Implementation of Long Short-Term Memory

Rachit Arora
University of Adelaide
Adelaide, Australia
a1788005@student.adelaide.edu.au

Abstract

Long Short-Term Memory (LSTM) network is an artificial recurrent neural network architecture capable of learning order dependence in sequence predictions problems. This work provides prefatory yet extensive study on LSTM. We have build a LSTM to predict Google stock price, Kaggle dataset by Rahul Shah[4]. We have implemented LSTM by using tensorflow and it's build in configurations.

1. Introduction

Predicting stock prices is one of the crucial problems in machine learning, this is done to predict return on stocks. The stock market is very dynamic in nature and is highly volatile, and thus there is a need of an efficient algorithm. In this work we will discuss LSTM, one of the popular deep learning models, in detail.

Long Short Term Memory recurrent neural network is a part of deep learning algorithms. It is different from the traditional neural network as it can process a sequence of data. The architecture consists of cell, input gate, output gate, forget gate. These are special Recurrent neural networks (RNN) capable of learning long term dependencies.

In this work LSTM is build for Google stock prices. A sequential order of various layers such as LSTM layers and dense layers are used to build the network. After checking the performance on the validation set the model is fine tuned by using various methods such as droupout and comparing root mean squared error.

1.1. The Dataset

The dataset used in this work is Google stock price Kaggle dataset by Rahul Shah[4]. There are six columns namely Date, Open (opening price of the stock), High (highest price of the stock for that day), Low (lowest price of the stock for that day), Close (closing price of the stock for that day), Volume. The dataset has 1258 rows in the train data and 20 rows in the test data. The first 5 rows of train data is displayed in figure 1.

Figure 1. The train dataset

	date	open	high	low	close	adj_close	volume
0	2015-11-25	107.510002	107.660004	107.250000	107.470001	101.497200	1820300
1	2015-11-27	107.589996	107.760002	107.220001	107.629997	101.648300	552400
2	2015-11-30	107.779999	107.849998	107.110001	107.169998	101.213867	3618100
3	2015-12-01	107.589996	108.209999	107.370003	108.180000	102.167740	2443600
4	2015-12-02	108.099998	108.269997	106.879997	107.050003	101.100533	2937200

2. Background

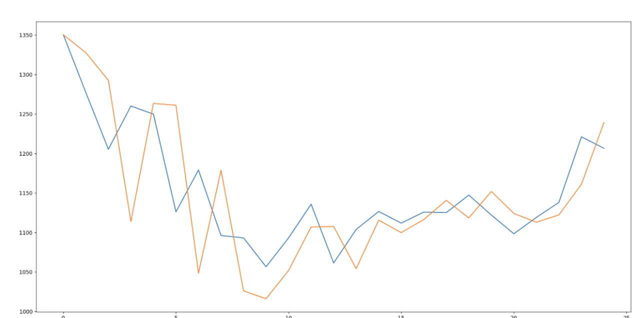
There are a number of researches already done research in this area, and have performed really well. We will discuss it in this section of the report.

2.1. Related work

Many researchers have have done research in stock prediction and some have come up with models. There are few on Google stock prediction and few on other companies stock prediction.

Michelangiolo Mazzeschi have done Google price prediction shown in Figure 2[3]. It shows the actual and the predicted closing prices. We can see that the model's performace is not too accurate.

Figure 2. Vanilla LSTM performance



Another research has been done in which LSTM model, Stacked LSTM and Attention-Based LSTM, along with the traditional ARIMA model, is used for prediction of stock prices. The results are shown in Figure 3[6].

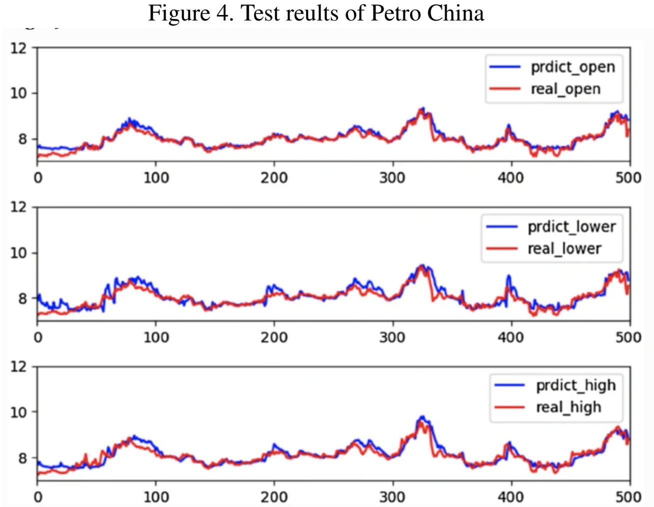
Figure 3. Mean squared error for three models

Mean Squared Error (*10^-3)			
Stock Ticker	LSTM	Stacked-LSTM	Attention-LSTM
XOM	25	25	11
GE	18	2	3
WMT	128	163	76
MSFT	40	27	16
IBM	28	7	6
AAPL	54	61	44
GOOG	35	37	26
GS	12	16	18
PFE	32	64	49
JNJ	57	123	41

In the Figure 3, it is clear that the Attention based LSTM model is better than the LSTM model because it assigns different weights to input features and thus automatically choose the relevant features.

Guangyu Ding Liangxi Qin have made a LSTM model to perform stock price prediction of Petro China and ZTE. The results ca be seen in Figure 4 and 5[2].

Figure 4. Test reults of Petro China



It can be seen that the LSTM model performs really well, predicted and the real prices are almost similar. Mean squared error was used to check the performance of the model. Because LSTM models are effective, it is used for stock price prediction.

In Figure 6[5], different experiments are carried out to check the performance of the neural networks. It is Deep Learning for stock price Prediction using Convolutional Neural Network and Long Short-Term Memory. Out of all the models the CNN with LSTM performs gives the best result because CNN is used for feature extractions and

Figure 5. Test reults of ZTE

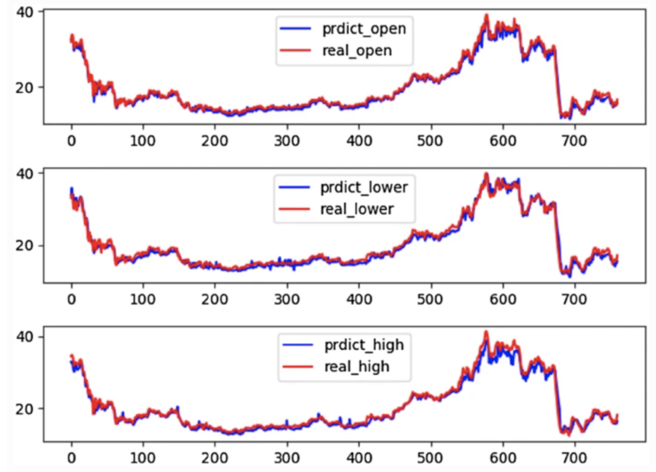


Figure 6. Performance of experiments

Metrics	γ	CNN3D	CNN3D-DR	LSTM-D	CNN3D + LSTM	CNN3D-D + LSTM	CNN3D-DR + LSTM
Accuracy	60/40	0.5082	0.4957	0.5202	0.4993	0.5693	0.5833
	65/35	0.5095	0.5009	0.5348	0.5130	0.5605	0.5728
	70/30	0.4798	0.4952	0.5339	0.5079	0.5991	0.6160
	75/25	0.4856	0.5200	0.5229	0.5150	0.5968	0.6316
	80/20	0.4878	0.5180	0.5175	0.5010	0.5653	0.5916
	Average	0.4942	0.5060	0.5258	0.5072	0.5782	0.5991
Precision	60/40	0.5019	0.4954	0.5124	0.4991	0.5671	0.5818
	65/35	0.5071	0.4985	0.5317	0.5029	0.5529	0.5650
	70/30	0.4904	0.4927	0.5150	0.5004	0.5929	0.6107
	75/25	0.5045	0.5165	0.5006	0.5222	0.5921	0.6281
	80/20	0.4949	0.5183	0.5089	0.5009	0.5620	0.5894
	Average	0.4998	0.5043	0.5137	0.5051	0.5734	0.5950
Recall	60/40	0.5017	0.4964	0.5107	0.4992	0.5635	0.5779
	65/35	0.5062	0.4975	0.5276	0.5027	0.5513	0.5621
	70/30	0.4938	0.4937	0.5130	0.5002	0.5910	0.6094
	75/25	0.5034	0.5155	0.5003	0.5218	0.5897	0.6249
	80/20	0.4951	0.5184	0.5087	0.5008	0.5596	0.5860
	Average	0.5001	0.5043	0.5121	0.5049	0.5710	0.5921
F-measure	60/40	0.4954	0.4951	0.4962	0.4986	0.5602	0.5754
	65/35	0.4989	0.4980	0.5154	0.5000	0.5507	0.5609
	70/30	0.4632	0.4923	0.5043	0.4989	0.5912	0.6096
	75/25	0.4791	0.5157	0.4911	0.5141	0.5894	0.6250
	80/20	0.4875	0.5163	0.5072	0.4995	0.5577	0.5845
	Average	0.4848	0.5035	0.5028	0.5022	0.5698	0.5911

LSTM network for stock price movement direction prediction. With the combination of the two, an efficient model is developed.

By comparing all the above different models, it can be inferred that the LSTM models performs better than the other models for stock price prediction.

3. Methodology

LSTM has the ability to process an entire sequence of data and thus are better than some of the deep learning algorithms. LSTM is specially designed to avoid long-term dependency problem.

Basic Recurrent Neural Network (RNN) have a repeating structure of simple modules such as a tanh layer. This can be seen in Figure 7[1]. This is a chain like structure with a single neural networ layer. If there is a long gap or more information to learn from then the RNN does not perform really well, i.e. RNN is not able to learn from them.

LSTM are also chain like structures but they are a bit different. Instead of a single neural network layer, there are four layers interacting with each other. The LSTM building block is shown in Figure 8.

Figure 7. Repeating module in basic RNN

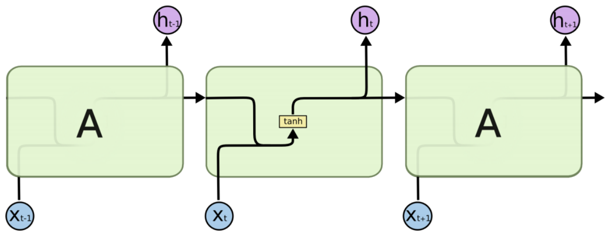
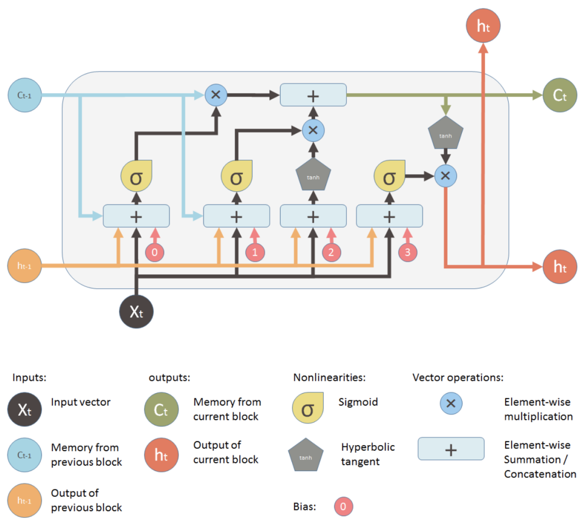


Figure 8. LSTM building block

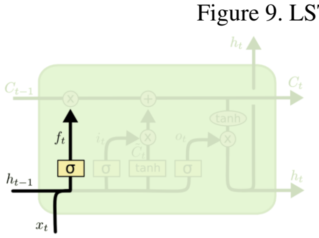


The network takes three inputs, input of the current time, an output from the previous block and memory from the previous block. The output h_t is the output of the current time and C_t is the memory os the current unit.

The decision is made by using current input, previous output and previous memory. Then a new output is generated and the memory is altered.

A step by step LSTM is explained from Figure 9 to 12[1]. The first step is what information we don't want, this decision is made by the forget gate layer, which in this case is the sigmoid layer.

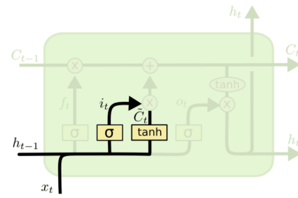
Figure 9. LSTM Step1



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The next step in Figure 10[1] we will decide what information we could store in the cell. This further has two steps in which sigmoid layer decides the values to be updated and the $tanh$ layer create new values that is known as the candidate values C_t. This values may or may not be added to the state.

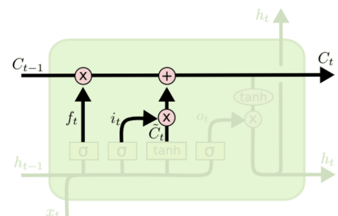
Figure 10. LSTM Step2



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

In next step we combine the above two mentioned steps and update the state as seen in Figure 11[1].

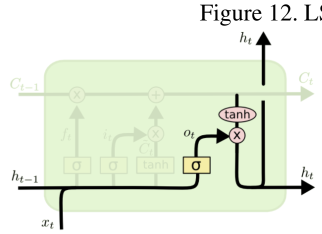
Figure 11. LSTM Step3



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

In next step we have to update the old cell state C_{t-1} to a new cell state C_t. This is done by multiplying C_{t-1} to f_t as shown in the Figure 11. This means that we will have will forget the things we decided to forget earlier and add i_t multiplied by C_t as shown in the Figure 11. New candidate values are scaled by how much we need to change each state value.

Figure 12. LSTM Step4



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

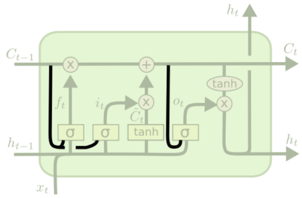
Finally we see what is the output, this again has two steps. The first step is to run a sigmoid layer to decide which part of cell state we want as output.

In the second step a tanh function is implemented on the cell state multiplied it with the sigmoid gate's output, to only output the part we want. The equation are displayed in the Figure 12[1].

The above explained is a basic LSTM model, there are a variants of LSTM's. Most LSTM's are different and it can be seen in the Figure 13[1].

It can be seen that peepholes connetions are added, it means that the gate layer looks like cell state.

Figure 13. LSTM



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \;+\; b_o\right)$$

3.1. Our architecture

In this experiment, following model architecture is adopted.

Figure 14. Model architecture

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_63 (LSTM)               (None, 10, 50)            10400
_________________________________________________________________
dropout_63 (Dropout)         (None, 10, 50)            0
_________________________________________________________________
lstm_64 (LSTM)               (None, 50)                20200
_________________________________________________________________
dropout_64 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_31 (Dense)             (None, 1)                 51
=================================================================
Total params: 30,651
Trainable params: 30,651
Non-trainable params: 0
```

A sequential organization is implemented as shown in Figure 14. We have used LSTM layers, droupout layers and dense layers in our architecture. We have started with 50 neurons, dropout of 0.2 and an adam optimizer.

Then we have fine tuned the model to find out the best optimizer, droupout and number of neurons.

4. Experimental analysis

4.1. Results

The data is explored and an error is found in the data. By checking the histograms in Figure 15 and Figure 16, we can see that the Close price highest value is more that 1200, whereas high and low prices are less than 850. Therefore, the data have some error.

Upon checking the history of the data we have found that 1000 shares were split into 2002 shares and some of the shares haven't been split. To correct the issue, we have divided some of the Close prices by 2.002, and hence the data error was rectified.

LSTM	RMSE
Validation before fine tuning	9.67
Validation after tuning droupout and units	9.47
Validation after tuning optimizer - Final	9.05

Table 1. Results before fine tuning

Figure 15. Histograms of Open and Low price

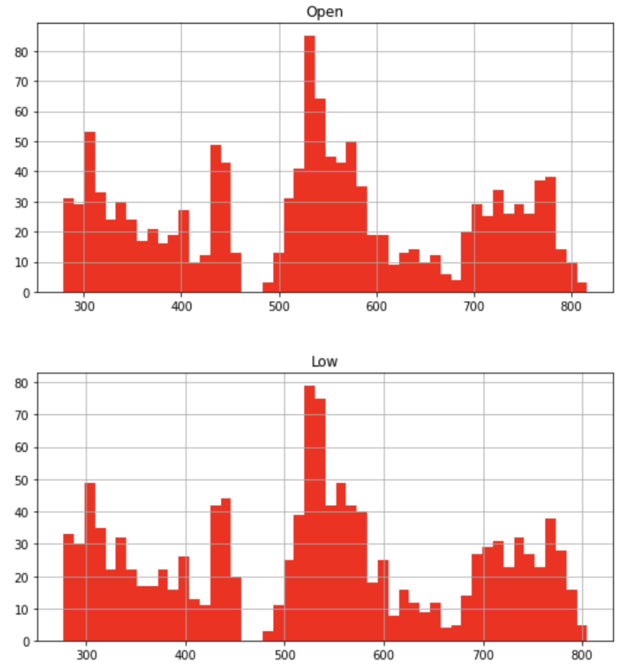


Figure 16. Histograms of High and Close price

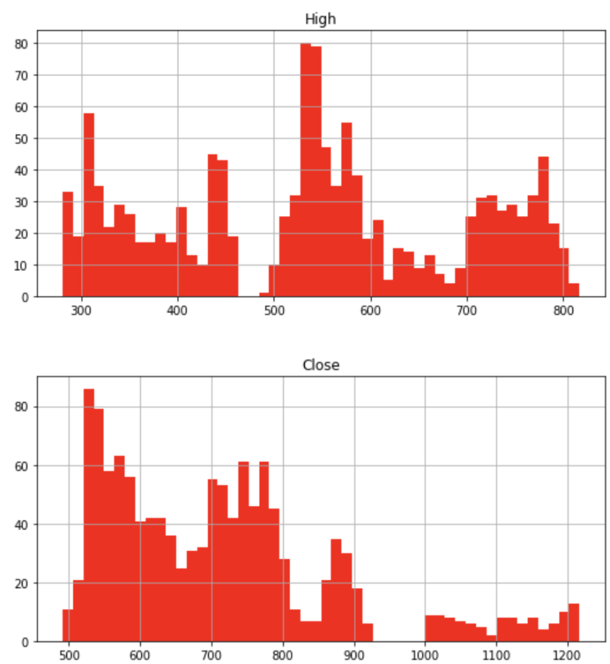


Figure 17 is the train, validation and the predictions of our model. Since the graph is not very clear, we have set a limit on the axis to get a zoom view in Figure 18. We can see that our predictions and the actual validation data is similar but there is still some error.

Figure 17. Train, Validation and Prediction of Close

Figure 18. Train, Validation and Prediction of Close - Zoom View

Figure 19. Checking optimizer

	optimizer	rmse
0	adam	9.356405
1	sgd	13.984245
2	rmsprop	10.112763
3	adagrad	14.909684
4	adadelta	14.846204
5	adamax	10.742085
6	nadam	9.058347

This error is calculated by finding the root mean squared error (rmse) which in this case, for our model before tuning, is 9.67.

We have further fine tuned the model to find the droupout value and the number of neurons value, that can be seen in Figure 20, and optimizer, that can be seen in Figure 19. The optimum number of neurons are 32, droupout rate is 0.2 and the optimizer for best results is nadam.

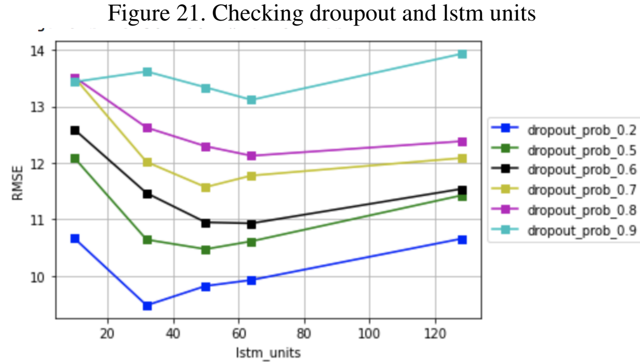
This is the best model we have achieved after fine tuning the parameters. The root mean squared error is 9.05 which is lower that what we achieved before fine tuning.

Figure 20. Checking droupout and lstm units

Time taken = 90.06555351018906 minutes

	lstm_units	dropout_prob	rmse
0	10	0.2	10.661097
1	10	0.5	12.085043
2	10	0.6	12.588331
3	10	0.7	13.510065
4	10	0.8	13.518815
5	10	0.9	13.434697
6	32	0.2	9.476239
7	32	0.5	10.640165
8	32	0.6	11.464290
9	32	0.7	12.015928
10	32	0.8	12.624712
11	32	0.9	13.618939
12	50	0.2	9.819341
13	50	0.5	10.471557
14	50	0.6	10.947340
15	50	0.7	11.570543
16	50	0.8	12.294441
17	50	0.9	13.337755
18	64	0.2	9.925140
19	64	0.5	10.611329
20	64	0.6	10.929054
21	64	0.7	11.775085
22	64	0.8	12.125935
23	64	0.9	13.115773
24	128	0.2	10.655376
25	128	0.5	11.422596
26	128	0.6	11.537601
27	128	0.7	12.085191
28	128	0.8	12.381877
29	128	0.9	13.929970

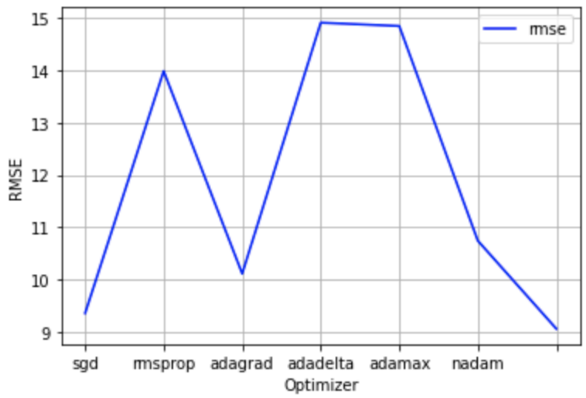
Figure 21. Checking droupout and lstm units



5. Code

The code can be found at the following link, the url is provided https://github.com/richie-1/LSTM.git

Figure 22. Checking optimizers



6. Conclusion

In this report Long Short Term Memory have been implemented. This first model is made at random. Then fine tuning is done and the graphs are studied to get the best parameters. Dropout layers are added to avoid overfitting. The final model is used to predict the stock prices and the performance is checked by calculating the root mean squared error. This model could be further improved by tuning the other parameters such as number of epochs.

In future work, experiments with different number of layers can help improve the performance of the model. In addition, other features could be used to do feature extraction, such as taking a mean of low and high price, which could help in improving the performance of the model.

References

[1] Colah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[2] G. Ding and L. Qin. Study on the prediction of stock price based on the associated network model of lstm. *International Journal of Machine Learning and Cybernetics*, 11(6):1307–1317, 2020.

[3] M. Mazzeschi. Google stock predictions using an lstm neural network. https://medium.com/towards-artificial-intelligence/google-stock-predictions-using-an-lstm-neural-network-dbe785949a96, 2019.

[4] R. Shah. https://www.kaggle.com/rahulsah06/gooogle-stock-price. https://www.kaggle.com/rahulsah06/gooogle-stock-price, 2019.

[5] C. Yang, J. Zhai, and G. Tao. Deep learning for price movement prediction using convolutional neural network and long short-term memory. *Mathematical Problems in Engineering*, 2020, 2020.

[6] Z. Zou and Z. Qu. Using lstm in stock prediction and quantitative trading.