Predicting Ethereum Prices using a LSTM Recurrent Network

Carolyne Pelletier **Akhil Dalal**
101054962 **100855466**

1. Introduction

Multivariate time-series modeling and forecasting constitutes an important problem with numerous applications in real-world domains. In particular, the financial market has an interest in gathering insights from the data the markets generate. Insights such as market trend and sentiment can help brokers choose a particular investment strategy. The recent rise in popularity of cryptocurrencies has created a largely unregulated and emotional market, making it highly volatile. This along with its very limited history, makes cryptomarkets difficult to predict.

In this report, we consider multivariate time series modelling from the Ethereum (ETH) cryptocurrency, where the data consists of various market conditions along with ETH daily closing prices. The goal in this supervised learning problem is to use past market conditions to predict future closing prices of ETH reflected in US dollars.

The Long Short-Term Memory (LSTM) recurrent neural network is chosen to solve this problem for its ability to learn long sequences of observations. More specifically, an LSTM cell can learn to recognize important input, store it in a long-term state, and learn to forget information it no longer needs [1]. This specialized ability makes it extremely attractive for a problem that includes time series data in which several different inputs (market conditions) affect a single output (ETH closing price prediction).

2. Long-Short Term Memory (LSTM)

To train a vanilla RNN on long sequences, you need to multiply its shared weights several times by itself. The product has the potential to either vanish or explode depending on the magnitude of the weight. This prevents the network from capturing the long term dependencies in the data [2] . To solve this issue, various types of cells with long term memory have been introduced -- LSTMs cells (Figure 1).

LSTM cells manage two state vectors: cell state and hidden state. The cell state c_(t) manages long term state throughout the LSTM layer, and the hidden state h_(t) manages the short term state within the cell.

The current input x_(t), and the previous hidden state h_(t-1) are fed to four different fully connected layers:

- The main layer g_(t) analyzes the current input and the previous hidden state, and stores this information in the cell state c_(t).
- The three other layers are gate controllers:
    - The forget gate f_(t) controls which parts of the cell state should be erased. Weak signals are blocked which prevents vanishing gradients.
    - The input gate i_(t) controls which parts of g_(t) should be added to the cell state.
    - The output gate controls which part of the cell state should be read in and sent for output at the current time step, both to the hidden state and to the cell's output y_(t) [1].

The gate controllers use the logistic activation function, therefore their outputs are always within the range of 0 to 1. These values indicate how much reading, writing, and forgetting to perform. Since the output of the current cell takes into account the state of the previous cell, this model can be trained to predict time series data like future ETH closing prices.

4. Methodology

4.1 Data Representation

The dataset, fetched from CoinMarketCap's API [3], ranges from January 1st, 2016 to December 16th, 2017. Data from before January 1st, 2016 was excluded due to the immaturity of the ETH market thereby making it an inaccurate representation of the ETH market at present. The market conditions are polled daily and include the Open, High, Low, Close (OHLC) prices along with Volume traded. A time series plot of the actual closing price can be seen in Figure 2, along with the training and testing set.

4.2 Training and Testing Sets

The ETH historical dataset is split into two parts: a training and a test set and spans nearly 24 months from January 1st, 2016 to December 16th, 2017. Eighty percent (80%) of the data is used in the training set and the remaining 20% is used in the testing set (Figure 2). Models are developed and trained using the training dataset and make predictions on the test dataset.

4.3 Sliding Window

When analyzing ETH time series data, the economic environment changes so drastically that it may not be reasonable to assume that a model's market conditions are constant over time. A common technique to assess constancy of a time series model is to compute market condition estimates over a sliding window of a fixed window size. Therefore, the data is segmented into a fixed window size which is continuously shifted over by one time-step, creating a set of time sequences.

4.4 Normalization

Since the nature of this data displays a large variance, the sliding window is normalized relative to the first entry to reflect percentage changes from the start of that window (data at index 0 is 0). This is done because small changes in the bigger numbers affect predictions by a lot more when compared to the same small change in a smaller number. The larger numbers have a bigger dominance during the optimization process, so normalization is required. Once the data in our sliding window is normalized, we feed it into an LSTM cell and record the prediction. The data is subsequently de-normalized for plotting purposes.

$$normalized_i = \frac{window\_element_i}{window\_element_0} - 1$$

$$un\_normalized_i = window\_element_0 * normalized_i + 1$$

4.5 Measuring Accuracy

The Root Mean Square Error (RMSE) is used to measure the accuracy of our network architectures. RMSE was chosen because it measures the difference between values predicted by a model and the values actually observed. The model with the lowest RMSE represents the best predictive performance as shown in section 6.

$$\boldsymbol{RMSE} = \sqrt{\frac{\sum_{t=1}^{n}(y_t - y_{hat_t})^2}{n}}$$

5. Network Architecture

The input to the network is a vector of market data split into sliding windows. The next component to the network consists of 2 stacked LSTM layers, each of which is made up of several LSTM cells. Each LSTM cell takes as input a single element from a sliding window. Experimentation with a single LSTM layer gave us unfavorable results and was therefore not included in section 6. To prevent overfitting, we then apply the dropout regularization technique

on the second LSTM layer. The output of the second LSTM layer is connected to a fully connected/dense linear layer which uses a linear activation function. The dense layers outputs a single number which is the prediction for that time step. There is a total of 123,305 weights in this network architecture.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (1, 15, 100)              42400
_________________________________________________________________
lstm_2 (LSTM)                (1, 100)                  80400
_________________________________________________________________
dropout_1 (Dropout)          (1, 100)                  0
_________________________________________________________________
dense_1 (Dense)              (1, 5)                    505
_________________________________________________________________
activation_1 (Activation)    (1, 5)                    0
=================================================================
Total params: 123,305
Trainable params: 123,305
Non-trainable params: 0
_________________________________________________________________
```

6. Experiments

To minimize the RSME loss, several network architectures and hyper parameters were tested. The best forecasting results (figure 3) were found with the following implementation:

- Network Architecture: 2 LSTM layers
- Hyper parameters:
  - Window Size: 15
  - Number of Days in Future Predictions: 5
  - Number of Neurons in LSTM cell: 100

The table below indicates the **RSME** values for each experiments.

- **FP** are days in future predictions. This represents the number of days the network is making predictions for. Predicting too far in the future gives inaccurate results compared to predicting a shorter time in the future.
- **WS** is the size of the sliding window. This is the number of days in the past that network looks at to make the prediction for the next FPs. Looking too far in the past reduces the accuracy of the future prediction.
- **Ne** is the number of neurons in an LSTM cell.

Network Architecture	WS = 15 FP = 5 Ne = 20	WS = 15 FP = 30 Ne = 20	WS = 15 FP = 30 Ne = 100	**WS = 15** **FP = 5** **Ne = 100**	WS = 30 FP = 5 Ne = 20	WS = 30 FP = 5 Ne = 100	WS = 30 FP = 30 Ne = 20	WS = 30 FP = 30 Ne = 100
2 LSTM layers	0.26	1.53	1.8	**0.24**	0.36	0.40	2.08	2.16

7. Conclusion

Due to the large number of external influences that affect ETH prices, predicting future prices has proved to be quite difficult. However, some useful patterns have emerged from our predictive model. For example, while the 30 day future predictions are not very accurate, the magnitude and direction of the predictions (slope) does give an indication of the volatility of the market. Predicting the volatility of a market could be useful to make a prediction regarding market environments allowing one to know what type of market they are currently in. Knowing what type of financial market one is in could help choose a trading strategy. For example, an arbitrage strategy might be more successful in producing high returns in a high volume environment. In future work, it would be interesting to see the

results of adding Bitcoin prices to our data since Bitcoin prices tend to heavily influence other cryptocurrencies in the market.

8. References

1. Géron, Aurélien, Hands-On Machine Learning with Scikit-Learn & TensorFlow p.402-407
2. Goodfellow, Bengio, Courville, DeepLearning, p.397-404
3. Coinmarketcap.com
4. https://deeplearning4j.org/lstm.html
5. https://towardsdatascience.com/lstm-by-example-using-tensorflow-feb0c1968537
6. https://www.tensorflow.org/versions/r1.1/api_docs/python/tf/contrib/rnn/BasicLSTMCell
7. https://github.com/tensorflow/models
8. https://github.com/tensorflow/models/blob/master/tutorials/rnn/ptb/ptb_word_lm.py
9. https://karpathy.github.io/2015/05/21/rnn-effectiveness/
10. https://distill.pub/2016/augmented-rnns/
11. https://github.com/mokemokechicken/keras_npi/blob/master/src/npi/add/model.py
12. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
13. https://dashee87.github.io/deep%20learning/python/predicting-cryptocurrency-prices-with-deep-learning/
14. https://medium.com/@binsumi/neural-networks-and-bitcoin-d452bfd7757e
15. http://www.jakob-aungiers.com/articles/a/Multidimensional-LSTM-Networks-to-Predict-Bitcoin-Price
16. https://dashee87.github.io/data%20science/deep%20learning/python/another-keras-tutorial-for-neural-network-beginners/
17. http://www.faculty.ucr.edu/~taelee/paper/lossfunctions.pdf
18. https://github.com/jaungiers/Multidimensional-LSTM-BitCoin-Time-Series
19. https://github.com/jaungiers/Multidimensional-LSTM-BitCoin-Time-Series/blob/master/Bitcoin%20LSTM%20Prediction.ipynb
20. https://www.youtube.com/watch?v=6niqTuYFZLQ
21. https://github.com/karpathy/neuraltalk2
22. https://datascience.stackexchange.com/questions/16350/how-many-lstm-cells-should-i-use
23. https://stackoverflow.com/questions/38080035/how-to-calculate-the-number-of-parameters-of-an-lstm-network
24. https://pdfs.semanticscholar.org/696c/2fa5697f58914921ff37d69ced44ddea143f.pdf
25 http://trap.ncirl.ie/2496/1/seanmcnally.pdf
26 https://www.mathworks.com/help/econ/rolling-window-estimation-of-state-space-models.html
27 https://link.springer.com/chapter/10.1007%2F978-0-387-32348-0_9
28 Keras Library from Keras.io

9. Figures

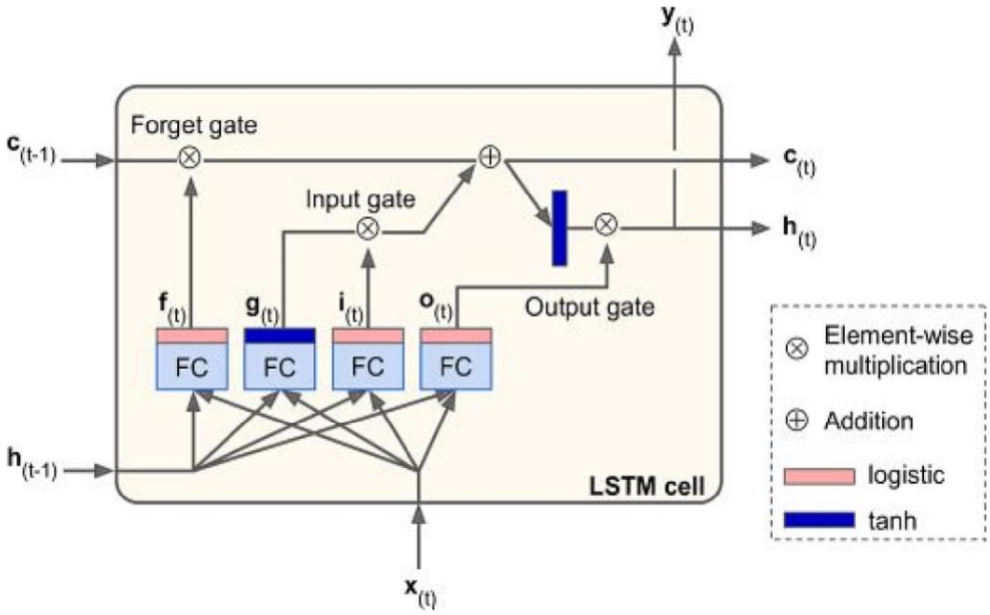

Figure 1 - LSTM cell image [1]

Figure 2 – Representation of the actual ETH closing price, split into training and testing sets

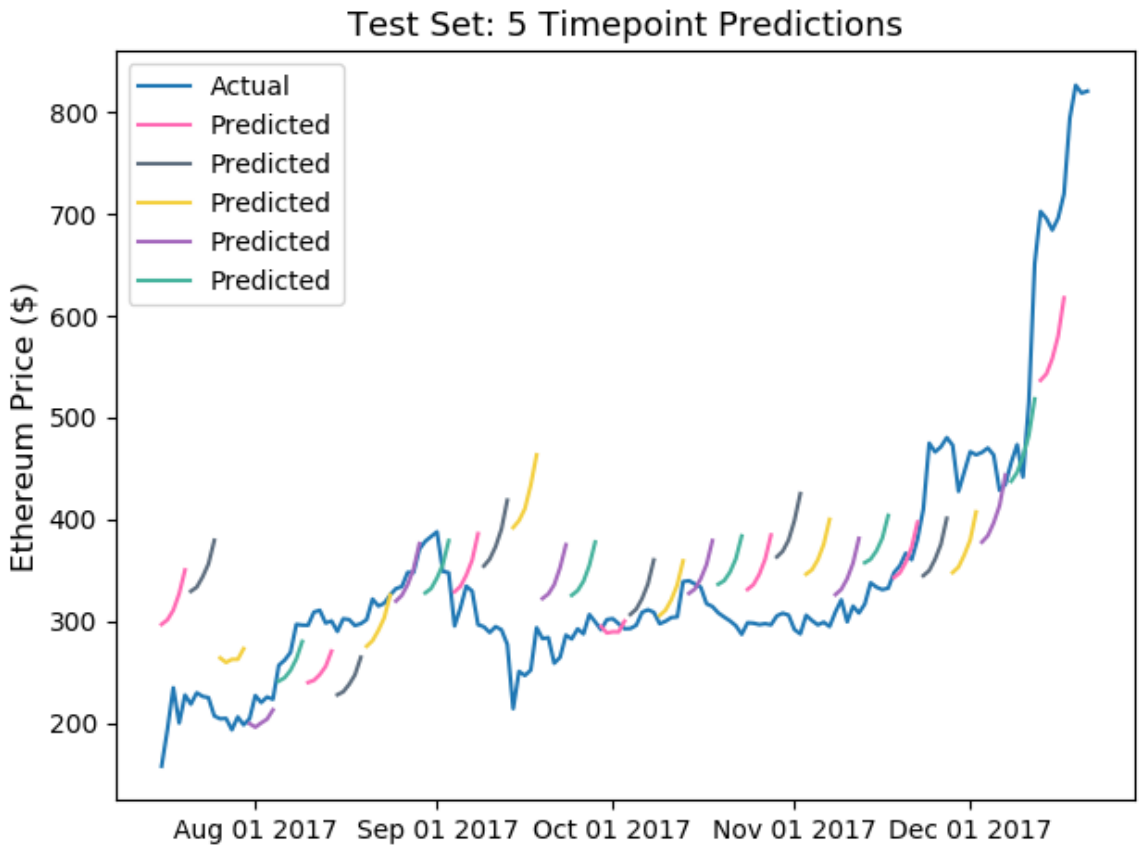

Figure 3 – This represents the best forecasting results with 2 LSTM layers, Window Size of 15, Predicting 5 days in the future with 100 neurons in each LSTM cell