**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

James Madison University

***Abstract* – Forecasting time series data is a topic of ongoing interest in finance and economics. Traditional forecasting methods such as Autoregressive Models (AR), Moving Average Models (MA), and Autoregressive Integrated Moving Average Models (ARIMA) have long been used for this purpose. Advances in technology have introduced deep learning techniques, including Long Short-Term Memory (LSTM) networks, a type of recurrent neural network adept at capturing long-term dependencies in sequential data. This study evaluates whether LSTM networks outperform traditional methods like ARIMA for forecasting Bitcoin prices. Results indicate that ARIMA models outperform LSTM models when applied to Bitcoin prices. The forecasting errors obtained from ARIMA were approximately 3% lower than the forecasting errors associated with LSTM, giving reason to believe that LSTM is not as superior as previously thought for forecasting financial time series.**

**Introduction**

Forecasting financial time series has long been challenging due to factors such as high volatility, non-stationarity, and complex interactions among economic variables. 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. Cryptocurrencies are decentralized digital assets that utilize blockchain technology to enable secure and transparent transactions without the need for centralized authorities. 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 compare traditional time series models, such as ARIMA, with deep learning approaches, such as Long Short-Term Memory (LSTM) networks, to identify which method provides the lowest out-of-sample forecast error for Bitcoin prices.

The Autoregressive Integrated Moving Average (ARIMA) model is one of the most widely used methods for forecasting time series data. This model combines two key components: the Autoregressive (AR) component, which captures the relationship between a time series and its past values, and the Moving Average (MA) component, which models the relationship between the time series and past forecast errors. Together, these components enable ARIMA to effectively capture both long-term and short-term dependencies in time series data [4]. By combining both the AR and MA components, ARIMA captures long-term dependencies through its Autoregressive (AR) terms and short-term dependencies through its Moving Average (MA) terms. Given these properties, we deemed ARIMA a suitable candidate for comparison with LSTM networks.

Advances in computing power have contributed to the growing popularity of deep learning methods for time series forecasting. This study focuses on Long Short-Term Memory (LSTM) neural networks, a type of recurrent neural network (RNN) designed to address the limitations of traditional RNNs, such as the vanishing gradient problem. LSTMs achieve this by using specialized memory cells that can retain relevant information over extended time periods while discarding irrelevant data.

Each LSTM memory cell contains three key components known as gates: the input gate, forget gate, and output gate. The input gate determines which new information should be added to the cell state. The forget gate decides how much of the existing information in the cell state should be retained or discarded. Finally, the output gate regulates which information is passed to the next memory cell. This architecture enables LSTMs to effectively model complex temporal dependencies in sequential data, making them well-suited for forecasting tasks.

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 for Bitcoin prices. Given this paper aims to replicate the findings of previous studies to assess whether the differences in prediction power for financial time series have been overstated, it would have been ideal to use the same datasets. However, much of the data utilized in the studies by Siami-Naimini et al. and Yiqing Hua is no longer freely available due to changes in pricing policies. Consequently, we utilized the last 10,000 1-minute closing price observations from Bitfinex, as this dataset was accessible within our budgetary constraints.

Our results challenge the conclusions of Siami-Naimini et al. and Yiqing Hua, suggesting that the predictive power of LSTM models for financial time series may be overstated compared to traditional methods. ARIMA not only demonstrated greater accuracy than LSTM over short-term time horizons but also outperformed LSTM in out-of-sample predictions using the same monthly NASDAQ index data analyzed in Siami-Naimini et al. Potential reasons for these discrepancies and detailed numerical comparisons are provided in the related work and results sections.

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**

Previous attempts have been made to forecast a financial time series using newly developed deep learning methods. Siami-Namini et al [2] found an 84-87% average reduction in forecasting error when using LSTM compared to ARIMA models. They attributed the results to the iterative optimization algorithms used in deep learning. While, 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, our exploratory findings found this stark difference in model performance could not be reproduced. The authors acknowledged that their ARIMA model was not optimized, which could lead to inaccurate results and introduce bias into their findings. This is likely why our results differ.

In a separate study, Yiqing Hua [1] concluded that LSTMs were more efficient and precise at predicting Bitcoin price fluctuations compared to ARIMA models. However, the author did not provide specific assessment metrics for the ARIMA results, making it difficult to verify or compare the performance of the models. Without appropriate assessment metrics, the conclusions drawn from this study lack sufficient empirical support.

Our methodology builds upon these works by optimizing ARIMA models using the Box-Jenkins method to accurately estimate ARIMA(*p, d, q*) parameters, thereby reducing bias in the findings. Additionally, we provide clear and consistent assessment metrics to facilitate a robust comparison of results.

**DATA and METHODS**

**Data Collection**

The dataset used in this study comprises of the last 10,000 observations of the 1-minute closing price of Bitcoin, extracted from Bitfinex via their API. All prices are measured in USD. To effectively train a machine learning algorithm, and ARIMA, the data must be divided into a training set and a test set (also known as a holdout set). For machine learning algorithms, it is also recommended to create a second holdout set, known as a validation set, to evaluate the model's ability to generalize to unseen data. The validation set is used during the hyperparameter tuning process to select configurations that enhance the model’s performance on new data while preventing data leakage.

However, determining the optimal allocation between these sets remains an ad hoc process, as the literature does not provide definitive guidance on an ideal split. In this study, a 70-30 split was employed, with the first 70% of observations used for training and the remaining 30% reserved for testing. This split was used in Siami-Namini et al and is the most commonly used split in research.

A graph showing a line graph

Description automatically generated with medium confidence

Figure 1: 1min Closing Price of Bitcoin

**Data Transformation**

To predict time series data using ARMA models, establishing data stationarity is a fundamental prerequisite for accurate prediction and analysis. Stationarity in time series analysis is defined by three key properties: a constant mean, a constant variance, and a constant autocovariance. If a time series does not exhibit these properties, applying an ARIMA model is likely to result in a poor fit and unreliable forecasts.

To this end, this paper adopts a standard approach in achieving stationarity by employing 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.

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 Bitcoin Price

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

**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.

There are multiple ways to find the optimal ARIMA order. We used the Box-Jenkins method for estimating ARIMA(*p, d, q*) [3]. The optimal order of integration, *d*, was chosen by the ADF test results from Table 1. The optimal order of *p* and *q* were chosen by iteratively estimating combinations of *p* and *q*, and choosing the model with the least BIC. This method suggests an ARIMA(1, 1, 0).

**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 2: 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*

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.

In this study, the LSTM model's hyperparameters—including the lookback period, hidden layer size, and number of hidden layers—were optimized using Optuna’s Bayesian Optimization algorithm. This method systematically explores the hyperparameter space to identify configurations that enhance model performance. Through this process, the optimal parameters were determined to be a lookback period of 26, a hidden size of 138, and a single hidden layer.

The lookback period of 26 means that the model uses the previous 26 time steps as input to predict the next value. The hidden size, which specifies the number of neurons in the LSTM layer, was set to 138. A larger hidden size increases the model’s capacity to learn complex patterns, but it also heightens the risk of overfitting.

The model was trained for 150 epochs, with each epoch representing a complete pass through the entire training dataset. A high number of epochs is commonly used in deep learning to enable the model to learn complex patterns in the data. However, excessive training epochs can lead to overfitting, diminishing the model's ability to generalize to unseen data. In this study, 150 epochs were determined to be sufficient, as additional training began to show signs of overfitting.

**Results**

Table 2 summarizes the results obtained from the ARIMA and LSTM models. The RMSE values from the training data were 45.70 for the ARIMA model and 44.92 for the LSTM model. However, when applied to the testing data, the RMSE values increased to 70.94 for the ARIMA model and 73.07 for the LSTM model. These results indicate that, on average, the ARIMA model achieves a 3% lower forecasting error compared to the LSTM model when predicting Bitcoin prices.

These findings contrast with the prevailing literature, which often reports significantly better performance for LSTM models. For example, Siami-Naimini et al. [1] reported an 84%-87% reduction in error rates when using LSTM models compared to traditional approaches like ARIMA. This discrepancy highlights the importance of dataset characteristics, model optimization, and evaluation criteria in influencing forecasting outcomes.

|  |  |  |
| --- | --- | --- |
| **Model** | **Train RMSE** | **Test RMSE** |
| ARIMA | 45.70 | 70.94 |
| LSTM | 44.92 | 73.07 |

Table 2: The RMSEs of ARIMA and LSTM models

**! insert here**

Figure 3: ARIMA Testing Predictions

**! insert here**

Figure 4: 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(0, 1, 1) which supports the idea that the results in this paper 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] Diebold, F. X. (2007). Elements of forecasting (4th ed.). Mason, OH: Thomson South-Western.