Problem 2: Recurrent Neural Networks for Regression

Answers:

1. Explanation of how you created your dataset.

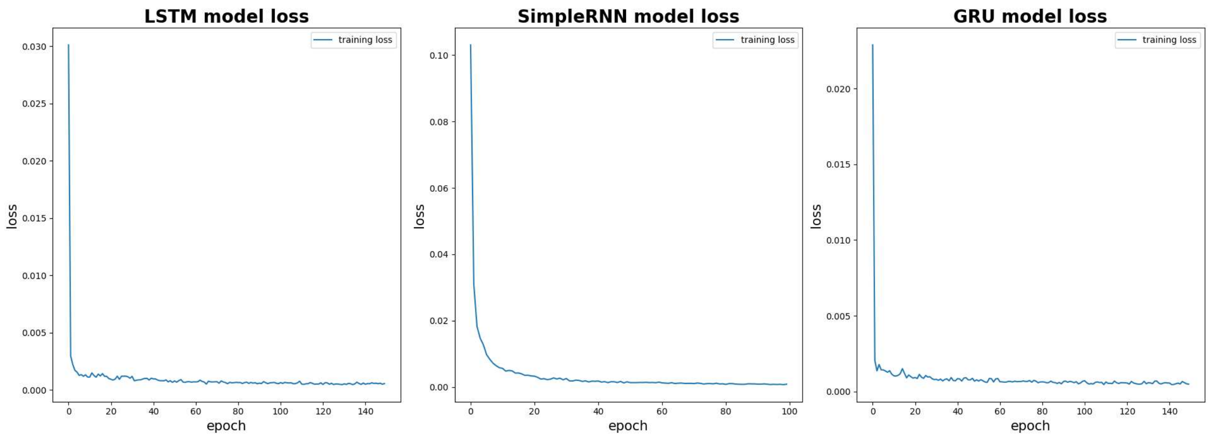
The given dataset was for the period of 5 years starting from 7-9-2015 to 7-8-2020. The entries in the csv files were in the order of latest entry as first entry. As a new dataset had to be created using past 3 days values. Dataset was reversed to read from past to present i.e. 7-9-2015 is now the first entry and the last entry is 7-8-2020.

Now for creating the dataset a function ‘create_dataset’ was made with two arguments data and n_past_days, where n_past_days represents number of days in past to look at for creating the new dataset. Once received both the arguments new data set is generated and returned in the form of a data frame. Column headers are also automatically adjusted as per requirement. For eg. In case of ‘Open’ new dataset with n_past_day = 30 will have three columns ‘Open_0’, ‘Open_1’, ….., ‘Open_28’, ‘Open_29’

With n_past_days = 3, we loose 3 days from the end as we don’t have past entries for those many days. The actual data set shape is 1259x6 and new dataset shape become 1256x13. We dropped ‘Date’ and ‘Closing value’ columns from the main dataset while generating new dataset. The new dataset have 12 features 3 for each main feature Volume, Open, High and Low.

2. Any pre-processing steps you followed
For pre-processing we used min-max scaler from sklearn to normalize the data. As all the models of neural networks performs better with normalized data. We applied normalization to the newly created dataset.
As in general scenario we need to maintain the scaler factor and scale is fit on the training set for any neural network application. Two Scaler were created and fitted, one on the train data features and other on train labels. Both the scalers were used back in test_RNN.py later by calling the scaler.pkl and scaler_label.pkl files.

3. All design steps you went through in finding the best network in your report and how you chose your final design.
We ran 3 different RNN networks Simple_RNN, GRU and LSTM with similar NN structure of 4 layers, 50 units per layer, batch size = 32 and ran all the models for 150 epochs. We recorded the loss vs epoch for each model and found that. Loss in Simple_RNN was higher then other two methods.



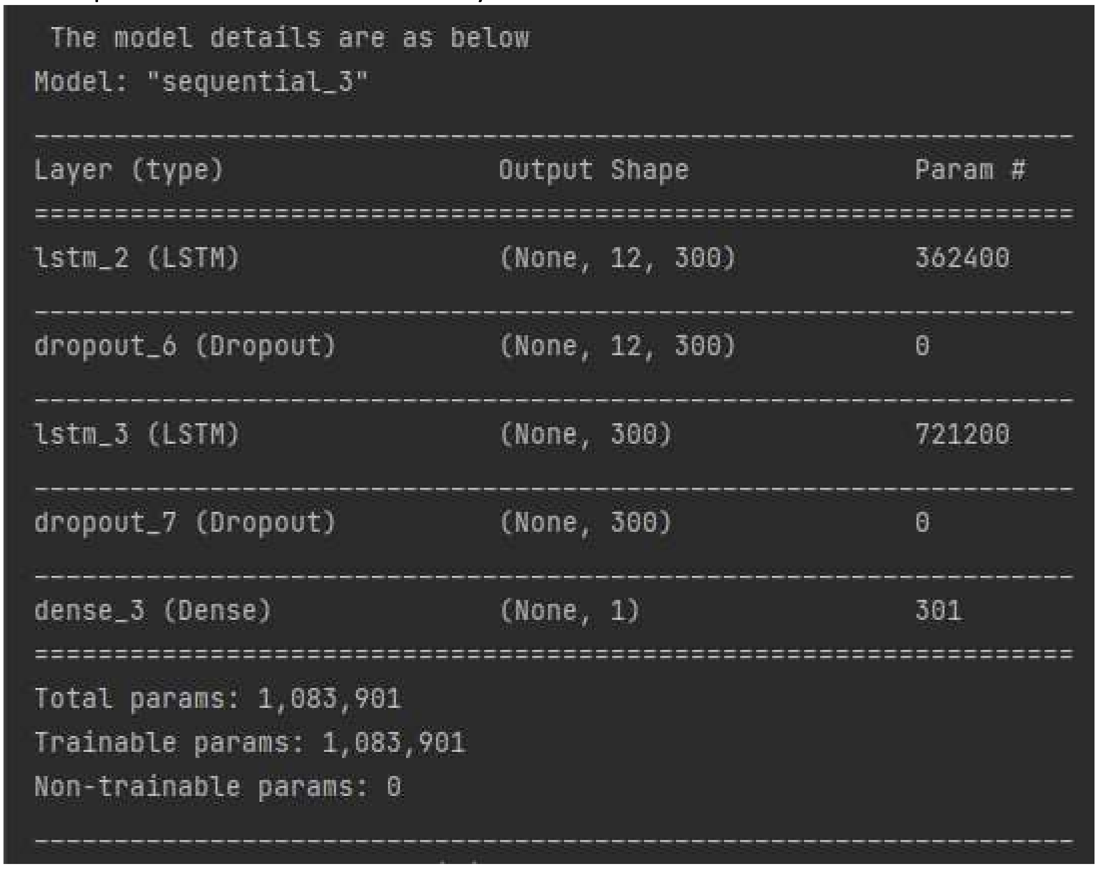
GRU vs LSTM we found that loss was almost similar in both the models but there are fluctuation in loss-per-epoch as seen in figure 1 (the generated plot after running train_RNN.py). LSTM on the other side gave a steady performance overall.

We choose LSTM as our final model. Now to fine tune it we computed Mean squared error to check the loss generated by the model. After many trial and error and referring to various theories of selecting number of layer and units in an LSTM model we concluded upon following architecture as described in next section.

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

The details of the architecture are as follows:
Units – 300 per layer
Number of Epochs – 500
Batch size – 32
Training algorithm – LSTM
Layers – 3, 1st layer have a return loops to 2nd layer
Loss function – Mean squared error
Optimizer – Adam

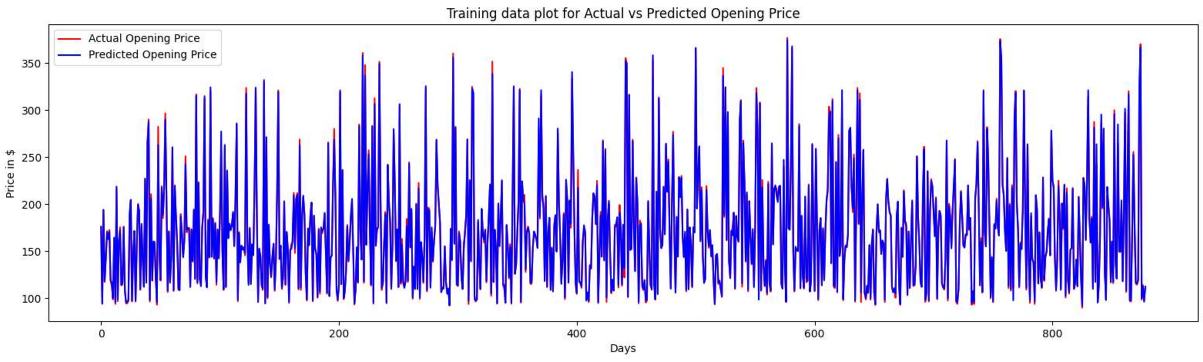
Also please see the summary of LSTM model on the terminal window.

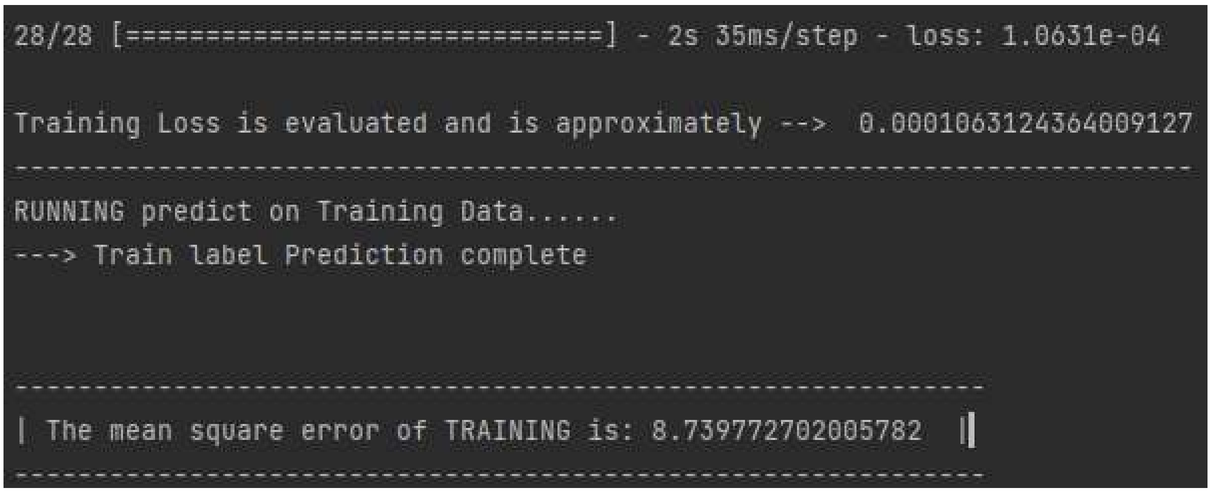


```
 The model details are as below
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 12, 300)           362400
_________________________________________________________________
dropout_6 (Dropout)          (None, 12, 300)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 300)               721200
_________________________________________________________________
dropout_7 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 301
=================================================================
Total params: 1,083,901
Trainable params: 1,083,901
Non-trainable params: 0
_________________________________________________________________
```

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

The output of training loop is computed and a graph of Original vs Predicted Opening price is shown in figure 2 generated from train_RNN.py

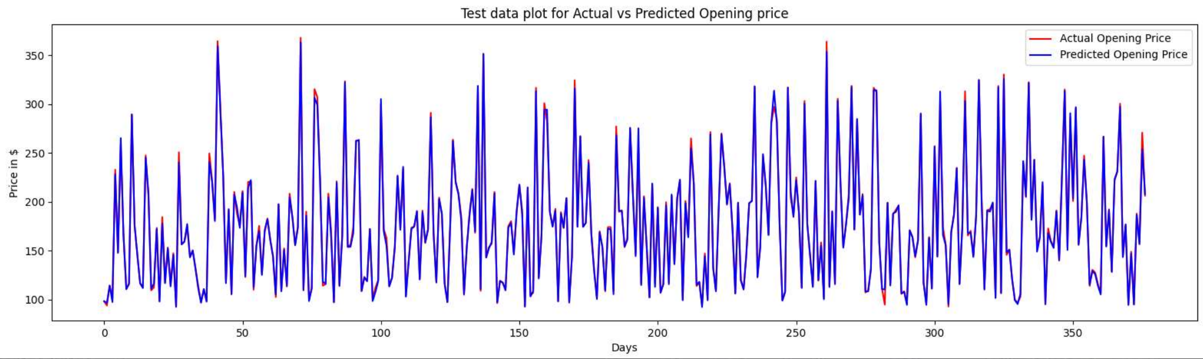


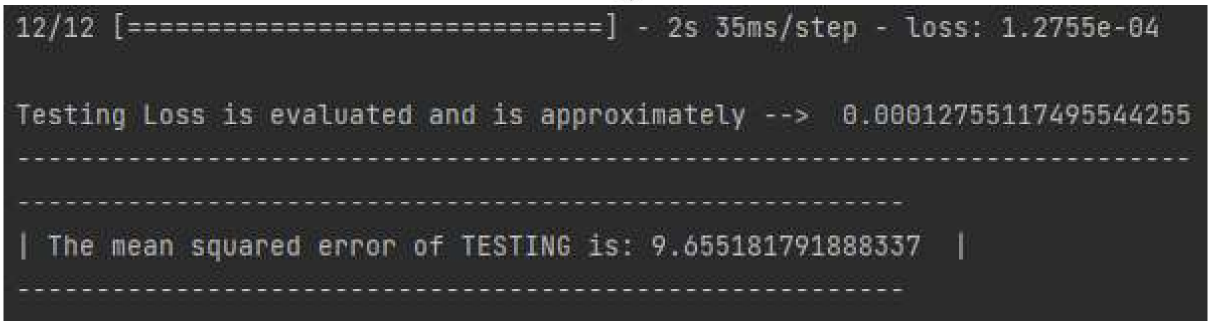


```
28/28 [==============================] - 2s 35ms/step - loss: 1.0631e-04

Training Loss is evaluated and is approximately -->  0.00010631243640091 27
--------------------------------------------------------------------------
RUNNING predict on Training Data......
---> Train label Prediction complete


------------------------------------------------------------
| The mean square error of TRAINING is: 8.739772702005782  |
------------------------------------------------------------
```

When we evaluate we see that the trained model incurred a loss of 1.0631e-04 as
Total overall mean squared error is: 8.734 between y_predict and actual y_train

6. Output from testing, including the final plot and your comment on it
From test_RNN.py figure 1 we can see the Actual vs Predicted Opening price.
Comment: As seen from the plot we could see the values closely follows the actual values.
And the loss value is very less which represents that we have a higher value of accuracy using the trained Group44_model.

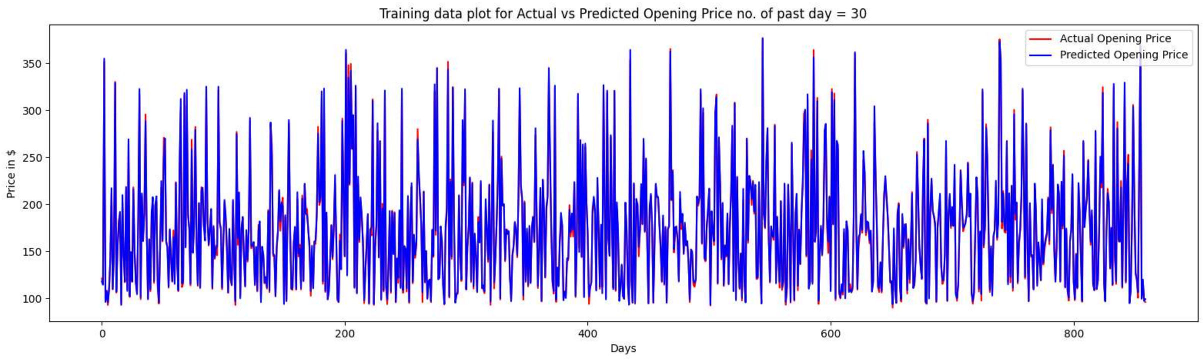




```
12/12 [==============================] - 2s 35ms/step - loss: 1.2755e-04

Testing Loss is evaluated and is approximately -->  0.00012755117495544255
--------------------------------------------------------------------------
------------------------------------------------------
| The mean squared error of TESTING is: 9.655181791888337  |
------------------------------------------------------
```

The mean squared error value for test dataset is = 9.655 between y_predict and actual y_test
The loss which is evaluated by running the model on test set is = 1.2755e-04

7. What would happen if you used more days for features (feel free to actually try it – but do not upload the datasets).
We used past 30 days data to compute the outputs. Following are the results as seen in figure 3 generated from train_RNN.py. In the actual plot figure you can see precisely by zooming in to different sections Overall the loss is higher than 3 days.


Training data plot for Actual vs Predicted Opening Price no. of past day = 30
Actual Opening Price
Predicted Opening Price
Price in $
Days
350
300
250
200
150
100
0
200
400
600
800