**Predicting Bitcoin Price with Long Short-Term Memory (LSTM) Neural Networks: A Comparative Study of Traditional and Deep Learning Approaches**

Christopher Ferguson and Stefan Maric

***Abstract* – Forecasting time series data is a topic of ongoing interest in finance and economics. Traditional ways of forecasting time series data include methods such as Autoregressive Models (AR), Moving Average Models (MA), and Autoregressive Integrated Moving Average Models (ARIMA). With improvements in technology, the methods used for forecasting time series data have expanded and now include deep learning algorithms. Deep learning comprises a variety of neural network architectures, with Long Short-Term Memory (LSTM) networks being a popular type of recurrent neural network that excels at capturing long-term dependencies in sequential data. This study aims to answer the question of whether the deep learning methods of forecasting, specifically Long Short-Term Memory (LSTM) Networks, outperform older, more traditional ways such as Autoregressive Integrated Moving Average (ARIMA) Models. The results from this paper indicate that ARIMA models outperform LSTM models when applied to Bitcoin prices. The forecasting errors obtained from ARIMA were approximately 7% lower than the forecasting errors associated with LSTM, giving reason to believe that LSTM is not as appropriate as previously thought for predicting financial time series.**

**Introduction**

Forecasting time series of financial data has long been a difficult task due to several unknowns, such as market volatility and changing economic conditions. Forecasting time series data for cryptocurrency made that task even more difficult. Cryptocurrency is an alternative medium of exchange that is growing rapidly in popularity. Each cryptocurrency has its own unique cryptographic foundation that enables secure transactions in a decentralized network. Trading strategies for cryptocurrencies are difficult to implement due to their continuous trading periods and high volatility. The largest and most prominent cryptocurrency is Bitcoin, which is the focus of this paper. The primary objective of this paper is to identify the forecasting method that provides the lowest out-of-sample prediction error rates for Bitcoin prices.

One of the most popular methods used in forecasting time series data is the Autoregressive Integrated Moving Average (ARIMA) model. This model combines both Autoregressive (AR) and Moving Average (MA) components to model a time series. The AR component of an ARIMA model is considered to be a long-term memory component due to its ability to be recursively written as lagged versions of itself beyond the order p. The MA component of an ARIMA model is considered to be a short-term memory component due to each error term being uncorrelated to errors from errors, forwards or backwards in time. In other words, the error terms are white noise processes that are identically and independently distributed. By combining both the AR and MA components we have effectively created a long short-term memory model which is why this paper will compare such a model to an LSTM neural network. ARIMA has been shown to outperform the accuracy of other traditional methods of forecasting time series data. [1]

Increasing computing power has made deep learning methods of forecasting increasingly popular. This study will focus on one specific deep learning method, Long Short-Term Memory (LSTM) Neural Networks. These are recurrent neural networks that use memory cells designed to retain information over time while forgetting irrelevant information. Each of these memory cells has three gates: input gates, output gates, and forget gates. The input gate controls the amount of information that enters the memory cell. The output gate controls the amount of information that is passed on to the next cell. And the forget gate controls the amount of irrelevant information that is forgotten or thrown away.

An in-depth explanation of the processes and implementations of both ARIMA and LSTM models will be examined in this paper, as well as a comparison of the forecasting accuracy of ARIMA and LSTM regarding Bitcoin prices. Since this paper seeks to replicate the results of previous works to determine whether the difference in prediction power for financial time series is over-stated, it would make sense to use the same data. However, due to a change in pricing, the data used in Siami-Naimini et al and Yiqing Hua is not available for free. This limitation led us to use just a year’s worth of daily Bitcoin pricing data, because this was available to us at our budget. Our results seem to disprove both papers we reference. Not only was ARIMA more accurate than LSTM in a short-term time horizon (1 year), it also handily outperformed LSTM in out-of-sample predictions using the same monthly NASDAQ index data used in Siami-Namini et al. This result was unexpected given the wisdom from Yiqing Hua that ARIMA precision falls dramatically. In the proceeding sections we discuss related work, data and methodology, accuracy metrics, in-depth explanations of the processes of both models, analysis of results, and a final discussion that will include our own critique of our work and future improvements.

**Related Work**

Prior attempts have been made to forecast a financial time series using newly developed deep learning methods. Siami-Namini et al [2] found an 85% improvement, on average, in prediction when using LSTM compared to ARIMA models. They credited the results to the iterative optimization algorithms used in deep learning. [2] Although, their results were based on changes in the monthly price of multiple stock market indices such as the NASDAQ and not cryptocurrencies such as Bitcoin. However, the authors noted that their ARIMA model was not optimized in any way, which can lead to inaccurate results. Yiqing Hua [1] found that after training LSTM it became more efficient and precise at predicting Bitcoin price fluctuations than ARIMA. Although Hua did conclude that in the short term ARIMA is still efficient but as the time horizon increases precision falls dramatically. In this paper we plan to expand on the prior research done by both Siami-Namini et al and Yiqing Hua with the hopes of improving their forecast accuracy as well as forecasting Bitcoin price and not stock market indices as done by Siami-Namini et al.

**DATA and METHODS**

**Data Collection**

The dataset is comprised of the daily price of Bitcoin from Jan 2022 to Jan 2023, using the closing price for that day, which is extracted from Yahoo Finance. The prices collected are measured in USD. To effectively train any machine learning algorithm, the data must be split into a training set and a holding set, also known as a test set. However, determining the optimal allocation of data allocated to each set is rather ad hoc, as there is no literature to suggest an absolute optimal allocation. Excluding the first observation, this paper uses the first 243 observations (67% of the data) as a training set, while the last 121 (33%) observations are used as a testing set. It is worth noting that the first observation is excluded due to the process of differencing the data, which will be further elaborated upon in the upcoming subsection.

**Data Transformation**

In order to predict time series data using ARMA models, the establishment of data stationarity is a fundamental prerequisite for precise prediction and analysis. Stationarity in time series analysis is characterized by the following properties: constant mean, constant variance, and constant autocovariance. If the time series were not to have the following properties, the underlying processes that determined one period would not lend any information to predict or analyze the next period.

To this end, this paper adopts a standard approach in achieving stationarity by employing the natural logarithm transformation on Bitcoin price, followed by a first-difference operation on the transformed series. This methodology ensures that the data is appropriately modified to meet the requisite stationarity criteria, thus laying the foundation for robust predictive modeling. Financial asset price time series that receive the described transformation are referred to as the log returns of Y. Therefore, in the context of Bitcoin, this paper will refer to its price data as the log returns of Bitcoin.

To test if the transformed time series is stationary, the Augmented Dickey-Fuller (ADF) Test is performed on the transformed series. According to the results of ADF test, the absolute value of the test-statistic is greater than the absolute value of the critical value at 5%, therefore the transformed dataset has no unit root and exhibits stationary characteristics.

Table 1: Augmented Dickey-Fuller Test for Log-Returns of Bitcoin Price

|  |  |
| --- | --- |
| Test - statistic | 5% Critical value |
| -13.0914 | -1.95 |
| -13.2046 | -2.87 |
| -13.1873 | -3.42 |
|  |  |

A picture containing text, antenna

Description automatically generated

Figure 2: Log Returns of Bitcoin

**Assessment Metric**

This paper uses the Root Mean Square Error (RMSE) to measure prediction accuracy. RMSE measures the difference between actual values and predicted values.

Where N is the total number of observations, *,* is the actual value of the time series at time step *i*, and is the predicted value from the model at time step *i*. This measure was specifically chosen due to its ability to penalize large prediction errors, thereby providing a more accurate assessment of the model's performance. Additionally, it offers the advantage of providing accuracy measures in the same units as the original data, making it easier to interpret the results.

**ARMA/ARIMA Models**

Autoregressive Moving Average Models (ARMA) were created to explain the processes that determine a time series based on its past behavior. ARMA models are a combination of two simpler models, AR (Autoregressive) and MA (Moving Average). An AR model describes the relationship between an observation and a number of lagged observations, while an MA model describes the relationship between an observation and a number of lagged forecast errors or shocks. ARMA models combine these two models by incorporating both lagged observations and forecast errors in a linear regression model. The model can be expressed as:

Autoregressive Integrated Moving Average Models (ARIMA) are an extension of the ARMA model, that accounts for the tendency for stochastic processes to be non-stationary by differencing the time series to transform the time series into a stationary stochastic process. The transformed dataset can then be estimated via an ARMA model [3].

The AR(*p*) model, where p represents the number of lagged versions of itself, can be written as a linear process:

Where *Yt* is stationary time series, c is a constant, represents the coefficients of the lagged observations from the time series, and is the white noise error term with a mean of zero and a variance of . Upon inspection, the AR(1) model displays some interesting properties:

*Yt-1* is also a function of itself from one period ago:

*Yt* can be recursively written as:

This can process can be repeated infinitely to show that *Yt* depends on all past observations of itself. Since this process of recursion goes back all the way to the beginning of the series, an AR(p) of any p, can be referred to as a *long-term memory* model.

Like the AR(*p*) model, the MA(q) model assumes that the error terms, , are a white noise process that are independently, and identically distributed, or “i.i.d” with a mean of 0 and a variance of . If the error terms are i.i.d, then it follows that the error terms from any period, *t*, are uncorrelated with the error terms from a period forward or backwards in time. This can be shown from an MA(1) model mathematically below:

As can be seen, the error terms, , for the MA(1) model only depend on the from last period, and the from today. As time moves forward, the from previous periods become uncorrelated with *Yt+h*. As the effect of from past periods completely drops to zero, it becomes apparent that the MA(q) model can be characterized as a *short-term memory* model. It is now evident that ARMA/ARIMA models can be seen as a type of long short-term memory model. This finding positions them as a prime candidate for comparison with LSTM neural networks, which are specifically designed to capture and incorporate long-term dependencies in time series data.

As a result of the BIC iteration (Table 1), the ARIMA model in this paper is an ARIMA(1, 0, 1).

**LSTM**

LSTM (Long Short-Term Memory) is a deep learning method that is a form of RNN (Recurrent Neural Network). RNN algorithms were developed to process sequential data, such as time series, and natural language, where the current state is dependent on the current input, as well as past inputs and states. Unlike RNN, LSTM is not susceptible to the exploding/vanishing gradient problem and distinguishes between long-term and short-term dependencies in a time series.

LSTM uses memory cells to control the flow of information that is used to make predictions. These memory cells use a series of gates to determine which information will be used to update the model’s decision-making process, and which information will be forgotten. The LSTM memory cell is composed of three information filtering gates, the forget gate, input gate, and output gate.

Diagram

Description automatically generated

Figure 5: LSTM memory cell

The forget gate of the memory cell determines how much of the previous cell state (*Ct-1*) information is discarded from the model. The gate takes the hidden state from the previous cell (*ht-1*) and value of the input at the current time step (*Xt*) as inputs to this process. The sigmoid function will output a value between 0 and 1. An output of 0 indicates that all information from the previous cell state will be discarded. An output of 1 indicates that all information from the previous cell state will be kept as relevant decision-making information.

*ft = σ(Wx,f xt + Wh,f ht-1* + *bf)*

The input gate of the memory cell determines how much of the information from *ht-1* and *Xt* will be added as memory to the cell state. This gate requires three computations 1) Potential memory to be added (*A*); 2) Percentage of potential memory to add to the cell state (*B)*; 3) Add the new information to the cell state (*Ct)*.

*A = tanh(Wx,A xt + Wh,A ht-1* + *bA)*

*B* *= σ(Wx,B xt + Wh,B ht-1* + *bB)*

*Ct = ft \* Ct-1 + (A\*B)*

The output gate determines how much of the new cell state, *Ct,* will become the new hidden state, *ht*. This allows the cell to selectively output relevant information while ignoring irrelevant or noisy information. This gate requires three computations 1) Potential hidden state (D); 2) Percentage of potential memory to be sent to hidden state (E); 3) Add the new information to the hidden state, *ht*.

D = *tanh(Ct)*

*E = σ(Wx,E xt + Wh,E ht-1* + *bE)*

*ht =D\*E*

In this paper, the LSTM model uses a lookback period of 10, which means that it takes the previous 10 time steps as input to predict the next value. The hidden size of the model is set to 60, which determines the number of neurons in the LSTM layer. The higher the number of neurons, the more complex the model becomes, which can improve its accuracy, but may also increase the risk of overfitting.

After processing the input sequence, the LSTM network will take the final hidden state and convert it into a linear value using a linear layer. To prevent overfitting, this model only uses the final time step in the sequence instead of the entire input sequence. By the final time step, the final hidden state will have incorporated information from the entire input sequence and should contain all the relevant information necessary for the next step prediction.

The model is trained for 1400 epochs, which represents the number of times the entire training dataset is processed during training. The use of a high number of epochs is common in deep learning models, as it allows the model to learn more complex patterns in the data. However, using too many epochs can lead to overfitting, which reduces the model's ability to generalize to new data.

**Results**

Table 2 presents the results obtained from both ARIMA and LSTM models. The RMSE values obtained from training the ARIMA and LSTM models were equivalent to 0.036 and 0.006, respectively. However, the testing data yielded RMSE values equivalent to 0.029 and 0.037 for ARIMA and LSTM models, respectively. The testing results show that, on average, ARIMA models lead to a 0.8% lower prediction error in log returns for Bitcoin. These findings contrast with the existing literature, which overwhelmingly favors LSTM models, often showing 84%-87% reduction in error rates [1].

|  |  |  |
| --- | --- | --- |
| **Model** | **Train RMSE** | **Test RMSE** |
| ARIMA | 0.0355 | 0.0292 |
| LSTM | 0.0056 | 0.0365 |

Table 2: The RMSEs of ARIMA and LSTM models

**Diagram

Description automatically generated with low confidence**

Figure 6: ARIMA Testing Predictions**Chart, line chart

Description automatically generated**

Figure 7: LSTM Testing Predictions

**Discussion**

The remarkable performance observed through deep learning-based approaches to the prediction problem is largely an illusion. Upon inspection, the ARIMA model optimal for the data in Siami-Namini et al [1], the fitting algorithm suggested an ARIMA(1, 1, 1) which supports the idea that the results in this paper are inaccurate overstate the error from ARIMA. Although deep learning models have shown impressive results in various applications, including natural language processing, computer vision, and speech recognition, their effectiveness in time series prediction tasks is questionable. The results from this paper suggest that traditional methods of forecasting outperform one instance of deep learning, LSTM, but not by a large margin.

There are numerous reasons explanations for the increased prediction error from the LSTM model in this paper. According to the training data results, the LSTM model outperforms the ARIMA models by a substantial margin of 3%, which indicates a significant difference in performance. This may indicate that the LSTM over-fit the training data and therefore lost its out-of-sample predictive power. The optimization methods used in this paper for the LSTM models hyperparameters were imperfect. Literature suggests various methods, such as grid search and dropout regularization, to improve model performance. Grid search involves systematically testing different combinations of hyperparameters to identify the combination that yields the best performance [4]. Dropout regularization involves randomly dropping out (i.e., deactivating) some of the model's neurons during training to prevent overfitting [5]. Additionally, the computer power used for optimization could be limiting the model's potential. The computer power available to research institutions greatly surpasses the capabilities of the machine we used. Future work could involve applying a more efficient optimization method, followed by regularization, to achieve a model that is less susceptible to overfitting.

**References**

[1] Hua, Yiqing. *Bitcoin Price Prediction Using ARIMA and LSTM* . 11 Dec. 2020, https://www.researchgate.net/publication/347611261\_Bitcoin\_price\_prediction\_using\_ARIMA\_and\_LSTM.

[2] S. Siami-Namini, N. Tavakoli and A. Siami Namin, "A Comparison of ARIMA and LSTM in Forecasting Time Series," 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, USA, 2018, pp. 1394-1401, doi: 10.1109/ICMLA.2018.00227.

[3] Bhatt, V. (2019). Applied Time Series Analysis.

[4] Brownlee, J. (2018). How to grid search hyperparameters for deep learning models in Python With Keras. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

[5] Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. arXiv preprint arXiv:1409.2329. Retrieved from <https://arxiv.org/abs/1409.2329>