Neuro-fuzzy Project

November 29th - February 16th 2025

Athanasios Kotoulas 03389

Pavlos Chatzis 03546

Contents

1 Introduction

Background on Bitcoin and the Importance of Price Prediction

Cryptocurrencies have gained significant attention over the past decade, revolutionizing the way digital transactions are conducted. Among these, Bitcoin stands out as one of the oldest and most widely recognized cryptocurrencies in the global financial market. Introduced in 2009 by an anonymous entity known as Satoshi Nakamoto, Bitcoin operates on a decentralized peer-to-peer network, allowing users to perform transactions without intermediaries. This innovation has led to its adoption as both a digital currency and an investment asset.

Bitcoin's popularity has grown alongside increasing speculation, making accurate price prediction a critical concern for investors and financial analysts. Successful forecasting of Bitcoin prices can inform trading strategies, mitigate financial risks, and provide insights into market behavior. Given Bitcoin's widespread influence, a robust predictive model holds value not only for individual investors but also for broader economic and technological research.

Challenges of Predicting Volatile Assets like Bitcoin

Predicting the price of Bitcoin presents significant challenges due to its inherent volatility. Unlike traditional financial assets, Bitcoin is influenced by a variety of factors including market sentiment, regulatory changes, technological advancements, and macroeconomic conditions. These factors contribute to rapid and often unpredictable price fluctuations, making it difficult to model future trends accurately.

Moreover, Bitcoin's decentralized nature means it operates independently of traditional financial institutions, reducing the availability of standardized data sources. The presence of sudden price spikes and crashes further complicates the development of accurate forecasting models. Addressing these challenges requires sophisticated computational methods capable of capturing non-linear dependencies and patterns in the data.

Overview of Traditional Models (e.g., ARIMA) Versus Advanced Neural Networks

Historically, statistical methods such as the AutoRegressive Integrated Moving Average (ARIMA) model have been widely used for time-series forecasting. ARIMA models are effective for capturing linear patterns and providing interpretable results. However, they struggle to capture the complex, non-linear relationships inherent in cryptocurrency price movements.

In contrast, advanced neural network architectures—particularly Long Short-Term Memory (LSTM) networks—have demonstrated superior performance in handling sequential data. LSTM models are a specialized form of recurrent neural networks (RNNs) capable of retaining long-term dependencies, making them well-suited for modeling time-series data. Recent research comparing various machine learning algorithms for cryptocurrency prediction has highlighted the effectiveness of multi-layer LSTM architectures in improving prediction accuracy. Based on this research, we selected a three-layer LSTM model as the foundation for our Bitcoin price prediction approach.

Objectives of the Project and the Scope of the Analysis

This project aims to develop a neural network-based methodology to predict the daily closing price of Bitcoin. Using a dataset containing minute-by-minute Bitcoin prices from January 1, 2021, to March 1, 2022, the primary objective is to design, train, and evaluate a three-layer LSTM model for accurate price prediction. Specifically, we aim to:

Implement a three-layer LSTM neural network for Bitcoin price prediction, informed by comparative research on machine learning algorithms.

Compare the performance of the LSTM model against a baseline ARIMA model.

Evaluate model accuracy by predicting Bitcoin prices for the last ten days of February 2022.

Analyze the computational efficiency of the model in terms of training and inference times.

Ensure modular code design to facilitate future adaptability and experimentation with different data formats.

Through this project, we seek to demonstrate the advantages of LSTM networks in capturing complex temporal patterns and provide a comprehensive performance comparison with traditional statistical methods.

2 Research

In order to create the network we sought, we first had to research and find the one that best fits our need to forecast the highly volatile price of Bitcoin. Luckily, since many others have tried to become rich before us, there is an abundance of projects and research available.

Our main source of information was the research paper [1] which was found by using keyword search and proved to be a really important part of understanding what and how we needed to move forward.

For the practical part of our project and the creation of our program we took inspiration from github projects. We looked at many different approaches from a variety of people but we settled to the one created by user *Ali619*. This project was well created and easily understandable, something important to us since we do not have an extensive background on python coding.

3 Methodology

After researching the subject and influenced by the results of [1] we chose an LTSM architecture for our network which seemed to have the best results for predicting prices as volatile as the Bitcoin prices.

Network Architecture

Our LSTM network is quite simple. It consists of only 1 hidden layer, and the input and output layers. The first two layers are LTSM layers with 50 cells each that use the ReLU activation function. The last layer is a fully connected (dense) layer that produces the result. This was the proposed architecture from [1].

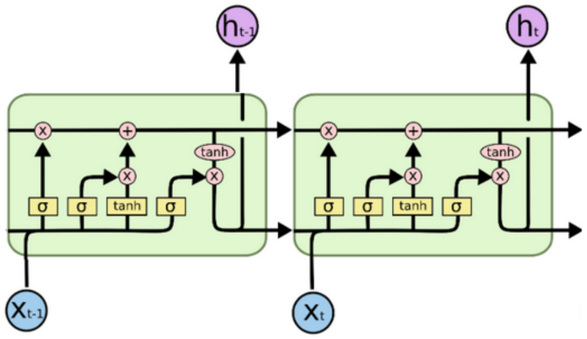


Figure 1: Two LSTM cells

The simplicity and performance of this architecture are what drove us to use it for our project since we could get good results with a relatively small need for training.

Training

Training the model took the bulk of our time on the project since we tested a lot of different configurations in order to find which parameters we needed to use to get the best results. Let's break down what these parameters are and how each affects the training of our model.

Parameters

Epoch is the simplest of the parameters but it can greatly affect the amount of time needed to train the model. Epoch shows the number of training cycles we want to perform based on our data set. The more iterations of training we perform the greater the accuracy of the model but more time is needed to complete the training, so there is a point of diminishing return. We found that our model could achieve really good on a relatively small amount of training cycles. For each of the models we tested the number of epochs was different and it depended on the other parameters.

Sequence Length is the most important parameter for the accuracy of out model. The sequence length refers to the amount of data the model looks at before making a prediction. The smaller the sequence length the less accurate the model is on long term forecasting but the better on short time predictions and easier to train. As the parameter gets larger we get better results for long term forecasting but the model needs a much bigger memory space and a lot more time to train. Choosing the right number for the sequence length is hard since we want to create an accurate model but we are very limited by the hardware at our disposal.

A diagram of the architecture:

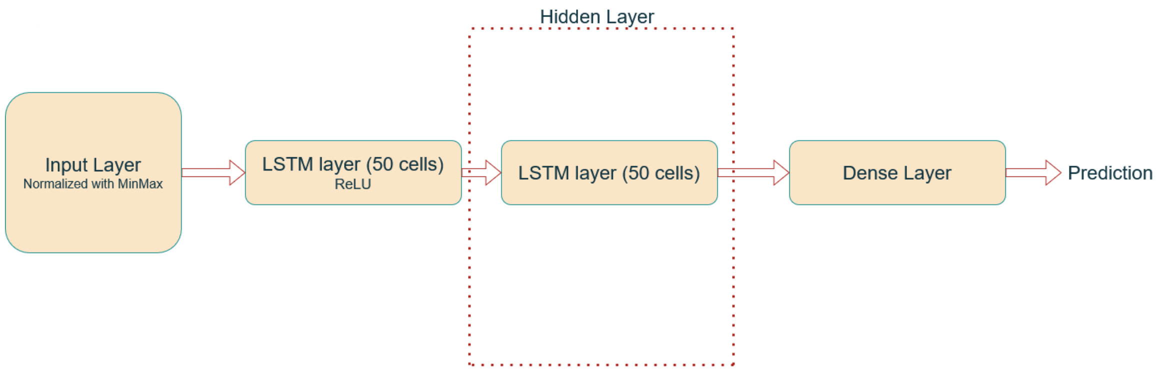


Figure 2: Neural Network Architecture Diagram

Batch size is important for the training of our model and refers to the number of samples processed in a forward and backward pass while training our model. Batch size can affect both the accuracy and the time needed to train the model. Larger batch sizes allow for quicker training but less accuracy and vise versa.

Training metrics

As our training metric we chose Mean Square Error (MSE) a widely used metric that is simple to calculate and creates accurate models. We did not want to overly complicate the metrics and thus complicating the network training phase. In [1] there where other metrics used more suited for price predictions but we chose not to use them.

Validation

Validation is an important phase while creating prediction models. The purpose of validation is to see how good the trained model is at recognizing patterns and making predictions. In order to validate the model historic data is used and the model makes multiple predictions on the already known data. Then the predictions are compared with the actual prices and by using a metric, in our model MSE was used once again, we can see if the model has been well trained. This is different from forecasting, where predictions are made based on the prices the model has predicted. More on forecasting later on.

Forecasting

This is the true goal of our model. Forecasting is when the model makes predictions for future prices without knowledge of those prices. Every prediction is made with the previous prediction as a fact and not the actual value. Forecasting is really hard to achieve especially in the long term since error is compounded, making the model really inaccurate. Forecasting is affected greatly by the *sequence length*. For more accurate predictions we need the sequence length to be quite large meaning that our model has more data to make the prediction on and is able to find trends and patterns. In our model we are limited by our data type and the hardware we posses.

4 Model

Here we will analyze the practical part of the project. We will see how the network was created, how the data was processed, what libraries were used and more technical details.

Input data wrapper

Before creating the model the first thing we needed to do was to create a wrapper that handles the data from our .csv file. We need this wrapper so our model is scalable and we have an easy way of expanding it. From the .csv file we were provided we need to extract the closing prices of bitcoin and create an array in order to have an input for our model. Using the **pandas** python library we can easily extract the closing prices and create a vertical array. Afterwards, we normalize the prices between he values [0,1] and manipulate the shape of this array based on the *sequence length* and create the 3D input that the LTSM model requires. Also, it is important to note that all data is of type float to increase the accuracy of the model and avoid overflowing.

Creating the model

To create the model we used the *Keras* python library which allowed us to easily create the model, compile, train and make predictions with it. Additionally, we can easily get the final weight and performance metric values. To compile the model we chose the *Adam* optimizer like we had been told.

Validation & Forecasting

The validation of the model is simple. We use an input array that has the historic values we want for the period that we want to make the predictions. When we input the array of sequences the model makes predictions for each minute of the period we chose but does not use the prediction it made before to make the next one. We chose to do our validation on the same period as the forecasting in order to show the difference it makes to have knowledge of the actual prices.

In the forecasting phase of our code the input to the model consists of only one sequence, the one just before the time we want to forecast for. After each prediction is made, the sequence is rolled and the oldest price is replaced with the one the model predicted. This is repeated as many times as we want, in our case for 10 days in minutes.

ARIMA forecasting

For the ARIMA model we were asked to compare our results with we chose the one provided in the *Statsmodels* python library. This model is really easy to use and since our main goal was not to create an ARIMA model we chose not to focus a lot on this part. The parameters for the ARIMA model where chosen with the help of AI and may not provide the best results. We made an attempt using another python library to find the optimal values but finally decided not to use it since it clashed with our main library. From the research we did on the ARIMA model the results we get seem to be correct.

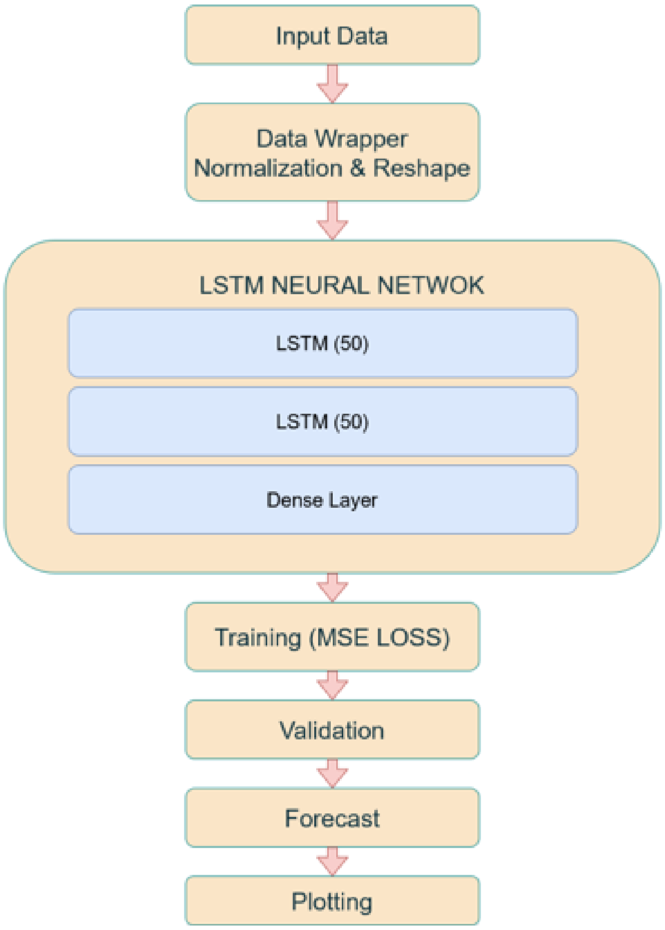
Plotting

As the primary way of showing our results we chose to create plots as it seemed the easiest way to verify our results. The *matplotlib* python library is the one we chose to use. This library makes it easy to create some plots that might be a little more complex.

Pseudocode

Below is the pseudocode of the entire project. It is really simplified since even our original code is quite small and understandable.

```
Get closing prices from .csv

Set parameters (epoch, sequence length)

Manipulate input array shape

Create model
LSTM(50 units, activation = 'relu')
LSTM(50 units, activation = 'relu')
dense(1, activation = 'linear')

Compile model (optimizer = 'adam', loss = 'mse')

Train model

Get MSE from training a & plot it

Validate the model for the last 10 days & calculate validation mse

Plot validation vs actual prices

Make forecast for last 10 days

Make ARIMA forecast for last 10 days

Plot actual vs LSTM vs ARIMA prices
```



5 Results

We made a number of experiments to see how the results vary but we established really quickly that to improve the forecasting ability of our model we needed more hardware. We needed to increase the value of the sequence length if we wanted the model to be more accurate but we where held back by the hardware we possessed as mentioned before. The results we got are not impressive but they are not bad to say the least. We will show 2 different experiments we made with different parameters to see how the affected the models performance.

The final more accurate model had a sequence length of 1440 which is equal to a day in minutes and was trained for 6 iterations. Let's call this **Model 1**. The model we will compare the results of **Model 1** with had a sequence length value of 420 which is equivalent to 7 hours of data and trained for 10 iterations. This will be referred to as **Model 2**.

Training

It is important to note that changing the hardware impacts the training time as a more powerful system is able to make the needed calculations a lot faster. Also, the batch size was 32 for all the models trained. To have meaningful comparisons the times provided bellow are from the same system even though **Model 1** was trained on another, more powerful, system.

Model 1: Complete training time: 35 hours, (1s/step)
Model 2: Complete training time: 13.5 hours, (300ms/step)

It is clear that the larger the sequence length the greater the computational overhead but the better the forecasting. During training the MSE is really low and stabilizes to a low value really quickly. For both models the progression of the MSE is shown in the figures bellow and also the final value after the training was completed is provided.

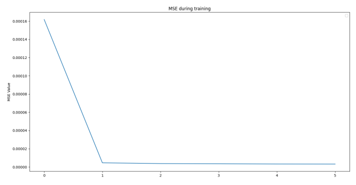


Figure 3: Model 1 MSE

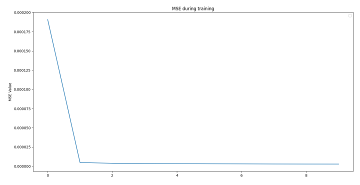


Figure 4: Model 2 MSE

Model 1: Final MSE: $2.9035e^{-6}$ **Model 2**: Final MSE: $2.7852e^{-6}$

The final weights of the trained model can be found in the *ltsm_ weights.txt* files with the corresponding names. We do not include them in the paper because of the sheer amount of data because of the architecture.

Validation

Validating the models is a lot more interesting. The validation was done with the exact same input for both models but the results are surprising. We check the accuracy of the models with the MSE metric. Shown in the graphs bellow are the validation predictions for both models.

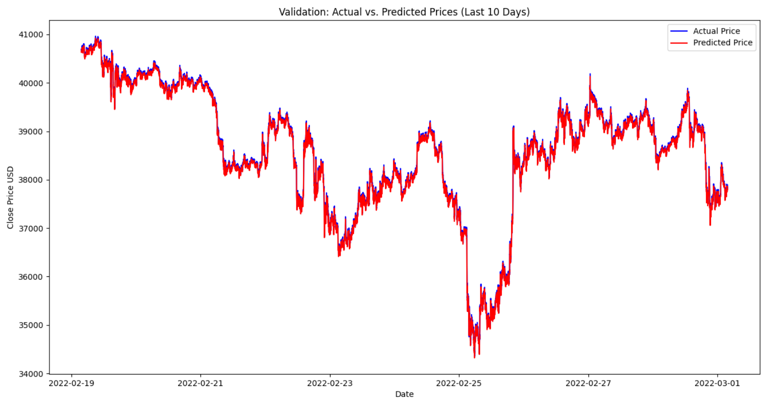


Figure 5: Model 1 Validation

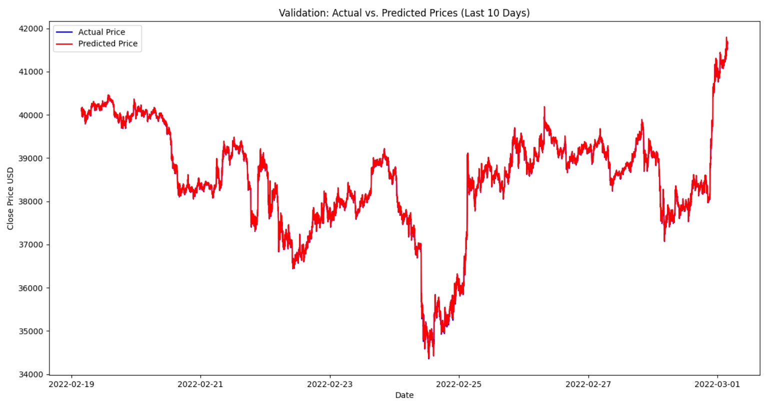


Figure 6: Model 2 Validation

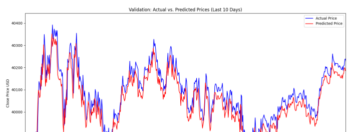


Figure 7: Model 1 close up

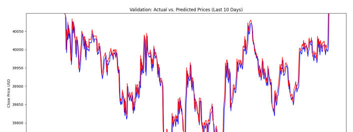


Figure 8: Model 2 close up

Model 1: Final validation MSE: 3183.69
Model 2: Final validation MSE: 2058.89

We see that during the validation phase the model with the higher sequence length is less accurate. This seems counter intuitive but we believe it is actually correct. A larger sequence length makes the model more accurate in long term predictions but in short terms ones, maybe because if the abundance of data and the greater complexity, the model is less accurate.

Forecasting

While forecasting is the main goal of this projects the results are not as good as we would hope for. As mentioned before, accurate forecasting is extremely difficult to achieve and we found that out the hard way. We cannot say that our model is accurate in any way and that is shown by the graphs bellow. What we can say is that it performs better than the ARIMA model and we believe that there is a lot of space for improving our model.

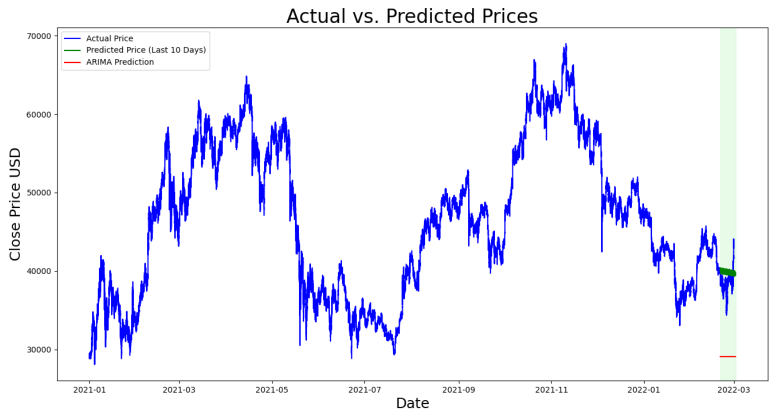


Figure 9: Model 1 Forecast

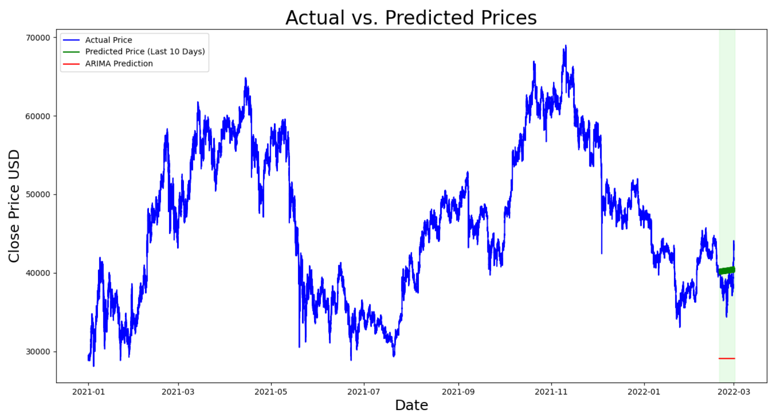
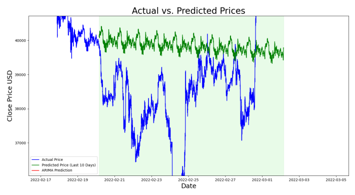


Figure 10: Model 2 Forecast



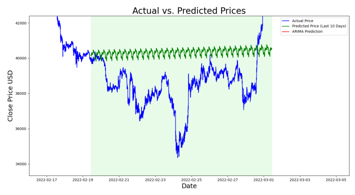


Figure 11: Model 1 close up

Figure 12: Model 2 close up

Let's discuss these results. We see that both models suffer from an oscillating effect. This seems to be a common problem with LTSM networks because of the feedback of their predictions. We also see that **Model 1**, our final model, is not accurate but seems to be better than **Model 2**. This is the result of the larger sequence length. We believe that if we want to forecast for th next 10 days, the sequence length should match or maybe even be greater than that time period. That would allow the model to recognize long term patterns and make a better forecast. But these are only our thoughts since we are unable to test for such great periods of time due to the sheer amount of data. If instead of the minute price of bitcoin we used the daily closing price we could maybe get better forecasting.

The ARIMA model performs a lot worse than our model since it essentially flat lines the price prediction. This is because of the volatility of the Bitcoin price. The ARIMA model is not good with handling volatile data.

Forecasting takes some time and is not instantaneous like validation. Here are the elapsed times for each model.

Model 1: Forecast elapsed time: 1161 seconds
Model 2: Forecast elapsed time: 1052 seconds

Both take around 20 minutes to complete forecasting and the time does not seem to be affected by changing the parameters and neither by changing the hardware since these times are taken from different machines and during experimentation this time is consistent.

6 Conclusion

In conclusion, this project aimed to develop and evaluate a methodology for predicting Bitcoin's closing price each minute. Through a review of existing methods and experimentation with different architectures, we concluded that a three-layer LSTM model performs the best, compared to other CNN architectures. Our model performs better than time series approaches such as ARIMA but still it doesn't produce satisfactorily results.

Despite our efforts, the project revealed several areas for improvement. The LSTM model required substantial computational resources and training time, which limited the extent of our experimentation. Furthermore, Bitcoin's volatile nature posed significant challenges, particularly for short-term forecasting. Another major limitation was the exclusion of external factors, such as social media sentiment, current market trends, and other macroeconomic variables, which are known to influence Bitcoin prices. Probably by incorporating these external factors could enhance the model's ability to capture complex market dynamics. As Jim Simons, founder of the Renaissance Technologies quant trading firm, said, "The market is how people feel about it." Incorporating these external factors, including regional and cultural attitudes towards Bitcoin, could significantly enhance the model's ability to capture complex market dynamics. We believe that the best prediction model would be one that integrates people's opinions about Bitcoin and includes a time-series approach that reflects how people feel and trade Bitcoin across different regions and cultural contexts.

Overall, this project shows the feasibility of using LSTM networks for Bitcoin price prediction given the proper resources and also provided a comparison with traditional statistical methods. Future work could explore hybrid models combining ARIMA and LSTM or investigate alternative deep-learning architectures such as Transformer models to further improve accuracy and efficiency. If this model had a 51 forecasting success rate, we would just set the minimum trading profit to be the transaction fees plus a good salary and then travel for a while.

References

[1] H. N. Gudavalli and K. V. R. Kancherla, *Predicting Cryptocurrency Prices with Machine Learning Algorithms: A Comparative Analysis*, Bachelor's Thesis, Blekinge Institute of Technology, 2023.