You are provided with a dataset for stock price prediction for 5 years with one sample per day (q2_dataset.py). Create a Recurrent Neural Network using the machine learning platform of your choice to predict the next day opening price using the past 3 days Open, High, and Low prices and volume. Therefore, each sample will have (4*3 =) 12 features.

1. **Explanation of how you created your dataset.**

The dataset was created keeping in mind that the next day's opening will be based on the past 3 days Open, High, and Low prices and volume.

- Step1:

  Loading the dataset into a data-frame. The original dataset contains 6 columns those are date, close/last, volume, open, high, low. The shape of the original data set is 1259rows x 6 columns.

- Step 2:

  The Date column had to be changed into date-time format so that the data could be sorted in ascending order of date (because 'predict the next day opening price using the past 3 days'). After the dates were sorted, we created an empty array of size 1259 rows x 6 columns.

- Step 3:

  Iterating through the for loop to create a dataset in which the open of the next day is calculated using the past 3 days Open, High, and Low prices and volume. We are iterating the for loop in such a way (iloc[i+3][3]) that every 4th element in the row i+3 and 4th column, will go to the 12^{th} column named Target. We have 12 columns such that we have 3 days open, high, low and volume.

2. **Any preprocessing steps you followed.**

- Firstly, we divided the newly created dataset into features and targets. The next step was to divide the dataset into 70% training and 30% testing.
- We normalized the training and testing datasets. The normalization method we used was MinMaxScaler. Normalization converts all the values and gets them between 0 to 1. The target data is not normalized.
- The data was then converted into a NumPy array, so that it can be reshaped further into a 3-Dimensional array. LSTM network expects the network to be 3-Dimensional in the form of number of samples, number of time steps and number of features, hence reshaping is needed.

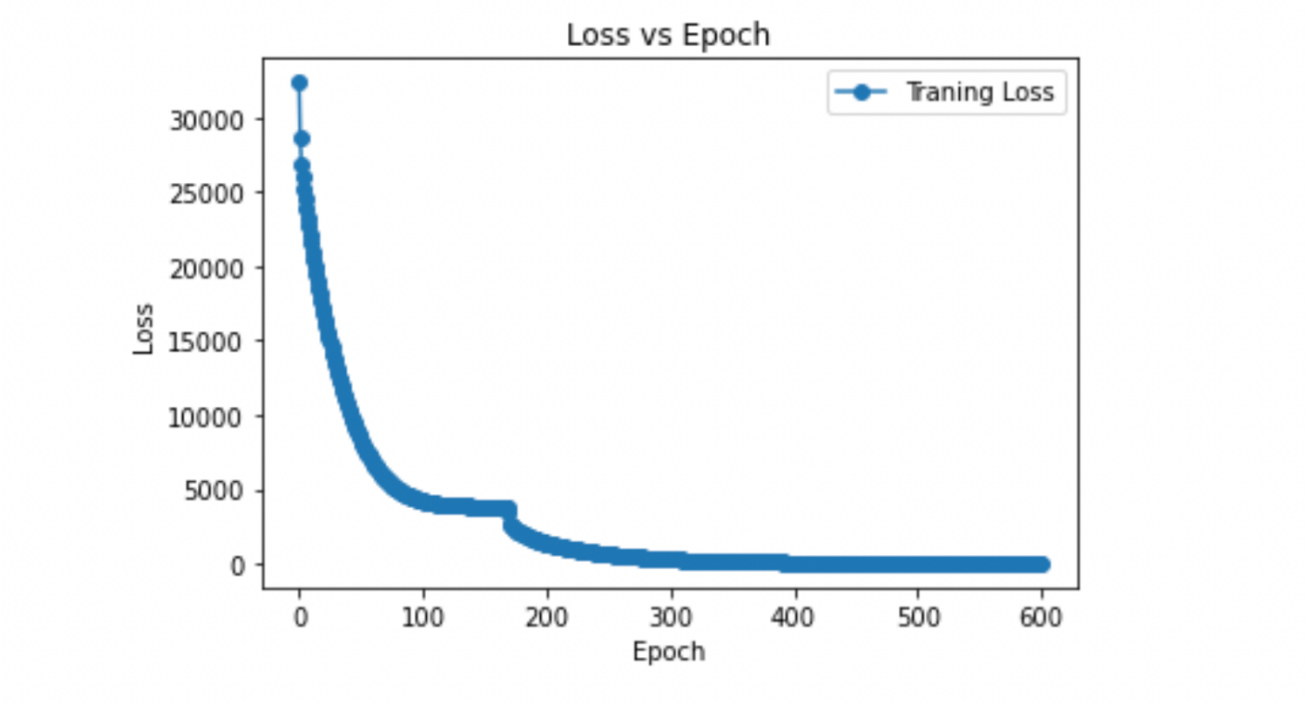
3. **All design steps you went through in finding the best network in your report and how you chose your final design.**

    - We started with a simple RNN network to begin with. We wanted a network that could remember results from past days and predict the next outcome based on the results of past few days. LSTM was the best suitable network for this job because LSTM networks are well-suited to classifying, processing, and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs.
    - To improve the outcomes on the testing set, we also attempted to change the unit numbers of each layer from tiny to large. As a loss function and an evaluation metric, we employed mean absolute error and mean square error, respectively. MSE calculates the squared average difference between the actual data and the anticipated data. MAE calculates the absolute average difference between the actual data and the predicted data. Because the features are normalised, but the targets are not, MAE is better at evaluating whether the predicted value is near to the genuine value.

4. **Architecture of your final network, number of epochs, batch size (if needed), loss function, training algorithm, etc.**

    - Our finals network consists of two fully connected hidden layers. First layer has 50 LSTM units/nodes, and the second layers has 150 LSTM units/nodes. It was observed that with more nodes in each layer, lesser epochs were needed to train the model better. But at the same times, if excessive nodes were used, there was usually a case of overfitting and was taking a huge amount of time to train. Also, it was observed that with increasing the nodes beyond a certain point, the loss was increasing as well.
    - We also tried adding the drop-out layer after both the hidden layers. There was a slight improvement in the loss because of that. The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting.
    - The number of epochs we chose were 600. After increasing the epoch beyond this point, there was no significant change in the loss, which means the network was converged. Mean square error (MSE) was the loss function that we used. The batch size is set to 32 by default, and the training method is defined as the "adam" optimizer, which is a common form of gradient descent that automatically tunes itself and produces good results.

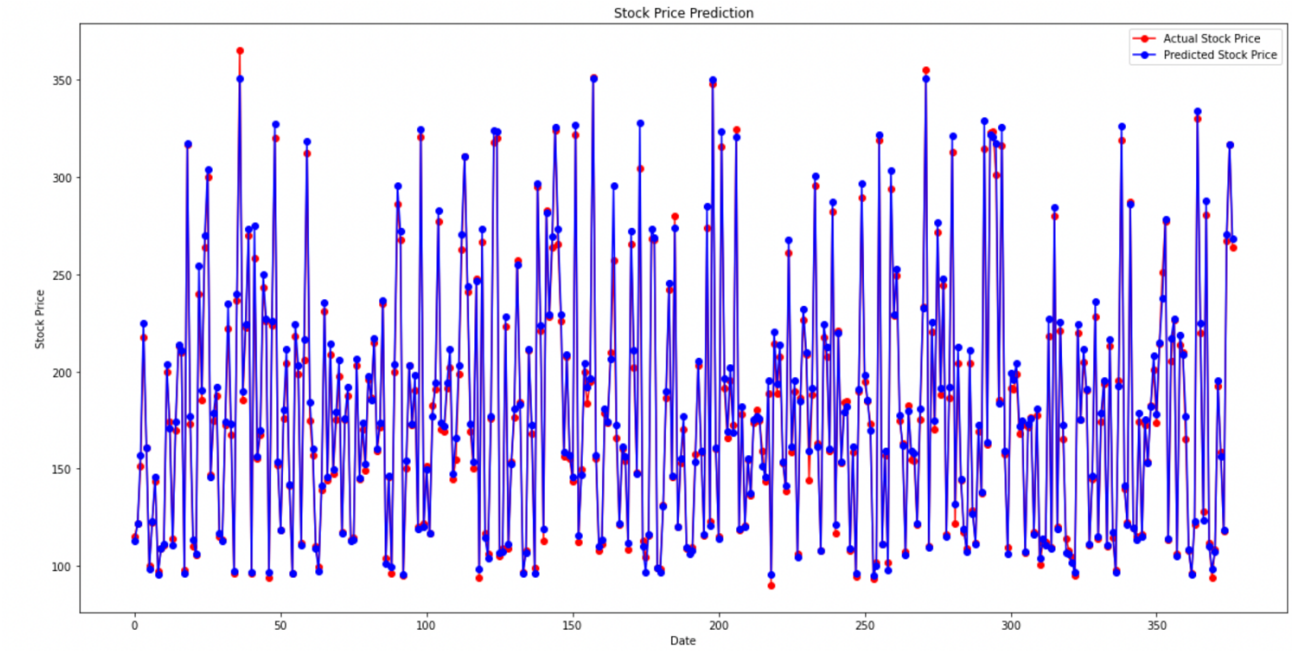
5. **Output of the training loop with comments on your output**



COMMENTS:

- As it is observed from the graph, which is the Loss vs Epoch (training loss), with increase in the e pochs the training loss goes on decreasing. The initial decrease in the loss is quite huge, with the loss dropping from 30000 to nearly about 5000 in a matter of 100 epochs. The decrease in loss st abilizes i.e. it converges after 150 epochs. By the time we reach 600 epoch the loss is very close t o 0. The final loss we get on the training dataset is about 19.1 and the MAE is 2.7179. This might be due to the influence of Adam optimizer. Adam updates the learning rate, as opposed to tradit ional stochastic gradient descent, which maintains a single learning rate that does not vary throu ghout training.

6. Output from testing, including the final plot and your comment on it.



COMMENTS:

- The stored model was used to make the predictions. The graph above the is a result of the testing dataset. The graph is a result of prediction made on the testing data set.
  Following losses were observed for the testing data:

  Total loss (mean absolute error) of the Test Data is: 3.4763709621277665
  Total loss (mean squared error) of the Test Data is: 25.147494232596216

- Compared to the training losses, we can say that our model performed good enough as we did not see any overfitting.
- As we can see from the graph, we were able to predict the pattern on the stocks compared to the real pattern and our model did a good job to follow the real prices of the stocks.
- At some points we see that there was a certain misjudgement in the prediction which can be further improved by using more days as a feature.

7. What would happen if you used more days for features?

- We tried increasing the number of days from 3 to 6. We observed that the model performed better. There was a minute decrease in the training loss and testing loss. LSTM works better if there are long term memories involved in training. To make this model work for more days as features, we need to tune in more hyper-parameters.