Question 1

In this question, I randomly sample 20% of the training set as training data, and according to Piazza, I split test dataset, half for testing dataset and half for validation.

According to the problem description, I choose batch size equal to 32, and use adam as the optimizer for MLP, CNN1, and CNN2.

To increase the speed of finding optimizing model, firstly I start by picture normalization by using `tf.keras.utils.normalize to preprocess attribute`

```
model_MLP.add(Dense(10, activation=tf.nn.softmax))
model_CNN_1.add(Dense(10, activation=tf.nn.softmax))
model_CNN_2.add(Dense(10, activation=tf.nn.softmax))
```

This is my output layer choice. Because this is a multi-classification problem, I choose softmax as the activation function in my output layer; Furthermore, because there are total 10 classes in the CIFAR10 dataset, the number of nodes of the output layer is 10.

As for the loss function, I choose 'categorical_crossentropy' because this is a multi-classification problem.

The description of the MLP in this question has two hidden layers, then I do the experiment to get the result for one hidden layer and three hidden layers.

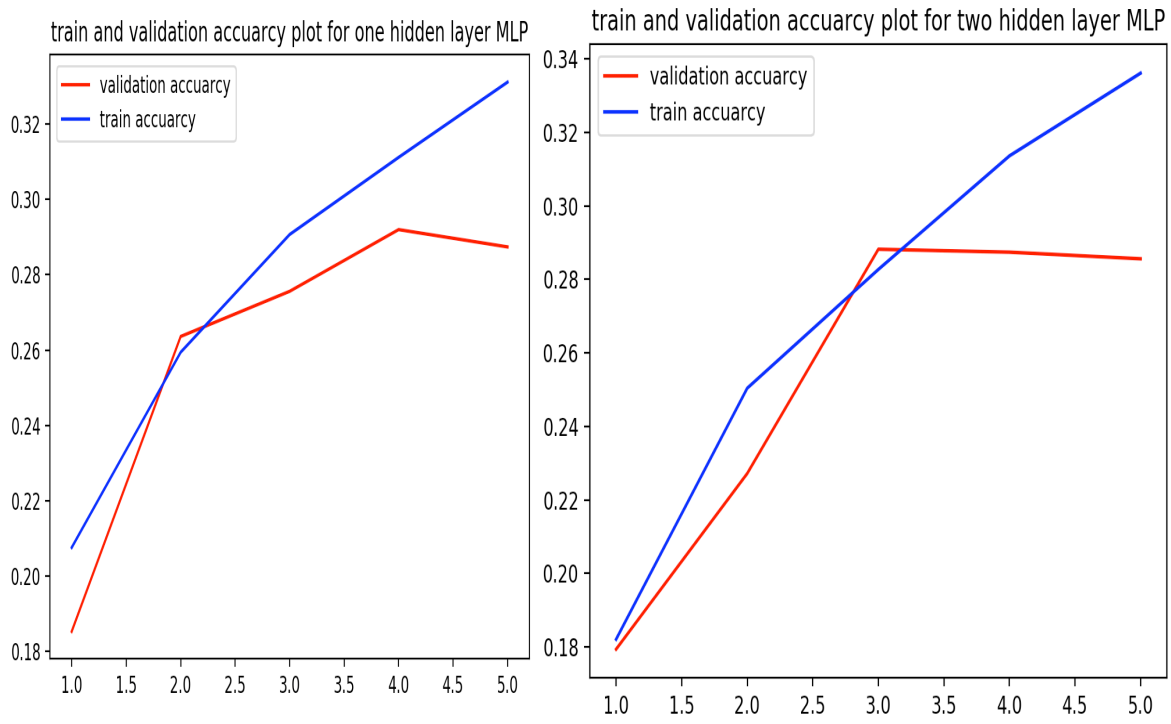


Figure 1

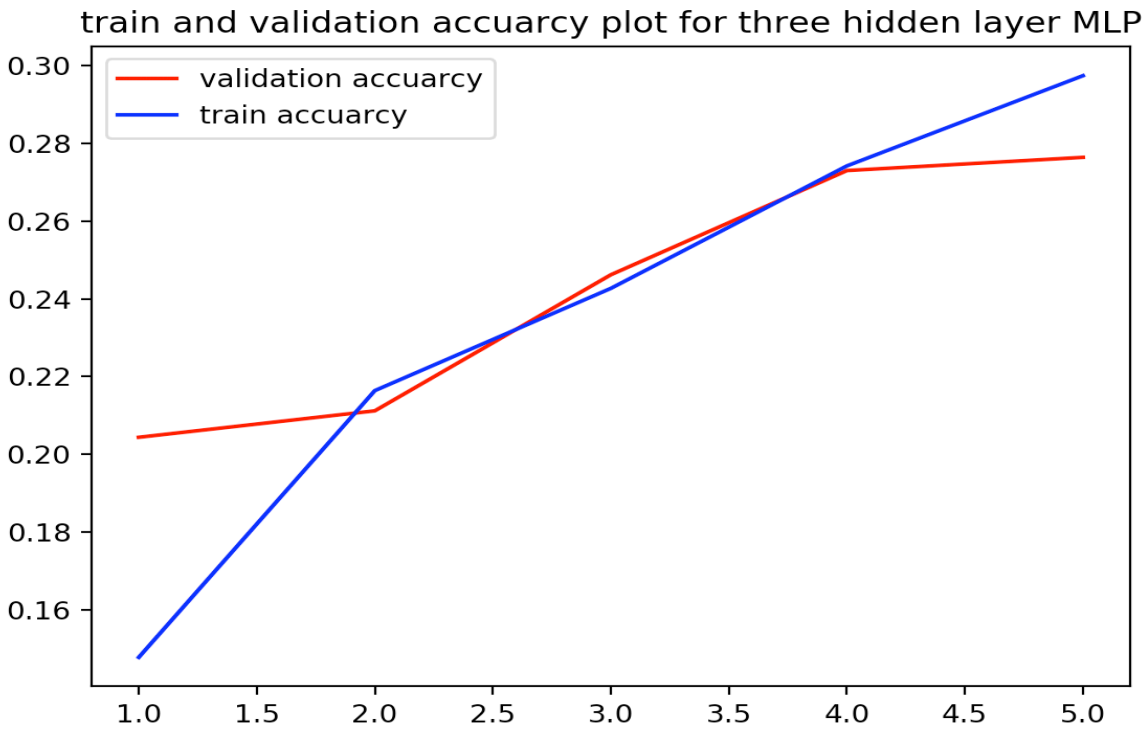


Figure 2

From Figure 1 and Figure 2, we can see that no matter how many hidden layers we choose for MLP, the training accuracy will increase with the increase of number of epochs. And we can see that at epoch 5, MLP having two hidden layers has a little higher accuracy. These three situations' validation accuracy are almost same around 28%. In a word, changing the number of hidden layers has a little impact on the results for this problem

The description of the MLP in this question has 512 nodes in each hidden layer, then I do the experiment to get the result for 256 and 1024 nodes for each hidden layer.

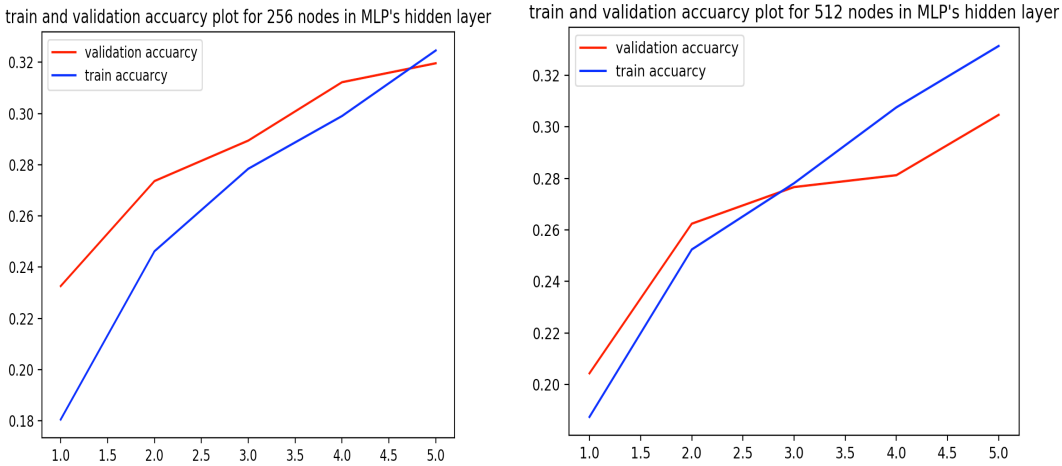


Figure 3

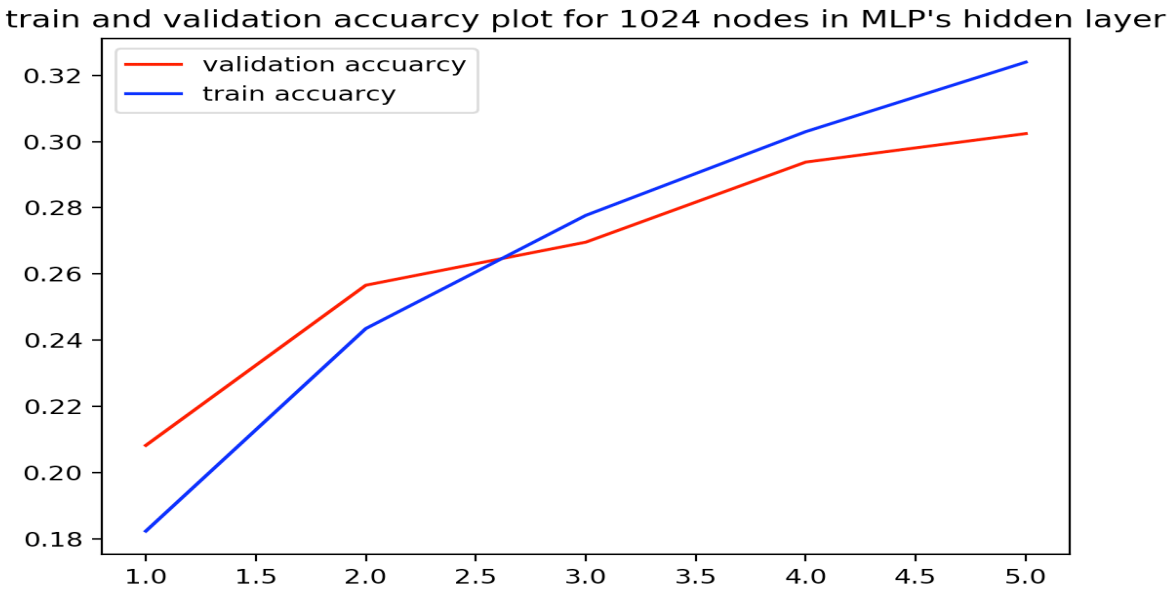


Figure 4

From Figure 3 and Figure 4, we can see that both validation accuracy and train accuracy have similar trend in these three situations. When there are 256 nodes in the hidden layer, the MLP has a little higher validation accuracy, at the last epoch. In conclusion, changing the number of nodes in the hidden layer has little impact in this problem for MLP.

```
MLP model test accuarcy:  0.3555999994277954
```

```
CNN1 model test accuarcy:  0.4837999939918518
```

```
CNN2 model test accuarcy:  0.451200008392334
```

We can get that the MLP test accuracy is lower than CNN models, this is mainly because that the convolution layer has a stronger ability to extract image features that classify the images.

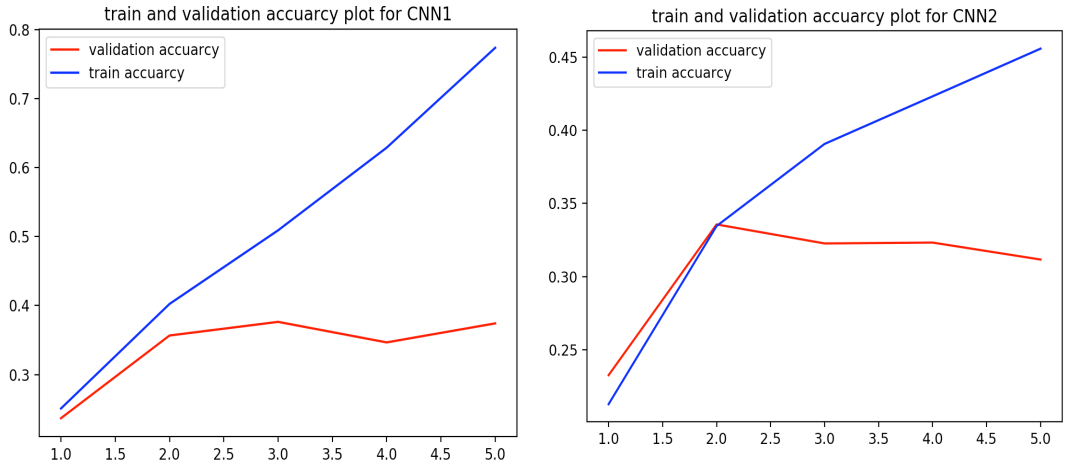


Figure 5

From Figure 5, we can get that CNN1 which has a much higher training accuracy compared with CNN2; however, the validation accuracy of CNN1 is only a little higher than that of CNN2; therefore, we can get that CNN1 has an overfitting problem. And CNN2 with two dropout layers in dense layer lessen the effect of such an overfitting problem, and it has a better training accuracy compared with MLP.

```
Epoch 1/5
313/313 [==============================] - 62s 199ms/step - loss: 2.0315 - accuracy: 0.2507 - val_loss: 2.3711 - val_accuracy: 0.2366
Epoch 2/5
313/313 [==============================] - 61s 196ms/step - loss: 1.6499 - accuracy: 0.4024 - val_loss: 2.0020 - val_accuracy: 0.3566
Epoch 3/5
313/313 [==============================] - 67s 213ms/step - loss: 1.3675 - accuracy: 0.5092 - val_loss: 1.9402 - val_accuracy: 0.3764
Epoch 4/5
313/313 [==============================] - 65s 208ms/step - loss: 1.0594 - accuracy: 0.6291 - val_loss: 2.1758 - val_accuracy: 0.3466
Epoch 5/5
313/313 [==============================] - 60s 190ms/step - loss: 0.6780 - accuracy: 0.7740 - val_loss: 2.3475 - val_accuracy: 0.3742
157/157 [==============================] - 3s 19ms/step - loss: 1.6403 - accuracy: 0.4760
CNN1 model test accuarcy:  0.47600001096725464
Epoch 1/5
313/313 [==============================] - 16s 52ms/step - loss: 2.1214 - accuracy: 0.2128 - val_loss: 2.1192 - val_accuracy: 0.2326
Epoch 2/5
313/313 [==============================] - 16s 50ms/step - loss: 1.8211 - accuracy: 0.3345 - val_loss: 1.8749 - val_accuracy: 0.3356
Epoch 3/5
313/313 [==============================] - 14s 46ms/step - loss: 1.6777 - accuracy: 0.3906 - val_loss: 2.1425 - val_accuracy: 0.3226
Epoch 4/5
313/313 [==============================] - 14s 45ms/step - loss: 1.5882 - accuracy: 0.4231 - val_loss: 2.0751 - val_accuracy: 0.3232
Epoch 5/5
313/313 [==============================] - 14s 46ms/step - loss: 1.5012 - accuracy: 0.4557 - val_loss: 2.2036 - val_accuracy: 0.3116
157/157 [==============================] - 2s 11ms/step - loss: 1.5379 - accuracy: 0.4562
CNN2 model test accuarcy:  0.4562000036239624
```

Figure 6: Training process for CNN1(up part) and CNN2(below part)

From Figure 6, we can clearly get that the CNN2 has a much less training time compared with that of CNN1. This could be caused by the drop out layer which drop some features which lessen the training computation task of the model; Furthermore, the pooling layer has the function that reduce the size of the feature maps outputted by convolution layers.

In a word, the two drop layers in CNN2 has the ability to lessen overfitting effect and increase the training efficiency by decreasing the training time of the model; Furthermore, the pooling layer also has a good effect on decreasing the training time.

When there are more epochs to train CNN model, the training accuracy could be very high and close to 1; however, because of overfitting, the validation accuracy could keep the accuracy around 35%. And this overfitting could also be caused by not enough number of training data because I only sample 20% of the original training dataset. As for the test accuracy, it could increase a little with the increase of the number of epochs

The network has the overfitting problem. At last, I give some recommendations to improve the network, Firstly, I could use only one convolution layer, which guarantees a higher training accuracy and test accuracy compared with MLP. Secondly, I will decrease the number of nodes in the CNN and dense layer to decrease the complexity of the network to improve validation accuracy; Thirdly, I will

add the drop out layer after the dense layer and also the max pooling layer after convolution layer to decrease the overfitting.

Question 2

Firstly, I load the original dataset, then slice out the date and close column. Then from the bottom to up, i.e. from the oldest data to the newest data, each continuous three days' open, high, low prices and volume as features (total 12 features), and after these continuous three days, the fourth day's open price as the target. Then according to the description of this problem, I randomized the data and split 70% of the dataset to training dataset, and 30% to the test dataset. The created training dataset and test dataset are saved in the data directory and the trained model is saved in the model directory.

Because we can easily get from the original dataset that the volume feature is magnitude larger than the other three features, I scale these four features to the same range between 0 and 1 using preprocessing.MinMaxScaler(feature_range=(0, 1)) scaler.

Firstly, this problem is like the linear regression problem, not a classification problem; therefore, I choose the 'mae' as the loss function, and the optimizer is 'adam' like question 1.

Then I start by using two, three, four LSTM layers to find the impact of the number of the LSTM layers

```
test_loss_RNN when number of LSTM layer is 2:  2.3911609649658203
test_loss_RNN when number of LSTM layer is 3:  2.4456374645233154
test_loss_RNN when number of LSTM layer is 4:  2.3039181232452393
```

We can get that the number of LSTM layers has little effect on the performance on the test dataset Therefore, I choose the number of LSTM layers is two for simplicity.

Then I run my model to find the impact of the number of nodes in the LSTM layers. For this, I choose 128, 256, and 512 respectively.

```
test_loss_RNN when number nodes in LSTM layer is 128:  3.0894198417663574
test_loss_RNN when number nodes in LSTM layer is 256:  2.8108386993408203
test_loss_RNN when number nodes in LSTM layer is 512:  2.4323344230651855
```

According the results and consider the training speed, I choose 512 nodes in the LSTM layer.

Then I want to find the number of dense layers impact (except for the output layer), in this part I choose 0, 1, 2, and 3 dense layers.

```
test_loss_RNN when the number of dense layer is 0:  3.320945978164673
test_loss_RNN when the number of dense layer is 1:  2.440660238265991
test_loss_RNN when the number of dense layer is 2:  3.663761615753174
```

```
test_loss_RNN when the number of dense layer is 3:  4.859391212463379
```

Then the number of the dense layer I choose is 1

Then I explore the effect of the number of nodes in the dense layer. In this part, I choose 128, 256 and 512.

```
test_loss_RNN when the number of nodes in dense layer is 128:  3.7909059524536133
test_loss_RNN when the number of nodes in dense layer is 256:  3.860928535461426
test_loss_RNN when the number of nodes in dense layer is 512:  3.400463104248047
```

Therefore, I choose 512 nodes in the dense layer, which is the same number of the nodes in the LSTM layer.

At last, I choose ‘relu’ as my activation function in my dense layer, because this problem is not the classification problem, and ‘relu’ has a very good convergence in training phase.

At last, my final design is two LSTM layer with 512 nodes, one dense layer with 512 nodes and ‘relu’ activation function, and one output layer, which output a scalar.

The number of epochs I choose is 100, use ‘mae’ as loss function, and validation split is 0.1.

Based on the below design steps and parameters choice, the output training loop is shown below.

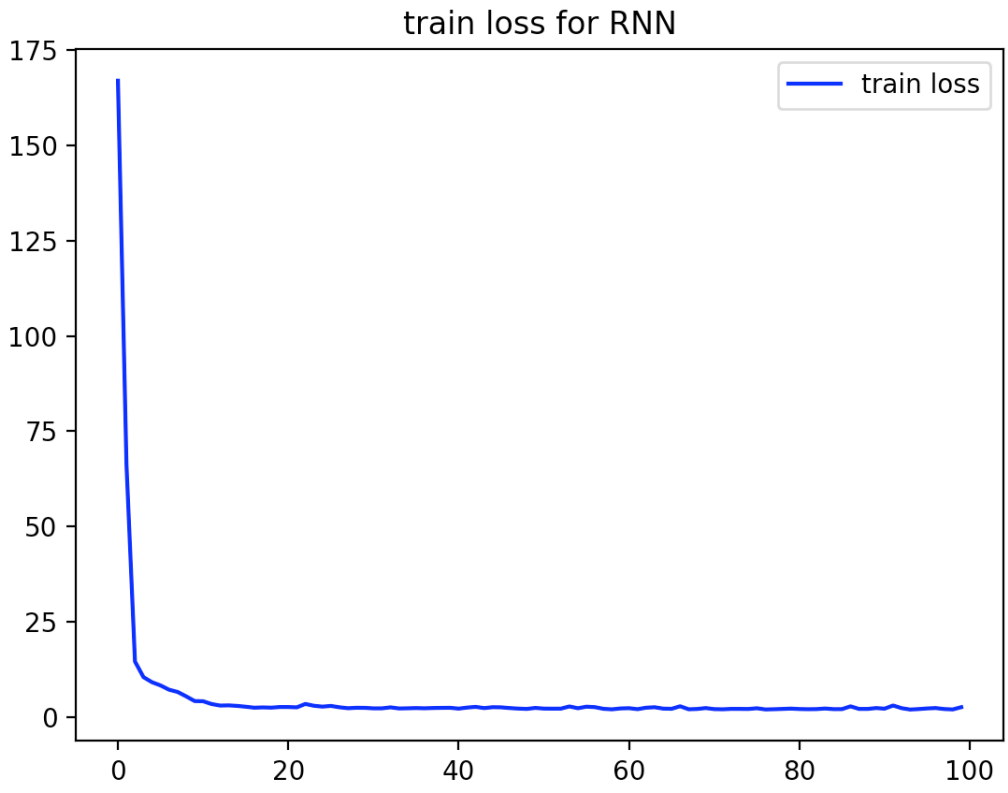
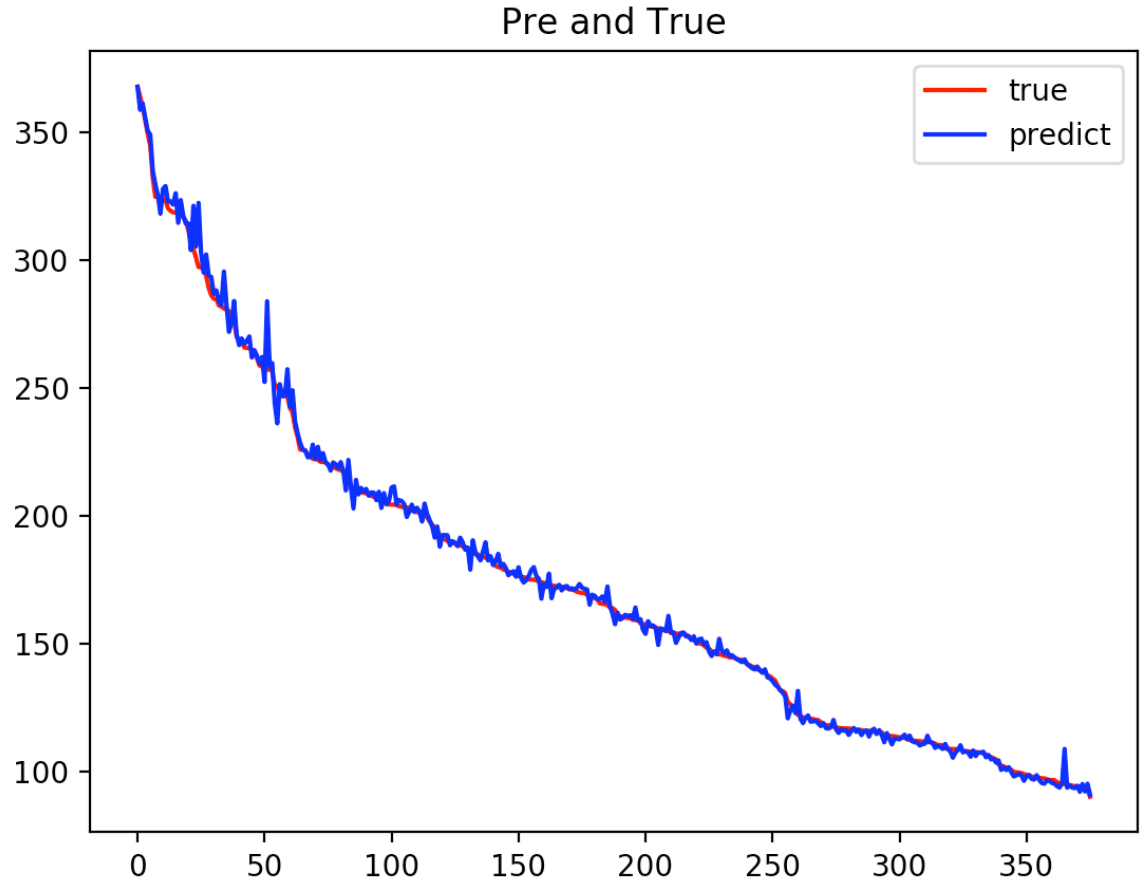


Figure 7

As shown in Figure 7, the train loss decreases with the increase number of epochs. At around 20 epochs, the train loss almost keeps stable; to optimize my design, we can choose 20 epochs as parameters to decrease the training time.




```
12/12 [==============================] - 0s 2ms/step - loss: 2.0788
test_loss_RNN:  2.0788331031799316
```

Figure 8

To have a beautiful plot, I sort the test dataset based on the target opening price. The predict and true plot, and the test process is shown in Figure 8. We can get that my designed model performance on test dataset is very good with only 2.0788 loss.

What would happen if you used more days for features?

There are some cases. When we only use a little more days for features, these features could be necessary to predict results, thus leading to a better result and converging faster; However, if we include a lot more features, this could lead to a larger training dataset to train the model. This is the reason why sometimes we need the feature reduction preprocessing. If we do not have enough training samples, this could lead to a higher loss; Furthermore, this increase of number of features could lead to an overfitting problem, which leads to a bad performance on test dataset.

Question 3

Preprocessing steps:

```
# load your training data
neg_train_file = glob.glob(r'data/aclImdb/train/neg/*.txt')
pos_train_file = glob.glob(r'data/aclImdb/train/pos/*.txt')
```

Firstly, navigate the train neg and train pos directory, return a txt file list.

```
def get_documents(FilePath, label, documents, all_words):
    for file_path in FilePath:
        documents_ele = []
        rev_ele = []
        f = open(file_path, "r")
        lines = f.readlines()

        for line in lines:
            line.strip()
            word_list = nltk.word_tokenize(line)
            for word in word_list:

                rev_ele.append(word.lower())
                all_words.append(word.lower())

        documents_ele.append(rev_ele)
        documents_ele.append(label)
        documents.append(documents_ele)


    f.close()
```

Then in this get_documents function I read each line in each file for both neg and pos files in the train dataset. After this, I do the word tokenization for each line. At last, I get the all words list which contains all the words consisted in the neg and pos files in the train dataset.

```
all_words = nltk.FreqDist(all_words)
word_features = list(all_words.keys())[:3000]


word_features_model = open("models/word_features_model.pkl", 'wb')
str = pickle.dumps(word_features)
word_features_model.write(str)
word_features_model.close()
```

After get this all word list, I use the nltk.FreqDist function get the list of words, which sorted according to the frequency of the words. Then I choose the top 15000 words with highest frequency out of the total around 700000 words. And the reason I choose 15000 is that the highest frequency could be some punctuations, such like ‘,’, ‘.’ etc. and could be some not important word, such like ‘a’, ‘the’, etc. The choosing process for this is shown at the bottom of

this report. And I choose 15000, which would enough to include those important words, which can help us classification. These 15000 words can help us to construct the feature vector to feed in the MLP network.

And this word features is also pickled for the test phase.

And the reason why I choose MLP is that it is a normal binary classification problem, and MLP has a good performance on binary classification problem; and this is also the reason why I choose sparse_categorical_crossentropy as loss function and choose sigmoid as activation function.

```
def construct_feature_vector(word_features,documents):
    attribute = []
    label = []
    for document in documents:
        feature_vectors = {}
        words = set(document[0])
        for word in word_features:
            if word in words:
                feature_vectors[word] = 1
            else:
                feature_vectors[word] = 0
        processed_data_ele = []
        vector = []
        for value in feature_vectors.values():
            vector.append(value)
        attribute.append(vector)
        label.append(document[1])
    for i in range(len(label)):
        if (label[i] == 'pos'):
            label[i] = 1
            continue
        else:
            label[i] = 0


    attribute = np.array(attribute, dtype = int)
    label = np.array(label,dtype = int)

    return  attribute, label
```

This construct_feature_vector function helps me to construct the feature vector. In the documents list, it contains all the word tokenized file. And then I loop this list, to examine all the words in each file's word list to check if the file contains the important word. If the file contains the word, then it will let that index element to be 1. At last, form a (3000,) length feature vector for each file. And because I need to feed the attribute and label to the network, I also make the label to be int (1 for pos, 0 for neg).

The construct_feature_vector function and get_documents function are written in my utils.py.

Then let's go into the model design part:

In this part, I explore number of Dense layers (except for the output layer), the number of nodes in each dense layer.

For find the best number of dense layers, I do experiment for 1, 2 and 3 dense layers.

```
NLP model accuarcy when the number of dense layer is 1:  0.8474799990653992
```

```
NLP model accuarcy when the number of dense layer is 2:  0.848360002040863
NLP model accuarcy when the number of dense layer is 3:  0.8464800119400024
```

From the results, we can get that the number of dense layers in this problem almost has no effect on the accuracy on the test datset; therefore, for the simpilicity of the network, I choose one dense layer.

Then I do experiment to explore the number of nodes in the dense layer (except for the output layer). In this part, I choose number of nodes 256, 512, and 1024 to do experiment.

```
NLP model accuarcy when the number of nodes in dense layer is 256:  0.8456799983978271
NLP model accuarcy when the number of nodes in dense layer is 512:  0.8496000170707703
NLP model accuarcy when the number of nodes in dense layer is 1024:  0.8486800193786621
```

Similar as the results of experiment on the number of dense layer, the number of nodes in dense layer also has a little impact on the accuracy on the test dataset. I choose 512, which shows a little higher accuracy.

Then I want to find the length of feature vector by doing experiments on length 1000, 3000, 5000.

```
NLP model accuarcy when length of feature vector is 1000:  0.8082799911499023
NLP model accuarcy when length of feature vector is 3000:  0.8498399985370636
NLP model accuarcy when length of feature vector is 5000:  0.8584399819374084
NLP model accuarcy when length of feature vector is 8000:  0.8626000285148621
NLP model accuarcy when length of feature vector is 15000:  0.8543199896812439
```

After this experiment on the length of feature vector, the length I choose is 8000 for now

At last, I want to do the experiment on the parameter epoch, because during the training process, I observe that this model has the overfitting problem; therefore, I want to explore a epoch to do the early stopping. The epoch I want to explore is 1, 3, 5

```
NLP model accuarcy when the epoch is 1:  0.8747599720954895

NLP model accuarcy when the epoch is 3:  0.8678799867630005

NLP model accuarcy when the epoch is 5:  0.8640000224113464
```

Therefore, to avoid the overfitting problem, I choose epoch is equal to 1.

Then I come back to experiment on the length of feature vector, when epoch = 1

```
NLP model accuarcy when the length of feature vector is equal to 15000:  0.8698800206184387

NLP model accuarcy when the length of feature vector is equal to 30000:  0.8695600032806396
```

At the final decision, I choose the length of the feature vector is 15000, because when it increases to 30000, the accuracy on the test dataset is almost same.

The final design of my model is one flatten layer, one dense layer with sigmoid activation function, one output layer with the sigmoid activation function, adam optimizer, sparse_categorical_crossentropy loss function, 1 epoch, and 15000 length feature vector. And I achieve around 87% accuracy on the test dataset.

As I mentioned in the question 2, a long feature vector could lead to overfitting problem. And in my experiment process, the overfitting problem is obvious, when I do experiment on the 100 epochs, the train accuracy is 100%, however not a large change on the validation accuracy and test accuracy. For example, when the feature vector length is equal to 3000, and 100 epochs training process is shown in below figure.

```
1407/1407 [==============================] - 12s 8ms/step - loss: 9.9312e-04 - accuracy: 0.9996 - val_loss: 1.0961 - val_accuracy: 0.8484
Epoch 46/100
1407/1407 [==============================] - 12s 8ms/step - loss: 2.0664e-04 - accuracy: 1.0000 - val_loss: 1.1471 - val_accuracy: 0.8512
Epoch 47/100
1407/1407 [==============================] - 12s 8ms/step - loss: 7.9582e-05 - accuracy: 1.0000 - val_loss: 1.1825 - val_accuracy: 0.8524
Epoch 48/100
1407/1407 [==============================] - 12s 8ms/step - loss: 6.6784e-05 - accuracy: 1.0000 - val_loss: 1.2151 - val_accuracy: 0.8520
Epoch 49/100
1407/1407 [==============================] - 12s 8ms/step - loss: 3.5295e-05 - accuracy: 1.0000 - val_loss: 1.2482 - val_accuracy: 0.8524
Epoch 50/100
1407/1407 [==============================] - 12s 8ms/step - loss: 3.3535e-05 - accuracy: 1.0000 - val_loss: 1.2798 - val_accuracy: 0.8516
Epoch 51/100
1407/1407 [==============================] - 12s 8ms/step - loss: 3.2695e-05 - accuracy: 1.0000 - val_loss: 1.3085 - val_accuracy: 0.8528
```

My experiment process later part is focus on this overfitting problem, and my strategy is to do early stopping, thus the epoch is 1. And as mentioned in question 2, some necessary feature

could lead to a higher accuracy on test dataset, thus my experiment increases from 3000 to 15000 to explore the best accuracy on the test case, to include those necessary words.