NANYANG TECHNOLOGICAL UNIVERSITY

CZ4042 CE/CZ4042: Neural Networks and Deep Learning

Assignment 1

Yong Hao

U1722282A

Contents

1. Part A: Classification Problem

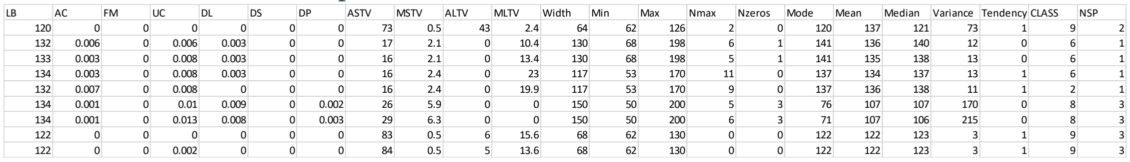
1.1 Introduction

Part A aims at building a feed-forward which 3 to 4 hidden layers neural networks to classify the Cardiotocography dataset containing measurements of fetal heart rate (FHR) and uterine contraction (UC) features on 2126 cardiotocograms classified by expert obstetricians and plot the model training result in regard of classification accuracy. Each record contains a feature space of 21 and 2 class labels. We will only use the NSP label for the scope of the assignment

The cardiotocograms were classified by three expert obstetricians and a consensus classification label with respect to a morphologic pattern and to a fetal state (N denotes Normal, S denotes Suspect and P denotes Pathologic) was assigned to each of the,. The aim is to predict the N, S and P class labels in the test dataset after training the neural network on the training dataset.

1.2. Methods

1.2.1. Normalization of Inputs



LB	AC	FM	UC	DL	DS	DP	ASTV	MSTV	ALTV	MLTV	Width	Min	Max	Nmax	Nzeros	Mode	Mean	Median	Variance	Tendency	CLASS	NSP
120	0	0	0	0	0	0	73	0.5	43	2.4	64	62	126	2	0	120	137	121	73	1	9	2
132	0.006	0	0.006	0.003	0	0	17	2.1	0	10.4	130	68	198	6	1	141	136	140	12	0	6	1
133	0.003	0	0.008	0.003	0	0	16	2.1	0	13.4	130	68	198	5	1	141	135	138	13	0	6	1
134	0.003	0	0.008	0.003	0	0	16	2.4	0	23	117	53	170	11	0	137	134	137	13	1	6	1
132	0.007	0	0.008	0	0	0	16	2.4	0	19.9	117	53	170	9	0	137	136	138	11	1	2	1
134	0.001	0	0.01	0.009	0	0.002	26	5.9	0	0	150	50	200	5	3	76	107	107	170	0	8	3
134	0.001	0	0.013	0.008	0	0.003	29	6.3	0	0	150	50	200	6	3	71	107	106	215	0	8	3
122	0	0	0	0	0	0	83	0.5	6	15.6	68	62	130	0	0	122	122	123	3	1	9	3
122	0	0	0.002	0	0	0	84	0.5	5	13.6	68	62	130	0	0	122	122	123	3	1	9	3

By observing the sample data set, we can note that there are relatively huge variations between features that might affect the training result of the model. For example: the mode, mean and median fields can reach over 100 while fields like AC and UC are in the range of [0, 1e-3].

```
def scale(X, X_min, X_max):
    return (X - X_min)/(X_max-X_min)

train_input = np.genfromtxt('ctg_data_cleaned.csv', delimiter= ',')
trainX, train_Y = train_input[1:, :21], train_input[1:,-
1].astype(int)
```

To eliminate the variations between fields to prevent a few features from having a dominant effect on training, we adopted min-max normalization technique such that all the input features lies in [0,1].

1.2.2. Split Training and Testing Data

We utilized the built in function ‘train_test_split’ from sklearn to randomly split the entire sample into training set and testing set with size ratio of 70:30. The function ‘train_test_split’ will split the data randomly by specifying the ‘random_state’ parameter. The split of the same sample with the same ‘random_state’ will always result in the same split, thus ensuring the consistency of the training result.

```
test_size = 0.3

train_X, test_X, train_Y, test_Y = train_test_split(trainX, trainY,
test_size = test_size, random_state=1)
train_X = scale(train_X, np.min(train_X, axis=0), np.max(train_X,
axis=0))
test_X = scale(test_X, np.min(test_X, axis=0), np.max(test_X,
axis=0))
```

The split result in a training set with size 1488 and testing set with size 638.

1.3. Experiments and Results

1.3.1. Question 1.

For this question, we will need to build a model that fulfils the following requirements:

1. The network is a feed-forward neural network with an input layer, one hidden layer of 10 neurons with ReLU activation function and one softmax layer.
2. The learning rate α = 0.01
3. L_2 regularization with wight decay parameter = 1e-6
4. Batch gradient descent with batch size = 32
5. Scaling of features (as we mentioned above)

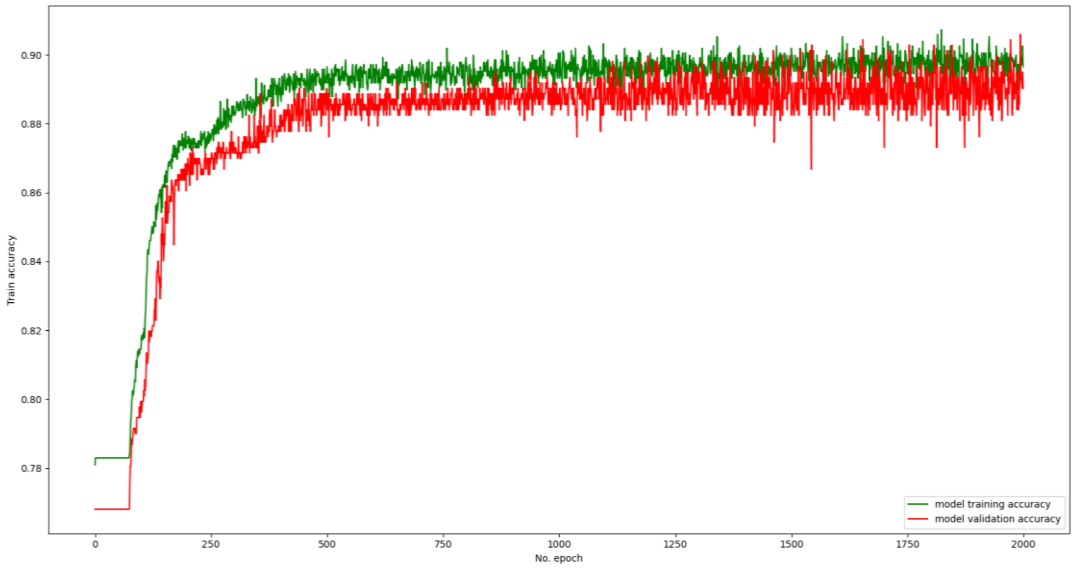
We implemented the following method to build the required model:

```
model = keras.Sequential()

model.add(keras.layers.Dense(num_neurons,
input_dim=21,activation='relu',kernel_initializer='random_normal',
bias_initializer='zeros',
kernel_regularizer=tf.keras.regularizers.l2(1e-6)))

model.add(keras.layers.Dense(NUM_CLASSES, activation='softmax',
                             kernel_initializer='random_normal',
                             bias_initializer='zeros'
                             ))

optimizer = keras.optimizers.SGD(learning_rate=0.01)

model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']
              )
```

The model is composed of 3 layers, an input layer, a hidden layer consists of 10 neurons with ReLU activation function and L2 regularization with weight decay parameter $\beta = 10{-}6$. The third layer and also the output layer has softmax activation function. The optimizer adopts the built-in SGD optimizer with learning rate equal to 0.01.

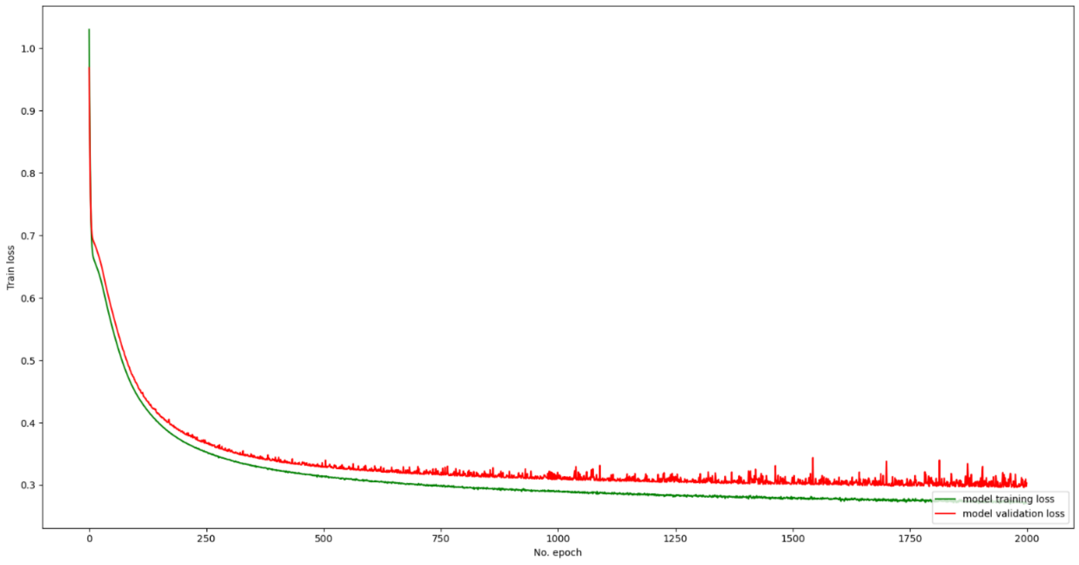
a) Use the training dataset to train the model and plot accuracies on training and testing data against training epochs

The training accuracies on training and testing set of the model is given by:



b) State the approximate number of epochs where the test error begins to converge.

The errors of the training set and testing set are given by:



We have implemented callback functions in the training session to log the loss of the testing set as follows:

Epoch	Testing Loss
0	0.97
100	0.46
200	0.35
300	0.34
400	0.33
500	0.32
600	0.32
700	0.32
800	0.32
900	0.31
1000	0.32

1100	0.31
1200	0.31
1300	0.31
1400	0.30
1500	0.31
1600	0.34
1700	0.30
1800	0.30
1900	0.30
2000	0.30

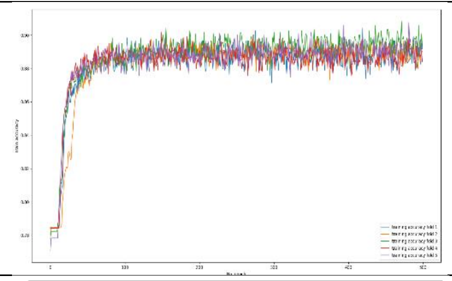
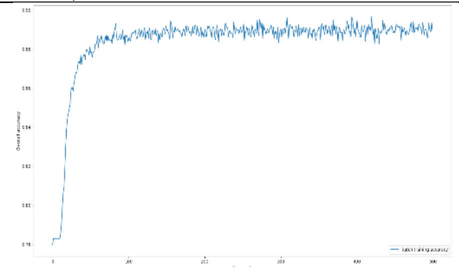
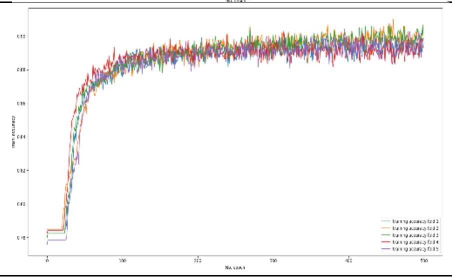
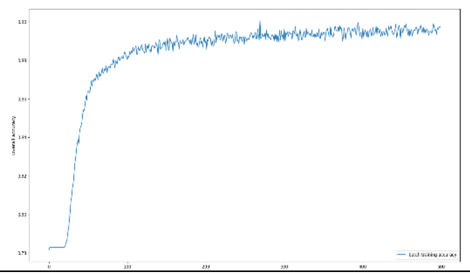
We can observe the testing loss reduced to 0.30 and converges around the 1500th epoch.

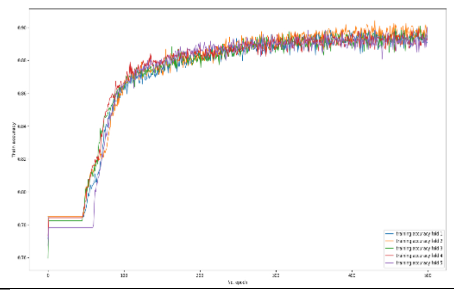
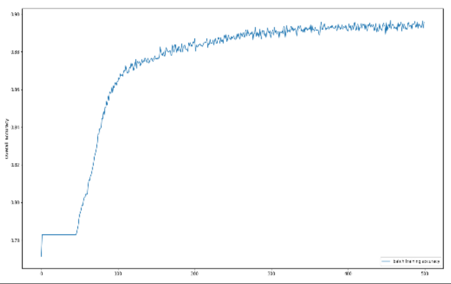
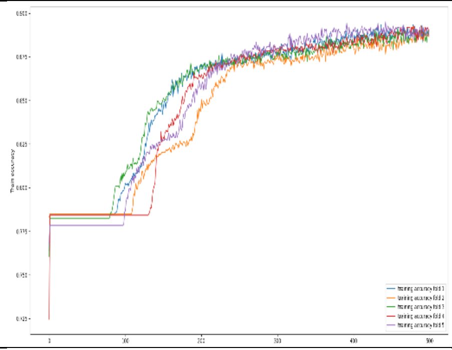
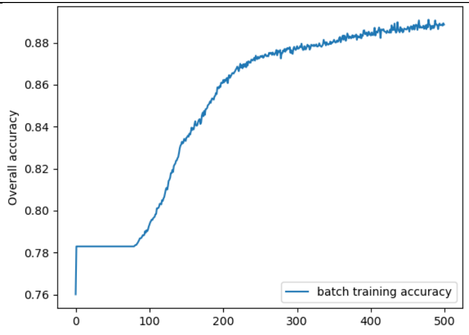
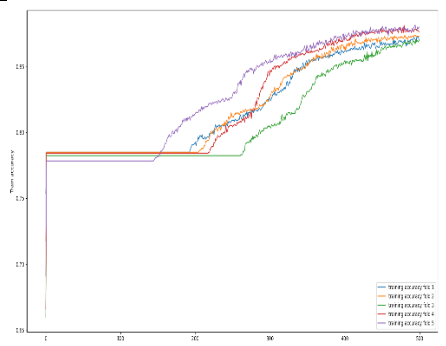
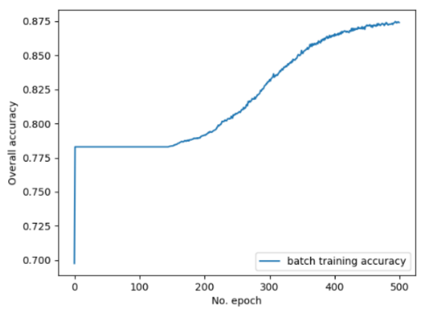
1.3.2. Question 2

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.

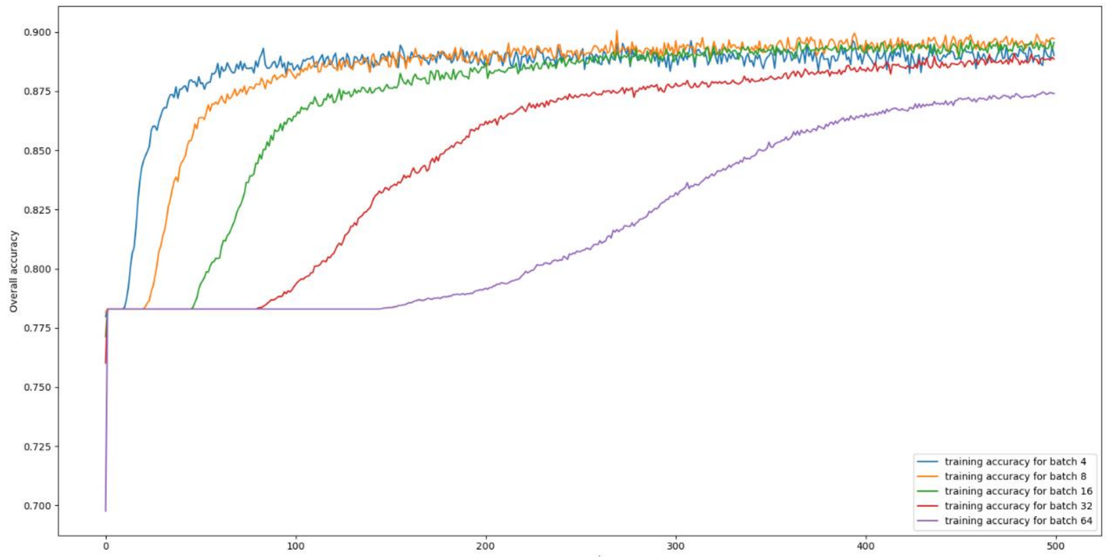
We implemented the built-in method of sklearn to split the training set into five folds for cross validation:

```
split_counter = 1

for train_index, test_index in KFold(n_split).split(train_X):
    train_x, test_x = train_X[train_index], train_X[test_index]
    train_y, test_y = train_Y[train_index], train_Y[test_index]

    histories['fold_'+str(split_counter)] =model.fit(train_x,train_y,
epochs=epochs, verbose = 0, batch_size=batch)

split_counter += 1
```

Batch Size	Accuracy for each fold	Overall accuracy (Mean of all the folds)
4		
8		

16		
32		
64		

To compare the cross-validation accuracy between different batch sizes, we plotted the following graph:



We implemented the following code to record the average training time for one epoch of each batch size:

```
current_time = time.time()

# logic to train the model for each batch size using 5-fold cross
validation

# Record the total training time for each batch size
endtime = time.time() - current_time
```

The average training time for each epoch is calculated as below:

Batch Size	Average Time for each epoch
4	3.072826560020447
8	1.531836547374725
16	0.9100194253921509
32	0.4580783643722534
64	0.26774400424957273

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

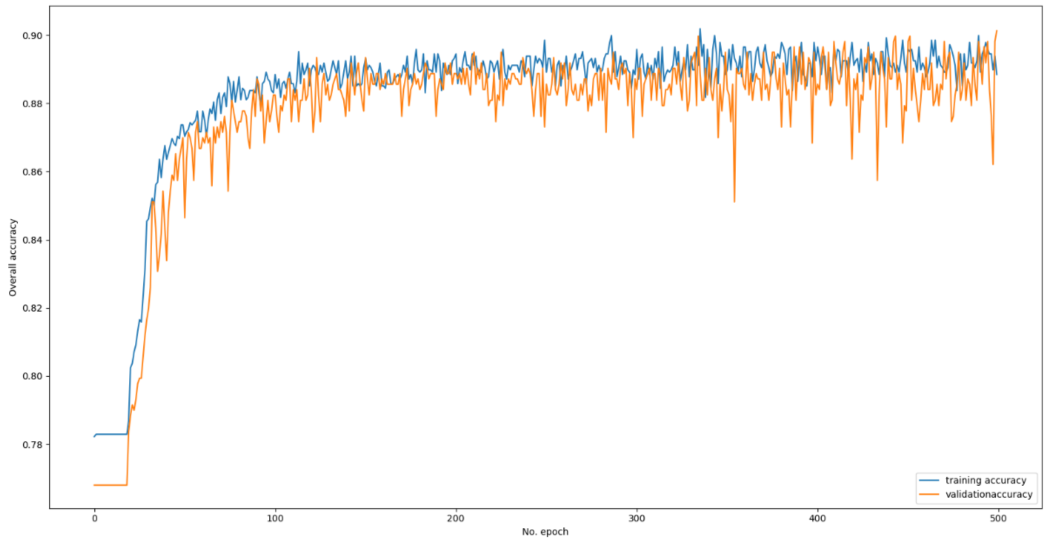
From the cross-validation result in section a) we can observe that the batch size 4, 8, and 16 have the highest training accuracies with batch size 8 being the highest. Comparably batch size 32 and 64 have the poorest training result after convergence. We can also observe that the speed of the increase of training accuracy is inversely proportional to the batch size, with batch size 4 increases the fastest.

From the average training time for each epoch we can observe that with the increase of batch size, the average time for each epoch reduces.

We select the optimal batch size to be 8 since it has the highest training accuracy and relatively acceptable average time for training each epoch.

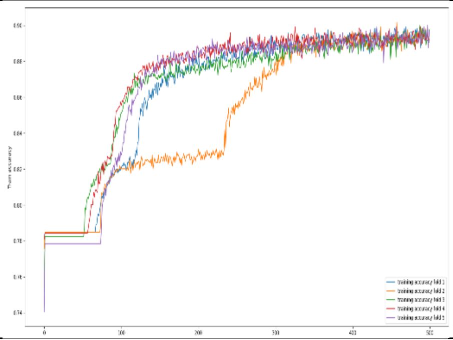
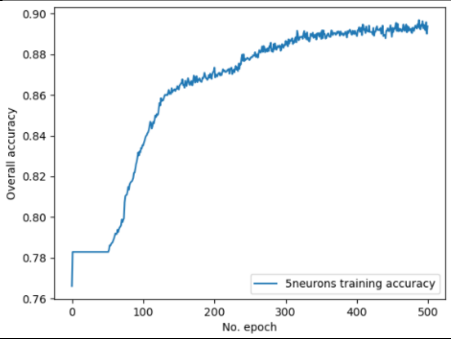
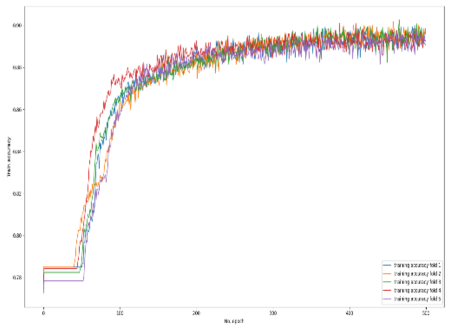
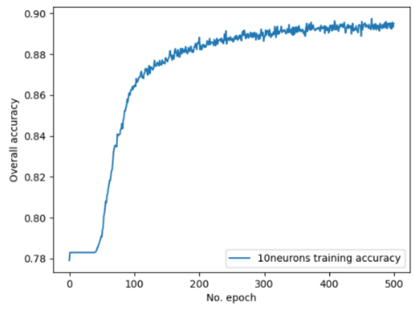
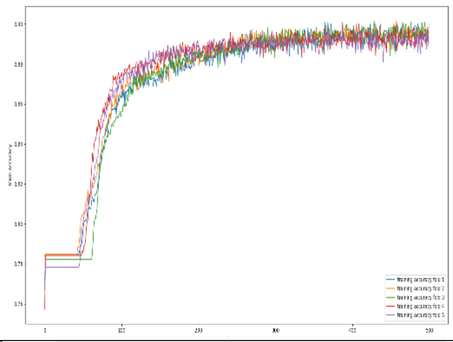
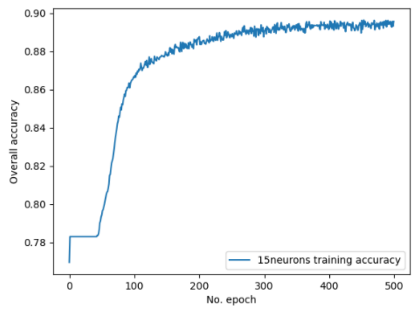
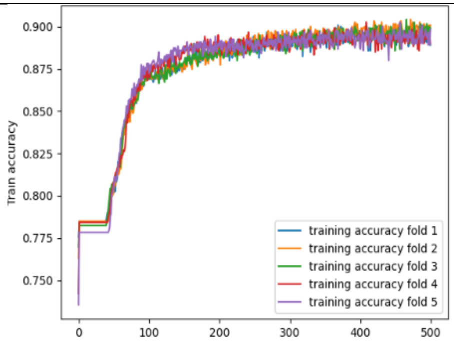
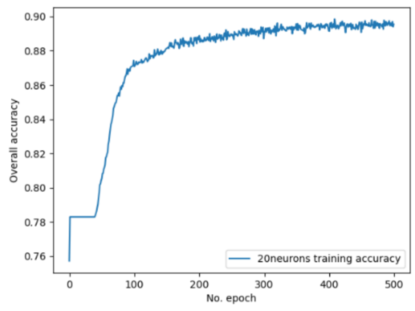
c) Plot the train and test accuracies against epochs for the optimal batch size.

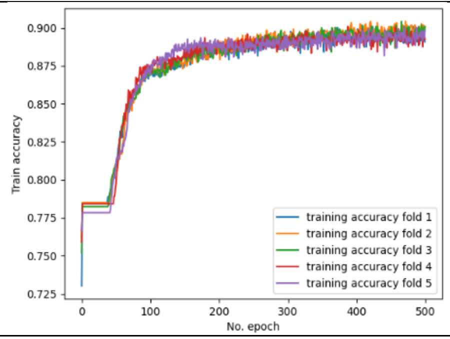
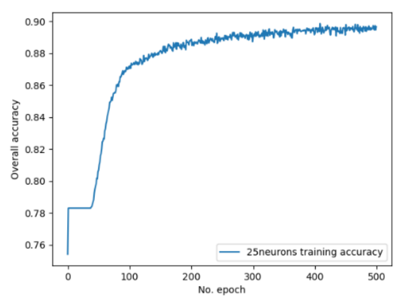
We chose the optimal batch size to be 8 and plot the train and test accuracies against epochs as following:



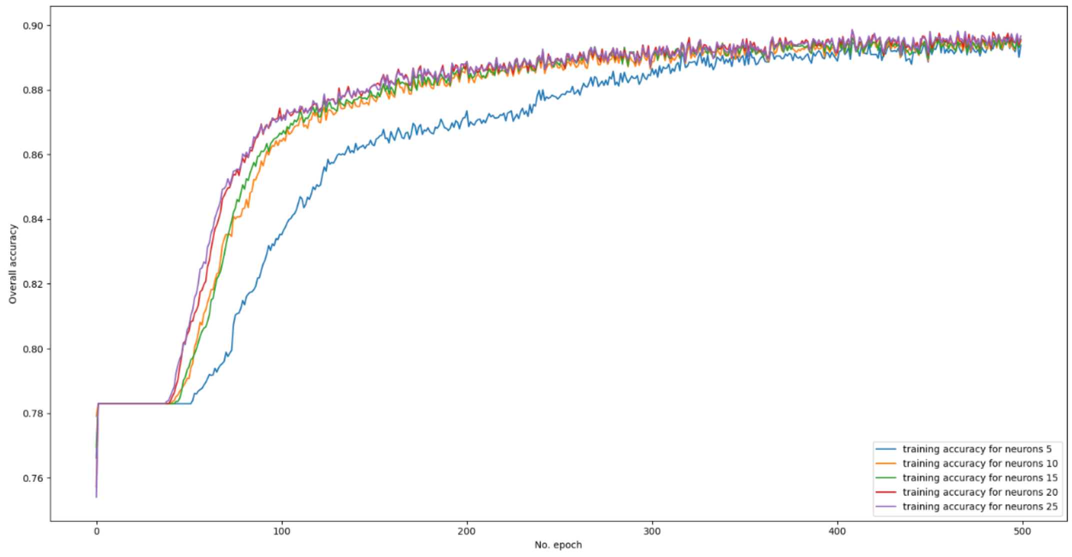
1.3.3. Question 3

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}.

Number of Neurons	Accuracy for each fold	Overall accuracy (Mean of all the folds)
5		
10		
15		
20		

25		

To compare the cross-validation accuracy between different number of neurons in the hidden layer, we plotted the following graph:



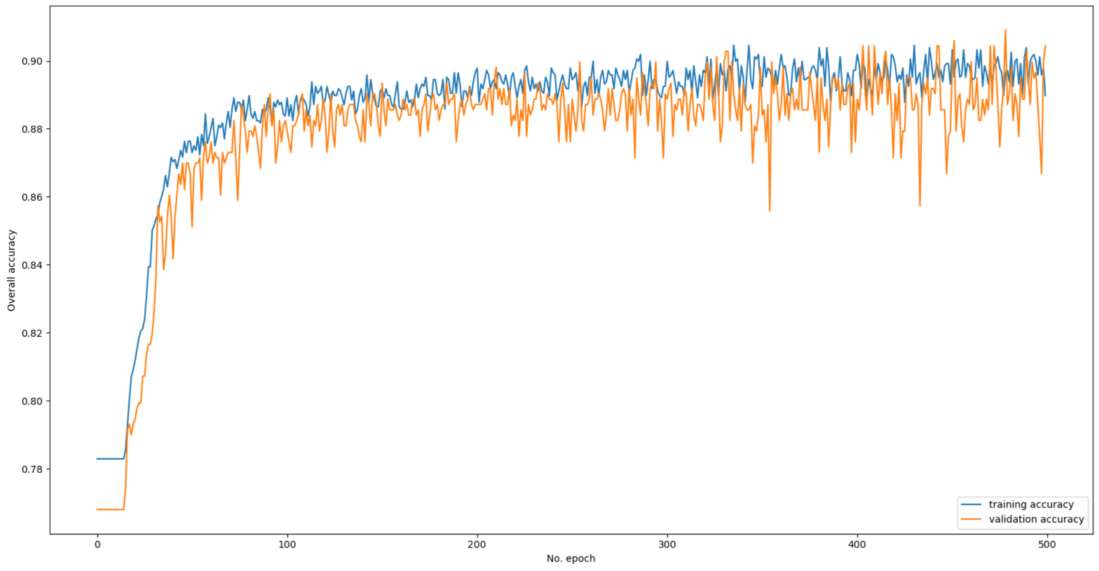
b) Select the optimal number of neurons for the hidden layer. State the rationale for your selection.

From the cross-validation comparison result above we can observe that the training accuracy for 25-neuron hidden layer has the highest training accuracy among other neuron sizes after convergence. Thus, the optimal number of neurons that we select is 25.

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

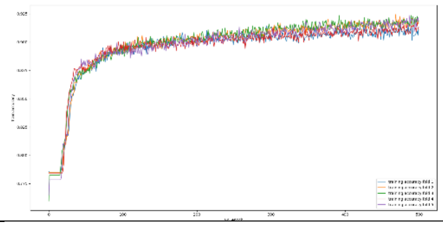
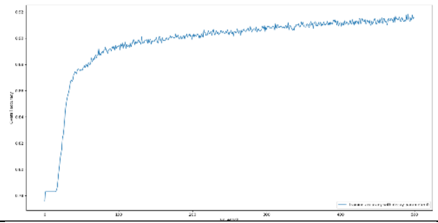
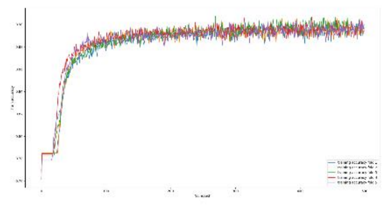
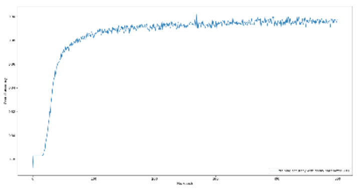
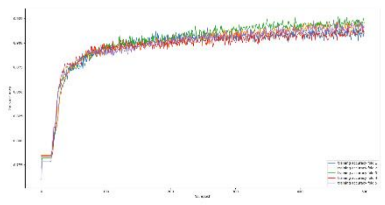
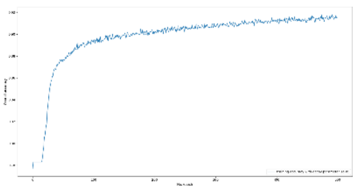
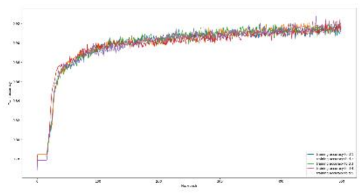
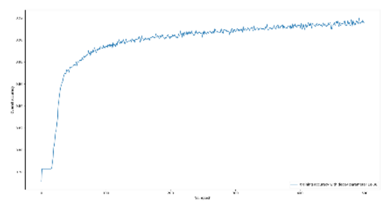
neurons.

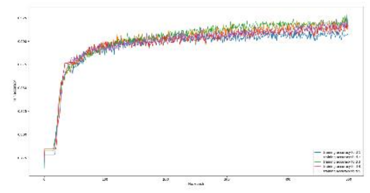
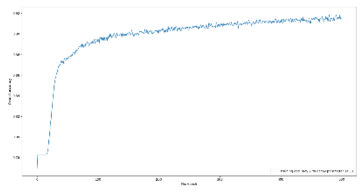
We chose the optimal number of neurons to be 25 and plot the train and test accuracies against epochs as following:



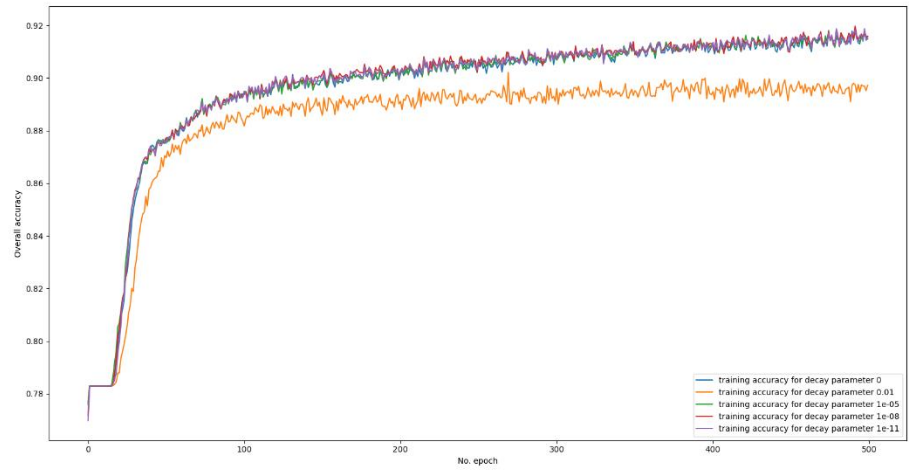
1.3.4. Question 4

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, 10e−3, 10e−6, 10e−9, 10e−12}.

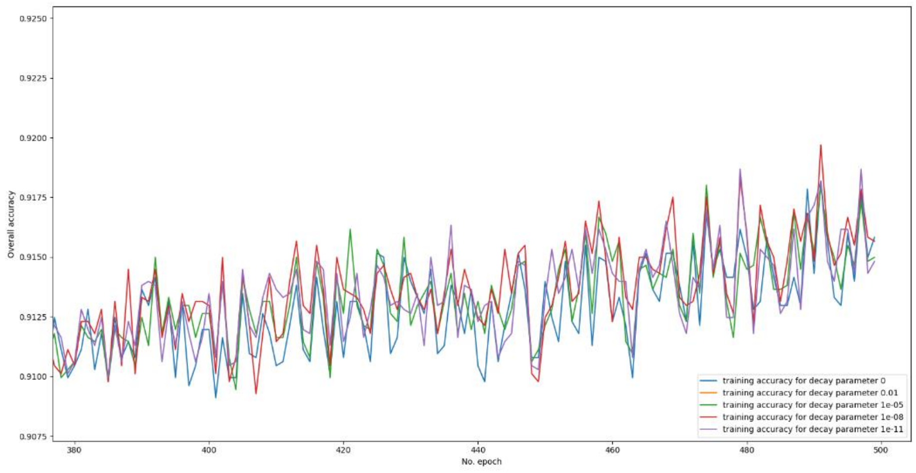
Decay Parameter	Accuracy for each fold	Overall accuracy (Mean of all the folds)
0		
10e−3		
10e−6		
10e−9		

10e−12		

To compare the cross-validation accuracy between different decay parameters in the hidden layer, we plotted the following graph:



b) Select the optimal decay parameter. State the rationale for your selection.
If we zoom in the comparison in section a) between training epoch 400 and training epoch 500, we can have the following pattern. From the following pattern we can observe that the training accuracy of decay parameter 1e-9 stands out of the other options for most of the time. Thus, we choose decay parameter 1e-9 to be the optimal decay parameter for the model.



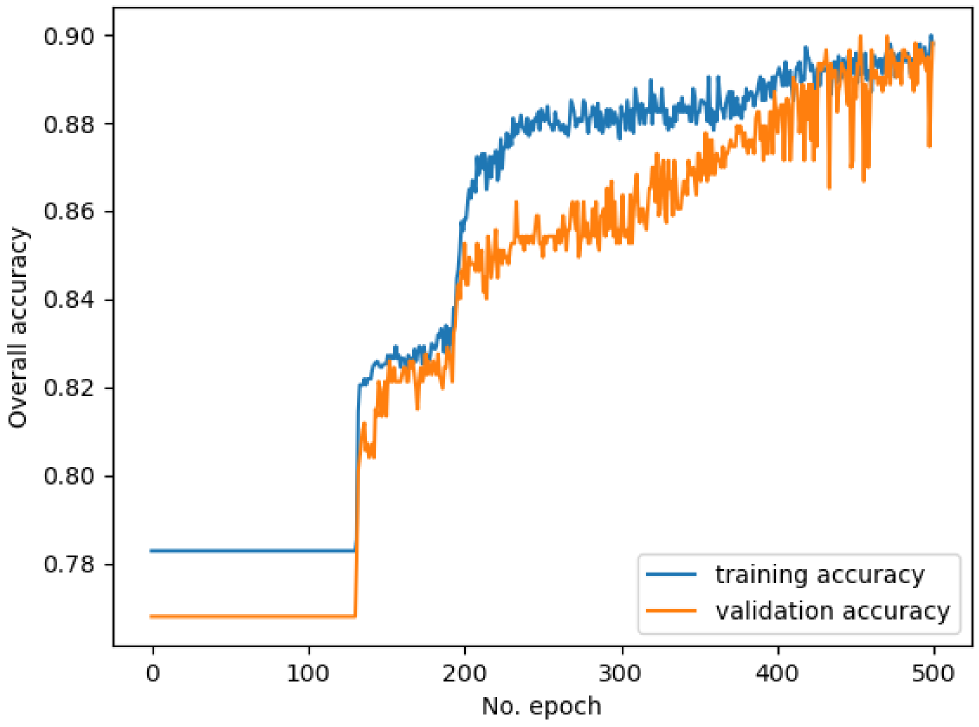
1.3.5. Question 5

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

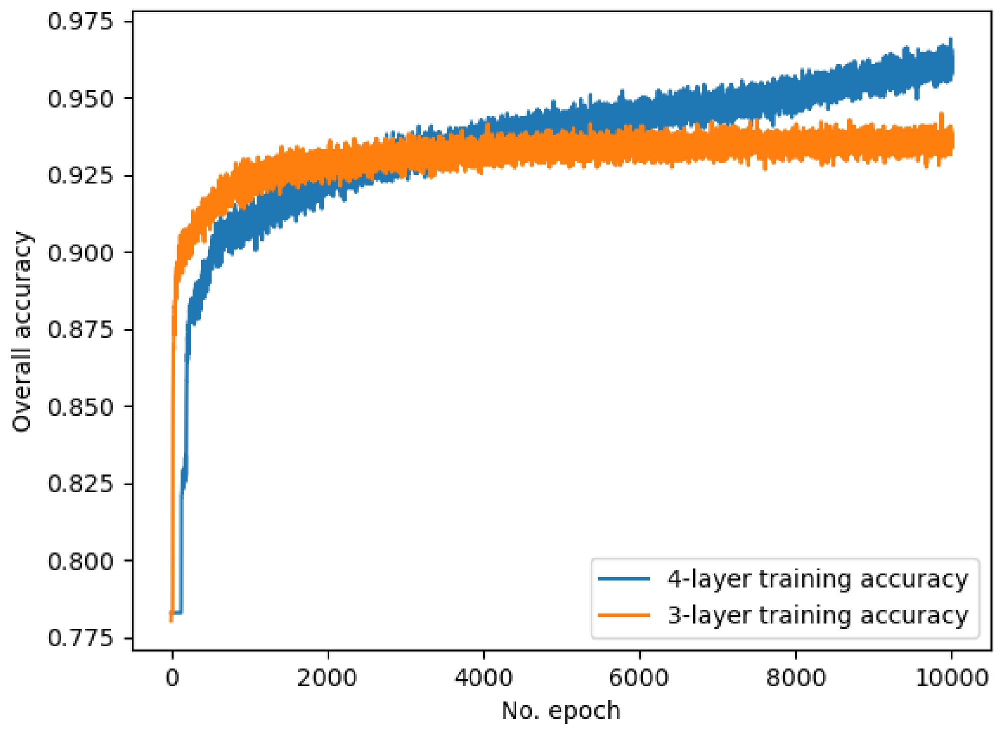
We created and trained the 4-layer model with the following requirements:

1. Having two hidden layers, each consisting 10 neurons
2. Training batch size if 32 and the decay parameter is 1e-6

We acquired the following graph for the training and testing accuracies against training epochs:



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



From the graph in section a) we can observe that the optimal 3-layer training model have a higher accuracy than 4-layer training model initially before the 4000th epoch. However, the training accuracy of the 4-layer model surpasses 3-layer model after the 4000th epoch and continues to increase while the accuracy of 3-layer model converges.

One possible reasoning for this behaviour is that with the increase in the number of hidden layers, the network becomes less volatile but more accurate to reflect the input data, and with more hidden layers the network becomes more capable to classify data with non-linear distribution which might explain why the 4-layer model can surpass the bottleneck experienced by the 3-layer model. The 4-layer network also has more parameters to train, so we could expect a higher accuracy with the increase of data size, for the current data size is only 1488 which is relatively small and not ideal for the network to learn complex mapping.

1.4. Conclusions

After experiments we can conclude that the optimal batch size is 8, optimal decay parameter is 1e-9, and the optimal number of neurons is 25. Additionally, 4-layer neural networks perform better than 3-layer neural network after large training epochs.

What we learned from the experiments is when choosing parameters for a network, in order to select the optimal set of parameters, oftentimes we will need to apply the strategy of trial and error.

2. Part B: Regression Problem

2.1 Introduction

Part B aims to use the data from the Graduate Admission Prediction which contains several parameters like GRE score, TOEFL score, University Rating and strength of Statement of Purpose and Letter of Recommendation, undergraduate GPA, research experience, that are considered to be important during the application for Master Programs. The assignment attempts to predict the chance of getting an admit ranging from 0 to 1.

2.2. Methods

2.2.1. Split Training and Testing Data

We utilized the built-in function ‘train_test_split’ from sklearn to randomly split the entire sample into training set and testing set with size ratio of 70:30. This is similar to what we have implemented in Part A.

```
test_size = 0.3

train_X, test_X, train_Y, test_Y = train_test_split(trainX, trainY,
test_size = test_size, random_state=1)
```

The split result in a training set with size 280 and testing set with size 120.

2.2.1. Normalization of Inputs

Serial No.	GRE Score	TOEFL Sco	University	SOP	LOR	CGPA	Research	Chance of Admit
1	337	118	4	4.5	4.5	9.65	1	0.92
2	324	107	4	4	4.5	8.87	1	0.76
3	316	104	3	3	3.5	8	1	0.72
4	322	110	3	3.5	2.5	8.67	1	0.8
5	314	103	2	2	3	8.21	0	0.65
6	330	115	5	4.5	3	9.34	1	0.9
7	321	109	3	3	4	8.2	1	0.75
8	308	101	2	3	4	7.9	0	0.68
9	302	102	1	2	1.5	8	0	0.5
10	323	108	3	3.5	3	8.6	0	0.45

By observing the sample data set, we can note that there are relatively huge variations between features that might affect the training result of the model. For example: the GRE scores can reach over 300 and TOEFL score over 100, while other features are only single-digits

```
train_X = (train_X - np.mean(train_X, axis=0)) / np.std(train_X,
axis=0)
test_X = (test_X - np.mean(test_X, axis=0)) / np.std(test_X, axis=0)
```

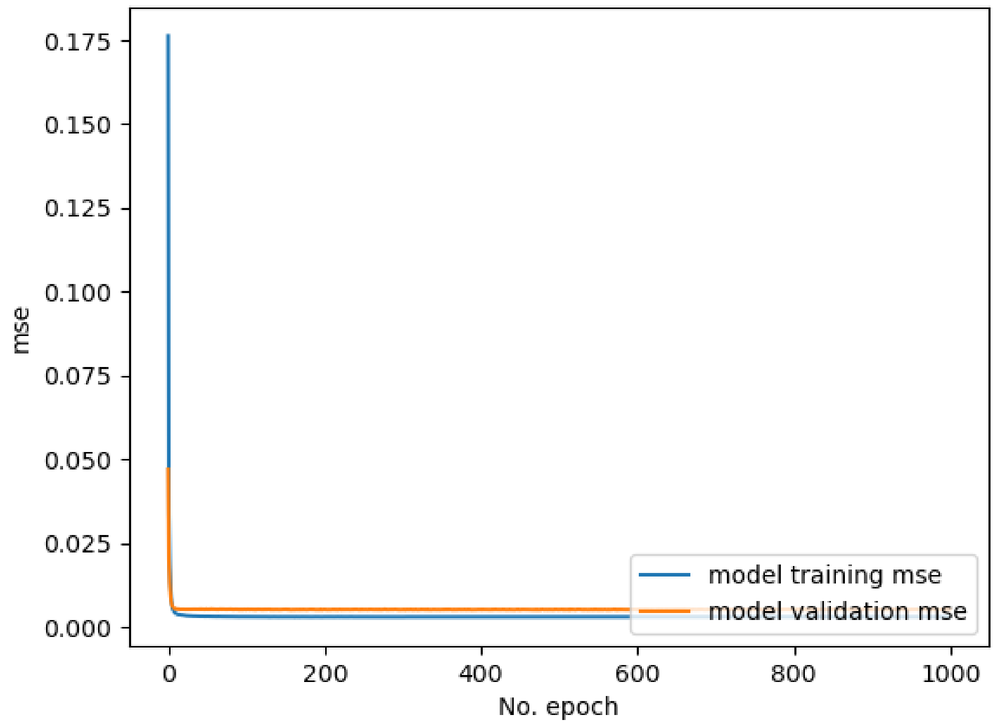
To eliminate the variations between fields to prevent a few features from having a dominant effect on training, we Normalize the input to have standard normal distributions N(0,1).

2.3. Experiments and Results

2.3.1. Question 1

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

We have built the model using the following implementation and trained it. We acquired the training mse and validation mse against the training epoch:

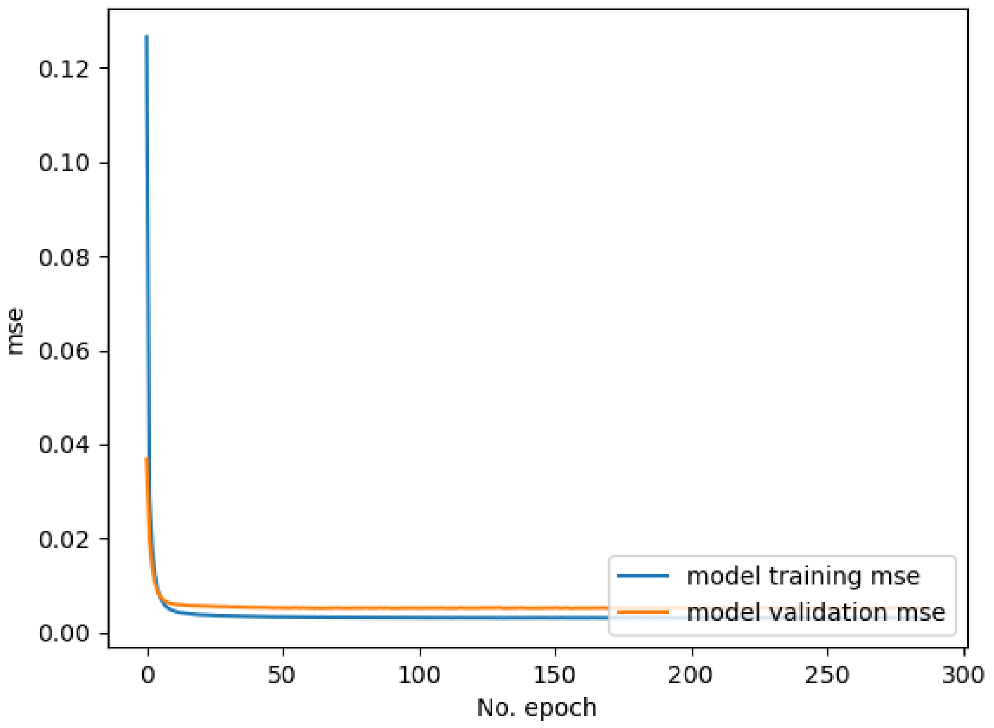


b) State the approximate number of epochs where the test error is minimum and use it to stop training.

We implemented the in-built EarlyStopping callback from Keras to record the minimum test error:

```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss',
patience=10)
```

We set the patience to be 10, which means the training will stop if the error has not improved in 10 consecutive epochs. We then acquired the following graph for training with early stopping implemented:



The log of the training with early stopping enabled is as below:

Epoch 288/1000

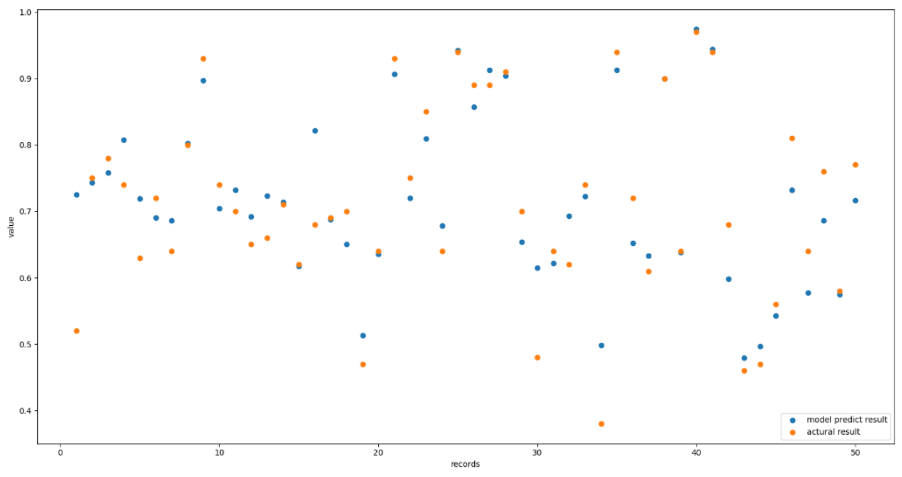
35/35 - 0s - loss: 0.0035 - mse: 0.0030 - val_loss: 0.0056 - val_mse: 0.0051

We can observe that the training stopped at epoch 288.

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

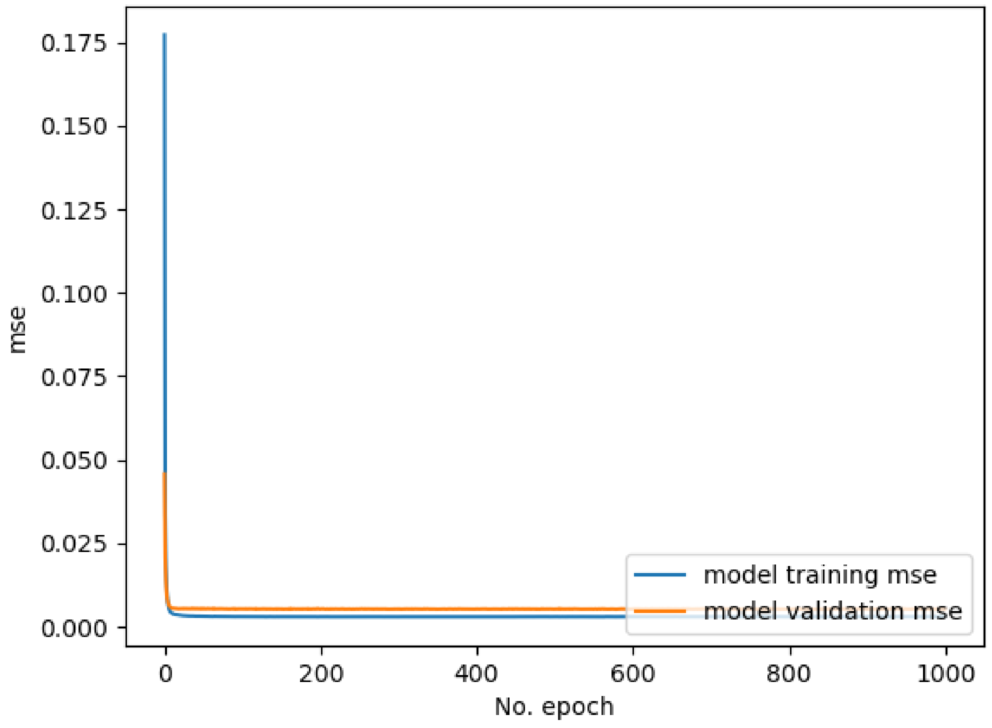
```
# Split the data randomly into 1:7, we will use predict_x to evaluate the model accuracy
rest_X, predict_X, rest_Y, predict_Y = train_test_split(trainX, trainY, test_size = 0.125, random_state=2)
result = model.predict(predict_X)
```

We split the data into 50:350 randomly, use the 50-sized sample to predict the values and compare the result with the actual values:



2.3.2. Question 2

We trained the model with full feature set and plotted the training mse and validation mse against the training epochs, and evaluated the mse on the testing set:



The model with full feature set has an mse of 0.005272765178233385 on testing set. We then adopted recursive feature elimination on each of the seven features, and recorded their validation MSEs on the testing set as below:

Feature to Remove	Validation MSE
GRE	0.005519551690667868
TOEFL	0.0053750695660710335
University Rating	0.005254045128822327
SOP	0.005348321981728077
LOR	0.005507092457264662
CGPA	0.00576110789552331
Research	0.00542471744120121

From the recorded validation MSE we can observe that removing the feature University Rating not only did not cause the validation MSE to increase, but caused the validation MSE to decease, denoting the network can produce better regression results. Thus, we eliminate the feature University Rating first.

After removing the feature University Rating, we recursively eliminate one of the remaining features and recorded the validation MSE to produce the following table:

Feature to Remove	Validation MSE
GRE	0.005464032292366028
TOEFL	0.005425973795354366
SOP	0.005357729736715555
LOR	0.0055208331905305386
CGPA	0.005893724504858255
Research	0.005567904096096754

We observe that removing the feature SOP caused the slightest increase in validation MSE.

We then have the following comparisons between 7 features, 6 features, and 5 features:

MSE		
7-feature	6-feature	5-feature
0.005272765178233385	0.005254045128822327	0.005357729736715555

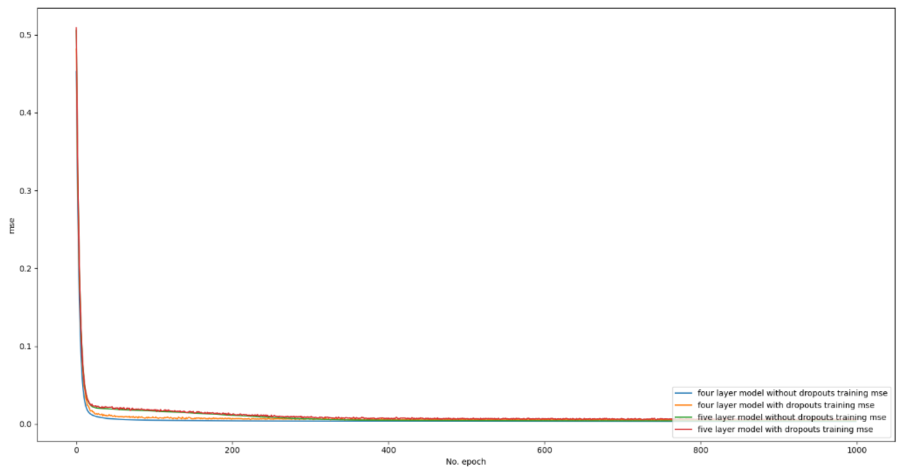
The pattern can be explained that the feature university rating does not have a strong correlation with the admission chance, thus removing it actually enhanced the prediction results. After removing the feature University Rating, we can observe that further removing any feature will increase validation MSE, meaning the remaining features all has a relatively strong influence on the admission chance. Thus, removing any of them would result in worse prediction results.

2.3.2. Question 3

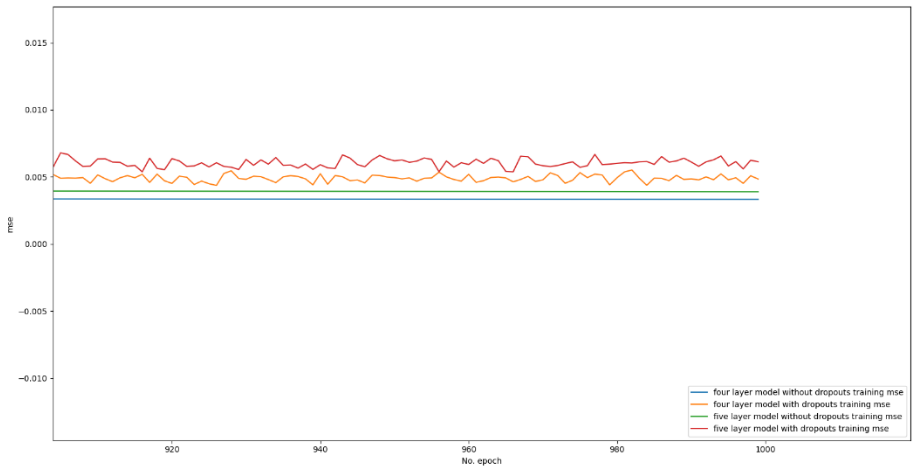
We are required to build four networks – four/five layers with and without dropouts. We’ll use the four-layer model with dropouts to demonstrate the method we implemented to build the models, and the rest are inferable:

```
# create 4 layer model with dropouts and record the mse
four_layer_model_with_dropouts = keras.Sequential()
four_layer_model_with_dropouts.add(keras.layers.Dense(50,
input_dim=6, activation='relu',
                                 kernel_initializer='random_normal',
                                 bias_initializer='zeros',

kernel_regularizer=tf.keras.regularizers.l2(1e-3)))
# Add dropout layer with keep probability of 0.8
four_layer_model_with_dropouts.add(keras.layers.Dropout(0.2))
four_layer_model_with_dropouts.add(keras.layers.Dense(50,
activation='relu',
                                 kernel_initializer='random_normal',
                                 bias_initializer='zeros',

kernel_regularizer=tf.keras.regularizers.l2(1e-3)))
four_layer_model_with_dropouts.add(keras.layers.Dropout(0.2))
four_layer_model_with_dropouts.add(keras.layers.Dense(1))
optimizer = keras.optimizers.SGD(learning_rate=1e-3)

four_layer_model_with_dropouts.compile(optimizer=optimizer,
                  loss='mean_squared_error',
                  metrics=['mse']
                  )
histories['four_layer_model_with_dropouts'] =
four_layer_model_with_dropouts.fit(train_X, train_Y,
                                        epochs=epochs,
                                        batch_size=batch_size,
                                        verbose = 0,
                                        validation_data=(test_X,
test_Y))
# evaluate the model on testing set
four_layer_model_with_dropouts_eval =
four_layer_model_with_dropouts.evaluate(test_X, test_Y)
print("Four layer model with dropouts evaluation result: " +
str(four_layer_model_with_dropouts_eval))
```

We build the networks accordingly and trained them on the training set, and plot their training MSEs as following:



If we zoom in around epoch 1000, we can observe the following pattern:



We can observe that generally the layer with dropouts perform better than layers without dropouts, and five-layer model performs better than layers with dropouts. We evaluated the models on the training set and recorded the MSEs of each network as following:

Network Model	MSE
Four layers without dropouts	0.005541009362787008
Four layers with dropouts	0.0053643519058823586
Five layers without dropouts	0.005530552938580513
Five layers with dropouts	0.005212908145040274

2.4. Conclusions

By doing recursive feature elimination we can enhance the performance of by eliminating relatively low correlated data that could disturb the prediction result.

We can also learn that adding dropouts after each hidden layer could enhance the performance of the network.