UDACITY ADVANCE MACHINE LEARNING NANODEGREE

STOCK PRICE PREDICTION

DEFINITION

---

1) **Project Overview**

All investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and make profitable investments and trades. A wealth of information is available in the form of historical stock prices and company performance data, suitable for machine learning algorithms to process.

Stock price prediction is a very vital research area throughout many past decades. Using statistical methods and stochastic analysis to make stock price prediction is the mainstream in last 20 years, while using machine learning techniques to predict stock price is becoming more and more prevalent in recent years.

In this project, I am making use of stock data(S&P 500) over last five years(1st Jan 2014 to 1st Jan 2019) of five trading companies that is, Google(GOOG), Apple(APPL), Amazon(AMZN), Facebook(FB) and Netflix(NFLX). Data is downloaded in csv from Yahoo Finance.

2) **Problem Statement**

This project focuses to build a stock price predictor that takes daily trading data over a certain date range (time-series data) as input, and outputs projected estimates for given query dates. Note that the inputs will contain multiple metrics, such as opening price (Open), highest price the stock traded at (High), how many stocks were traded (Volume) and closing price adjusted for stock splits and dividends (Adjusted Close), though we only need to predict the Adjusted Close price so this is the most prominent concern for us.

The challenge of this project is to accurately predict the future closing value of a given stock across a given period of time in the future. We will make use of Keras to create LSTM RNN models. Also we will utilize Pandas dataframe for performing significant operations over time-series stock data. The performance can be measured based on prediction vs actual price for the stock.

3) **Evaluation Metrics**

For this project we will focus on regression based evaluation metrics since it's a clear cut regression problem, so the important metrics for such scenario would be r-square and root-mean-squared-error. r-square can help us in determining how much variation in the dependent variable is explained by the variation in the independent variables. While, root-mean-squared-error can provide what is the average deviation of the prediction from the true value, and it can be compared with the mean of the true value to see whether the deviation is large or small.

We will also measure the performance of our LSTM model over benchmarking Linear Regression model.

ANALYSIS

---

1) **Data Exploration**

As mentioned already we will be dealing with five different stocks. Primarily I am using GOOG data to train for all of our model and will verify robustness later with same model applied to other trading companies stock data.

Data exploration phase consists of following steps:

i) Reading stock data which we have downloaded already and is stored in ./data/* folder then taking a glimpse of what the stock data contains actually. This contains last five year stock data of Google (from 1 Jan 2014 to 1 Jan 2019).

In [2]:
```
# Load stock data from .csv file
stocks = load_data('./data/GOOG.csv')

stocks.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2013-12-31	552.526367	556.878052	549.555664	556.734009	556.734009	2733400
1	2014-01-02	554.125916	555.263550	550.549194	552.963501	552.963501	3666400
2	2014-01-03	553.897461	554.856201	548.894958	548.929749	548.929749	3355000
3	2014-01-06	552.908875	555.814941	549.645081	555.049927	555.049927	3561600
4	2014-01-07	558.865112	566.162659	556.957520	565.750366	565.750366	5138400

ii) Transform this data to separate out our primary focus of features and labels.

```
In [3]: # Transform stock data to separate out feature set and label set
        stocks = transform_data(stocks)

        stocks.head()
```

Out[3]:

	Date	Relative_Date	Volume	Open	High	Low	Close	Adj Close
0	2013-12-31	0	2733400	552.526367	556.878052	549.555664	556.734009	556.734009
1	2014-01-02	1	3666400	554.125916	555.263550	550.549194	552.963501	552.963501
2	2014-01-03	2	3355000	553.897461	554.856201	548.894958	548.929749	548.929749
3	2014-01-06	3	3561600	552.908875	555.814941	549.645081	555.049927	555.049927
4	2014-01-07	4	5138400	558.865112	566.162659	556.957520	565.750366	565.750366

iii) Now we calculate standard statistics on our data. Our primary features are Open, High, Low and Volume while labels are Close or Adjusted Close, Adjusted Close being the prominent one.

```
In [4]: # Displays statistics over data insights.
        display_statistics(stocks)
```

```
Data Statistics...

Open --->
Min:  491.94281          Mean : 787.285428951    Std:  215.252833686     Max:  1271.0
High --->
Min:  493.261566         Mean : 793.849246497    Std:  217.640945524     Max:  1273.890015
Low --->
Min:  484.891632         Mean : 780.097597724    Std:  213.205546288     Max:  1249.02002
Close --->
Min:  489.854309         Mean : 787.114218611    Std:  215.435792582     Max:  1268.329956
Volume --->
Min:  7900       Mean : 1923273.1533     Std:  1120658.78407     Max:  11204900
```

Note: Date is not an important feature also since it is not contiguous due to holidays and all, it makes us to introduce Relative Date which makes our data to good on this contiguous abnormality.

2) **Exploratory Visualization**

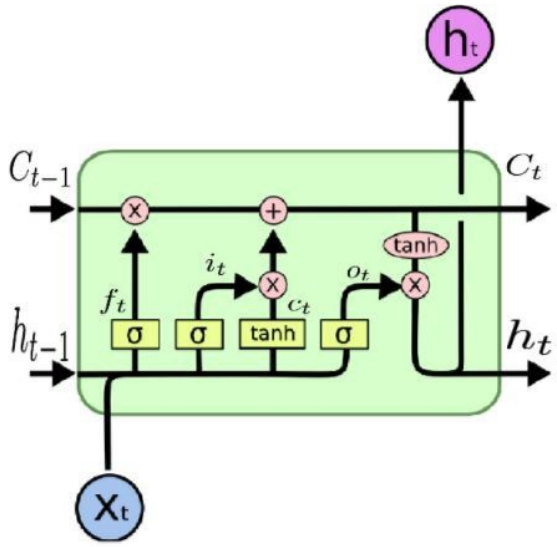
Our primary predicting price target for a stock is Adjusted Close, so lets visualize it in a plot to learn about its behaviour over five years of time. As the plot clearly signifies that how Adjusted Close value evolves over last five years from 1st Jan 2014 to 1st Jan 2019. `Adjusted Close` is more significant than `Close` cause it also take into account other stock metrics like dividend.

```
In [5]: # Visualize `Adj Close` which is our primary price target for stock data.
visualize(stocks, title='Google Trading', column='Adj Close',x_label='Trading Days',y_label='Adj Close Price')
```

3) **Algorithms and Techniques**

For this project, the time-series data can be effectively handled by LSTM-RNN model, hence, it's the best possible solution to use a LSTM Neural Net model capable of learning from such sequential data. We will make use of Keras over tensorflow backend to create LSTM RNN models. Also we will utilize Pandas dataframe for performing significant operations over time-series stock data. The performance can be measured based on prediction vs actual price for the stock.

LSTM-RNN(Long Short Term Memory - Recurrent Neural Net):
Its the state-of-the-art algorithm for most of the sequential data and time-series data. It consists of memory cell unit which is controlled with the help of gates. It has the mechanism of forget gate which helps it maintaining memory span longer though only the significant part of it same as humans have tendency to(that is remembering the past which is significant to us and forgetting other past moment). It has a following cell structure with sigmoid and tanh activation.



LSTM cell structure

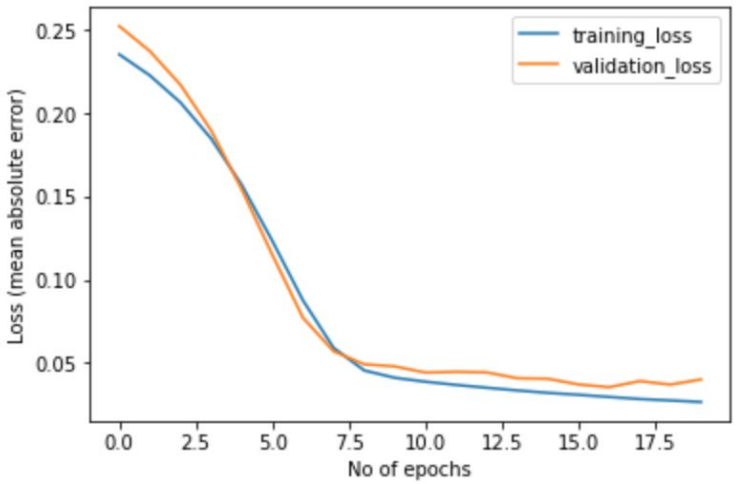
So due to LSTM tendency to maintain memory over longer span of time effectively it proves to be a better model to train on time-series data.

We have designed two LSTM networks as follows:

i) Simple LSTM model: It is a single layered LSTM model. We have used batch_size of 32 and number of epochs as 20 for training purpose. As the history loss plot clearly signifies how the loss function is converging. We have choose mean absolute error(mae) as loss metric and adam optimizer to achieve the following.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                4736
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 4,769
Trainable params: 4,769
Non-trainable params: 0
_________________________________________________________________
```

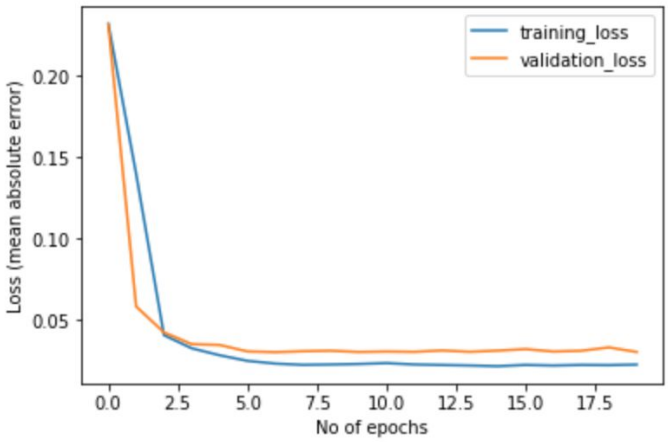
In [12]:
```
# Plot improve over loss over no of epochs
plot_loss(history)
```



ii) Improved LSTM model: It is multi-layered LSTM model with dropout to avoid overfitting. All other training parameters like batch_size, epochs, loss metric and optimizer are same for it as for simple LSTM model. Tried with increasing more layers but that does not give any advantage rather model was overfitting the data much in that case.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 1, 128)            68096
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 128)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 256)               394240
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 462,593
Trainable params: 462,593
Non-trainable params: 0
_________________________________________________________________
```

```
In [17]: # Plot improve over loss over no of epochs
         plot_loss(history)
```



4) **Benchmark**

The benchmark model for this project would be Linear Regression model since for a time series data like this it can be effectively used to compare other models in order to validate and check their performance with respect to it.

Our benchmark Linear Regression model summary is as follows:

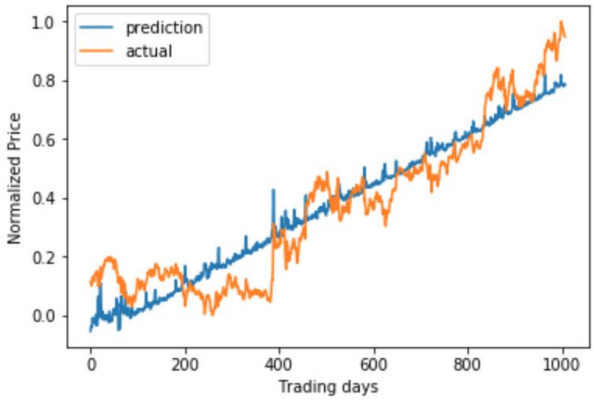
```
In [7]: # Building benchmark model and displaying model summary
        model_benchmark = build_benchmark_model(X_train, y_train)

        Model coefficient: [[ 0.86965828  0.1940259 ]]
        Model intercept: [-0.10091482]
```

The evaluation metrics and plots of the benchmark model are as follows:

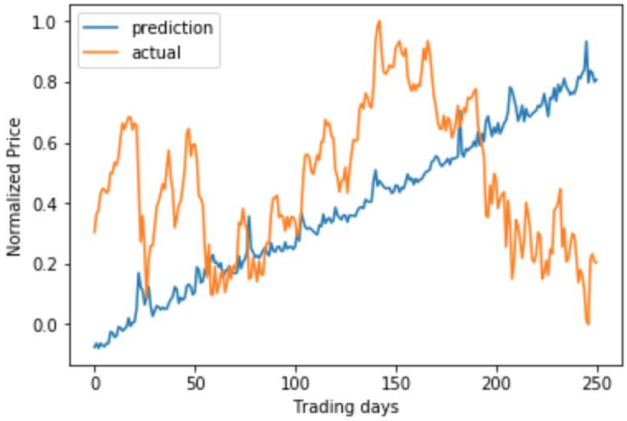
In [8]:
```
# Training prediction plot
predict_and_plot(model_benchmark, X_train, y_train)
```

```
r2_score: 0.8677523182737249
mean_squared_error: 0.009114244694205126
root_mean_squared_error: 0.09546855343098651
```



In [9]:
```
# Testing prediction plot
predict_and_plot(model_benchmark, X_test, y_test)
```

```
r2_score: -1.415025928569825
mean_squared_error: 0.12773974706658645
root_mean_squared_error: 0.3574069768017777
```



Methodology

---

1) **Data Preprocessing**

This step focuses on the preprocessing of stock data. It consists of following parts:

i) Normalization of feature set and label set. Since each of stock feature has different kind of value range, likewise we have seen when displaying statistics. So it requires to normalize them in some range say (0.0, 1.0) to perform any later computation. Sklearn `Min Max Scaling` performs a crucial role to do this.

ii) Now we can split the data into training phase and testing phase and do the necessary scaling again.

Data Preprocessing for Benchmark model

In [6]:
```
""" Preprocessing data."""

# Normalizing features('Relative_date' and 'Volume') and label('Adj Close')
X, y = get_normalized_feature_label(stocks, features_col=[0,1], labels_col=[6])

# Splitting dataset into training and testing.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Min max scaling of dataset to range of (0.0, 1.0)
X_train, y_train = min_max_scaling(X_train, y_train)
X_test, y_test = min_max_scaling(X_test, y_test)

# Displays size for Training and Testing dataset
print("No of training feature points: {}".format(X_train.shape))
print("No of training label points: {}".format(y_train.shape))
print("No of testing feature points: {}".format(X_test.shape))
print("No of testing label points: {}".format(y_test.shape))
```

```
No of training feature points: (1008, 2)
No of training label points: (1008, 1)
No of testing feature points: (251, 2)
No of testing label points: (251, 1)
```

Data Preprocessing for LSTM model

In [10]:
```
# Normalizing features('Volume','Open','High','Low') and label('Adj Close')
X, y = get_normalized_feature_label(stocks , features_col=[1,2,3,4], labels_col=[6])

# Splitting into training, validation and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1)

X_train, y_train = min_max_scaling(X_train, y_train)
X_val, y_val = min_max_scaling(X_val, y_val)
X_test, y_test = min_max_scaling(X_test, y_test)

# Reshape feature set in order to feed into LSTM network.
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_val = np.reshape(X_val, (X_val.shape[0], 1, X_val.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

2) **Implementation**

This includes our module implementation for benchmark model, simple LSTM model and improved LSTM model.

i) Benchmark model: We have chosen Linear Regression as our benchmark model since it clearly depicts that all other model should be atleast better than this.

```
"""Benchmark model."""

def build_benchmark_model(X, y):
    """ Building Linear Regression model as the benchmark model."""
    model = LinearRegression()
    model = model.fit(X, y)

    """Displays model summary."""
    print("Model coefficient: {}".format(model.coef_))
    print("Model intercept: {}".format(model.intercept_))

    return model
```

ii) Simple LSTM model: This is a single layered LSTM network which perform quite well enough when we compare to our benchmark model. It consists of single LSTM layer and then dense layer as the output layer. We are using mean absolute error(mae) as the loss metric and Adam as the optimizer.

```
def build_simple_model(X_train, y_train, X_val, y_val):
    """ Build single-layered LSTM RNN model."""
    model = Sequential()

    model.add(LSTM(32, return_sequences=False, input_shape = (X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='mae', optimizer='adam', metrics=['mean_squared_error'])

    history = model.fit(X_train,
                        y_train,
                        epochs = 20,
                        validation_data=(X_val,y_val),
                        batch_size = 32)

    """Displays model summary."""
    model.summary()

    return model, history
```

Some of the problems/complications faced while implementing these models are:
i) Rescaling data points in order to make LSTM network more effective over time series data.
ii) Reshaping feature set in order to fed into LSTM network for training, validation as well as testing purpose.
iii) Tried many variants of multi-layered LSTM model but choosing more than 2 layers is making model overfit the data. So this was also a nice challenging part.
iv) Stucking moments in some of the pandas/numpy operations.

3) **Refinement**

Though simple LSTM model performs significantly well enough with comparison to benchmark model as well. But still we can explore on improvements to this single layered model with 32 LSTM cells only. Hence we will try to introduce more number of layers with more number of LSTM cell network and what we observed that on increasing more than 3 layers only training time increases while model wholly try to overfit the data. Eventually it's better to go with 2-layered LSTM network with more no of LSTM cell as that in simple LSTM model. Also we introduce LSTM network with dropout to avoid overfitting to the time series data. In this way we got our improved LSTM model. This performs significantly well enough in predicting other company stock price as well.

```
def build_improved_model(X_train, y_train, X_val, y_val):
    """ Build multi-layered LSTM RNN improved model using appropriate dropouts."""
    model = Sequential()
    # First layer LSTM
    model.add(LSTM(128, return_sequences=True, input_shape = (X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(0.2))
    # Second layer LSTM
    model.add(LSTM(256, return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='mae', optimizer='adam', metrics=['mean_squared_error'])

    history = model.fit(X_train,
                        y_train,
                        epochs = 20,
                        validation_data=(X_val,y_val),
                        batch_size = 32)
```

Results

---

1) **Model Evaluation and Validation**

We have used r2_score and root_mean_squared_error to evaluate model correctness. Also we have plot prediction vs actual price to visualize the same in a time-series curve.

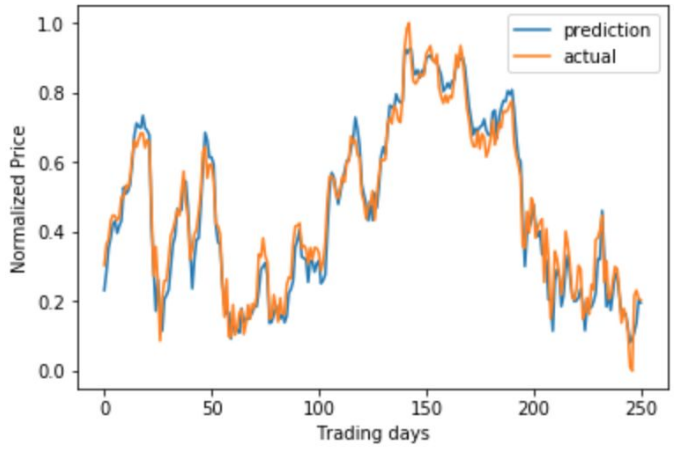
i) Simple LSTM Evaluation:
Characteristics of model includes:

- Single layered LSTM network with 32 LSTM cell unit
- batch_size = 32 and no of epochs = 20
- Loss metric used is mean absolute error and Adam optimizer is used.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                4736
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 4,769
Trainable params: 4,769
Non-trainable params: 0
_________________________________________________________________
```

```
In [15]: # Testing prediction plot
         predict_and_plot(model_simple, X_test, y_test)
```

```
r2_score: 0.9508455680487824
mean_squared_error: 0.0025999616113308844
root_mean_squared_error: 0.05098981870266734
```



ii) Improved LSTM Evaluation:

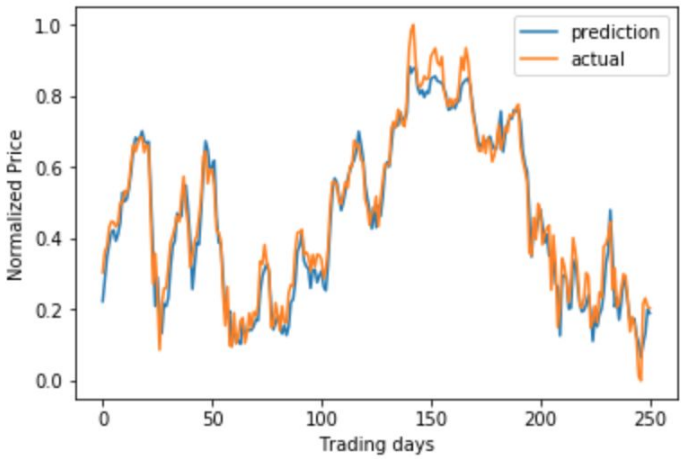
Characteristics of model includes:

- Multi layered (2-layered) LSTM network with each layer having 128, 256 LSTM cell unit respectively.
- batch_size = 32 and no of epochs = 20
- Loss metric used is mean absolute error and Adam optimizer is used.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 1, 128)            68096
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 128)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 256)               394240
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 462,593
Trainable params: 462,593
Non-trainable params: 0
_________________________________________________________________
```

In [20]:
```
# Testing prediction plot
predict_and_plot(model_improved, X_test, y_test)
```

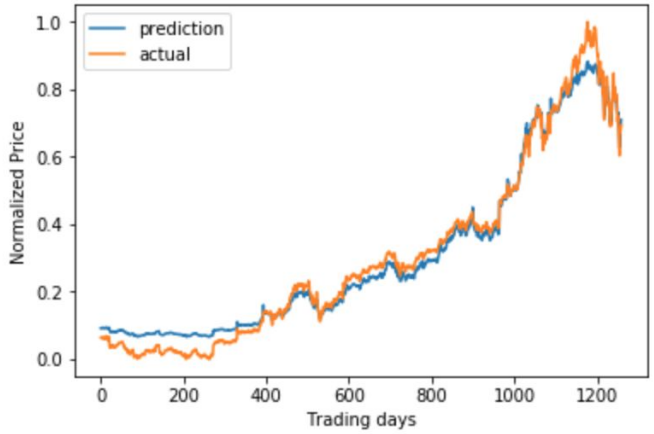
```
r2_score: 0.9538591171436863
mean_squared_error: 0.002440563737129292
root_mean_squared_error: 0.04940206207365531
```



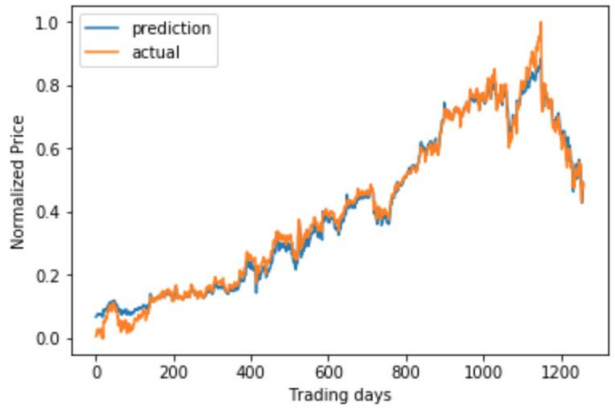
Mean absolute error as a loss function proves to be a better choice with respect characteristics of the stock data. Also our improved model has r2_score of **0.95** for testing data which depicts its robustness. In order to further verify and evaluate the robustness of model we checked the same model trained on Google stock data to use and perform prediction on other company stock data like Amazon(AMZN), Facebook(FB), etc..

Following figures depicts the same and this we will discuss more in Justification section.

```
AMZN Trading
r2_score: 0.9831487380471252
mean_squared_error: 0.001246867053278604
root_mean_squared_error: 0.03531100470502934
```




```
FB Trading
r2_score: 0.9914342767987853
mean_squared_error: 0.0005442728422325693
root_mean_squared_error: 0.023329655853281877
```

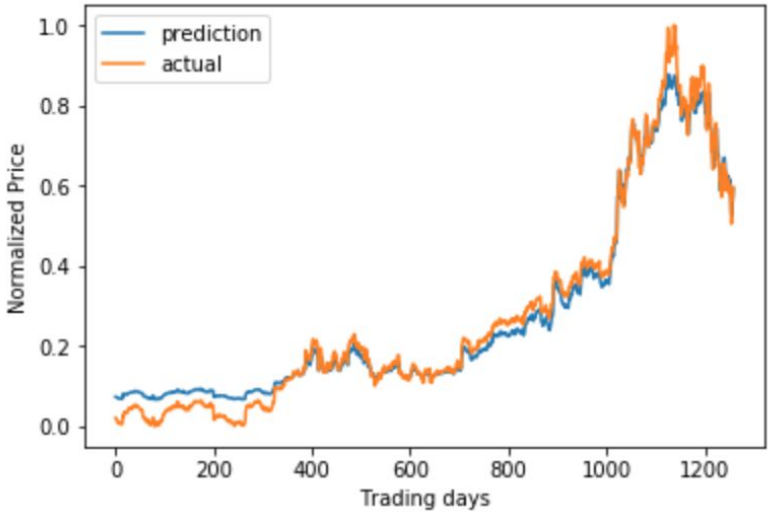


2) **Justification**

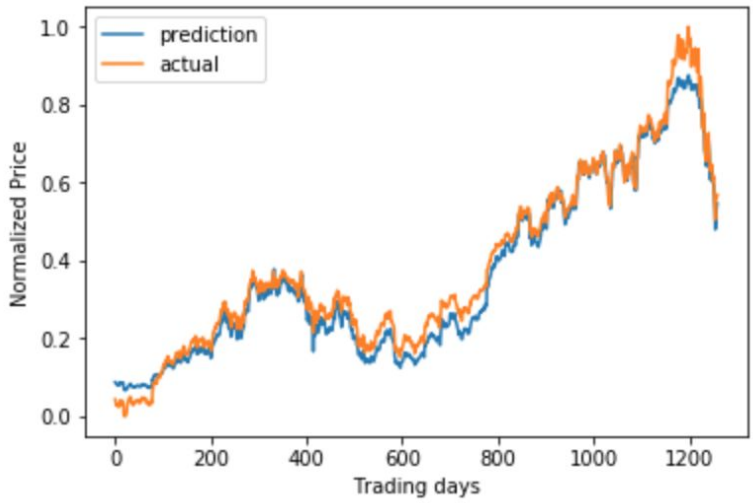
Our improved LSTM model proved to be significantly well enough than what benchmark model we have used. Evaluation metrics clearly depicts the same. The r2_score for our benchmark model was **0.86** for training and **-1.41** for testing while our improved LSTM model has not only achieves r2_score of **0.98** for training but also performs enough well with testing data having r2_score of **0.95** for testing. So this justifies that our improved model is significantly well and robust.

In order to justify correctness and robustness of our final improved model we have tried the same model behaviour on other company stock trading data like Apple(APPL), Netflix(NFLX), Amazon(AMZN), etc.. We have visualize the same in plots and it clearly proves how the model is generalized in stock prediction with feature set as (Open, High, Low, Volume) as the final ones.

```
NFLX Trading
r2_score: 0.984126291770032
mean_squared_error: 0.0010567463119250291
root_mean_squared_error: 0.03250763467133573
```



```
AAPL Trading
r2_score: 0.9765285714395611
mean_squared_error: 0.001259258739464848
root_mean_squared_error: 0.035486035837563594
```

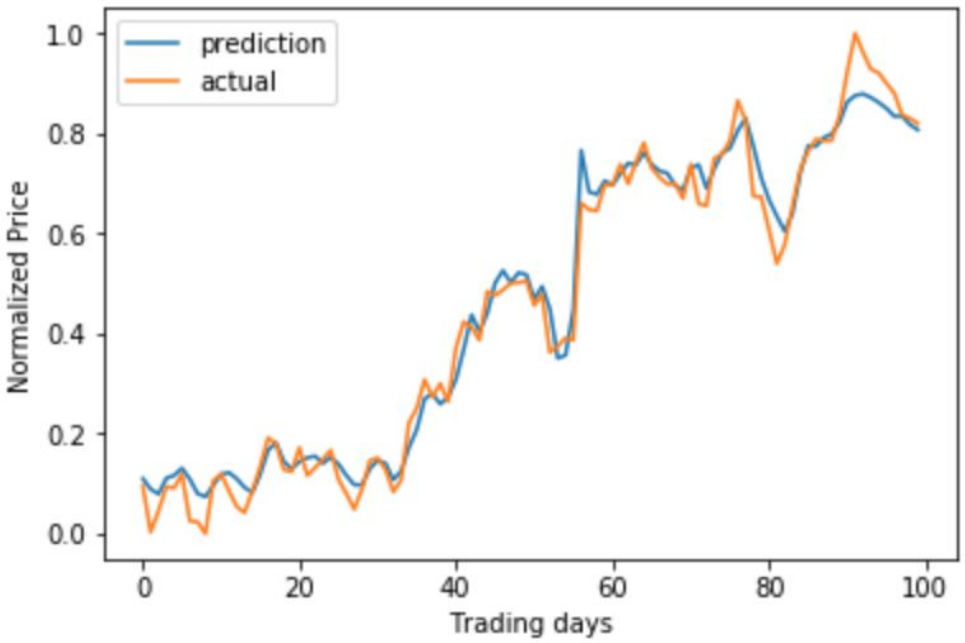


Conclusion

---

1) Free-Form Visualization

Following is a `**Google Trading**` visualization from one of our model prediction vs actual normalized stock price. We can clearly identify that model is not so sensitive to sudden spike moments which may occur due to many external metrics and reasons. This we will discuss in more detail in improvement section later on. So hence this is the most challenging part of how we can deal with such spikes in data.



2) Reflection

A quick glimpse of what we aimed, how we started and eventually ended upon is as:

1) Our problem domain deals with Stock price prediction using historical time series stock data.
2) We have use iPython notebook and python important modules/libraries like pandas, numpy, sklearn(for preprocessing and benchmarking), keras for LSTM model and matplotlib for visualization of figures.
3) Data collection: Download last five years data from Yahoo Finance for the important S&P 500 stocks (Google, Apple, Facebook, Netflix and Amazon).
4) Data Preprocessing: We have use pandas to load data, normalize it and split it into training/testing data points. We have used Min Max Scaling for normalization.
5) Build benchmark model: We have use Linear Regression for benchmarking results.
6) Build simple LSTM model: Single layered LSTM model. This performs significantly well with respect to Benchmark model.
7) Build improved LSTM model: Multi-layered LSTM complex network with dropout to handle over overfitting. This is very robust and generalized for other company stock data as well.
8) Visualize Results: Use matplotlib extensively to visualize more and more curve to identify, validate and evaluate our model behaviour on time series stock data.
9) Check the robustness of the improved model.

3) Improvement

Improvement is the term which never stops you to explore more and more. Same happens here initially we have simple LSTM model which performs significantly well but still we explore more and more and observed that 2-layered LSTM model quite suited to our stock data learning. But still now as all the prediction curves depicts that our model is not sensitive to sudden high/low spike which happens due to other stock affecting metrics like P/E ratio (Price to Earning) and also all other national level concerns like Election, etc.. Hence if we can explore more to find the relation between other features and this sudden spike we can definitely fed that feature to our network to make it more robust and accurate.

REFERENCES

---

1) Murtaza Roondiwala, Harshal Patel, Shraddha Varma : `Predicting Stock Prices Using LSTM`, International Journal of Science and Research (IJSR). (2017)

2) Bao, Wei, Jun Yue, and Yulei Rao : `A deep learning framework for financial time series using stacked autoencoders and long-short term memory`. (2017)