Predicting Bitcoin Price Using Recurrent Neural Networks with Long Short-Term Memory

Bishoy Boktor
Department of Computer Science
Middle Tennessee State University
Murfreesboro, USA
bb5p@mtmail.mtsu.edu

Yonathan Feleke
Department of Computer Science
Middle Tennessee State University
Murfreesboro, USA
ywf2a@mtmail.mtsu.edu

Ryan Florida
Department of Computer Science
Middle Tennessee State University
Murfreesboro, USA
rjf2u@mtmail.mtsu.edu

Kyle Lutz
Department of Computer Science
Middle Tennessee State University
Murfreesboro, USA
kjl3s@mtmail.mtsu.edu

Joel Norris
Department of Computer Science
Middle Tennessee State University
Murfreesboro, USA
jrn3i@mtmail.mtsu.edu

Karla Robles
Department of Computer Science
Middle Tennessee State University
Murfreesboro, USA
knr3p@mtmail.mtsu.edu

***Abstract*—This paper shows how recurrent neural networks with long short-term memory can be used to estimate the average price and percent-change of the average price of Bitcoin a number of days in advance. We chose to use the long short-term memory model because of its proven ability to produce state-of-the-art results when dealing with sequential data. Using this architecture and Bitcoin data collected per-minute over a period of up to a little over six years, we were able to obtain rather shocking results. These results, though seemingly unbiased, still raise major concerns and require further investigation.**

***Index Terms*—Recurrent Neural Networks, Neural Networks, Long Short-Term Memory, LSTM, Bitcoin, Cryptocurrency, BTC, RNN, Crypto**

I. Introduction and Background

A. Recurrent Neural Networks

Recurrent neural networks (RNNs) are deep and powerful sequence-based models that have been used in many areas of research such as audio generation [1] [2], video [3], text generation [4], image generation [5], and a variety of others. The main drawback of using RNNs is that they are notoriously difficult to train because of the infamous "exploding" [6] and "vanishing" [7] gradient problems. However, these problems can be dealt with, in turn, by imposing norm-constraints on the gradients to keep them from exploding and then using Long Short-Term Memory (LSTM) [8] to mitigate the attenuation of the gradient. It should be noted that there are alternative models that also work around these problems [9] [10] [11], but the LSTM model is the most widely used because of its broad generality and ease of implementation.

B. Recurrent Neural Networks with Long Short-Term Memory

The LSTM model is trained on the data set one step at a time in order to predict what will most likely come next in the sequence. Once a prediction is made, it is fed back into the network in order to influence all future decisions. This is to say that the prediction at a given time step is completely dependent upon all previous time steps. The LSTM architecture has, at the time of this paper, given rise to state-of-the-art results in a variety of problems, including speech synthesis [12].

Though RNNs also have long and short-term memory, the main difference between RNNs and LSTMs is that LSTMs come with specialized "memory cells" that offer an extra memory resource [13]. These memory cells enable LSTMs to store data for an arbitrary amount of time, in theory. The memory cells of vanilla LSTM models, i.e. those with forget gates, are governed by the following set of equations:

$$\begin{aligned} a &= f(W \cdot [x_t; y_{t-1}] + b) \\ g_i &= \sigma(W_{g_i} \cdot [x_t; y_{t-1}] + b_{g_i}) \\ g_o &= \sigma(W_{g_o} \cdot [x_t; y_{t-1}] + b_{g_o}) \\ g_f &= \sigma(W_{g_f} \cdot [x_t; y_{t-1}] + b_{g_f}) \\ c_t &= g_i \odot a + g_f \odot c_{t-1} \\ y_t &= g_o \odot f(c_t) \end{aligned}$$

Here x_t and y_t are our input and output vectors, respectively, at time t, $\sigma(\cdot)$ is the familiar sigmoid activation function (with carrying capacity and gain both set to unity), and f is an activation function—the standard is $\tanh(\cdot)$. The input and output gate values, g_i and g_o, are used to regulate incoming and outgoing data, and they also take part in protecting the memory cell value. The forget gate, g_f, can be used to purge the memory cell when its data no longer needs to be stored. The more the input and output gates become "closed off", and the more the forget gate become "open", the less and less the memory cells undergo changes.

C. Bitcoin

Cryptocurrencies have become a very hot topic lately and the one that started it all, Bitcoin (BTC), is starting to really raise some eyebrows. So, what is Bitcoin? To put it simply, Bitcoin is an electronic means by which people can exchange value with one another sort of like how physical paper money can be passed between people. The main difference here though is that this vehicle of value is completely digital and can be passed between people across the world in seconds. So, one may ask, what makes this different from just wiring money to someone? Well, first, the cost is minimal as there are trivial fees to pay to the cryptocurrency miners— these fees are typically on the order of cents whereas those associated with money wires are astronomical in comparison. Second, one of the biggest draws to adopting Bitcoin is that it allows transactions to be made without a third party, i.e. a bank, being involved in those transactions. Therefore, we are now at a point where we can truly take control of our own finances and choose to do with them as we will, not what a bank will allow us to do with them. Though, not to be cliché, with this great power comes great responsibility. A complete description of how the technology behind BTC is implemented can be found in Nakamoto's paper [14] and in Narayanan's book [15].

II. Methods

A. Data Preprocessing

Since we have BTC price data read in every minute, and our goal is to predict a next-day price, we began preprocessing our data by partitioning the data into daily subsets. We then found the daily averages. [1] After finding the daily averages, we then broke our data set up into training and testing sets, in which 95% of the data was used for training and the remaining 5% was used for testing. We chose this particular split to reduce the likelihood of the testing set containing a value out of the range of the training set. Following the split, we performed the following min-max normalization on both data sets:

$$x_i^* = \frac{x_i - \min(X_{\text{train}})}{\max(X_{\text{train}})} - \min(X_{\text{train}})$$

where X_{train} is the set of training data feature vectors, and $x_i \in X$ such that $X = X_{\text{train}} \cup X_{\text{test}}$. Note the abuse of notation here: since $\dim(\underline{x}_i) = 1, \underline{x}_i \in X$, we just refer to each feature as x_i.

B. Constructing the Neural Network

For the actual neural network, we used two layers consisting of one LSTM layer and one dense layer. A 20% dropout was applied following the LSTM layer [16] during training in hopes of preserving generality. For the LSTM layer's activation function, we used the "scaled exponential linear unit" [17]. We shall denote this activation function as $\text{selu}(\cdot)$ from this point. The $\text{selu}(\cdot)$ activation function is given by:

$$\text{selu}(x) = \lambda \begin{cases} x & x > 0 \\ \alpha e^x - \alpha & x \le 0 \end{cases}.$$

We chose to use $\text{selu}(\cdot)$ because it empirically outperformed the $\tanh(\cdot)$ and *ReLU* activation functions. The choice of optimizer for this net, also due to empirical findings, was *adadelta* [18] [19] as it was observed to out-perform the standard *rmsprop*. This being a regression problem, our loss function was, of course, the mean-squared error function given by:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2.$$

A visualization of this network is displayed in Figure 1.

[1]It should be noted that some of our data sets were originally in Yen, so there were some cases in which we had to convert Yen to USD.

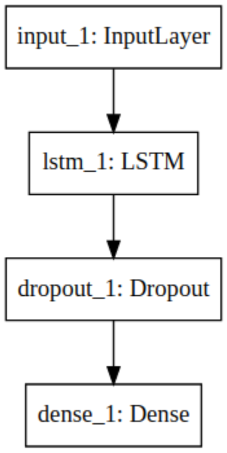

Fig. 1. Structure of our neural network.

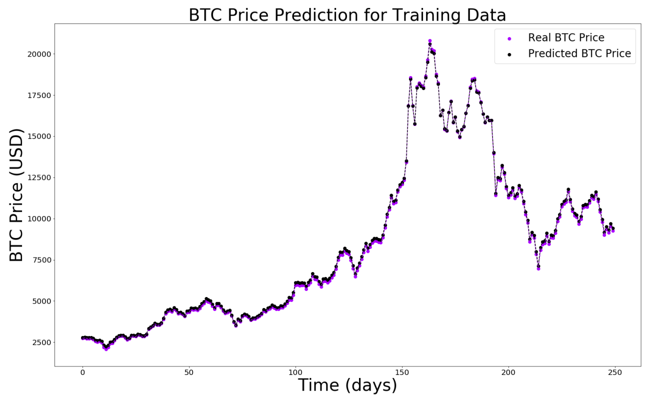

Fig. 2. Training Predicted Price After 500 Epochs.

C. Training The Network

During the training phase, we stuck with the standard $80/20$ validation split of the training set. It was observed that this model over fits relatively quickly, so we kept the number of epochs fairly low (500 in the case of our final results). See Figures 2 and 3. Though each of our four data sets had its own ideal number of epochs to avoid over fitting, 500 seemed to yield very promising results across all of them. It is also important to note that the training/testing data was not shuffled since we are trying to get the net to predict what is coming next.

D. Testing The Network

For testing, we treated each "next day" as that of the current day. This is to say that, given n training days, the predicted

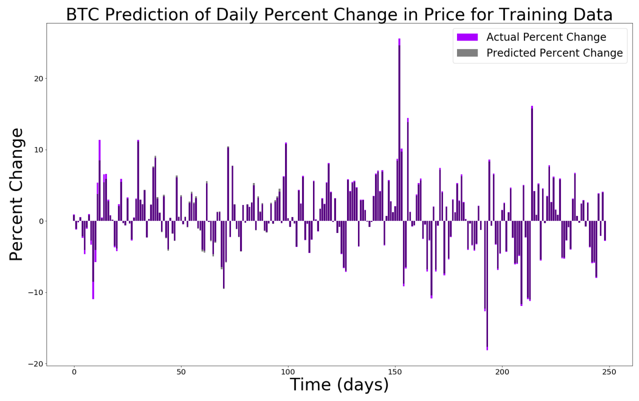

Fig. 3. Training Predicted Percent Change After 500 Epochs.

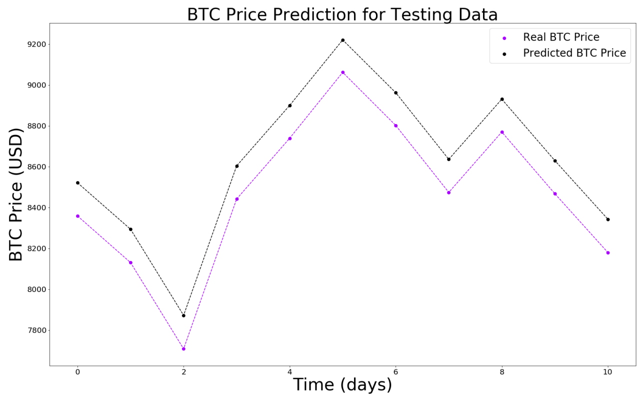

Fig. 4. Testing Predicted Price

prices for each day $n+1, n+2, n+3, \ldots$ were all treated as the predicted price based on day n and not their immediate predecessor. To clarify: we did not predict the price for day $n+1$ and then feed that result into the net for the price prediction of day $n+2$; instead, we predicted each of the day $n+i$ prices, in turn, based on that of day n.

III. Results

Our results were surprisingly, in the most unsettling of ways, amazing as in they create the illusion of our network being profoundly successful. This is to say that our network seems to be capable of predicting general pricing trends (Figure 4) and percent-price change (Figure 5) for multiple days in advance with a strikingly high degree of accuracy. As can be observed in the aforementioned figures, our model is always over-shooting the actually price by a generous amount but it is, however, able to predict whether the price will be higher or lower on a given day than it was on the previous day. What is really exciting is the accuracy our model obtains for the percent-change between the days; it seems to be able to almost perfectly predict the this, which begs the following question: are we, unknowingly, introducing a bias somewhere in this model?

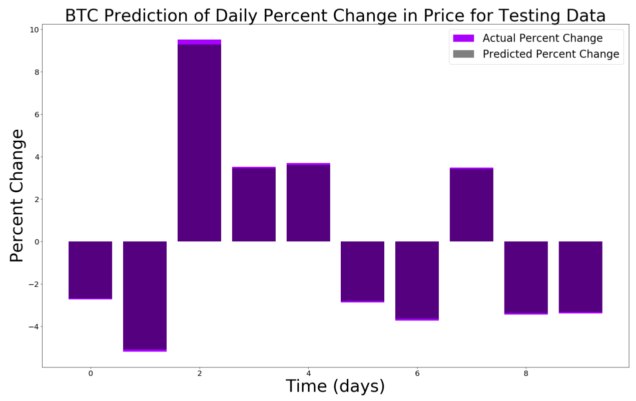

Fig. 5. Testing Predicted Percent Change

IV. Discussion and Conclusion

Due to the astounding efficacy of the LSTM model, computational scientists have been able to accomplish some truly amazing feats in the context of machine learning. Among the many problems we can attempt to solve and—at the very least—learn from, we decided to focus on the problem of predicting price trends for a revolutionary financial and technological achievement that has been gaining a lot of attention: cryptocurrenies. Since cryptocurrencies are so new and have sparked an interest in many people as of late, they seem like a natural choice for us to explore with the application of the LSTM framework in order to see if we may predict reasonable future trends in the prices of cryptocurrencies, if not the prices themselves. Bitcoin, since it was “the one that started it all”, offers us the most robust sets of pricing data since it has been around for quite some time now (circa 2009) and because it has peaked so much interest in, not only the scientific community, but also specialists from many other fields and the general public. Though our results seem to be “too good to be true”, and try as we might, we were not able to identify any obvious bias we were adding to the model. It should be noted that the results of this project should not be taken at face-value and that **we are not suggesting anyone use these results for their own investing endeavors. As always, do your proper due-diligence before investing, stay skeptical, and remember: if something seems too good to be true, then it probably is.**

References

[1] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” 2017, URL https://arxiv.org/pdf/1609.03499.pdf.

[2] N. Agarwala, Y. Inoue, and A. Sly, “Music composition using recurrent neural networks,” 2017, URL https://web.stanford.edu/class/cs224n/reports/2762076.pdf.

[3] I. Sutskever, G. Hinton, and G. Taylor, “The recurrent temporal restricted boltzmann machine,” 2008, URL https://papers.nips.cc/paper/3567-the-recurrent-temporal-restricted-boltzmann-machine.pdf.

[4] I. Sutskever, J. Martens, and G. Hinton, “Generating text with recurrent neural networks,” 2011, URL http://www.cs.utoronto.ca/ ilya/pubs/2011/LANG-RNN.pdf.

[5] A. van den Oord, N. Kalchbrenner, and K. Kavvukcuoglu, “Pixel recurrent neural networks,” 2016, URL https://arxiv.org/abs/1601.06759.

[6] G. Philipp, D. Song, and J. G. Carbonell, “The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions,” 2017, URL https://arxiv.org/pdf/1712.05577.pdf.

[7] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” 1998, URL https://arxiv.org/pdf/1503.00075.pdf.

[8] S. Hochreiter and J:urgen Schmidhuber, “Long short-term memory,” 1997, URL http://www.bioinf.jku.at/publications/older/2604.pdf.

[9] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Gated feedback recurrent neural networks,” 2015, URL http://proceedings.mlr.press/v37/chung15.pdf.

[10] R. Dey and F. M. Salem, “Gate-variants of gated recurrent unit (gru) neural networks,” 2017, URL https://arxiv.org/pdf/1701.05923.pdf.

[11] A. V. Peddada and A. L. Tsang, “Eqnmaster: Evaluating mathematical expressions with generative recurrent networks,” 2015, URL https://cs224d.stanford.edu/reports/PeddadaAmani.pdf.

[12] H. Zen, Y. Agiomyrgiannakis, N. Egberts, F. Henderson, and P. Szczepaniak, “Fast, compact, and high quality lstm-rnn based statistical parametric speech synthesizers for mobile devices,” 2016, URL https://arxiv.org/pdf/1606.06061.pdf.

[13] Z. C. Lipton and J. Berkowitz, “A critical review of recurrent neural networks for sequence learning,” 2015, URL https://arxiv.org/pdf/1506.00019.pdf.

[14] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008, URL https://bitcoin.org/bitcoin.pdf.

[15] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, “Bitcoin and cryptocurrency technologies a comprehensive introduction,” 2015.

[16] T. Bluche, C. Kermorvant, and J. Louradour, “Where to apply dropout in recurrent neural networks for handwriting recognition?” 2015, URL https://pdfs.semanticscholar.org/3061/db5aab0b3f6070ea0f19f8e76470e44aefa5.pdf.

[17] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, “Self-normalizing neural networks,” 2017, URL https://arxiv.org/pdf/1706.02515.pdf.

[18] M. D. Zeiler, “Adadelta: An adaptive learning rate method,” 2012, URL https://arxiv.org/pdf/1212.5701.pdf.

[19] S. Ruder, “An overview of gradient descent optimization algorithms,” 2017, URL https://arxiv.org/pdf/1609.04747.pdf.