Machine Learning Engineer Nanodegree

Build a stock predictor model

Ankit Maurya
June 16th, 2018

I. Definition

Project Overview

Investment firms, hedge funds and even individuals have been using financial models to better understand market behaviour 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.

In this project we will predict a timeseries composed of stock prices using machine learning . Big investment firms are using more and more of machine learning algorithms to predict the stock prices . The financial advisors are getting replaced with machine learning algorithms. Here we will predict the stock price of a large company listed on NYSE stock exchange , given it’s historical performance.

We will use Long short-term memory models which are a special case of RNNs, to predict the stock prices here . Recent research shows the LSTM are the most useful variant of RNNs.

We will be using quandl python api to obtain data set for GE closing stock prices.

Problem Statement

The challenge of this project is to accurately predict the closing value of a given stock across a given period of time in the future. I will predicting the Adj.Close of a stock listed on NYSE stock exchange. The adjusted closing price is a stock closing price after it has been amended to include any dividend, split, or merge.

I will be using LSTM for the prediction . The goal is to first build a model using linear regression and then use LSTM to improve the prediction.

Metrics

The model will be predicted evaluated the MSE and MAE between the actual and predicted close values for the stock. Also we will compare the delta between the performance of the benchmark model (linear regression model) and the LSTM model.

II. Analysis

Data Exploration

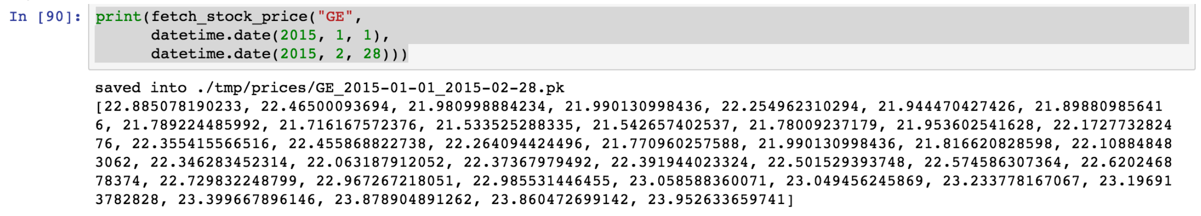
A snippet of the data set looks like this . We will be using quandl python api to get the data. We will be using data from 2015 to 2016.

$ curl "https://www.quandl.com/api/v3/datasets/WIKI/GE/data.csv"
Date,Open,High,Low,Close,Volume,Ex-Dividend,Split Ratio,Adj. Open,Adj. High,Adj. Low,Adj. Close,Adj. Volume
2016-12-30,31.63,31.795,31.51,31.6,25452804.0,0.0,1.0,30.812025142156,30.972758121874,30.695128429635,30.782800964026,25452804.0
2016-12-29,31.74,31.8769,31.7,31.71,16092077.0,0.0,1.0,30.919180461968,31.05254012817,30.880214891127,30.889956283838,16092077.0
2016-12-28,31.84,31.97,31.67,31.7,18885134.0,0.0,1.0,31.016594389069,31.143232494301,30.850990712997,30.880214891127,18885134.0
2016-12-27,31.89,32.045,31.85,31.9,15655819.0,0.0,1.0,31.06530135262,31.216292939627,31.026335781779,31.07504274533,15655819.0
2016-12-23,31.87,31.94,31.77,31.88,14988300.0,0.0,1.0,31.0458185672,31.114008316171,30.948404640098,31.05555995991,14988300.0
2016-12-22,31.91,31.95,31.78,31.82,24193137.0,0.24,1.0,31.08478413804,31.123749708881,30.958146032809,30.997111603649,24193137.0

The format is a CSV, and each line contains the date, the opening price, the highest and the lowest of the day, the closing, the adjusted, and some volumes. The lines are sorted from the most recent to the least. The column we're interested in is the Adj. Close, that is, the closing price after adjustments.

The adjusted closing price is a stock closing price after it has been amended to include any dividend, split, or merge.

I have created an api fetch_stock_price which returns the stock price for the requested symbol, ordered from the from_date to the to_date.



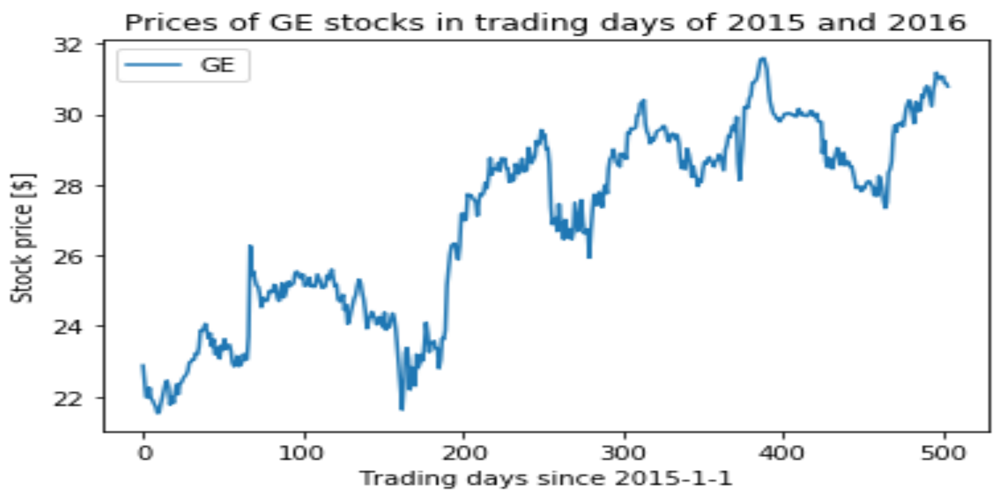
```
In [90]: print(fetch_stock_price("GE",
               datetime.date(2015, 1, 1),
               datetime.date(2015, 2, 28)))

saved into ./tmp/prices/GE_2015-01-01_2015-02-28.pk
[22.885078190233, 22.46500093694, 21.980998884234, 21.990130998436, 22.254962310294, 21.944470427426, 21.89880985641
6, 21.789224485992, 21.716167572376, 21.533525288335, 21.542657402537, 21.78009237179, 21.953602541628, 22.1727732824
76, 22.355415566516, 22.455868822738, 22.264094424496, 21.770960257588, 21.990130998436, 21.816620828598, 22.10884848
3062, 22.346283452314, 22.063187912052, 22.37367979492, 22.391944023324, 22.501529393748, 22.574586307364, 22.6202468
78374, 22.729832248799, 22.967267218051, 22.985531446455, 23.058588360071, 23.049456245869, 23.233778167067, 23.19691
3782828, 23.399667896146, 23.878904891262, 23.860472699142, 23.952633659741]
```

I did not found any abnormality in data, no missing values no negative values etc.

Exploratory Visualization

To visualize the data I have used matplotlib library . I have plotted the Adj. Close of GE stocks from 2015-1-1 to 2016-12-31.



Algorithms and Techniques

The goal of this project was to study time-series data and explore as many options as possible to accurately predict the Stock Price. Through my research i came to know RNN works on sequences: they accept multidimensional signals as input, and they produce a multidimensional output signal.

Inside, an RNN has multiple stages; each stage is connected to its input/output and to the output of the previous stage. Thanks to this configuration, each output is not just a function of the input of its own stage, but depends also on the output of the previous stage (which, again, is a function of its input and the previous output). This configuration ensures that each input influences all the following outputs, or, from the other side, an output is a function of all the previous and current stages inputs.

LSTM models are an evolution of RNNs: with long RNNs, the training phase may lead to very tiny or huge gradients back-propagated throughout the network, which leads the weights to zero or to infinity: that's a problem usually expressed as a vanishing/exploding gradient. To mitigate this problem, LSTMs have two outputs for each stage: one is the actual output of the model, and the other one, named memory, is the internal state of the stage.

Thus, LSTM were a good choice for time series analysis.

The following parameters can be tuned for LSTM:

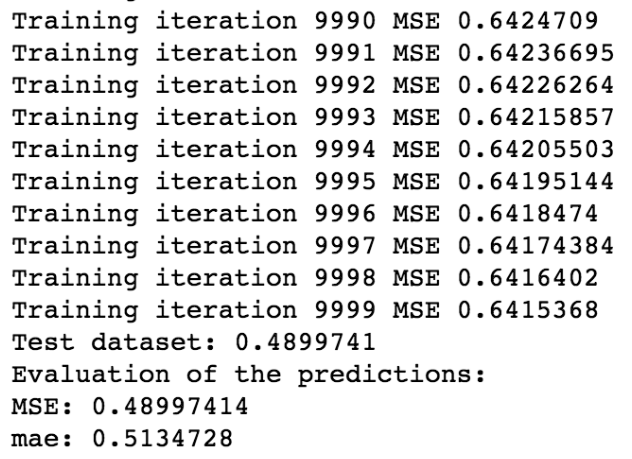
- learning_rate – Tried learning rates like 0.001, 0.1 ,0.01, 0.07 etc and finally settled on 0.01 as it gave the best performance.
- n_epochs – How many times to run through the training process . Tried 1000,5000 ,10000 etc. Finally settled on 5000 as it gave the best performance.
- n_embeddings - Number of neurons in the network, used 64 .
- feature_size – Number of features in a particular batch . Used 20 as there are approximately 20 working days in a month , hence we will have 20 closing stock values.

Benchmark

A linear regression model was built to predict future stock prices , which can be used a benchmark for our future model. The model will be predicted evaluated the MSE and MAE between the actual and predicted close values for the stock.

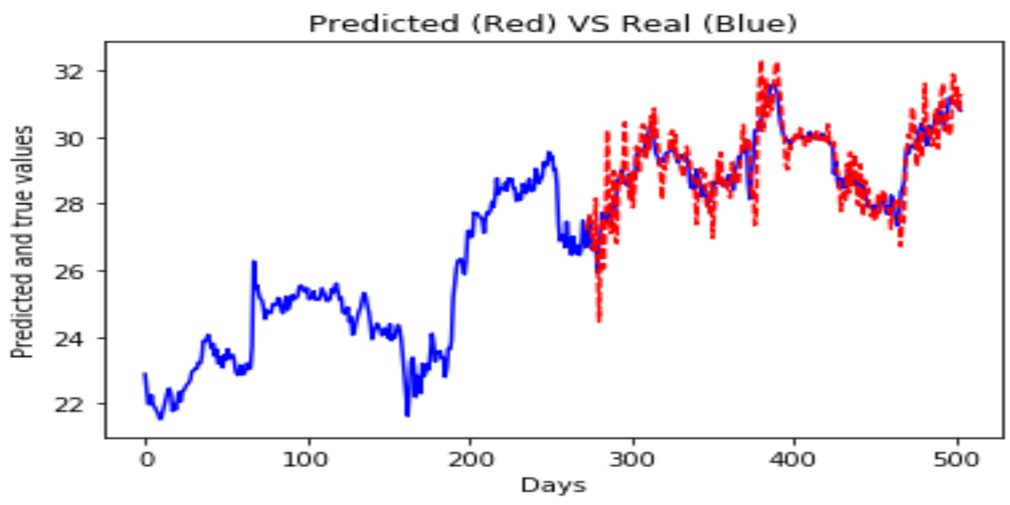
feat_dimension = 20
train_size = 252
test_size = 252 - feat_dimension

Same data split will be used for the LSTM model.
10000 epochs were used .



```
Training iteration 9990 MSE 0.6424709
Training iteration 9991 MSE 0.64236695
Training iteration 9992 MSE 0.64226264
Training iteration 9993 MSE 0.64215857
Training iteration 9994 MSE 0.64205503
Training iteration 9995 MSE 0.64195144
Training iteration 9996 MSE 0.6418474
Training iteration 9997 MSE 0.64174384
Training iteration 9998 MSE 0.6416402
Training iteration 9999 MSE 0.6415368
Test dataset: 0.4899741
Evaluation of the predictions:
MSE: 0.48997414
mae: 0.5134728
```

The figure shows that the benchmark model has a MSE of 0.48 and MAE of 0.51 on the test data.



III. Methodology

Data Preprocessing

Financial data can be expensive and hard to extract, that's why in this experiment we use the Python library quandl to obtain such information. The library has been chosen since it's easy to use, cheap (XX free queries per day), and great for this exercise, where we want to predict only the closing price of the stock.

Quandl is an API, and the Python library is a wrapper over the APIs.
The format retuned is a CSV, and each line contains the date, the opening price, the highest and the lowest of the day, the closing, the adjusted, and some volumes. The lines are sorted from the most recent to the least. The column we're interested in is the Adj. Close, that is, the closing price after adjustments.

I have built a function which is able to cache calls and specify an initial and final timestamp to get the historical data beyond the symbol.

The returned object of the function fetch_stock_price is a mono-dimensional array, containing the stock price for the requested symbol, ordered from the from_date to the to_date. Caching is done within the function, that is, if there's a cache miss, then the quandl API is called. The date_obj_to_str function is just a helper function, to convert datetime.date to the correct string format needed for the API.

The code is in jupyter notebook . Attaching a snapshot of the function .

```python
def fetch_stock_price(symbol,
                      from_date,
                      to_date,
                      cache_path="./tmp/prices/"):
    assert(from_date <= to_date)

    filename = "{}_{}_{}.pk".format(symbol, str(from_date), str(to_date))
    price_filepath = os.path.join(cache_path, filename)

    try:
        prices = load_pickle(price_filepath)
        print("loaded from", price_filepath)

    except IOError:
        historic = quandl.get("WIKI/" + symbol,
                              start_date=date_obj_to_str(from_date),
                              end_date=date_obj_to_str(to_date))

        prices = historic["Adj. Close"].tolist()
        save_pickle(prices, price_filepath)
        print("saved into", price_filepath)

    return prices
```

I have also created a format_dataset function. The code is provided in jupyter notebook.

```python
def format_dataset(values, temporal_features):
    feat_splits = [values[i:i + temporal_features] for i in range(len(values) - temporal_features)]
    feats = np.vstack(feat_splits)
    labels = np.array(values[temporal_features:])
    return feats, labels
```

Given the timeseries, and the feature size, the function creates a sliding window which sweeps the timeseries, producing features and labels (that is, the value following the end of the sliding window, at each iteration). Finally, all the observations are piled up vertically, as well as the labels. The outcome is an observation with a defined number of columns, and a label vector.

The train and test data can thus be found using the following implementation

```python
stock_values = fetch_stock_price(symbol, datetime.date(2015, 1, 1), datetime.date(2016, 12, 31))
minibatch_cos_X, minibatch_cos_y = format_dataset(stock_values, feat_dimension)

train_X = minibatch_cos_X[:train_size, :].astype(np.float32)
train_y = minibatch_cos_y[:train_size].reshape((-1, 1)).astype(np.float32)
test_X = minibatch_cos_X[train_size:, :].astype(np.float32)
test_y = minibatch_cos_y[train_size:].reshape((-1, 1)).astype(np.float32)
```

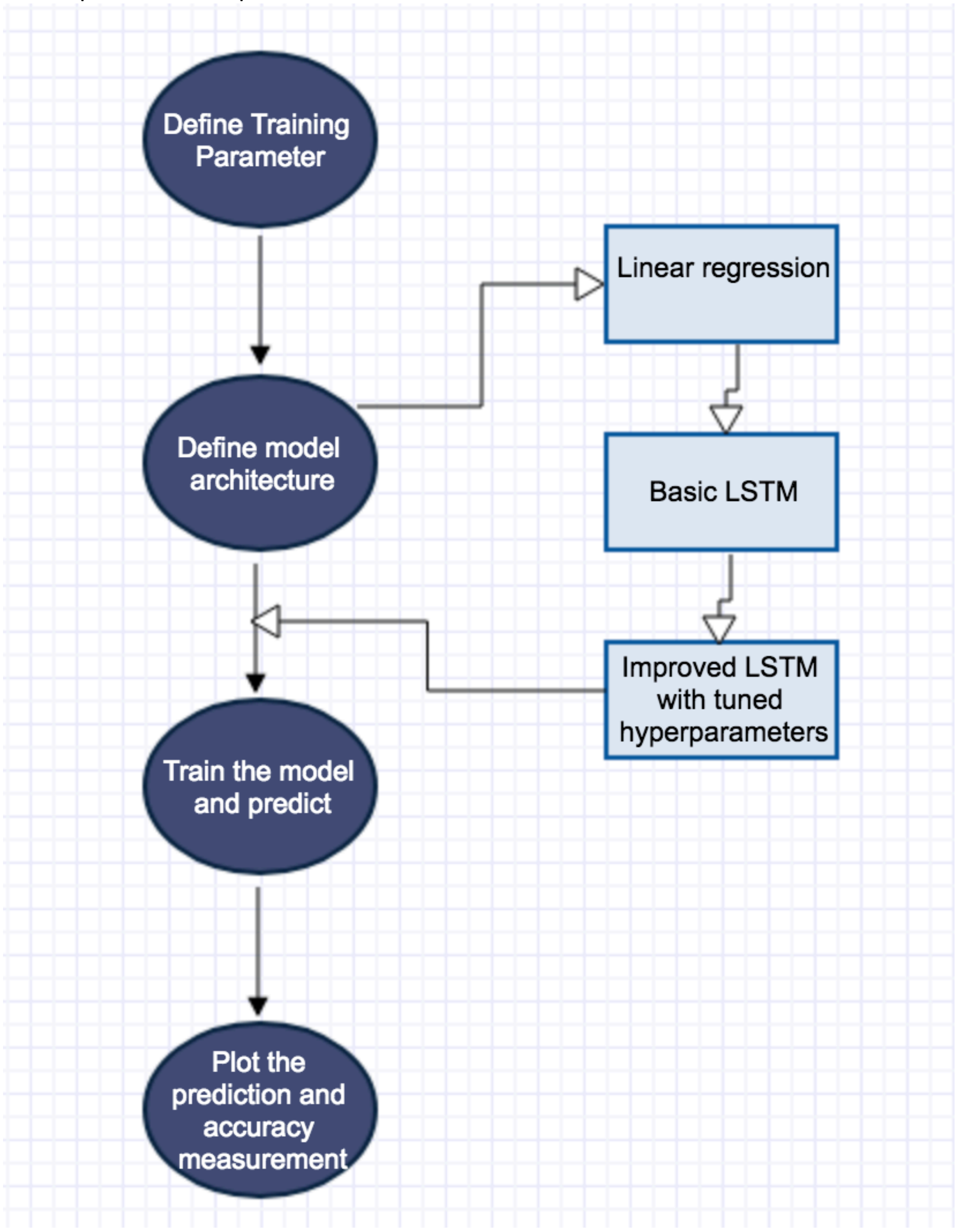
We have kept the following parameters.
symbol = "GE"
feat_dimension = 20
train_size = 252
test_size = 252 - feat_dimension

- We are predicting the stock price of GE hence, symbol = "GE".
- We have a feature size of 20 , as there are approximately 20 working days in a month and hence 20 stock prices.
- We are using one year training data from 1st Jan 2015 to 31st Dec 2015. There are 252 working days in 2015 .

- We will be predicting the stock prices of next 1 year i.e 2016.

Implementation

The implementation process can be summarized as below


Define Training Parameter
Define model architecture
Linear regression
Basic LSTM
Improved LSTM with tuned hyperparameters
Train the model and predict
Plot the prediction and accuracy measurement

I have described all the steps in the notebook with the proper code .

Some code insights :

- Linear regression model :
    1. Split the dataset into testing and training .
    2. I opted for the most classic way of implementing it, that is, the multiplication between the observation matrix with a weights array plus the bias. What's coming out (and the returned value of this function) is the array containing the predictions for all the observations contained in x:

```
def regression_ANN(x, weights, biases):
    return tf.add(biases, tf.matmul(x, weights))
```

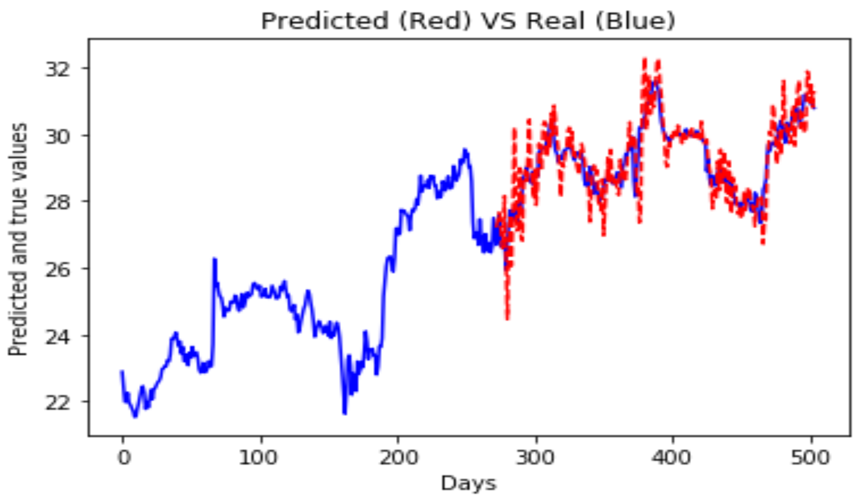
3.Train the model using for 1000 epochs

```
# For each epoch, the whole training set is feeded into the tensorflow graph
for i in range(n_epochs):
    train_cost, _ = sess.run([cost, train_op], feed_dict={X_tf: train_X, y_tf: train_y})
    print("Training iteration", i, "MSE", train_cost)

# After the training, let's check the performance on the test set
test_cost, y_pr = sess.run([cost, y_pred], feed_dict={X_tf: test_X, y_tf: test_y})
print("Test dataset:", test_cost)
```

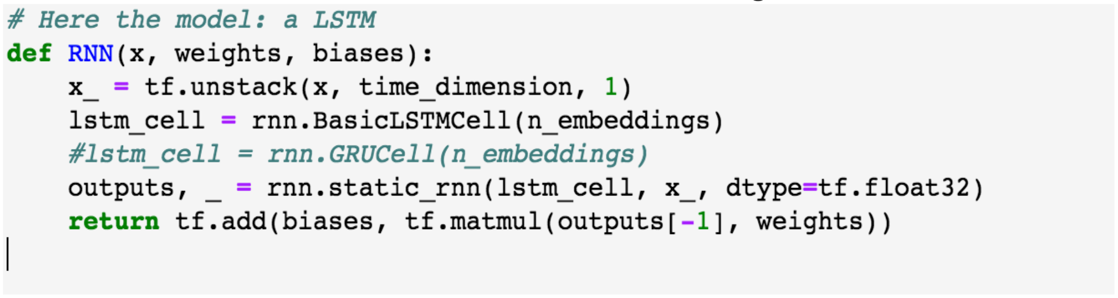
4.Plot the prediction and accuracy.

```
Training iteration 9990 MSE 0.6424709
Training iteration 9991 MSE 0.64236695
Training iteration 9992 MSE 0.64226264
Training iteration 9993 MSE 0.64215857
Training iteration 9994 MSE 0.64205503
Training iteration 9995 MSE 0.64195144
Training iteration 9996 MSE 0.6418474
Training iteration 9997 MSE 0.64174384
Training iteration 9998 MSE 0.6416402
Training iteration 9999 MSE 0.6415368
Test dataset: 0.4899741
Evaluation of the predictions:
MSE: 0.48997414
mae: 0.5134728
```



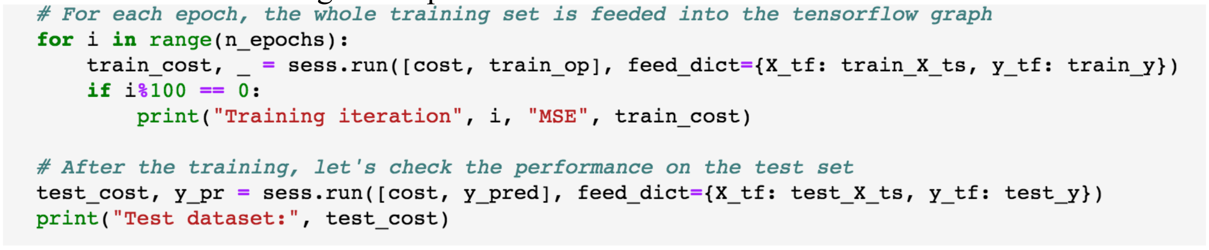
- Improved LSTM model
    1. Split the data into training and testing .

2. We will use an LSTM with a variable number of embeddings.



```
# Here the model: a LSTM
def RNN(x, weights, biases):
    x_ = tf.unstack(x, time_dimension, 1)
    lstm_cell = rnn.BasicLSTMCell(n_embeddings)
    #lstm_cell = rnn.GRUCell(n_embeddings)
    outputs, _ = rnn.static_rnn(lstm_cell, x_, dtype=tf.float32)
    return tf.add(biases, tf.matmul(outputs[-1], weights))
```

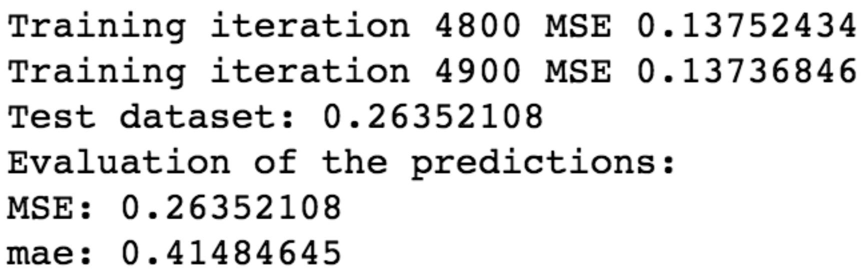
3. Train the model using 5000 epochs.



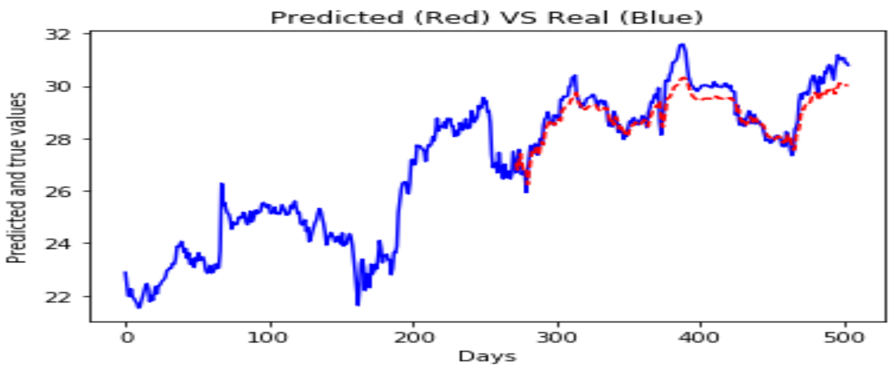
```
# For each epoch, the whole training set is feeded into the tensorflow graph
for i in range(n_epochs):
    train_cost, _ = sess.run([cost, train_op], feed_dict={X_tf: train_X_ts, y_tf: train_y})
    if i%100 == 0:
        print("Training iteration", i, "MSE", train_cost)

# After the training, let's check the performance on the test set
test_cost, y_pr = sess.run([cost, y_pred], feed_dict={X_tf: test_X_ts, y_tf: test_y})
print("Test dataset:", test_cost)
```

4.Plot the prediction and accuracy.



```
Training iteration 4800 MSE 0.13752434
Training iteration 4900 MSE 0.13736846
Test dataset: 0.26352108
Evaluation of the predictions:
MSE: 0.26352108
mae: 0.41484645
```



Refinement

For improving the performance I worked upon tuning the hyperparameters for the LSTM model.

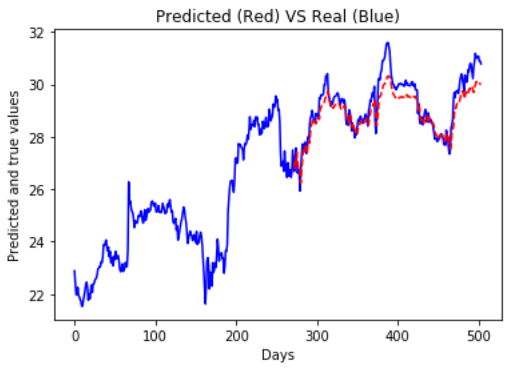
To improve I have done the following :

- I tried the learning rate of 0.01,0.1 and 0.001 , 0.01 gave the best result.
- I tried epochs of 1000,5000 and 10000 . 5000 gave the best result
- I tried n_embeddings from 64, 128 to 256 , 256 gave the best result .

With :

```
learning_rate = 0.01
n_epochs = 5000
n_embeddings = 256
```

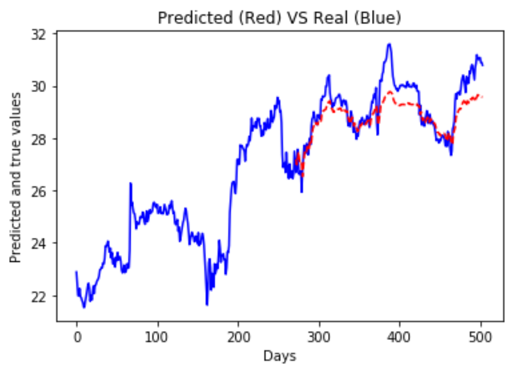
```
Evaluation of the predictions:
MSE: 0.26352108
mae: 0.41484645
```



With :

```
learning_rate = 0.01
n_epochs = 1000
n_embeddings = 256
```

```
Test dataset: 0.47434026
Evaluation of the predictions:
MSE: 0.47434032
mae: 0.5500248
```



Clearly we improved the MSE from 0.47 to 0.26 after tuning the parameters.

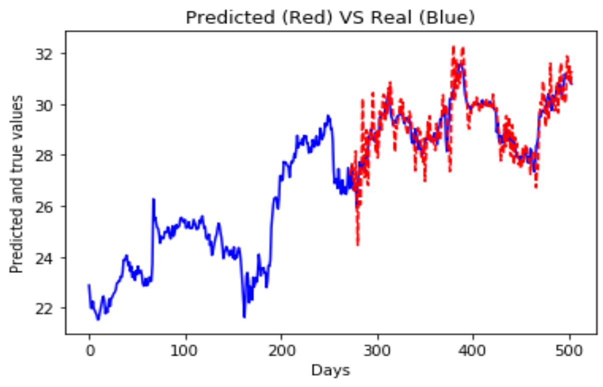
IV. Results

Model Evaluation and Validation

With each model I have refined and fine tuned my paraments and improved the MSE significantly:

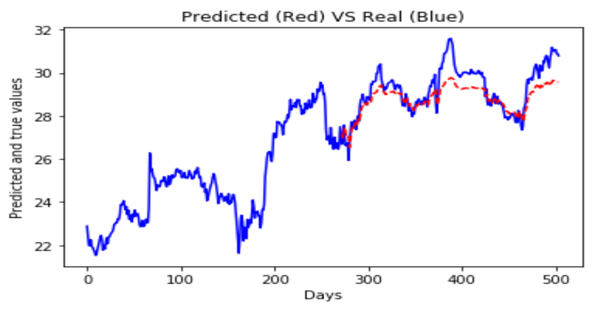
- Model 1 : Simple linear regression

```
MSE: 0.48997414 (on test data)
MAE: 0.513 (on test data)
```



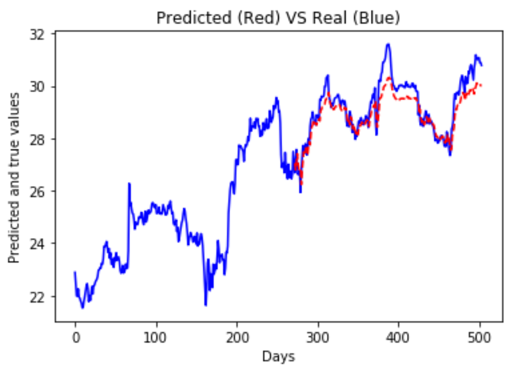
- Model 2 : LSTM with initial parameters

```
MSE: 0.47434032 (on test data)
mae: 0.5500248
```



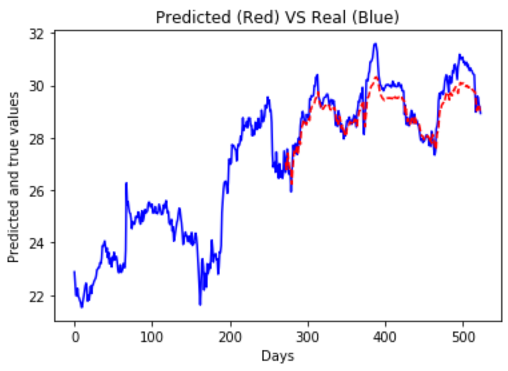
- Model 3: LSTM with tuned parameters

```
MSE: 0.26352108 (on test data)
mae: 0.41484645
```



Robustness check

For checking the robustness I increase the test data from 12 months to 13 months, still got an `MSE: 0.27168366` which is almost equal to the final model with test data of 12 months.



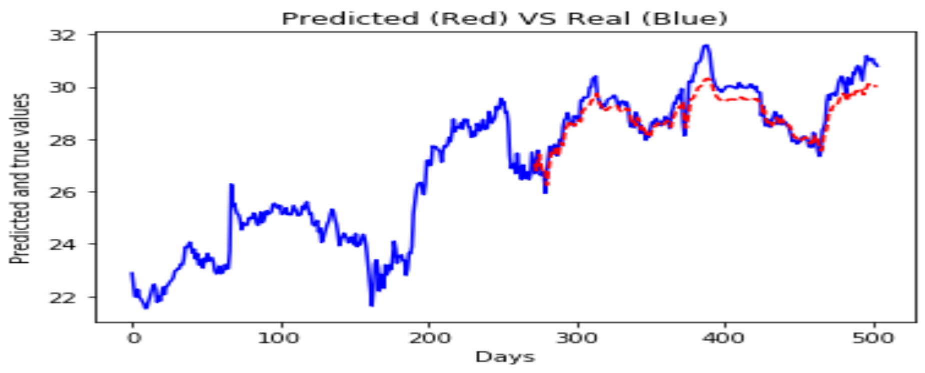
Justification

Comparing the benchmark model to the improved LSTM model, the final MSE ranges from 0.489(Linear regression model) to 0.263(improved LSTM model) which is an improvement of 46%.
The clearly shows that the final model has surpassed the benchmark model.
The MAE is 0.414 , they are dollars! With a learner, we would have predicted on average 0.4 a dollar closer to the real price the day after.

V. Conclusion

Free-Form Visualization

I have already discussed all the important features of the datasets and their visualisation in the above sections. But to conclude my report I would choose my final model visualization, which is improved version of LSTM by fine tuning parameters. I was able to predict a year of stocks prices with a MSE of 0.26 and MAE of 0.41.



Reflection

The steps undertaken in the project were :

- How to collect the historical stock price information

    1. We used Python library quandl to obtain the stock data

- Set up the infrastructure

    1. We used a iPython Notebook for the code
    2. Install the required libraries(like quandl,numpy,matlibplot,tensorflow etc.)
    3. We used one year data for training i.e 2015 year and the next year for testing i.e 2016 year.

- Use regression to predict the future prices of a stock

    1. We built a benchmark model using linear .regression
    2. After tuning the parameters we got a MSE of 0.48 and MAE of 0.51

- Develop LSTM model

    1. We developed the LSTM model and got a MSE of 0.47 and MAE of 0.55
    2. After tuning the hyperparameters we got a MSE of 0.26 and MAE of 0.414.

- Visualize the performance

    1. We plotted the observed results vs the predicted results.
    2. We analysed the differences between different models based on MSE and MAE.

I started this project with the hope to learn a completely new algorithm, i.e, Long-Short Term Memory and also to explore a real time series data sets.

The most difficult part of this project was to tune the hyper parameters for the LSTM model . Each time I changed the parameters the model building used to take a lot of time(in minutes), hence it was the most difficult part to get the correct hyperparameters.

Improvement

I can think of the following things which can be improved in this project:

- Evaluate another algorithm like GRUCell instead of LSTM. GRUCell is a recent RNN algorithm and has shown some promises. It would be good to compare the results of both these algorithms
- Train the model on other stock prices and see how well it can predict the stock prices.
- Try to reduce the MSE and MAE even more by tuning parameters/changing algorithms.