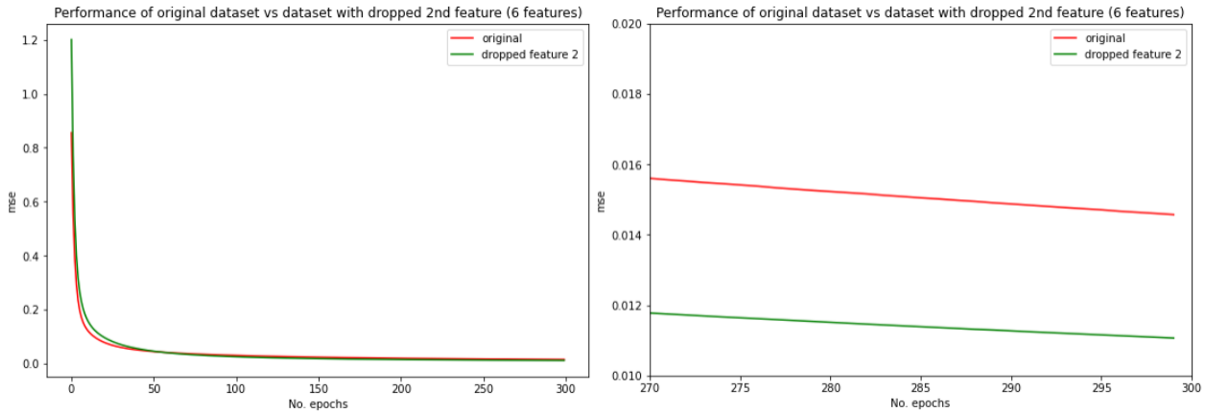

Figure 24: Performance of original dataset vs dataset without index 2 (University)

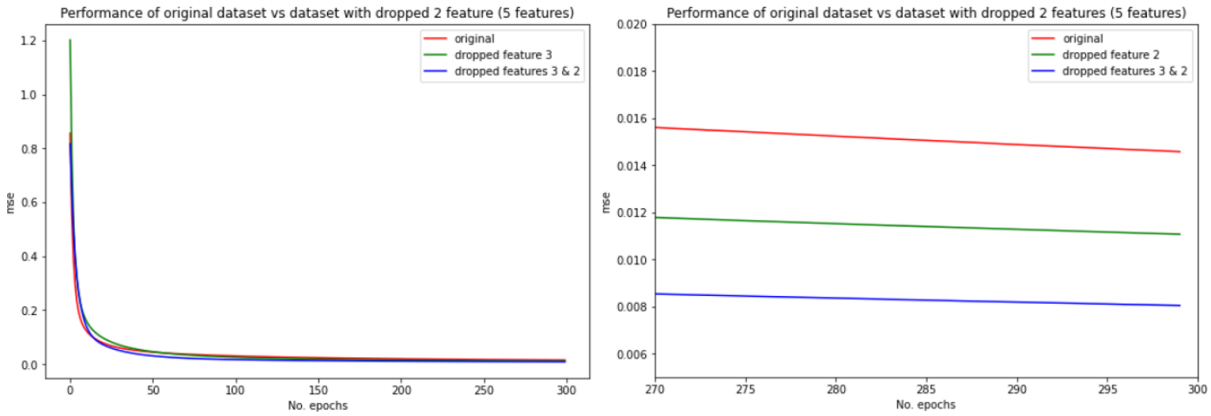

Figure 24: Performance of original dataset vs dataset without index 2 & 3(University & SOP)

Although both the MSEs of the datasets without University and/or SOP were better than that of the original dataset, all three curves follow an extremely similar trajectory, as seen in Figure 24. The difference in their final MSEs after 300 epochs were also very similar, in fact they seem to be converging towards the same point. This suggests that beyond 300 epochs, the difference between the MSEs of the three datasets will likely continue to decrease. This means that the effects of features University and SOP on the model is minimal, especially for models trained with large number of epochs.

Question 3:

- 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.

In order to introduce dropouts to each model, the Dropout function in keras was used as shown in Figure 19. The parameters used for this question is as follows:

```
NUM_CLASSES = 7

epochs = 300
batch_size = 32

decay_beta = 10**-3
learning_rate = 10**-3
hidden_layer = 50

seed = 10
```

The comparisons of performance for each model is shown below:

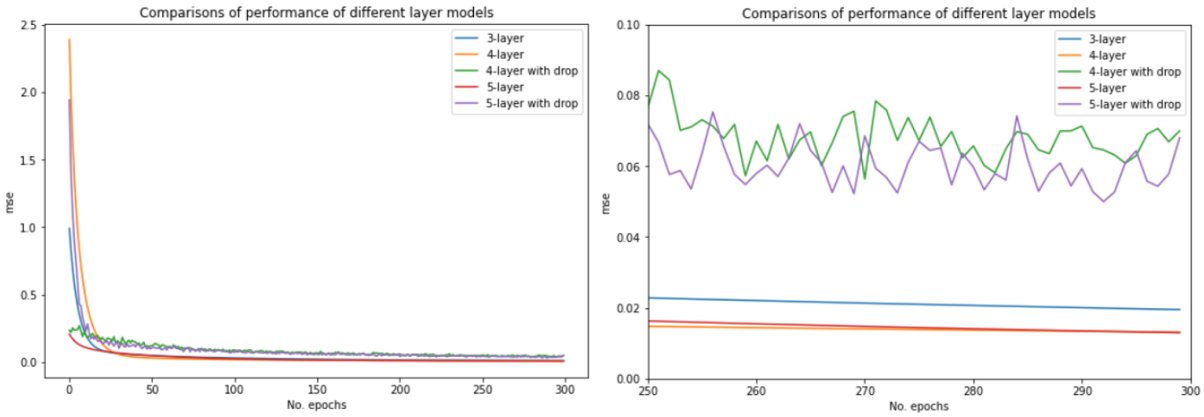

Figure 25: Comparisons of performance of different layer models (train MSE)

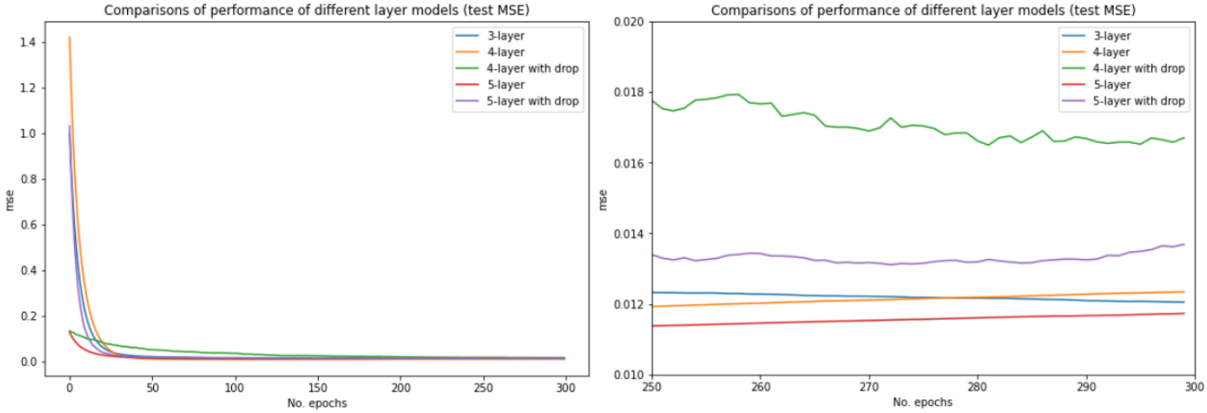

Figure 26: Comparisons of performance of different layer models (test MSE)

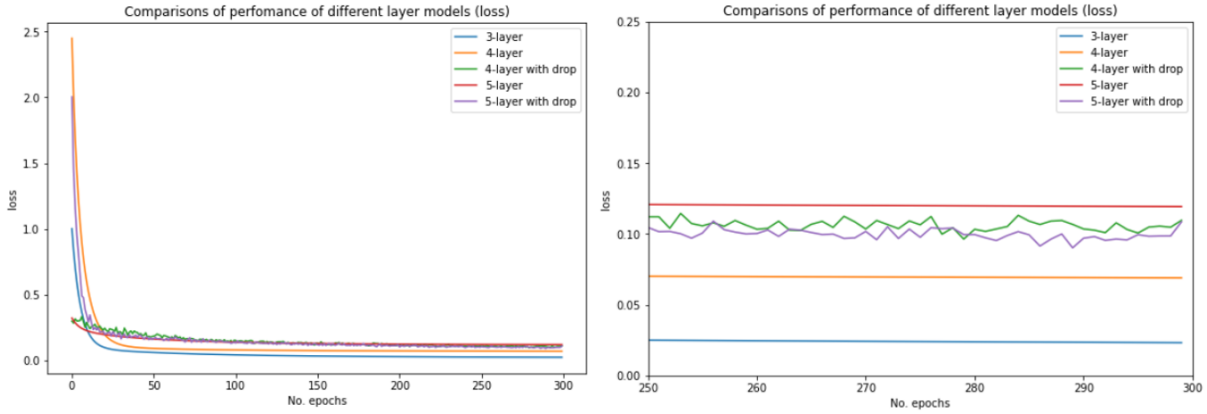

Figure 27: Comparisons of performance of different layer models (train loss)

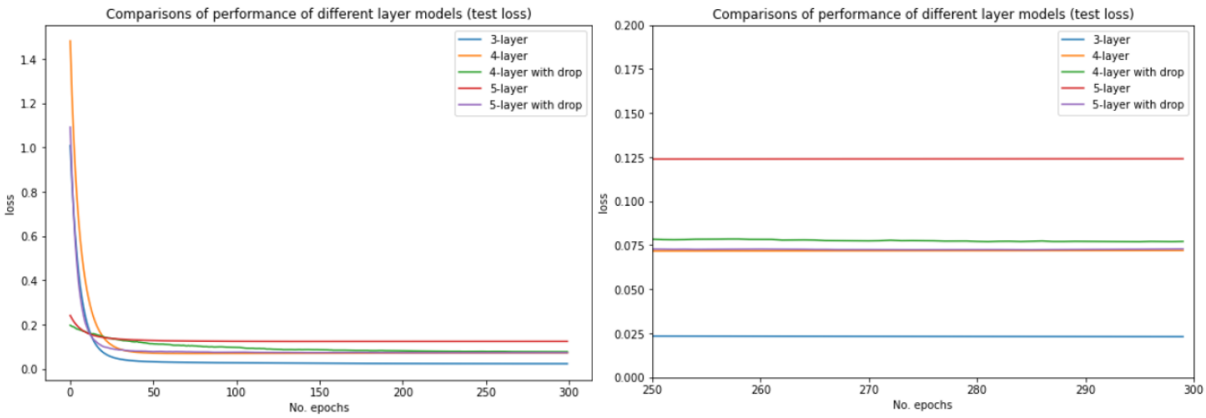

Figure 28: Comparisons of performance of different layer models (test loss)

From the results illustrated by Figures 25 to 28 above, the following observations can be made:

- The MSEs of the models with dropouts were evidently worse than their original counterparts.
- Among the models with dropouts, the performance was better for models with more hidden layers.
- The 4-layer and 5-layer models performed better in terms of MSE for training data as compared to the 3-layer model. However, the MSE for test data for the 3-layer model seems to be converging towards a lower point than that of both the 4-layer and 5-layer models.
- The 5-layer with dropout model performed better than the original 5-layer model in terms of loss for training data. This was the only case where a model with dropout performed better than its original counterpart.
- The 3-layer model's performance in terms of training and test loss was significantly better than all other models in this question.

Conclusion

From the results in questions 1 and 2, is can be observed that the recursive feature elimination (RFE) method is effective in minimizing the mean squared error (MSE) for a given dataset. However, as mentioned in the conclusion of Part A, the results from these models are all subject to randomness. Because of this, there is a probability that the RFE method may choose to eliminate different features depending on the performance of each feature during the current run. This, again, means that there is no guarantee that the feature eliminated was in fact the feature that causes the minimum drop in the result.

From the results of question 3, it can be observed that increasing the number of layers in the network hindered the model performance instead of improving it as it did with the dataset in Part A. This may be due to the fact that the dataset used in Part B was significantly smaller than that of Part A (400 entries compared to more than 2000). Thus, increasing the number of layers in the model push the model complexity beyond the optimal point for the dataset, causing overfitting of data. This is further supported by the fact that random dropouts in the 5-layer model actually improved the performance of the model compared than its original counterpart.