Watch Profits Soar to the Ether: Analyzing the Potential for Various RNNs of Predicting Future Ether Prices

Andrew Wisniewski
University of Oregon
awisniew@uoregon.edu

Ethan Reinhart
University of Oregon
ereinha3@uoregon.edu

Abstract

Blockchains in the decentralized web space have opened a new market for investing, with a current potential worth billions of dollars. Predicting the price of cryptocurrencies on blockchains could create significant investment opportunities. Unfortunately, the volatility of cryptocurrency prices makes them difficult to predict compared to stable fiat currencies. In this work, we analyze the market trends of Ether, one of the most widely used cryptocurrencies, and attempt to train various Recurrent Neural Networks to predict future Ether prices. We examine Long Short-Term Memory (LSTM) variants and Gated Recurrent Units (GRUs), comparing and analyzing the performance of four different models. Setting a constant of a 10-day future prediction, our results indicate that the GRU model had the highest comparative accuracy. While these results are not accurate enough for capital gains on investment, we look to improve these in the future with more data.

1 Introduction

Blockchain and cryptocurrency – while much of the public opinion usually associate these terms with news about scams and fraud, there is an alternative perspective that reveals a huge decentralized space with billions of dollars in investments [1]. Ethereum is the most popular blockchain to date, with its native token, Ether, being one of the most popular cryptocurrencies [2]. Since its conception in 2013, the price of Ether has fluctuated from double to quadruple digits. The potential for investment within blockchains leads to investors doing their best to speculate on token prices.

Predicting the future prices of cryptocurrencies like Ether is a complex task due to the volatile and dynamic nature of the market. Traditional financial models often fall short of capturing the intricacies of this emerging market. However, advancements in machine learning, particularly Recurrent Neural Networks (RNNs), offer promising tools for making more accurate predictions.

RNNs are a class of neural networks where connections between nodes form a directed graph along a sequence. This architecture allows them to exhibit temporal dynamic behavior, particularly suited for time-series data, such as cryptocurrency price movements. Unlike traditional neural networks, RNNs can use their internal state (memory) to process sequences of inputs, which is essential for predicting prices based on historical data.

Our approach in predicted Ether prices leverages variants of RNNs like Long Short-Term Memory (LSTM), Bidirectional Long-Short Term Memory (BiLSTM), and Gated Recurrent Units (GRUs), attempting to predict time-series data with high volatility. By capturing the patterns and trends of past price movements, we believe these networks can provide insights that would be valuable for traders and investors. Due to the blockchain being a ledger of transactions, data for our models was widely abundant.

Our approach began with aggregating 12 features related to the blockchain (e.g., average transactions, total addresses) and creating a correlation matrix to determine whether these features could be related to each other. This process provided insights into which features had a high correlation with Ether price and could potentially be useful for our model in making predictions.

2 Background

RNNs encompass a class of neural networks capable of handling sequential data by considering temporal relativity. This allows for chronological sequences to be predicted with much higher accuracy than prior models. Unlike feed-forward neural networks, which process each input independently, RNNs maintain an internal state that allows them to retain information about previous inputs while processing the current input. This sequential processing capability makes RNNs well-suited for a wide range of tasks involving sequential data, including time series prediction, natural language processing, and speech recognition [3].

RNN architecture consists of an input layer, recurrent connections, a hidden state, and an output layer. The input layer receives the sequential input data to the model. The recurrent connections are the defining feature of RNNs. These use the hidden states of the model, or internal representations of the input sequence given the time step. This creates the structure for time steps, where at each time step, the network takes the previous hidden state along with the input vector for that step and produces the next hidden state for the network. At each hidden state, the network creates an output, where all outputs are combined at the end to produce an output.

While standard RNNs offer powerful modeling capabilities, they suffer from certain limitations that can hinder their performance, particularly in tasks requiring long-term memory. The vanishing gradient problem, caused by the exponential decay of gradients over long sequences, makes it challenging for standard RNNs to capture dependencies across distant time steps. As a result, they often struggle to learn and retain information over extended sequences, leading to difficulties in modeling long-range dependencies.

2.1 LSTMs

Long Short-Term Memory (LSTM) networks address the limitations of standard RNNs by introducing specialized memory cells and gating mechanisms [3]. These memory cells allow LSTMs to selectively store or discard information over multiple time steps, effectively overcoming the vanishing gradient problem and enabling them to capture long-term dependencies in sequential data. The key components of an LSTM cell include the input gate, forget gate, output gate, and cell state. The cell state is responsible for controlling long-term memory storage and information control. The input gate determines how much new information should be added to the cell state, the forget gate determines how much information should be retained from the previous cell state, and the output gate determines which parts of the cell should be used to compute the output and next cell state. The cell state coupled with the gated components provides the model with long-term memory, impacting predictions.

2.2 BiLSTM

BiLSTMs were created to enhance LSTM architecture by also considering a backward pass over the data [3]. BiLSTMs consist of a forward LSTM layer, a a backward LSTM layer, and a merge layer, combining the contextual information from both directions. This merge layer is typically executed by averaging or concatenating the results from forward and backward LSTMs. BiLSTMs improve contextual understanding from the basic LSTM model, allowing for better-informed predictions. BiLSTMs can capture even longer-term dependencies due to the combination of the forward and backward pass. Further, this bi-directional pass over the data creates resilience to noisy data. The architecture described translates to improved performance on many sequential analysis tasks: Natural Language Processing (NLP), Speech Recognition, and Time Series Prediction.

2.3 GRU

GRUs are another variant of RNNs that are similar to standard LSTMs, with a slightly simplified structure [3]. GRU architecture consists of two main gates: an update gate and a reset gate. The update gate decides how much past information will be relayed to future states. This is somewhat analogous to the combination of forget gates and input gates of the LSTM model. The reset gate determines how much information the model should forget in a similar process to the forget gate in an LSTM. The GRU also consists of a candidate creation process for the current hidden state by combining the current input and the previous hidden state. The output is then the combination of the previous hidden state along with the current hidden state. With this simplified architecture, GRUs offer a more efficient alternative to LSTMs while retaining the capability to handle long-term dependencies in sequence data.

3 Methods

This section will cover the dataset and methods we used to train and evaluate our models.

3.1 Data

Our data was sourced from etherscan.io, an aggregator of blockchain data. The data starts on 7/30/2015, the conception of the Ethereum blockchain, and ends on 6/1/2024. The data was collected on a daily basis. The features used to create our correlation matrix were:

- Number of Unique Addresses
- Addresses Sending Transactions
- Addresses Receiving Transactions
- Ether Price
- Highest Gas Fee
- Gas Used
- Transaction Growth
- Addresses Created
- Average Gas Price
- Active Addresses

3.2 Features

After creating the correlation matrix for our potential features, we needed to choose which ones to include in our model. For the single-feature implementation of the LSTM, we chose daily Ether price as the feature. The single-feature LSTM predicts the y-value (Ether price) based on a series of x-values (days in the past).

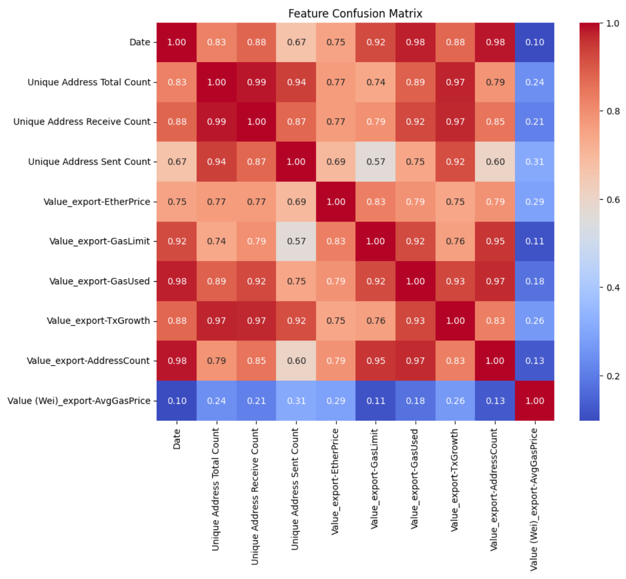

Figure 1: Feature Correlation Matrix

For our two-feature LSTM, we initially selected the daily gas limit, which represents the highest gas fee within a day. We chose this feature because a higher gas fee typically indicates a higher volume of transactions on the blockchain, suggesting increased trading activity and demand for Ether, which can drive up the price. Conversely, a lower gas limit suggests lower transaction volumes, potentially indicating reduced demand and a subsequent price drop.

Moreover, the gas limit had the highest correlation to Ether price in our correlation matrix, with a correlation coefficient of 0.83. This strong correlation led us to believe that including the gas limit as a feature could enhance our model's ability to predict future Ether prices more accurately. However, in practice, we found that using 'Active Addresses' provided the best accuracy in our predictions. The number of daily active addresses seemed to offer more precise insights into user engagement and transaction activity on the blockchain.

3.3 Loss and Accuracy

Our model uses the mean-squared error (MSE) as the loss metric. We chose MSE because it penalizes larger errors more significantly due to its quadratic nature, placing a higher penalty on predictions that are further from the actual value. This helps to ensure that the model prioritizes reducing larger errors, leading to potentially more accurate overall predictions.

For evaluating accuracy in regression models, scaled root-mean-squared error (RMSE) is commonly used. This scaling is essential as our input data was normalized before input. Since we are predicting prices, we wanted a more intuitive metric to assess how well our model's predictions align with actual prices. We calculated the mean squared error as our loss function for our model and then took the square root to obtain the RMSE. We then inverse-scaled the values using the scaler applied to Ether prices to determine the RMSE in dollars.

4 Experiments

4.1 Experimental Procedure

Our experiments began by splitting the data into 80% training data and 20% testing data. The training data consisted of the earlier portion of the dataset, while the testing data was drawn from the most recent portion. This approach allows our models to be trained on historical data and tested on the most recent data. The intention behind this was to evaluate the model's ability to make predictions on recent, unseen data, simulating real-time prediction scenarios.

We additionally recognize the issue of information leaking. We recognize that parsing our data such that we have a sequence of prices in days and a corresponding price 10 days in the future introduces the possibility of leaking 9 days of information. We mitigated this by partitioning before we parsed it, such that our training data would never see examples in our testing data. We did not initially realize this and were surprised to see our model decrease in accuracy following this. Still, we are confident that this is the correct decision for long-term prediction accuracy.

4.2 Model Parameters

We defined a class trainArgs, which would be used to tune and keep hyperparameters constant across all models except for epochs. Epochs were changed due to different models' tendency to overfit depending on how they learn from the data. Keeping hyperparameters constant across all models allows a fair comparison of each model's capabilities relative to one another. Further, we initialized a seed of 42 to for all models such that the distribution of initial weights would be the same across multiple hyperparameter tuning iterations. The trainArgs hyperparameters defined were:

1. Days in Future = 10
   - Number of days in the future the model is expected to predict a value for
   - While days in future is not something that was tuned throughout the process, it still functions as a hyperparameter as it affects the model's accuracy. We elected 10 days in the future as it allows for flexibility in investments while still keeping a relatively short-term scope.
2. Time Step = 200
   - Number of days provided as sequential input to the model for each data point
   - We elected to use 200 days to provide each data point with a sufficient amount of temporal information while keeping the dataset very similar to its original size.
3. Dropout = 0.2
   - A decimal value corresponding to the number of nodes whose output will be set to 0 at the end of each layer
   - We elected to incorporate a dropout rate of 20% after each RNN layer of our model. This allows our model to better generalize to the unpredictable nature of cryptocurrency price fluctuations.
4. Learning Rate = 0.001
   - A decimal value that affects the speed of convergence towards the optimal solution
   - This is a conventional value used for learning rates. We tried with a magnitude larger and smaller and achieved the best results with the given value.
5. Epochs = 20
   - Number of epochs used to train each model

- After tuning the number of epochs, we found training for over greater than 20 epochs tended to cause model overfitting whereas training for under 20 epochs tended to cause underfitting.
- This number remained constant across all models other than the BiLSTM where we found using half of this provided optimal results

6. Neurons = 64
    - Number of neurons used in each LSTM or GRU layer
    - We kept this constant across all models such that hyperparameters could be more accurately tuned. We selected this number by experimenting with different numbers of fully connected nodes between layers and found this suitable for our dataset.

7. Conv = 64
    - Number of neurons used in the convolutional layer prior to the output layer
    - We also kept this constant across all models as this increased accuracy compared to a single output layer.

These hyperparameters were tuned across all models but were optimized for Model 1 (1-Feature LSTM). These hyperparameters were initialized based on conventions but tuned throughout the training process.

4.3 Model Architecture

1. 1-Feature LSTM

    In this model, we use two LSTM layers, each followed by a dropout layer with the priorly specified dropout rate. The input of this model is a vector sequence of length equal to the time step. The neurons in an LSTM determine the dimensionality of the hidden state vector. This number of neurons stayed constant across all LSTM implementations to better analyze performance. The first layer returns a sequence of vectors while the second layer returns a single vector. The second layer is fully connected to the convolutional Dense layer. This Dense layer is fully connected to the output layer.

2. 2-Feature LSTM

    This model uses the same architecture as the prior LSTM, other than a modified expected input. This now expects a matrix sequence, instead of a vector sequence, of the same length as the prior.

3. BiLSTM

    This model uses a very similar architecture to the LSTM, utilizing bidirectional LSTM layers instead. Both forward and backward passes make use of an LSTM that passes over the data in the forward and backward direction, respectively. These each have the same structure as the first LSTM. These are then fully connected to the convolutional Dense layer and then fully connected to the output layer.

4. GRU

    This model has the same structure as the previous models, utilizing 2 GRU layers instead of LSTMs. Each of these layers is fully connected to the convolutional Dense layer and then fully connected to the output layer.

4.4 Performance Analysis

The graph of our model predictions for the test data and RMSE accuracy is shown below:

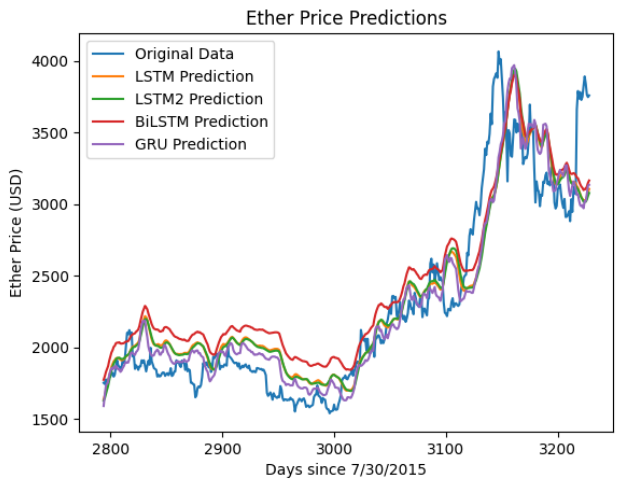

Figure 2: Test Data Prediction Graph

As shown in our graph and RMSE table, the GRU model performed the best. We were surprised with how poorly the BiLSTM did relative to the other

Model	Overall RMSE (in USD)
1-Feature LSTM	252.01
2-Feature LSTM	254.84
BiLSTM	294.21
GRU	242.47

Table 1: RMSE values (in USD) for different models

models. We hypothesize the reason for this is that backward regression over the data is not beneficial in predicting regression for future data points. We originally thought not enough learning was occurring, but training over a greater number of epochs on this model increased loss. We hypothesize that the 2-Feature LSTM performed worse than the 1-Feature due to the noise added by the additional feature. We hypothesize this is likely due to the natural unpredictability of compounding factors that influence price, other than past prices themselves. We note that GRU performed with the lowest loss. We believe this is due to the simple, yet powerful, architecture present in this model. We argue this is the best model for our dataset, as it can accurately capture temporal dependencies with less training required, essential for a small dataset such as the one used to train this model.

5 Conclusion

In this paper, we compare the results of RNN variants with similar hyperparameters for predicting Ether prices. Our results indicate that a GRU model would be the best choice to tune and use for predicting cryptocurrency prices. As mentioned in the results, the models encounter difficulty when predicting the price of cryptocurrencies 10 days into the future — likely due to their volatility and the compounding factors that impact them.

We conclude that more work could be done with these models to achieve higher accuracy. Our future work on these models will include finding more data, specifically with shorter intervals between data points. Hourly or 10-minute intervals would provide sufficient information to improve the accuracy over volatile data. If we had more time, we would have also liked to engineer different features to optimize the performance of the GRU and determine whether it could be useful in trading.

Acknowledgements

We would like to thank Professor Thien and GE Steven Walton for a great term!

References

1. Federal Trade Commission. "FTC Data Shows Huge Spike in Cryptocurrency Investment Scams." FTC, 17 May 2021, `https://www.ftc.gov/news-events/news/press-releases/2021/05/ftc-data-shows-huge-spike-cryptocurrency-investment-scams`.

2. CoinGecko. "Most Popular Cryptocurrencies." CoinGecko, `https://www.coingecko.com/research/publications/most-popular-cryptocurrencies`.

3. "RNNs." CS472: Machine Learning, University of Oregon, `https://classes.cs.uoregon.edu/24S/cs472/notes/RNNs.pdf`.

Section Authorship

- Andrew: Abstract, Introduction, Methods (3.1, 3.2, 3.3), Experiments (4.1, 4.4), Conclusion
- Ethan: Introduction, Background (2.1, 2.2, 2.3), Experiments (4.1, 4.2, 4.3, 4.4)