Stock Prediction with Machine Learning Models

MA2132 Journal

Lee Jia Jie, Prannaya Gupta
M19207

20 September 2019

Question

What would be an optimal architecture for a model to predict stock prices with as high accuracy as possible, given the constraints of either a FFNN, LSTM, GRU or CNN, and the number of 36 different models which may be trained to optimize its hyperparameters?

Contents

Done By: **Lee Jia Jie** and **Prannaya Gupta** (M19207)

Part 1 Abstract

In this Mathematics Journal, we have predicted stocks by training neural networks on data from Yahoo Finance. This was done using the programming Language `Python` on the Google Colaboratory Platform and using the Machine Learning Models: LSTM, GRU, CNN and FFNN. First, we found the optimal model by deriving the data from a Yahoo Finance into a `NFLX.csv` file and pre-processing pipeline. We then split the data and trained the 36 different derived models and, after finding LSTM as the better of the models, found the most optimal architecture for the LSTM model. We lastly unscaled and compared the prices against historical data and found the Opening Price for Netflix on tomorrow, the 21st of September, 2019 to be **$287.181**. Based on this, we concluded that a LSTM model with the architecture - CuDNNLSTM layer with 256 neurons, dropout layer with a 0.4 dropout rate, flatten layer, dense layer with 512 neurons, dropout layer with a 0.4 dropout rate and the final dense layer with 1 output neuron, yields the most accurate results. In the end we tested against Apple, NVIDIA and DBS to see if it recognised their patterns as well. In the end used it to predict day high and low and finally find the day high and low predictions for the next 10 days (21 September - 1 October 2019).

(Word Count: **225** words)

Part 2 Introduction

In this Mathematics Journal, we are going to predict stocks by training neural networks on data from Yahoo Finance.

I had just been studying machine learning when this Statistics Journal was assigned and thought that building a simple stock prediction system would be a good way to practise what I had learned and at the same time, complete the project. Furthermore, I could benefit from my father's expertise as a stock trader, and I gained from him a basic overview of how the stock market worked.

For this project, we will be training and evaluating the models on five years' worth of Netflix's stock price data taken from Yahoo Finance.

We will use the `Python` programming language and the machine learning and data science libraries: `tensorflow`, `scikit-learn`, `pandas`, `numpy`, `matplotlib` and `pandas_datareader` to automatically download the data. Finally, all the code will be written in an IPython notebook and executed on Google Colaboratory, which provides free hosted GPU runtime.

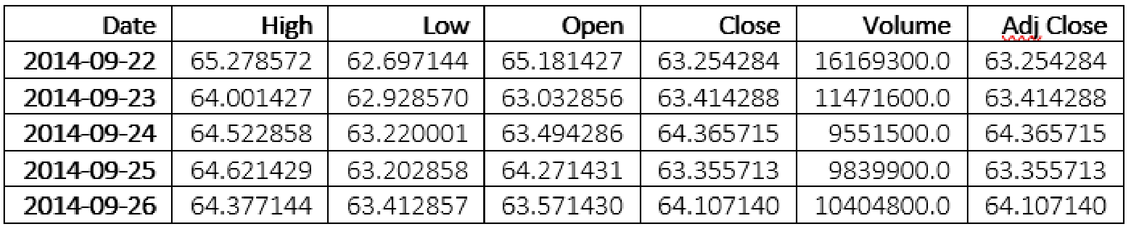
We are taking the loss of each model as the mean squared error and our aim is to obtain as low a value as possible; it shall basically be our measure of accuracy.

Part 3:
Finding the optimal model

3.1 Downloading the Data

For the most recent data to be easily downloaded, we automate it using the pandas_datareader module. However, for the purposes of this journal we will just save the downloaded data in `data/NFLX.csv`, so that it can be loaded quickly when the code is re-run.

Here are the first 5 rows of the csv file:



Date	High	Low	Open	Close	Volume	Adj Close
2014-09-22	65.278572	62.697144	65.181427	63.254284	16169300.0	63.254284
2014-09-23	64.001427	62.928570	63.032856	63.414288	11471600.0	63.414288
2014-09-24	64.522858	63.220001	63.494286	64.365715	9551500.0	64.365715
2014-09-25	64.621429	63.202858	64.271431	63.355713	9839900.0	63.355713
2014-09-26	64.377144	63.412857	63.571430	64.107140	10404800.0	64.107140

Table 1: Sample data from `NFLX.csv`

3.2 Pre-processing Pipeline

The data must be pre-processed before it can be used to train the model. The following steps are the transformations that will be made:

- ➔ Remove the “`Adj Close`” column as it is essentially equivalent to the “`Close`” column and therefore useless.
- ➔ Calculate the 10- and 30-days moving average of the closing price and add them as columns to the table.
- ➔ Remove days with `NaN`[1] values, i.e. missing values.
- ➔ Normalize the data, i.e. scale each value to between 0 and 1. This can be done easily by subtracting the minimum value and of that column and dividing the result by the range of the data, that is

$$\frac{X - X_{min}}{X_{max} - X_{min}}$$

where X is a column from the table.

To achieve this, two subclasses of `BaseEstimator` and `TransformerMixin` (from `sklearn.base`) are created, the first of which updates the features and thus satisfies the first 2 points, and the second performing the third transformation. For normalization, the `MinMaxScaler` class (from sklearn.preprocessing) is used. `sklearn.base.Pipeline` is used to create the pipeline.

[1] Not a Number

3.3 Splitting the Data

We will now split our data into input and output data for the models to train on, the former comprising the high, low, close, volume and 10- and 30-days moving average of the closing price, of the past 100 days, and the latter being the opening price of the next day. Thereafter, the input and output data will be split into sections for training, validation and testing, with a ratio of 70:15:15.

The model's weights and biases will be trained and optimized with the training data set, while the validation set is used to check for overfitting during training. This basically means that the model learns to memorize all the input and output values and does not generalize well to new data. Finally, the test set is used to evaluate the model's performance after training is complete.

To achieve everything described above, we implemented 2 functions, `split_input_output` and `split_train_val_test`, which do as their names suggest. The input and output data returned by `split_input_output` is passed to `split_train_val_test`, which finally returns the six arrays as desired.

3.4 Training Different Models

We will be training 4 different types of machine learning models in this project. Their architectures can be seen in the code at the end of this report.

The first model is a basic Feed-Forward Neural Network (FFNN), containing only dense layers.

The second model uses Long Short-Term Memory (LSTM), which is designed for time-series prediction. We will use the CuDNNLSTM layer from tensorflow.keras.

The third model uses Gated Recurrent Unit layers (GRU). Like the LSTM, it is designed for such tasks as stock prediction but is simpler and more computationally efficient. We will use the CuDNNGRU layer from tensorflow.keras.

Finally, the fourth model is a Convolutional Neural Network (CNN). This model is not commonly used in stock prediction and time-series prediction in general, and is included mostly for variety and entertainment value.

After training the four models, the losses on the test set are shown on the right:

Model	Test loss
FFNN	0.0085
LSTM	0.0017
GRU	0.0019
CNN	0.0057

Table 2: Model test scores

Considering the graph of validation loss, it can be seen that the LSTM performs best, followed by the GRU model. The standard FFNN performs the worst of all.

Shown on the right is a graph comparing the models' predictions and the actual opening price.

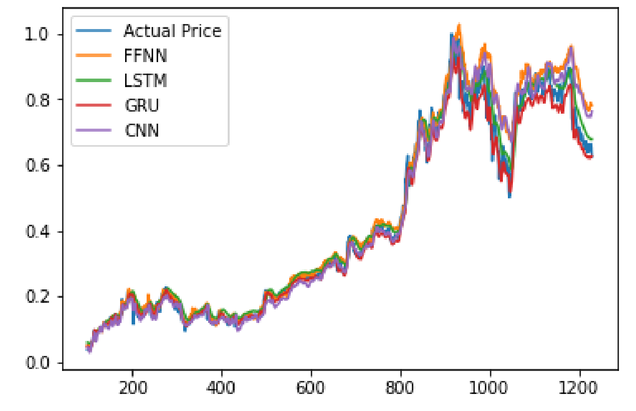


Figure 1: Comparison of the actual opening price and predictions

In the next section, we will optimize the LSTM's hyperparameters.

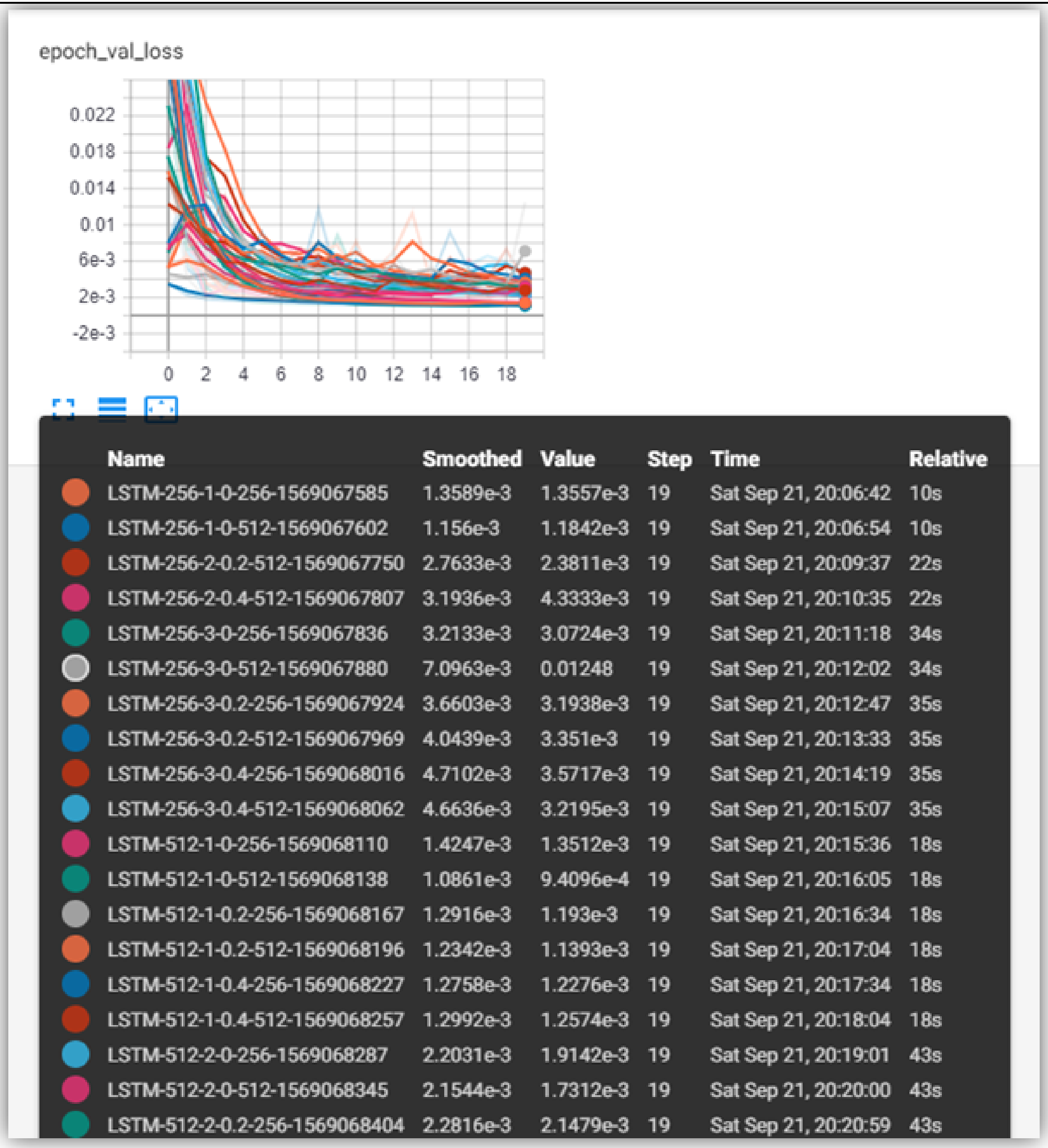
3.5 Finding the Optimal Architecture for the LSTM

It is now time to fine-tune the LSTM model, and we do this by trying out multiple combinations of hyperparameters, such as the learning rate, decay, number of layers, layer sizes, etc. Possible values for each of these hyperparameters are stored in a list, and all their combinations are used to train separate models which will them be compared with the test set.

Due to time constraints, we are only able to tweak a few of these parameters, and the ones we have chosen are as follows:

- ➔ the number of neurons in the LSTM layers. There are 2 possible values chosen: 256 and 512.
- ➔ the number of LSTM layers. There are 3 possible values: 1, 2, or 3.
- ➔ the dropout rate of the layers. There are 3 possible values: 0, 0.2, or 0.4.
- ➔ the number of neurons in the fully-connected layers. There are 2 possible values, the same as that of the LSTM (256 and 512).

There are 36 possible combinations, and thus 36 models will be trained. After training, TensorBoard is used to compare the models' performances. The results are as shown below (Each model's name is in the format `LSTM-{LSTM neurons}-{LSTM layers}-{dropout rate}-{Dense neurons}-{timestamp}`):



	Name	Smoothed	Value	Step	Time	Relative
●	LSTM-256-1-0-256-1569067585	1.3589e-3	1.3557e-3	19	Sat Sep 21, 20:06:42	10s
●	LSTM-256-1-0-512-1569067602	1.156e-3	1.1842e-3	19	Sat Sep 21, 20:06:54	10s
●	LSTM-256-2-0.2-512-1569067750	2.7633e-3	2.3811e-3	19	Sat Sep 21, 20:09:37	22s
●	LSTM-256-2-0.4-512-1569067807	3.1936e-3	4.3333e-3	19	Sat Sep 21, 20:10:35	22s
●	LSTM-256-3-0-256-1569067836	3.2133e-3	3.0724e-3	19	Sat Sep 21, 20:11:18	34s
○	LSTM-256-3-0-512-1569067880	7.0963e-3	0.01248	19	Sat Sep 21, 20:12:02	34s
●	LSTM-256-3-0.2-256-1569067924	3.6603e-3	3.1938e-3	19	Sat Sep 21, 20:12:47	35s
●	LSTM-256-3-0.2-512-1569067969	4.0439e-3	3.351e-3	19	Sat Sep 21, 20:13:33	35s
●	LSTM-256-3-0.4-256-1569068016	4.7102e-3	3.5717e-3	19	Sat Sep 21, 20:14:19	35s
●	LSTM-256-3-0.4-512-1569068062	4.6636e-3	3.2195e-3	19	Sat Sep 21, 20:15:07	35s
●	LSTM-512-1-0-256-1569068110	1.4247e-3	1.3512e-3	19	Sat Sep 21, 20:15:36	18s
●	LSTM-512-1-0-512-1569068138	1.0861e-3	9.4096e-4	19	Sat Sep 21, 20:16:05	18s
●	LSTM-512-1-0.2-256-1569068167	1.2916e-3	1.193e-3	19	Sat Sep 21, 20:16:34	18s
●	LSTM-512-1-0.2-512-1569068196	1.2342e-3	1.1393e-3	19	Sat Sep 21, 20:17:04	18s
●	LSTM-512-1-0.4-256-1569068227	1.2758e-3	1.2276e-3	19	Sat Sep 21, 20:17:34	18s
●	LSTM-512-1-0.4-512-1569068257	1.2992e-3	1.2574e-3	19	Sat Sep 21, 20:18:04	18s
●	LSTM-512-2-0-256-1569068287	2.2031e-3	1.9142e-3	19	Sat Sep 21, 20:19:01	43s
●	LSTM-512-2-0-512-1569068345	2.1544e-3	1.7312e-3	19	Sat Sep 21, 20:20:00	43s
●	LSTM-512-2-0.2-256-1569068404	2.2816e-3	2.1479e-3	19	Sat Sep 21, 20:20:59	43s

Figure 2: Validation loss against number of epochs

LSTM-256-1-0-256-1568463366	0.0005474354467450587
LSTM-256-1-0-512-1568463375	0.0004492358456927297
LSTM-256-1-0.2-256-1568463384	0.001225208372959648
LSTM-256-1-0.2-512-1568463393	0.0007193362199947895
LSTM-256-1-0.4-256-1568463402	0.0009116612504893805
LSTM-256-1-0.4-512-1568463412	0.0004236454558421803
LSTM-256-2-0-256-1568463423	0.0017564928405644263
LSTM-256-2-0-512-1568463437	0.0023046846885014984
LSTM-256-2-0.2-256-1568463452	0.006309656870058354
LSTM-256-2-0.2-512-1568463468	0.0009144440267632222
LSTM-256-2-0.4-256-1568463485	0.0016942301395294422
LSTM-256-2-0.4-512-1568463502	0.0018224813235814081
LSTM-256-3-0-256-1568463520	0.002481179139302934
LSTM-256-3-0-512-1568463542	0.0016386102739488705
LSTM-256-3-0.2-256-1568463565	0.001755849872407613
LSTM-256-3-0.2-512-1568463590	0.0013674528536605922
LSTM-256-3-0.4-256-1568463615	0.00296733299996156
LSTM-256-3-0.4-512-1568463641	0.0015114462153766961
LSTM-512-1-0-256-1568463669	0.0007310977883582168
LSTM-512-1-0-512-1568463690	0.0006145404237459469
LSTM-512-1-0.2-256-1568463712	0.0007105961558409035
LSTM-512-1-0.2-512-1568463734	0.0006120822207509156
LSTM-512-1-0.4-256-1568463756	0.000960952670464073
LSTM-512-1-0.4-512-1568463780	0.0004549121336929281
LSTM-512-2-0-256-1568463803	0.0008366446312078658
LSTM-512-2-0-512-1568463836	0.0007797060322927256
LSTM-512-2-0.2-256-1568463870	0.001109833661827971
LSTM-512-2-0.2-512-1568463906	0.0009010062998105937
LSTM-512-2-0.4-256-1568463941	0.0019515789606991936
LSTM-512-2-0.4-512-1568463977	0.002409091345308458
LSTM-512-3-0-256-1568464014	0.003118468977182227
LSTM-512-3-0-512-1568464062	0.0009144910732763545
LSTM-512-3-0.2-256-1568464110	0.002377724928288337
LSTM-512-3-0.2-512-1568464160	0.004663190593504731
LSTM-512-3-0.4-256-1568464210	0.0014436753872466986
LSTM-512-3-0.4-512-1568464262	0.0012397152696982684

Table 3: Model test results

It is obvious that the model with 256 neurons in its LSTM layers, 1 LSTM layer, a 0.4 dropout rate and 512 neurons in its final Dense layer performed the best in the test set.

3.6 Unscaling and Comparing the Prices

For the stock predictions to be useful, they must of course be unscaled to obtain the actual prices. We have thus implemented a function, `unscale_price`, which basically takes the same `MinMaxScaler` used to scale the data, obtains the x_{min} and x_{max} stored in the class, and uses those values to unscale the data.

Now that the actual prices have been obtained, the predictions can be compared with the actual price to see the actual error of the model in terms of money.

Here are the first five rows of the actual price predictions:

Date	Predicted	Actual
2015-03-27	58.167995	59.330002
2015-03-30	57.471416	59.715714
2015-03-31	57.224842	60.110001
2015-04-01	57.108532	59.642857
2015-04-02	56.827667	59.071430

Table 4: Sample predictions on Netflix data

As you can see, the model has an error of around a few dollars. The mean absolute error of all the predictions is calculated to be around $4.20.

Unfortunately, this is too high and definitely not practical in stock trading.

3.7 Tomorrow’s Opening Price

We used the LSTM model to predict the opening price of the next day (21/09/19), which was unscaled and yielded:

$287.181

Part 4: Conclusion and other improvements

4.1 Conclusion and Future Improvements

Our answer to the question is apparent: Given our constraints, we have found that an LSTM model with the following architecture yields the most accurate predictions:

- ➔ CuDNNLSTM layer with 256 neurons
- ➔ Dropout layer with a 0.4 dropout rate
- ➔ Flatten layer
- ➔ Dense layer with 512 neurons
- ➔ Dropout layer with a 0.4 dropout rate
- ➔ The final Dense layer with 1 output neuron

4.2 Testing the LSTM on other Companies

We decided to test the best performing model on the stock prices of other randomly selected companies to see if it could recognise their patterns as well. The results are as follows:

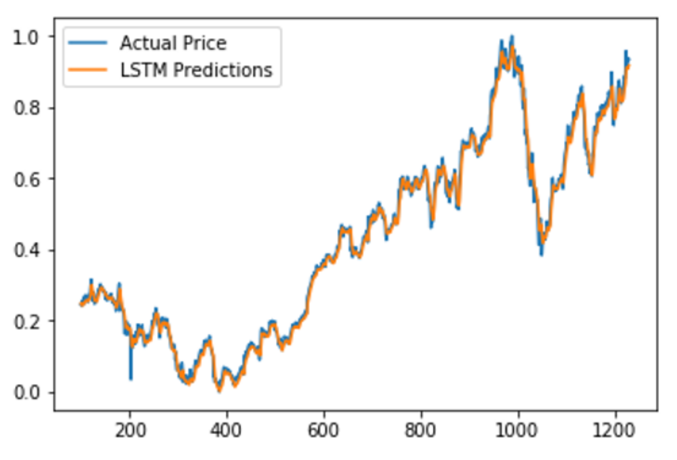


Figure 3: Apple predictions and the actual opening price (loss: 4.1550e-04)

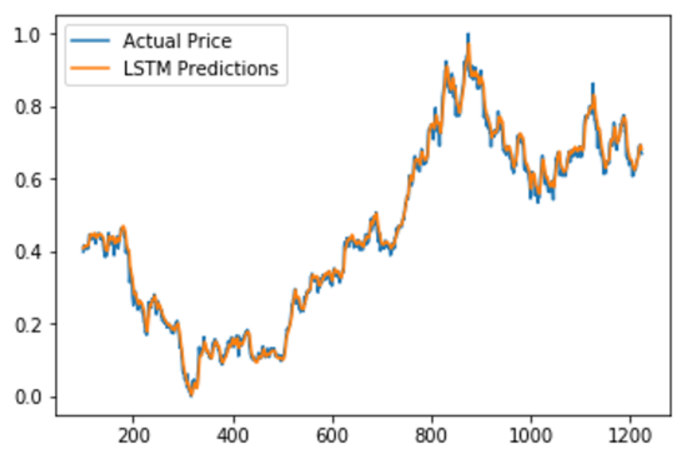


Figure 4: DBS predictions and the actual opening price (loss: 3.0266e-04)

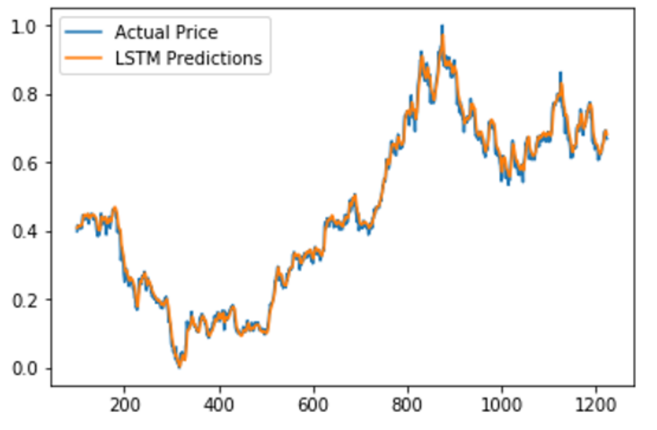


Figure 5: NVIDIA predictions and the actual opening price (loss: 3.0817e-04)

4.3 Day High and Low

As the model's predictions of the opening price is not accurate enough for practical use in the stock market, we have decided to try using it to predict the day high and low instead, which, given the insufficient accuracy, would be more useful as it would help a stock trader decide when to buy and sell more, as the price approaches the predicted day high or low.

We used the architecture of the best performing LSTM model as discussed above and trained 2 separate models to predict the high and low prices respectively.

Here are the predictions after training:

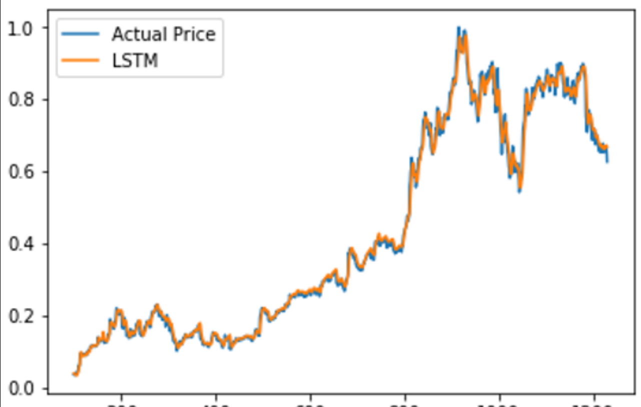


Figure 6: Model predictions and the actual high price (loss: 5.1280e-04)

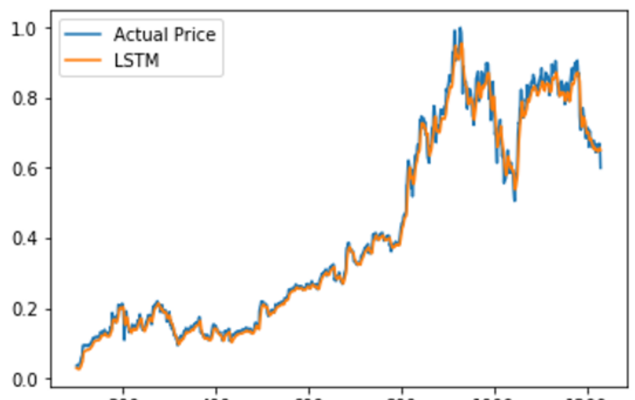
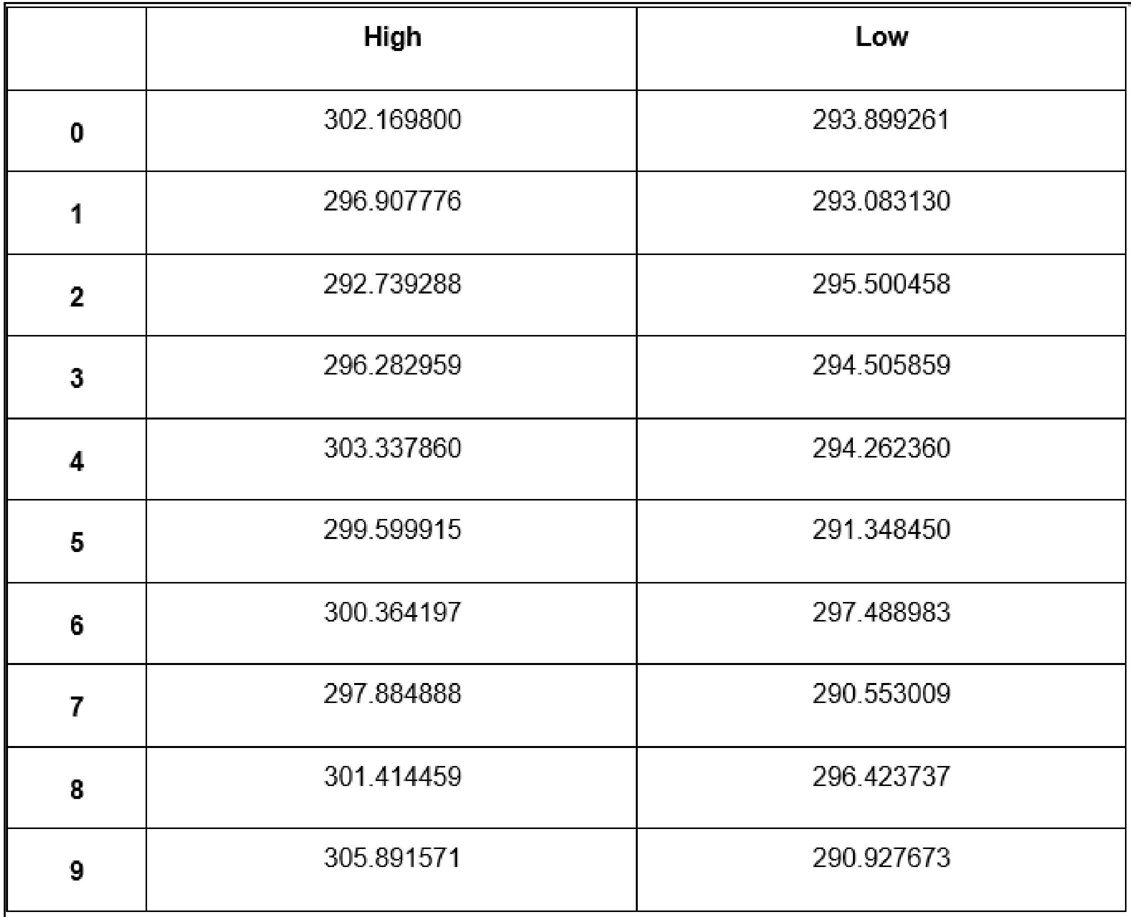


Figure 7: Model predictions and the actual low price (loss: 7.2225e-04)

4.4 High and Low Predictions

Next, we trained a model with 20 outputs for the high and low prices of the next 10 days based on the architecture of the best performing LSTM model mentioned above. It had quite a high loss of 0.0031. We unscaled the values using the above discussed function, and the following prices were obtained:



	High	Low
0	302.169800	293.899261
1	296.907776	293.083130
2	292.739288	295.500458
3	296.282959	294.505859
4	303.337860	294.262360
5	299.599915	291.348450
6	300.364197	297.488983
7	297.884888	290.553009
8	301.414459	296.423737
9	305.891571	290.927673

Table 5: Predictions for the next 10 days on Netflix

Of course, as the amount of output neurons has increased its accuracy would definitely be affected, and the optimal model would probably require a larger neural network.

Report Word Count: **1687** Words (with titles)

Part 5 References

- ➔ https://sg.finance.yahoo.com/quote/NFLX?p=NFLX&.tsrc=fin-srch. (2019). Netflix, Inc. (NFLX) Stock Price, Quote, History & News
- ➔ TensorFlow. (2019). tf.keras.layers.CuDNNLSTM | TensorFlow Core r1.14. [online] Available at: https://www.tensorflow.org/api_docs/python/tf/keras/layers/CuDNNLSTM.
- ➔ TensorFlow. (2019). tf.keras.layers.CuDNNGRU | TensorFlow Core r1.14. [online] Available at: https://www.tensorflow.org/api_docs/python/tf/keras/layers/CuDNNGRU
- ➔ Docs.scipy.org. (2019). NumPy Reference — NumPy v1.17 Manual. [online] Available at: https://docs.scipy.org/doc/numpy/reference/
- ➔ Scikit-learn.org. (2019). API Reference — scikit-learn 0.21.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/classes.html
- ➔ YouTube. (2019). Recurrent Neural Networks (RNN) - Deep Learning w/ Python, TensorFlow & Keras p.7. [online] Available at: https://www.youtube.com/watch?v=BSpXCRTOLJA
- ➔ YouTube. (2019). Stock Prediction using LSTM Recurrent Neural Network. [online] Available at: https://www.youtube.com/watch?v=zwqwlR48ztQ
- ➔ YouTube. (2019). TensorFlow Tutorial #23 Time-Series Prediction. [online] Available at: https://www.youtube.com/watch?v=6f67zrH-_IE
- ➔ AnalyticsVidhya (2019). Essentials of Deep Learning : Introduction to Long Short Term Memory. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/
- ➔ wikiHow. (2019). How to Write a Scientific Paper. [online] Available at: https://www.wikihow.com/Write-a-Scientific-Paper [Accessed 21 Sep. 2019].

Part 6 Code

The code is more than 700 lines long in total and thus could not be printed here.

However, here is the link to the IPython Notebook in Google Colaboratory:
https://colab.research.google.com/drive/1P2a9xDqS-32rlKZubLD2nsjVs1veFlLS

To prevent errors, please run the cells in order, but to save time you can skip the cells training and evaluating different combinations of LSTM models, which requires a total of around 30 minutes to run.

Of course, if you run the code you may find different results from the ones mentioned in this report, but the difference should be marginal.