Report

- ## Prepare Dataset:

1] The provided csv file was read using ‘pandas’ library
2] 3 sets of data were created for:
- **Input Data:** which contained (Open, High, Low, Volume) parameters from last 3 days. Hence each input row now contained 12 input parameters, each.
- **Labels:** which contained the expected ‘Open’ values for the corresponding days.
- **Dates:** which contained the dates for the corresponding days. (Later used for plotting the graphs)

3] The 3 sets were divided into training and testing sets (70% and 30% respectively)
4] All 3 training sets were concatenated & saved in **‘./data/train_data_RNN.csv’**
5] All 3 testing sets were concatenated & saved in **‘./data/test_data_RNN.csv’**

- ## Pre-processing Steps:

1] The input values were scaled to a range of (0, 1).
- i.e.: each of the input values ware mapped to a corresponding value ranging from 0 to 1

2] The input was then reshaped to adhere to the following format for the RNN:
- **Format:** [Samples, Timesteps, Features]
- In our case: a] ‘Samples’ = Total number of records/inputs
  b] ‘Timesteps’ = Number of previous days (3) being considered
  c] ‘Features’ = Number of features (4) being used from each day

- ## How Final Design was chosen ?

1] First, different types of Recurrent Neural Networks were considered:
a] Simple RNN: found to be the fastest of the lot (when it comes to training)
b] GRU (Gated Recurrent Units): Results were similar to LSTM, but LSTM produced slightly better results.
c] LSTM (Long Short-Term Memory): Achieved minimum loss for Test Predictions, although was the slowest.

Implementation for first 2 has been commented in the code. One can easily try them out, if needed.

2] We tried a number of configurations – including single layer & multi-layered (2-5) – with varying units in the range of [32, 64, 128, 256]. Eventually we found 3 consecutive LSTM layers – each with 64, 128, and 256 units respectively, to provide very efficient results.

3] We decided to utilize **LSTM**, since it achieved best results in terms of loss.

- ## Final Network Architecture:

1] Final Design:
- **Input:** 3 timesteps – with 4 features each
- **LSTM-Layer-1:** with 64 Units, with default activations (‘relu’ – activation, ‘sigmoid’ – recurrent activation)
- **LSTM-Layer-2:** with 128 Units, with default activations
- **LSTM-Layer-3:** with 256 Units, with default activations
- **Dense-Layer:** With only 1 neuron and no explicit activation; acting as the final output layer

This is not a classification problem. Hence, we decided not to use any explicit activation at the output layer.
We had also tried adding Dropout layers, but they were no help in improving the results.

2] Number of Epochs = 250: Having tried different values, it was observed that the loss values were fluctuating very closely post epoch = 200. We took this as an indication of possible convergence & decided to restrict the epochs.

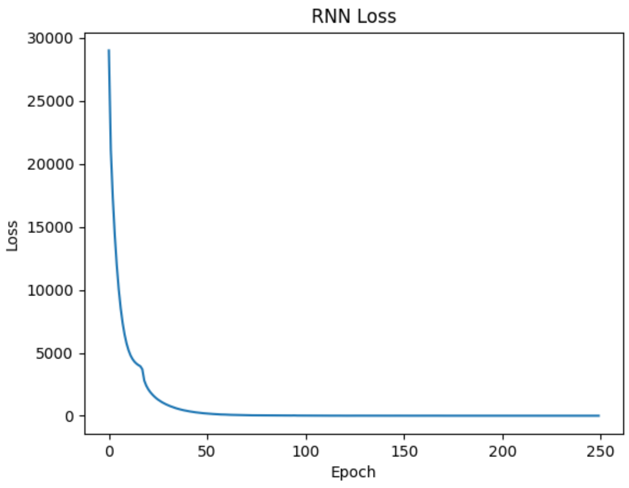
3] Loss Function = Mean Square Error: This was one of the most popular options available to accurately measure the loss; and it suited our needs, as the problem does not involve classification.

4] Batch Size = 16

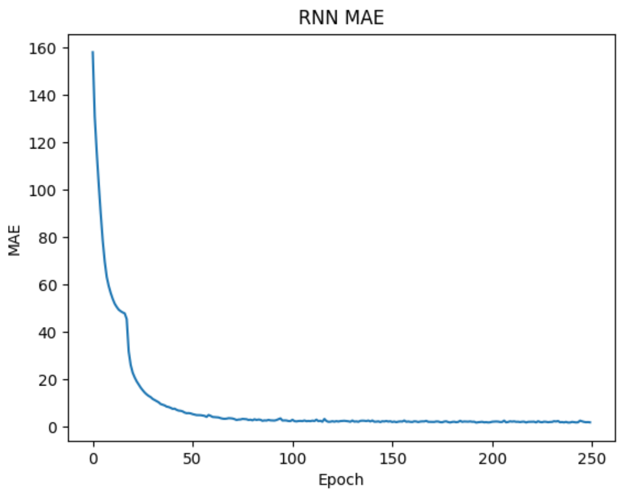
5] Optimizer = 'Adam', with default initial learning rate

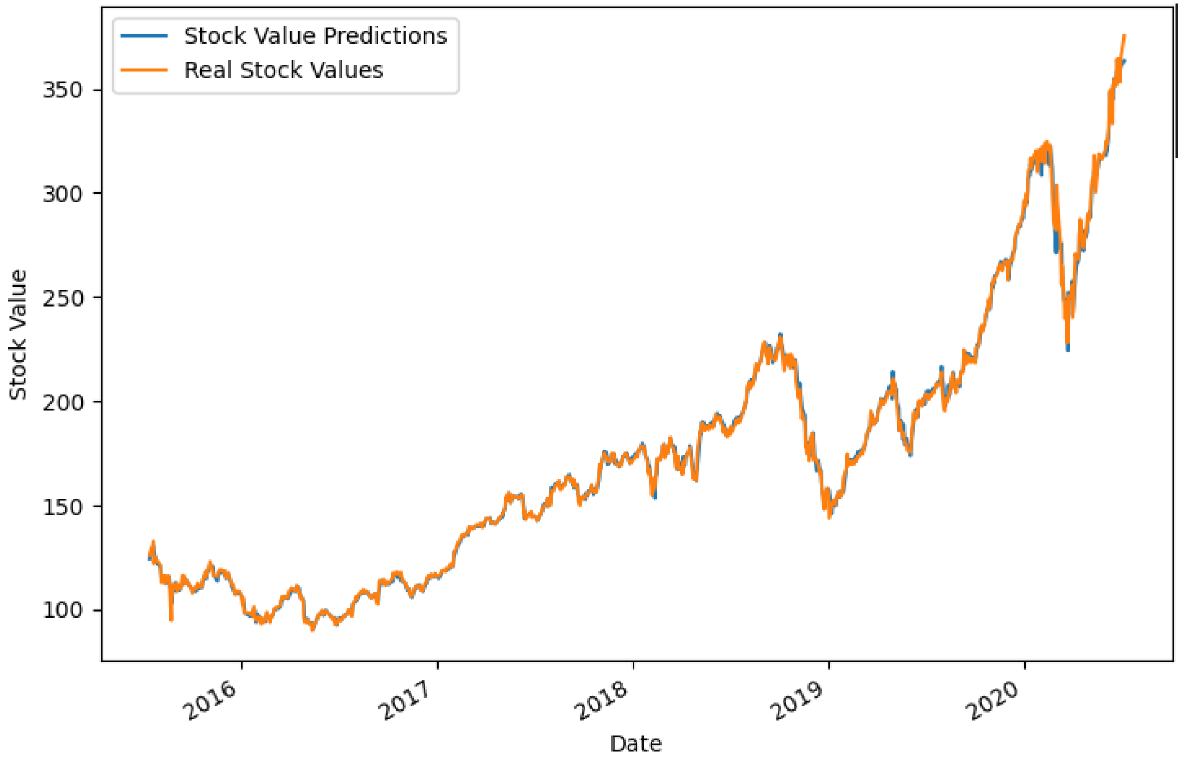
- **Output of Training Loop:**

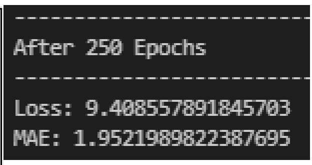
Epoch vs Loss



Epoch vs MAE







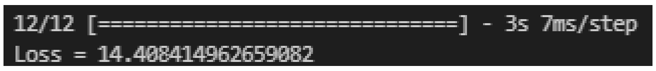
```
------------------------
After 250 Epochs
------------------------
Loss: 9.408557891845703
MAE: 1.9521989822387695
```

After training for 250 epochs, the latest **'Loss'** value (MSE - Mean square error) was ~9.4085, which is very less, specially compared to the initial loss of over 25000. And the MAE (Mean Absolute Error) which was used as a metric as well, was also down to ~1.95.

As we can see in the graph, the loss decreases drastically during the initial epochs and then improves in smaller chunks as it is nearing convergence; as is expected.

The **Date v/s Stock Value** graph shows how the network is able to predict for the training data (at the end of training), which is consistent with the low loss value and backs our decision to stop at epochs=250, as it appears to be converging well.

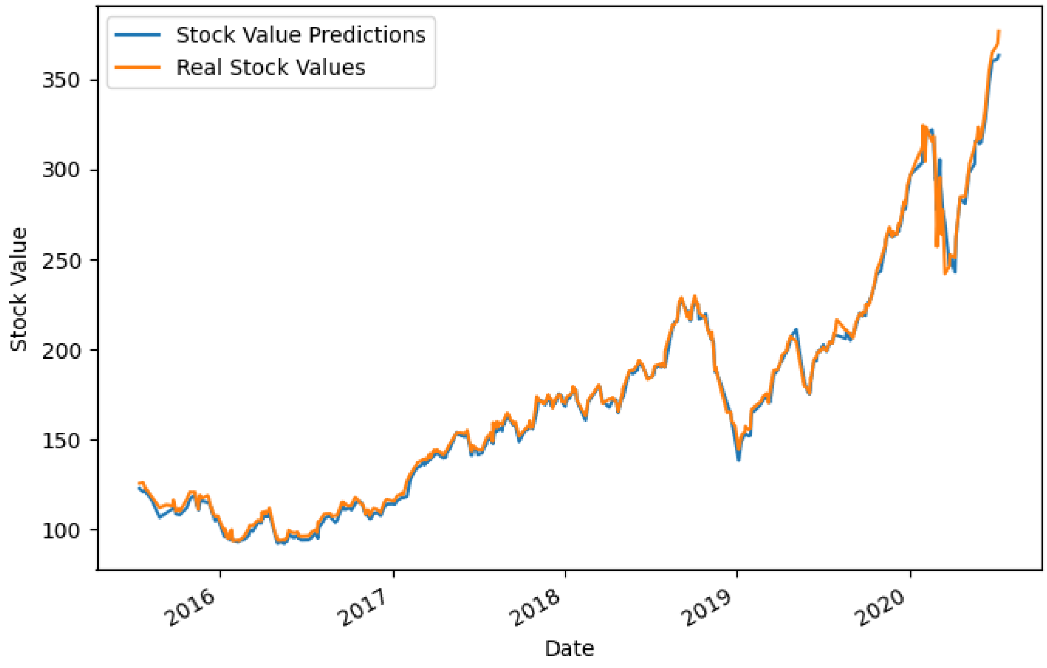
- ## Output of Testing:



```
12/12 [==============================] - 3s 7ms/step
Loss = 14.408414962659082
```

On testing the network with the remaining 30% input data, which the network was not exposed to; we were able to achieve good results with a total loss = ~ 14.40.

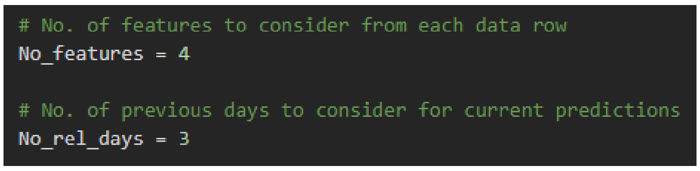
Below is the plot showing the difference (in values), between the predicted stocks and the actual stocks that were expected.



- As we can see, some stocks appear to have been predicted very well.
- We can also observe some other stocks which were not predicted as accurately, but still are very close to the actual values.
- The above graph is reflective of the low loss and appears to provide efficient results.

- ## Experiment with more days for features:

- We tried with 6 days instead of 3.
- And observed a higher loss value (~ 19) for the test set.
- One can easily try this option by simply changing the value of **'No_rel_days'** to 6 in both – training as well as testing files.



```
# No. of features to consider from each data row
No_features = 4

# No. of previous days to consider for current predictions
No_rel_days = 3
```

References:

1. https://www.tensorflow.org/guide/keras/rnn
2. https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
3. https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN
4. https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU