Stock Price Prediction with RNN

Information Technology

Submitted By

Name: **Kunal Anand**
Reg. No.: **169108078**

Under the Guidance of Dr. Ashish Kumar Dadhich

Manipal University Jaipur
(Jaipur-Ajmer Express Highway, Dehmi Kalan,
Near GVK Toll Plaza, Jaipur, Rajasthan 30300)

CERTIFICATE

Date: 27th April 2019

This is to certify that the project titled **STOCK PRICE PREDICTION** is a record of the bonafide work done by **KUNAL ANAND** (169108078) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech) in **(Information Technology)** of Manipal University Jaipur, during the academic year 2018-19.

Dr Ashish Kumar Dadhich

Project Guide, Dept of Computer Science & Engineering

Manipal University Jaipur

Dr Pankaj Vyas

HOD, Dept of Information Technology

Manipal University Jaipur

ABSTRACT

Predicting securities market costs may be a complicated task that historically involves intensive human-computer interaction. Due to the related nature of stock costs, typical batch process strategies cannot be used with efficiency for stock market analysis. we tend to propose an on-line learning rule that utilizes a kind of repeated neural network (RNN) referred to as Long Short Term Memory (LSTM), wherever the weights are adjusted for individual information points mistreatment random gradient descent. This can offer additional correct results once compared to existing stock worth prediction algorithms. The network is trained and evaluated for accuracy with varied sizes of information, and the results are tabulated. A comparison with respect to accuracy is then performed against a man-made Neural Network.

LIST OF TABLES

Table No	Table Title	Page No
1.	Training Set	13
2.	Test Set	14

LIST OF FIGURES

Figure No	Figure Title	Page No
1.	RNN Model	6
2.	LSTM Model	7
3.	Sigmoid Function	9
4.	ReLu Function	9
5.	Proposed Model	11
6.	Project Workflow	12
7.	Predicted & Actual Plot	15
8.	Predicted & Actual Plot Zoomed in	16

CONTENTS

Title	Page No.
1. Abstract	1
2. List of Tables List of Figures	4
3. Background Overview	6
4. Methodology	11
5. Implementation & Analysis	15
6. Future Work & Conclusion	18
7. References	19

BACKGROUND OVERVIEW

Traditional approaches to stock market analysis and stock price prediction include fundamental analysis, which looks at a stock's past performance and the general credibility of the company itself, and statistical analysis, which is solely concerned with number crunching and identifying patterns in stock price variation. The latter is commonly achieved with the help of Genetic Algorithms (GA) or Artificial Neural Networks (ANN's), but these fail to capture correlation between stock prices in the form of long-term temporal dependencies. Another major issue with using simple ANNs for stock prediction is the phenomenon of exploding / vanishing gradient [4], where the weights of a large network either become too large or too small (respectively), drastically slowing their convergence to the optimal value. This is typically caused by two factors: weights are initialized randomly, and the weights closer to the end of the network also tend to change a lot more than those at the beginning.

An alternative approach to stock market analysis is to reduce the dimensionality of the input data [2] and apply feature selection algorithms to shortlist a core set of features (such as GDP, oil price, inflation rate, etc.) that have the greatest impact on stock prices or currency exchange rates across markets [10]. However, this method does not consider longterm trading strategies as it fails to take the entire history of trends into account; furthermore, there is no provision for outlier detection.

Proposed System

We propose an online learning algorithm for predicting the end-of-day price of a given stock (see Figure 2) with the help of Long Short Term Memory (LSTM), a type of Recurrent Neural Network (RNN).

About RNN

Recurrent Neural Network (RNN) was designed, like any other Neural Network, to function like a specific part of the human brain. When looking at our brain's cerebrum, we can divide it into the Temporal, Parietal, Occipital and Frontal lobe. Out of these, the Frontal lobe is the part which deals with short-term memory and remembers what happened in the immediate present and use it for decision making in the near future. It is this Frontal lobe that the RNN tries to replicate.

About LSTM

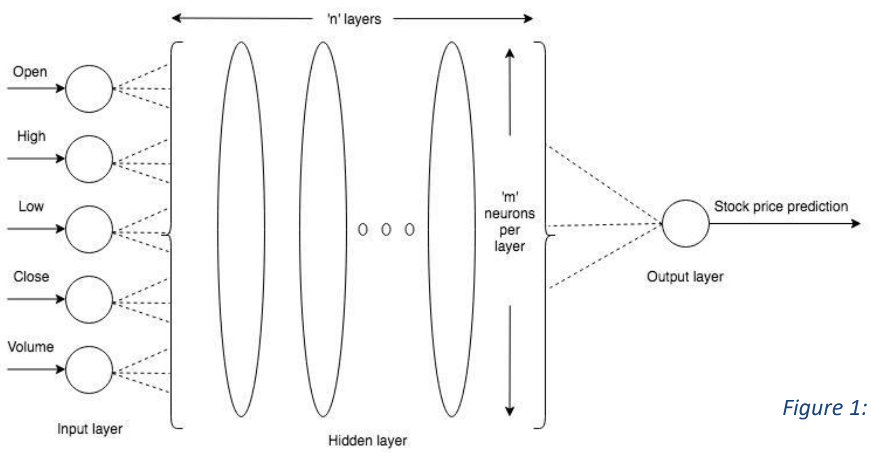


Figure 1: RNN Model

LSTM's are a special subset of RNN's that can capture context-specific temporal dependencies for long periods of time. Each LSTM neuron is a memory cell that can store other information i.e., it maintains its own cell state. While neurons in normal RNN's merely take in their previous hidden state and the current input to output a new hidden state, an LSTM neuron also takes in its old cell state and outputs its new cell state.

An LSTM memory cell, as depicted in figure, has the following three components, or gates:
1**. Forget gate**: the forget gate decides when specific portions of the cell state are to be replaced with more recent information. It outputs values close to 1 for parts of the cell state that should be retained, and zero for values that should be neglected.
2. **Input gate** : based on the input (i.e., previous output o(t-1), input x(t), and previous cell state c(t-1)), this section of the network learns the conditions under which any information should be stored (or updated) in the cell state
3**. Output gate**: depending on the input and cell state, this portion decides what information is propagated forward (i.e., output o(t) and cell state c(t)) to the next node in the network.

Thus, LSTM networks are ideal for exploring how variation in one stock's price can affect the prices of several other stocks over a long period of time. They can also decide (in a dynamic fashion) for how long information about specific past trends in stock price movement needs to be retained in order to more accurately predict future trends in the variation of stock prices.

<u>Advantages of LSTM</u>

The main advantage of an LSTM is its ability to learn context specific temporal dependence. Each LSTM unit remembers information for either a long or a short period of time (hence the name) without explicitly using an activation function within the recurrent components.

An important fact to note is that any cell state is multiplied only by the output of the forget gate, which varies between 0 and 1. That is, the forget gate in an LSTM cell is responsible for both the weights and the activation function of the cell state. Therefore, information from a previous cell state can pass through a cell unchanged instead of increasing or decreasing exponentially at each time-step or layer, and the weights can converge to their optimal values in a reasonable amount of time. This allows LSTM's to solve the vanishing gradient problem – since the value stored in a memory cell isn't iteratively modified, the gradient does not vanish when trained with backpropagation.

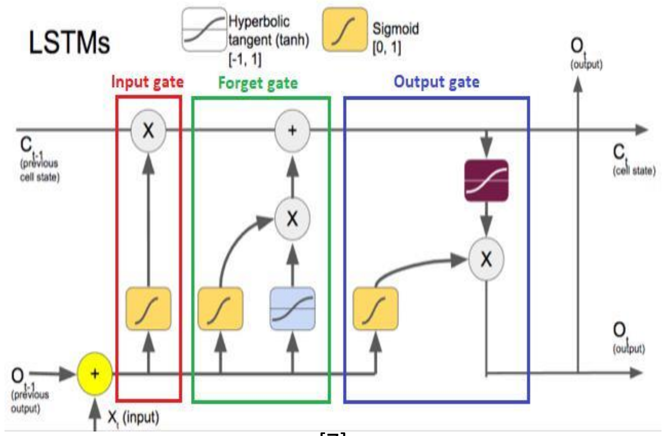
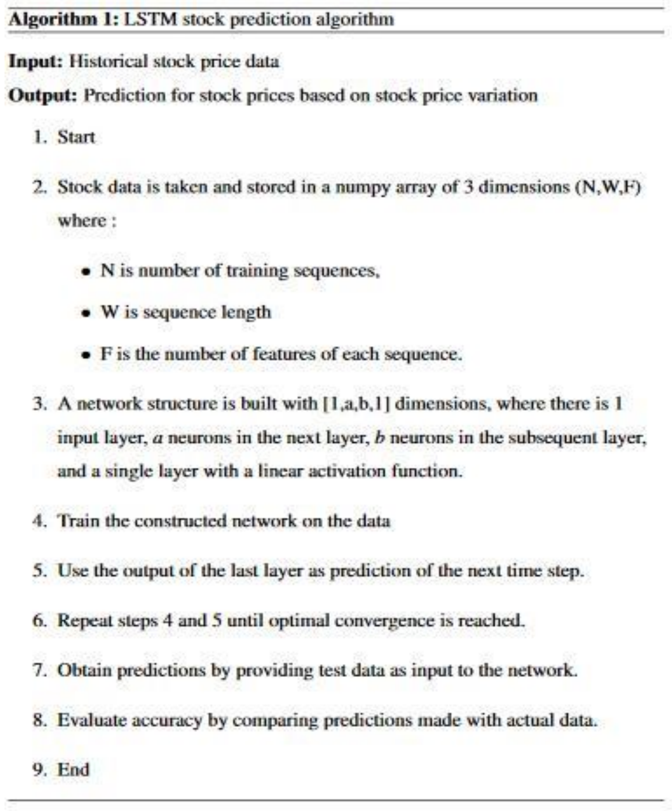


Figure 2 LSTM Model

Stock prediction Algorithm Used



```
Algorithm 1: LSTM stock prediction algorithm
Input: Historical stock price data
Output: Prediction for stock prices based on stock price variation
  1. Start
  2. Stock data is taken and stored in a numpy array of 3 dimensions (N,W,F)
     where :
       • N is number of training sequences,
       • W is sequence length
       • F is the number of features of each sequence.
  3. A network structure is built with [1,a,b,1] dimensions, where there is 1
     input layer, a neurons in the next layer, b neurons in the subsequent layer,
     and a single layer with a linear activation function.
  4. Train the constructed network on the data
  5. Use the output of the last layer as prediction of the next time step.
  6. Repeat steps 4 and 5 until optimal convergence is reached.
  7. Obtain predictions by providing test data as input to the network.
  8. Evaluate accuracy by comparing predictions made with actual data.
  9. End
```

Terminologies used

Given below is a brief summary of the various terminologies relating to our proposed stock prediction system:

1. **Training set** : subsection of the original data that is used to train the neural network model for predicting the output values
2. **Test set** : part of the original data that is used to make predictions of the output value, which are then compared with the actual values to evaluate the performance of the model
3. **Validation set** : portion of the original data that is used to tune the parameters of the neural network model
4. **Activation function**: in a neural network, the activation function of a node defines the output of that node as a weighted sum of inputs.

$$Activation function = \Sigma(Input * weights) + Bias$$

Here, the sigmoid and ReLU (Rectified Linear Unit) activation functions were tested to optimize the prediction model.

a. Sigmoid – has the following formula

$$y = 1/(1+e^{-x})$$

and graphical representation

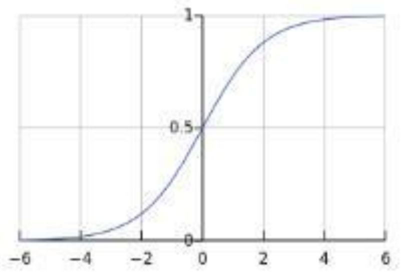


Figure 3 Sigmoid Function

$$y = max(0,x)$$

b. ReLU – has the following formula

and graphical representation

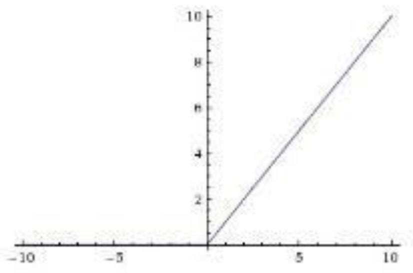


Figure 4 ReLU Function

5. **Batch size** : number of samples that must be processed by the model before updating the weights of the parameters
6. **Epoch** : a complete pass through the given dataset by the training algorithm
7. **Dropout**: a technique where randomly selected neurons are ignored during training i.e., they are "dropped out" randomly. Thus, their contribution to the activation of downstream neurons is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass.
8. **Loss function** : a function, defined on a data point, prediction and label, that measures a penalty such as square loss which is mathematically explained as follows –

$$l(f(xi), yi) = (f(xi) - yi)^2$$

9. **Cost function**: a sum of loss functions over the training set. An example is the Mean Squared Error (MSE), which is mathematically explained as follows:

$$MSE() = \Sigma Ni = 1(f(xi) - yi)^2/N$$

10. **Root Mean Square Error (RMSE)**: measure of the difference between values predicted by a model and the values actually observed. It is calculated by taking the summation of the squares of the differences between the predicted value and actual value, and dividing it by the number of samples. It is mathematically expressed as follows:

$$\sqrt{\frac{\Sigma(y_{predicted} - y_{actual})^2}{N}}$$

    In general, smaller the RMSE value, greater the accuracy of the predictions made.

METHODOLOGY

Project Workflow

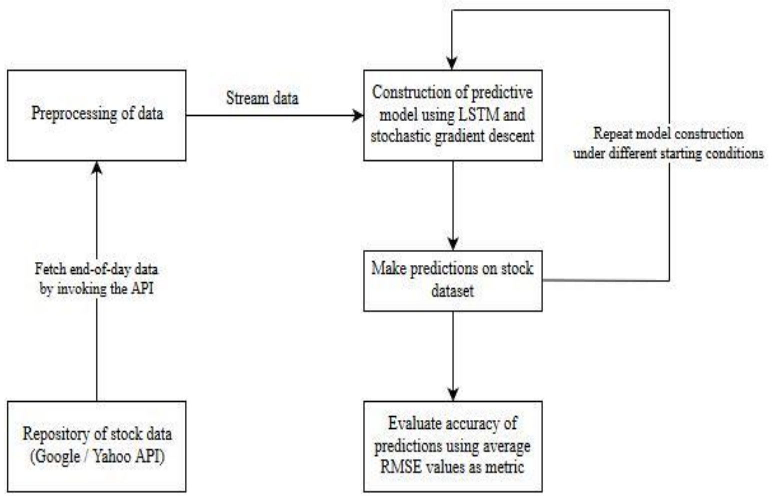

Preprocessing of data
Stream data
Construction of predictive model using LSTM and stochastic gradient descent
Repeat model construction under different starting conditions
Fetch end-of-day data by invoking the API
Make predictions on stock dataset
Repository of stock data (Google / Yahoo API)
Evaluate accuracy of predictions using average RMSE values as metric

Figure 5 Project Workflow

About the Dataset

We are using Google's Stock price from 5 years till now from a financial website (Yahoo! Finance). Benchmark stock market data (for end-of-day prices of various ticker symbols i.e., companies) was obtained from two primary sources: Yahoo Finance and Google Finance. These two websites offer URL-based APIs from which historical stock data for various companies can be obtained for various companies by simply specifying some parameters in the URL, but we choose to download the CSV dataset.

The obtained data contained five features:

1. Date: of the observation
2. Opening price: of the stock
3. High: highest intra-day price reached by the stock
4. Low: lowest intra-day price reached by the stock
5. Volume: number of shares or contracts bought and sold in the market during the day
6. OpenInt i.e., Open Interest: how many futures contracts are currently outstanding in the market

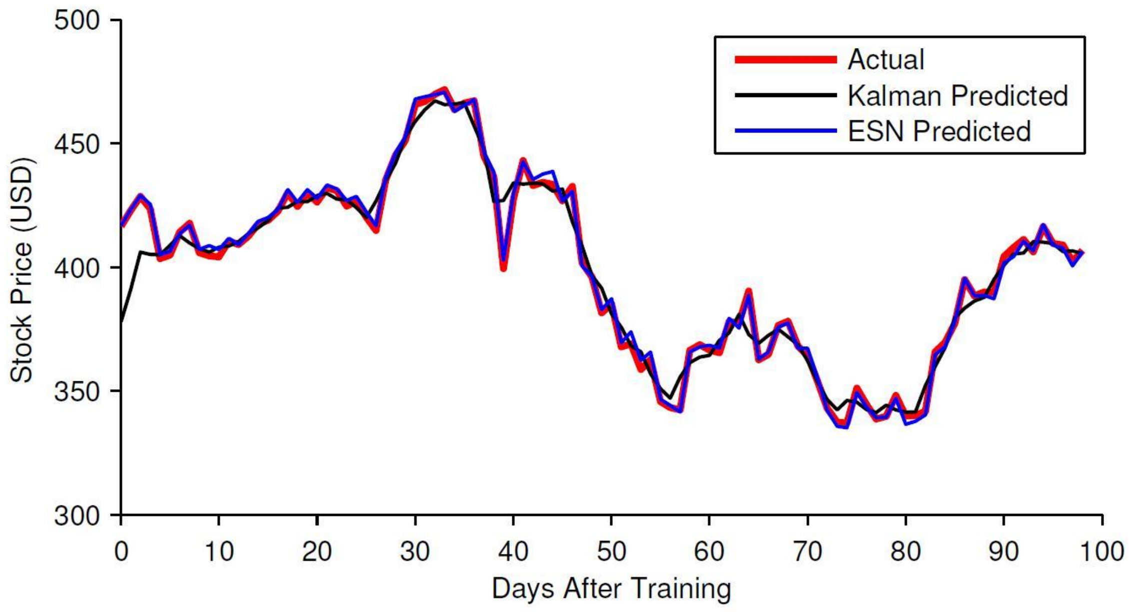


Figure 6 Proposed Model

We will also train our LSTM on 5 years of data. We can see that their predictions are quite close to the actual Stock Price. We can try to get the same accuracy from our model as well.

We assume that the present day is January 01, 2019. We will then get the Google Stock price for the previous 5 years. Once we train our LSTM, we will try to predict the stock price for the month of January 2019.

Data Preprocessing

We have the training set of 5 years of Google Stock price. The test set contains the stock price for January 2019. We will first import it. As we can see in the dataframe, the dates range from 2012 to 2016.
We can see that there are 3 types of stocks - Open, High, Low and Close. There is also a Volume column which contains the Volume of stocks for Google. We will focus on the Open stock price and will predict this price for January 2019.

The input for RNN will NOT be Date and Open Stock Price, it will just be the Open Stock Price for different time frames. We will get this by using just the Open Stock Price form our Dataframe using iloc.

For feature scaling, we have 2 options - Standardisation and Normalisation. Since LSTMs use many sigmoid functions, which work in 0s and 1s, it also makes sense to use Normalisation, that converts the data between 0 and 1. But you can check the results of both the methods and choose for yourself.

Date	Open	High	Low	Close	Volume
1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1/4/2012	331.27	333.87	329.08	666.45	5,749,400
1/5/2012	329.83	330.75	326.89	657.21	6,590,300
1/6/2012	328.34	328.77	323.68	648.24	5,405,900
1/9/2012	322.04	322.29	309.46	620.76	11,688,800
1/10/2012	313.7	315.72	307.3	621.43	8,824,000
1/11/2012	310.59	313.52	309.4	624.25	4,817,800
1/12/2012	314.43	315.26	312.08	627.92	3,764,400
1/13/2012	311.96	312.3	309.37	623.28	4,631,800
1/17/2012	314.81	314.81	311.67	626.86	3,832,800
1/18/2012	312.14	315.82	309.9	631.18	5,544,000

Table 1 Training Data

We will next try to determine what the input (X_train) and output (y_train) will be. The input will be the value that changes with time i.e. the current open stock price (time t). The output would obviously be the future value of the same i.e. the near future open stock price (time t+1). The trick behind choosing the ranges would be that the prediction would be a day after the current value. The training set contains 1258 values. The input should be therefore restricted to 1257. The output on the other hand, cannot contain the 0th days prediction, so it will start from 1 and end at 1258.

The use of LSTM involves the prediction along the time of a particular value. The input that we have is currently 2-dimensional - we have 1257 rows and 1 column. We need to add another dimension to the input in account of time. This process is called reshaping. This format of input is required by Keras and the arguments have to be in the order of batch_size(number of rows), timesteps(the number of time intervals or days between any 2 rows, in this case it will be 1) and input_dim (number of columns). These 3 arguments are encapsulated together and come after the original data as the argument of Numpy's reshape function.

Building the RNN

We will first import 3 classes. The Sequential class that will initialise our RNN. The Dense class will create the output layer of our RNN. And finally, the LSTM class which will make our RNN have "Long Memory".

The method to create the RNN will be the same as the one used to create ANN and CNN. We will then use other methods of the Dense and LSTM classes to add the layers and compile it, and eventually fit it.

We are predicting a continuous variable and are hence using a regression model instead of a classification model and call our object 'regressor'. To this regressor object, we will add the LSTM layer, which in itself will take the input layer as input. The arguments we add to the LSTM layer are: units - number of memory units, activation function - can be tanh or sigmoid. Other arguments will be default. But there will be an additional argument - the input shape argument to specify the format of our input layer. This argument would be none and 1 - none to specify that model can expect any time step and 1 because we have just 1 column of input. The optimal number of memory units that we can use is 4, the activation function is sigmoid, and the input_shape would be (None, 1).

The next layer that we will add is the Output layer. We will use the Dense class, with the argument being units, everything else being default. The units are the number of neurons that should be present in the output layer, which is dependent on the dimensions of the output. So, our units argument for Dense class will have value 1.

To compile all the layers into a single system, we will use the compile function along with its arguments. Optimizer can be RMSprop or Adam. Both the optimizers gave similar results but RMS was memory heavy, so we went forwarded with Adam. But usually, RMSprop is recommended in Keras documentation. The Loss argument decides the manipulation of weights, so for this we should be using Mean Squared Error for continuous variable. For test set, we might use Root Mean Square Error in its place. Other arguments will be default.
Now we will fit this regressor to the training dataset. We will use the fit method for this. The important arguments include the input, output, batch_size and epochs. We will keep the default batch size of 32 but will change epochs to 200 for better convergence.

Date	**Open**	**High**	**Low**	**Close**	**Adj Close**	**Volume**
1/24/2019	1076.48	1079.475	1060.7	1073.9	1073.9	1361300
1/25/2019	1085	1094	1081.82	1090.99	1090.99	1119100
1/28/2019	1080.11	1083	1063.8	1070.08	1070.08	1284300
1/29/2019	1072.68	1075.15	1055.865	1060.62	1060.62	1021800
1/30/2019	1068.43	1091	1066.85	1089.06	1089.06	1279800
1/31/2019	1103	1117.33	1095.41	1116.37	1116.37	1538300
2/1/2019	1112.4	1125	1104.89	1110.75	1110.75	1462200
2/4/2019	1112.66	1132.8	1109.02	1132.8	1132.8	2576500
2/5/2019	1124.84	1146.85	1117.248	1145.99	1145.99	3552200
2/6/2019	1139.57	1147	1112.77	1115.23	1115.23	2105600
2/7/2019	1104.16	1104.84	1086	1098.71	1098.71	2044800

Table 2 Test Set from 2019

With the passage of fitting, we see that the loss keeps on decreasing. But we will get accurate results only if we have the same loss in the test set.

IMPLEMENTATION AND RESULTS

Making Predictions and Visualising the Result

The methods of getting the Test Dataset is same as that Training Dataset. We will just rename it to real_stock_price so that we can distinguish between prediction and actual values.

Next, we use our model to make predictions on the test dataset. But we should keep in mind that every prediction is for the next day and not the present. The input will be the real_stock_price. The model that we have made is on scaled values. When used as it is, it will give incorrect predictions. So we will convert the input using the same "sc" object used for scaling the training data. We will also have to reshape the data according to the format expected by the predict method in a 3d format.
We will now use the regressor model to make predictions on the input and store it in the predicted_stock_price. The argument would obviously be the input. We now have the predicted stock price for the January 2017.

But this output will be scaled. We will have to use the inverse transform method of the same "sc" object we had used to scale the data to get the proper predicted values. This is the final prediction.
Now we visualize our predictions with the actual stock prices of Google. For this we use the pyplot module. We have some arguments with pyplot, like the use of color. For real stock prices, we will use Red. We will also include a label mentioning the real stock price. We will keep blue the color for Predicted Stock price and change the label as well. We will also add the axis labels and title and display it.

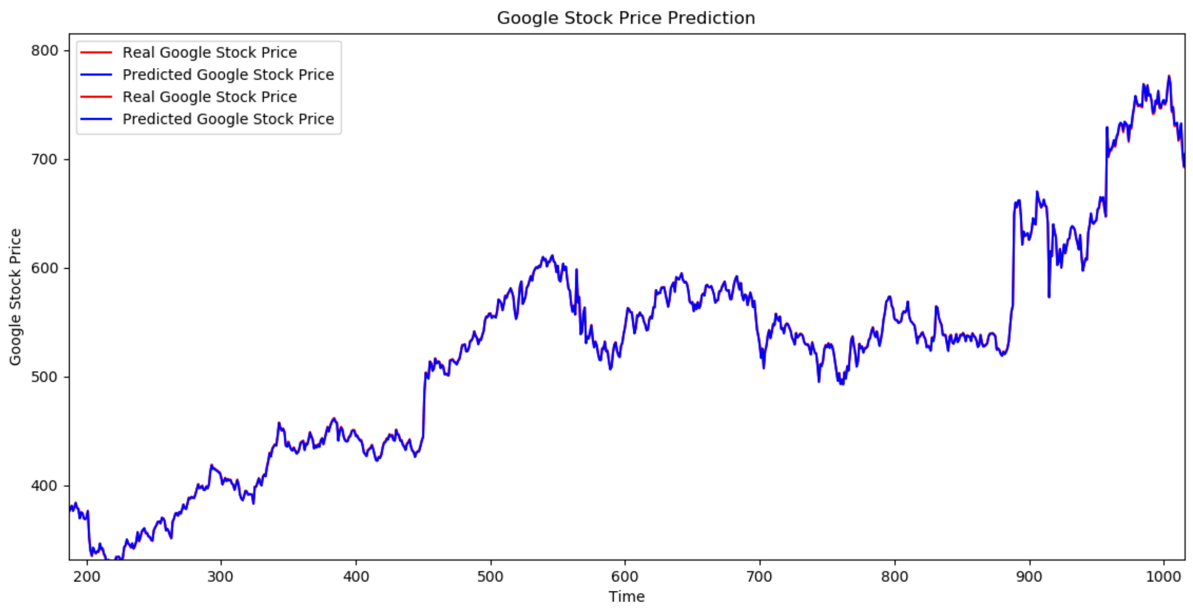


Figure 7 Predicted & Actual Plot

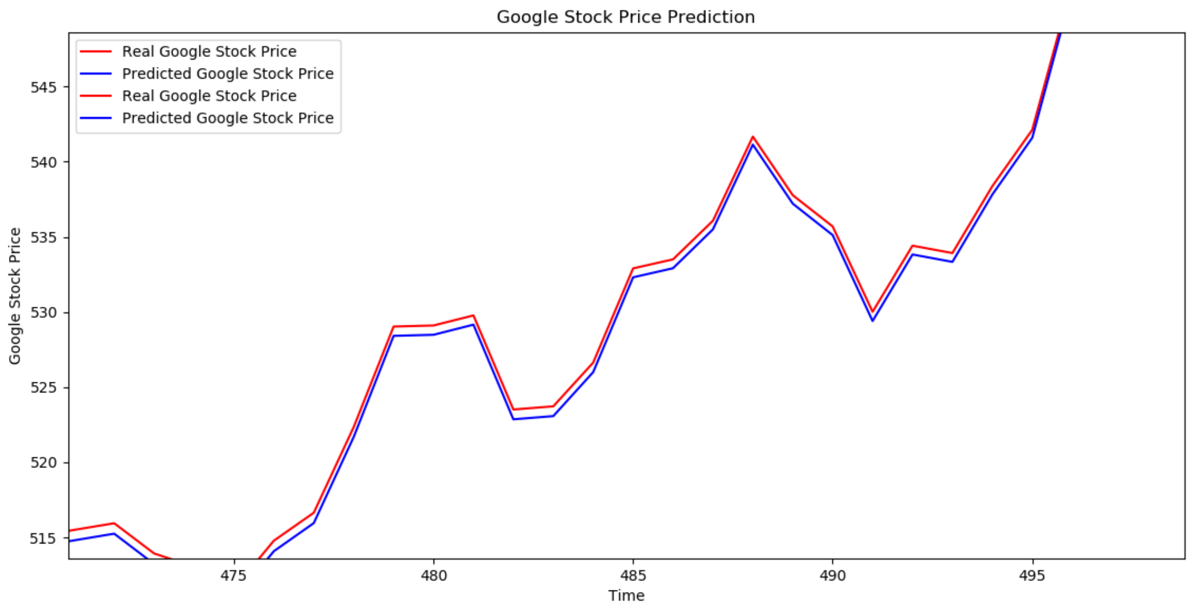


Figure 8 Predicted & Actual Plot Zoomed In

What's important to note is that this is a 1 time-step prediction i.e. input is of time t and prediction is of time t+1. We should note that we were able to make these predictions for 20 days only because we had the stock price for 20 days. This would not have been possible if we had the stock price for just 1 day. It would be wonderful to make predictions for a long future, but we would hardly get such amazing predictions. In finance, there is the Browning Motion, which makes future values of stock prices independent of the past, so it would be impossible to make long term predictions for stock price.

Further Analysis

We will take this a little further and make predictions for the stock price from 2012 to 2016. The method of getting the input, scaling and reshaping has not changed.

We already have the regressor and Xtrain from previous work. We just have to make predictions using it. And then we can unscale the predictions to regain the actual values of the stocks.

We compare our predictions with the actual prices through visualisation. At first, we are not able to see the real stock price. But after zooming in, we can see that the LSTM has very accurately predicted the prices and the Blue graph is mostly overlapping the Red one.

Evaluating the RNN

RNNs are evaluated using the Root Mean Square (RMSE) of the test set. We will the root using the math library. The mean squared error function is taken from the scikit-learn's metrics module. These 2 functions when combined, will calculate the rmse for the test sets actual values and the predictions we made. The actual RMSE value does not tell us anything about the size of the test set and how the error correlates to it. We need this in percentage. 800 is the average value of the stock in the test set, so we will divide the RMSE value by 800 to get a percentage value. We see that the value is close to 0.4% which is a very low error rate and good for our predictions.

CONCLUSION AND FUTURE WORK

Although the LSTM we have designed predicts quite accurately the stock prices of Google, the reason it is so accurate is because it is learning at time-step of 1. This leads to a reset of hidden layer, and this process goes on and the model is not learning anything useful. This output is not relevant because of this 1 time-step learning.

To make improvements in our model, we need to increase the time-step.

The results of comparison between Long Short Term Memory (LSTM) and Artificial Neural Network (ANN) show that LSTM has a better prediction accuracy than ANN.

Stock markets are hard to monitor and require plenty of context when trying to interpret the movement and predict prices. In ANN, each hidden node is simply a node with a single activation function, while in LSTM, each node is a memory cell that can store contextual information. As such, LSTMs perform better as they are able to keep track of the context-specific temporal dependencies between stock prices for a longer period of time while performing predictions.

An analysis of the results also indicates that both models give better accuracy when the size of the dataset increases. With more data, more patterns can be fleshed out by the model, and the weights of the layers can be better adjusted.

At its core, the stock market is a reflection of human emotions. Pure number crunching and analysis have their limitations; a possible extension of this stock prediction system would be to augment it with a news feed analysis from social media platforms such as Twitter, where emotions are gauged from the articles. This sentiment analysis can be linked with the LSTM to better train weights and further improve accuracy.

REFRENCES

[1] Nazar, Nasrin Banu, and Radha Senthilkumar. “An online approach for feature selection for classification in big data.” Turkish Journal of Electrical Engineering & Computer Sciences 25.1 (2017): 163-171.

[2] Soulas, Eleftherios, and Dennis Shasha. “Online machine learning algorithms for currency exchange prediction.”
Computer Science Department in New York University, Tech. Rep 31 (2013).

[3] Suresh, Harini, et al. “Clinical Intervention Prediction and Understanding using Deep Networks.” arXiv preprint arXiv:1705.08498 (2017).

[4] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio.
“On the difficulty of training recurrent neural networks.”
International Conference on Machine Learning. 2013.

[5] Zhu, Maohua, et al. “Training Long Short-Term Memory With Sparsified Stochastic Gradient Descent.” (2016)

[6] Ruder, Sebastian. “An overview of gradient descent optimization algorithms.” arXiv preprint arXiv:1609.04747.(2016).

[7] Recht, Benjamin, et al. “Hogwild: A lock-free approach to parallelizing stochastic gradient descent.” Advances in neural information processing systems. 2011.

[8] Ding, Y., Zhao, P., Hoi, S. C., Ong, Y. S. “An Adaptive Gradient Method for Online AUC Maximization” In AAAI (pp. 2568-2574). (2015, January).

[9] Zhao, P., Hoi, S. C., Wang, J., Li, B. “Online transfer learning”. Artificial Intelligence, 216, 76-102. (2014)

[10] Altinbas, H., Biskin, O. T. “Selecting macroeconomic influencers on stock markets by using feature selection algorithms”. Procedia Economics and Finance, 30, 22-29. (2015).