Using ML Techniques to Implement a Profitable Bitcoin Trading Strategy

CS229 Final Project

Bryn Hughes[1], Rayan Rizvi[2], and Nate Fleischli[3]

[1] *Stanford University Department of Computer Science, brynmh@stanford.edu*
[2] *Stanford University Department of Economics, rrizvi@stanford.edu*
[3] *Stanford University Department of Computer Science, nathan23@stanford.edu*

December 6, 2021

Cryptocurrency trading volume has increased dramatically since 2009, when Satoshi Nakamoto first implemented Bitcoin and made it publicly available. Especially over the past two years, where cryptocurrency trading has been increasing at an exponential rate. In this study, we use cryptocurrency market data across 14 different tokens as input to attempt to utilize a recurrent neural network, specifically a long short-term memory, to implement a trading strategy that can profitably trade USD with Bitcoin over an extended period of time. In the end, we were able to achieve this relatively successfully, with our primary takeaway showing the difficulty of predicting asset targets removing market movements.

1 Introduction

At a high-level our project focuses on using different ML techniques to implement a profitable bitcoin trading strategy. While the field of quantitative finance, in reference to applying machine learning to markets, has been around for some time, applying machine learning to cryptocurrency trading is a relatively new field. Prior to this project, our team had experience with creating trading algorithms in traditional financial applications. Since, as mentioned in our abstract, the popularity of blockchain and cryptocurrencies is dramatically increasing, we thought that choosing this project topic would be relevant, timely, and challenging. We also believed that we could create somewhat of a novel approach given that this is a relatively new field. We found that our topic was important because of the increasing role that blockchain and cryptocurrency technologies are playing in our lives especially with regard to financial applications, or DeFi.

To define the problem more specifically, we will discuss our inputs, outputs, and utilized algorithms here. Our input is time-series price data of 14 different cryptocurrencies. We then use something similar to a recurrent neural network, specifically a long short-term memory (LSTM), to output a predicted target, which refers to 15-minute market residualized returns. More information on the inputs will be featured in the dataset and features section, more information on the LSTM (along with other algorithms we tried but did not function as well as the LSTM) will be featured in the methods section, and more information on the target will be featured in the experiments/results/discussion section.

2 Related Work

In this section we will describe five different approaches that people have attempted when solving the same or similar problem as we are. To break them into multiple groups, we will first discuss approaches that utilize RNNs (recurrent neural networks) or LSTMs (long short-term memory) and the second discuss approaches that do not utilize RNN/LSTMs.

The first example is Jaquart, Dann and Weinhardt's study on short term bitcoin market prediction via machine learning. In contrast to our project, this study analyzes the predictability of the bitcoin market with different prediction horizons ranging from 1 to 60 minutes. This is something that we would like to extend our project with and will be discussed in the future work section as well. This study tests various machine learning models and found, similar to our study as well, that

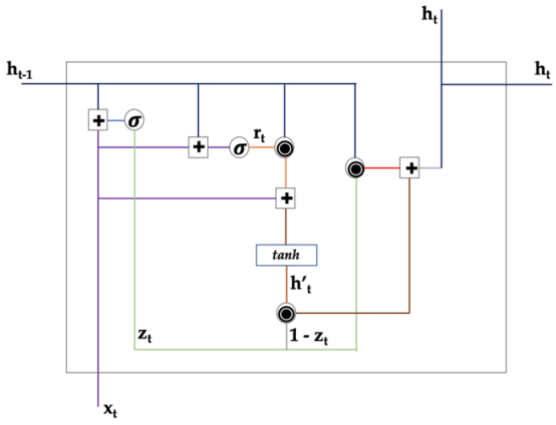

Figure 1: *Architecture of a GRU [1]*

recurrent neural networks are "especially well-suited for the examined prediction tasks". Another part of this study that could be a possible extension of our project is using blockchain-based and interest-based features. In our project, our inputs were exclusively asset-based features [2].

The second example to consider that utilizes an RNN is Hamayel and Owda's study on cryptocurrency price prediction using three different variations on RNNs. In contrast to our project, this study utilizes an LSTM but also utilizes a GRU (gated recurrent unit) and a bi-LSTM (bidirection LSTM algorithm) to solve the problem. A GRU only has two gates (reset and update) as opposed to the LSTM which has three gates. This leads to the GRU being computationally more efficient than the LSTM. The metric of accuracy utilized in this project is mean absolute percentage error. Comparing the three algorithms used, the GRU performed significantly better than the LSTM and bi-LSTM algorithms. While our program exclusively predicts bitcoin prices, this study predicted the values of Bitcoin, Ethereum, and Litecoin, with Ethereum being the most accurate prediction. If it weren't for the data/memory constraint we faced, testing our algorithm on different tokens such as Ethereum would be a good extension [3].

Another RNN-type study to consider, similar to the previous, is UC Berkeley's Dutta, Kumar, and Basu's study on using a GRU (gated recurrent unit) to predict bitcoin prices. They also conclude that a GRU performs better for this specific problem. Figure 1 above shows how a GRU performs, and you can see our explanation of the LSTM model below to be able to fully understand how the GRU is different from the LSTM [1].

Now we will turn toward different implementations that do not use recurrent neural networks as the primary approach to solve the problem. The first approach to consider is Koker's study on using machine learning for cryptocurrency trading. In this study, Koker presents a model for active training based on reinforcement machine learning. The scope of his study is using five different cryptocurrency tokens. The performance metric utilized to provide feedback is the Sortino ratio, which is referenced in eqn. 1 and thus the gradient descent step in eqn. 2, with R referring to trading returns. The sortino ratio is chosen over the sharpe ratio because as Koker explains, sharpe ratio discourages models from making trades with large positive returns because of the impact on volatility [4].

$$Sortino = \frac{mean(R)}{\sqrt{mean(min(R,0)^2)}} \tag{1}$$

$$\Delta\theta = \eta \frac{dS(\theta)}{d\theta} \tag{2}$$

The last approach to discuss is Lahmiri and Bekiros' study on intelligent forecasting with ML trading systems in intraday bitcoin market. While no recurrent neural networks are used, the authors experiment with many different models, including an SVR, GRP, RT, kNN, FFNN, BRNN, and RBFNN. Given constraints, we will explore their usage of the BRNN (bayesian regularization neural network) model which showed the highest level of accuracy with fast convergence. As they explain, the BRNN reduces the possible effects of overfitting by adding an additional term to the sum of squared error of the neural network model. With the additional component the objective function F becomes:

$$F = \beta E_d + \alpha E_w \tag{3}$$

where E_d is the sum of squared errors, E_w is the sum of the square of network weights, with α and β as regularization parameters [5].

All in all, it seems that the best performing model was in fact an RNN, but seemingly the gated recurrent unit outperformed the LSTM as evident in Hamayel and Owda's study and the UC Berkeley study as well. An improvement to our project could be using a GRU as opposed to the LSTM that we utilized.

3 Dataset and Features

The dataset that we are utilizing is time-series price data for 14 different cryptocurrency tokens, where one line represents the minute data for one token. The features that we utilize are the close price, asset weight, target value, which I will describe below. The target refers to 15 minute residualized returns. At a high-level, the target is meant to be the 15 minute log return of the specific asset minus the market returns. Due to the fact that cryptocurrency asset returns are highly correlated, it is import that we perform this linear residualization. It is important to note that when calculating the market returns, each asset has a weight that corresponds to their trading volume. Thus bitcoin, for example, contributes more heavily to our calculation of the market

Table 1: *Assets and their Weights*

Weight	Asset Name
2.3978952727983707	Bitcoin Cash
0,4.30406509320417	Binance Coin
6.779921907472252	Bitcoin
1.3862943611198906	EOS.IO
2.0794415416798357	Ethereum Classic
5.8944028342648505	Ethereum
2.3978952727983707	Litecoin
1.6094379124341003	Monero
1.791759469228055	TRON
2.0794415416798357	Stellar
4.406719247264253	Cardano
1.0986122886681098	IOTA
1.0986122886681098	Maker
3.5553480614894135	Dogecoin

return than litecoin, let's say. Here is the exact math of our target calculation. First, with prices P^a for each asset a, the return is the following:

$$R^a(t) = log\frac{P^a(t+16)}{P^a(t+1)} \tag{4}$$

Then, our weighted market return is the following, with w^a representing the weight of asset a:

$$M(t) = \frac{\sum_a w^a R^a(t)}{\sum_a w^a} \tag{5}$$

We then calculate the target as:

$$Target^a(t) = R^a(t) - \beta^a M(t) \tag{6}$$

with β^a representing the rolling average of asset a's weight. To illustrate the different assets and their corresponding weights, we refer to table 1 above [6].

It is important to make clear that while we are only predicting the bitcoin target, we utilize the entire dataset in order to remove the market movement from our bitcoin target calculation. In terms of the size of the dataset, we are using 19.36 million rows for our training and 4.84 million rows for our testing. This is a 4:1 training to testing ratio. In terms of preprocessing, the biggest bump we ran into was handling NaN values. A NaN price close value would subsequently cause a NaN target value, so it was important that we handled the NaN values properly. This was especially true because given the large nature of the dataset (24.2 million rows) there were upwards of 300,000 NaN rows.

We decided that given the nature of our data being time-series data, we did not have the option to drop or ignore NaN values. This is because when we are looking 15-minutes ahead or 60-minutes back for example, a NaN value could cause downstream effects. We decided to handle NaN values using linear interpolation where we would fill in NaN close price values by looking at previous and future closing values and interpolating the missing value. We then recalculated the targets for the data after interpolation.

4 Methods

To approach our problem, we used three different machine learning techniques to try to predict our bitcoin returns. In each subsection we will describe the algorithm that we used and how we applied it to our problem.

4.1 Logistic Regression

The first technique we utilized was logistic regression. Logistic regression is derived from linear regression with the primary difference being that logistic regression is used for classification problems. In the context of our problem, we utilized a binary output logistic regression that predicted whether the target was positive (meaning the price of the asset increases) or if the target was negative (meaning the price of the asset decreases). To dive deeper into logistic regression, let's start with the hypothesis function which is defined as,

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}} \tag{7}$$

and the sigmoid function is defined as,

$$g(z) = \frac{1}{1+e^{-z}} \tag{8}$$

. Logistic regression then uses gradient ascent to maximize the log-likelihood of the parameters to fit the parameters for our model. We then have the stochastic gradient ascent rule as the following:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \tag{9}$$

With fitted parameters, we utilize the hypothesis function to make predictions, in our case predicting whether the target value will be positive or negative [7].

4.2 1-Dimensional Convolutional Neural Network

The second technique we utilized was multiple 1-dimensional convolutional neural networks. We will first provide background on what a standard convolution neural network is, what it means to be 1-dimensional, and then discuss how we applied it to our problem. The CNN is a specific type of neural network that is most often applied to two dimensional image data. The main differentiator of this type of neural network is the convolutional layer where a linear operation is applied that multiplies the input data by a

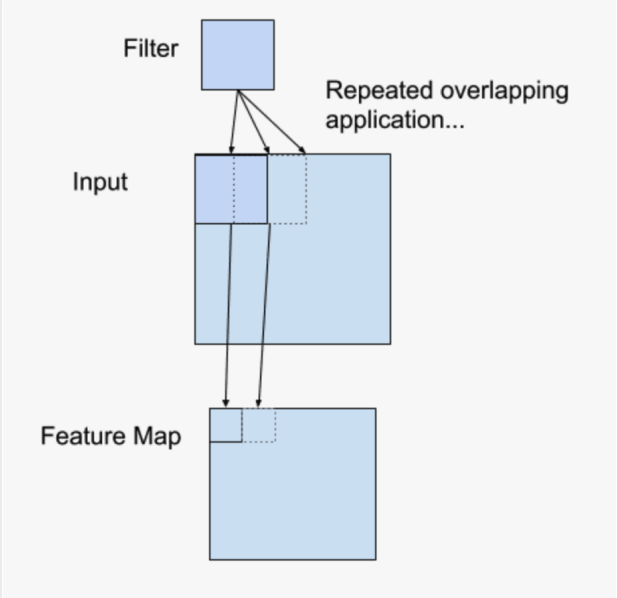

Figure 2: *Common 2d CNN [8]*

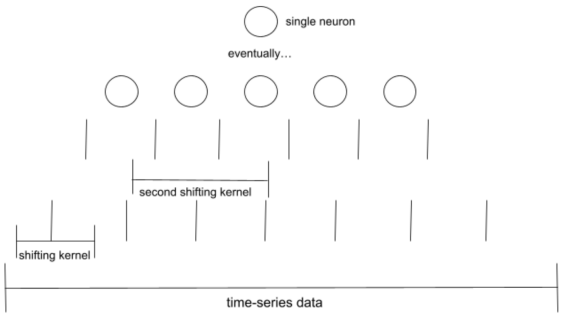

Figure 3: *Our Application of the CNN*

specific set of weights, which is referred to as the kernel [8]. The kernel moves, in this case slides on our time-series data to apply the set of weights to different sets of overlapping inputs. A graphic of a common 2d CNN is feautured in figure 2. In our problem, we utilize a CNN which moves along only 1 dimension, our time-series data. We utilize multiple 1d CNNs eventually resulting in one single neuron for our predictions. See figure 3 for a sketch of how we applied it to our problem.

4.3 RNNs and LSTMs

The final technique to explore is recurrent neural networks (RNNs) and specifically our implementation of a long short-term memory (LSTM). To begin with, it is important to understand the value of an RNN. Traditional neural networks are unable to use prior events to inform current events in a model. On the other hand, recurrent neural networks are able to overcome this issue because they are neural networks with loops, which allows information to persist. One can visualize

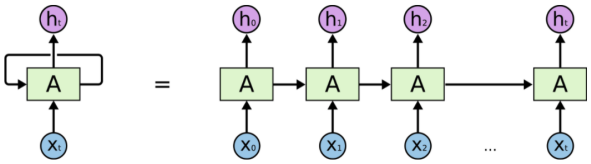

Figure 4: *Unrolled RNN [9]*

the structure of an RNN with figure 4 with inputs x, outputs h, and A representing a neural network.

In our implementation, we use a more robust version of an RNN that is referred to as an LSTM. LSTMs solve the limit of an RNN where it can not fully be relied on to properly recollect prior information in a long-term capacity. While LSTMs have a lot going on under the hood, we will concisely explain the key functions of the LSTM. Refer to figure 5 for a visual representation of how an LSTM functions. The top line is labelled the cell state which stays through every layer of the series. The gates (forget, input, output) allows us to remove or add additional information to the cell state. Using the input data and the previous hidden state, the forget gate will decide which information is valuable and which isn't. The input gate then transforms the values after the current state and previous hidden state are passed into the sigmoid function. The output gate then takes in the values of the current and previous hidden state and determines the value of the next hidden state [10].

As evident in the diagram, the $sigmoid$ and $tanh$ are heavily utilized by the LSTM. Refer to equation 8 for the definition of the sigmoid function. The $tanh$ function refers to the hyperbolic tangent function which transforms values between -1 and 1. The $tanh$ function is defined here below [11]:

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} = 2sigmoid(x) - 1 \qquad (10)$$

5 Experiments/Results/Discussion

Given space constraints, we will focus our results discussion on our implementation of the LSTM. Our 1d CNN model and logistic regression models performed slightly worse than the LSTM, but converged in the same fashion as is described in the LSTM section. The LSTM performed best most likely due to the fact that it is best-suited to time-series data and has more parameters than the other two models. Given our large amount of data and use of the Adam optimizer, which penalizes overfitting, our model did not overfit the training set. We know this as the testing accuracy was very similar to our training accuracy.

First, table 14 displays the values of our hyperparameters. The reasoning behind our choice of hyperparameter values is as follows:

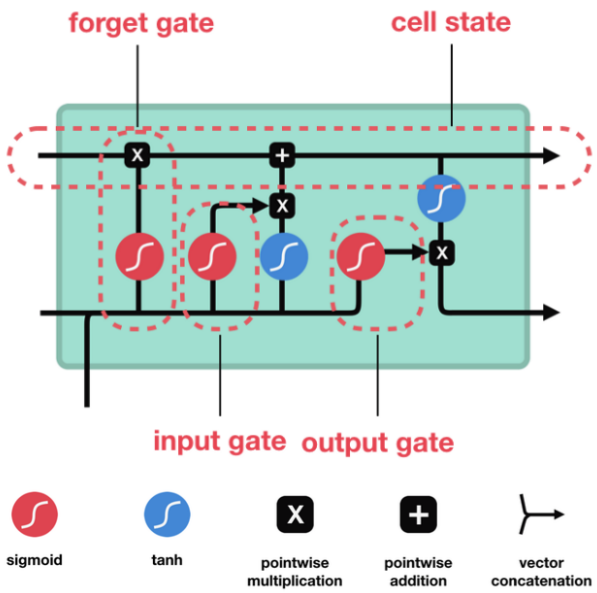

Figure 5: *Architecture of an LSTM [12]*

Table 2: *Hyperparameters*

Hyperparameter	Value
Number of Epochs	50
Batch Size	128
Optimizer	Adam
Loss Function	MSE
Learning Rate	5×10^{-5}

- Initially the models were tested with more than 50 epochs, but more epochs were not necessary.
- The batch size was chosen to be a large power of two that was still compatible with our computing memory constraints.
- The Adam optimizer was chosen given its reliability and ability to prevent over fitting. Most hyperparameters (excluding the learning rate) for the Adam optimizer were left alone as they performed sufficiently (for example the β values).
- Other choices for the loss function such as the pearson correlation coefficient or sign agreement loss were tested, but neither resulted in increased performance, or resulted in worse performance, and thus MSE loss won out because of its simplicity.
- The learning rate was adjusted to stabilize convergence and rate of convergence appropriately. This was done using a coarse to fine random search.

Our primary metric used to measure performance was sign agreement:

$$\text{sign\_ag}(\hat{y}, y) = \mathbf{1}\{\text{sign}(\hat{y}) = \text{sign}(y)\} \tag{11}$$

We now turn to our discussion of results. The trained results did not model the target as closely as desired. As seen in figure 6, the predicted target values converged to the mean of the true target values, rather

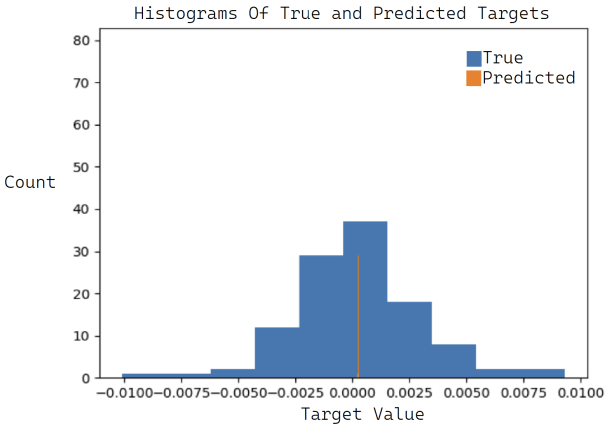

Figure 6: *Histogram of True and Predicted Values*

than estimating target values for that input. Initially, we thought this was because the learning rate was too high, but after testing different values we saw this was not the case. Training multiple times and with varying models gave the same results. In fact, even when using a subset of the data and attempting to over fit the data, the model would converge on the average values. Our conclusion from this is that the distributions of $S|T > 0$ and $S|T \leq 0$ where S is the past 60 minutes of price data and where T is the target, are very similar. This would explain the convergence of our model. We believe that this is because the price is largely affected by three factors: value (affected by exogenous factors), price dynamics, and noise. By subtracting market trends, we largely removed value from our target. We hoped the model would be able to utilize dynamics to predict price changes, but the target was more noise based than expected, with only a small percentage of the target explainable by price dynamics.

6 Conclusion/Future Work

To summarize our project, we attempted to predict a market-residualized target of bitcoin prices using logistic regression, a 1-dimensional convolutional neural network, and a long short-term memory RNN (which performed the best). Through our analysis of related work on the subject and our own implementation, we came to the conclusion that predicting market-residualized returns may not be easily achieved through the use of an LSTM. Through our analysis, we determined this was the case because outside of market movements, most fluctuations in price can be attributed to noise with a small percentage attributed to external dynamics. If we had more time and resources, there are many things we would like to try. Examples include using a GRU as opposed to an LSTM, adding more non-price inputs, or trying our model on multiple tokens (ethereum, litecoin, etc.) to name a few.

7 Contributions

In terms of the distribution of work, Nate led the logistic regression model, Rayan led the 1d convolutional neural network implementation, and Bryn handled the implementation of the LSTM. Bryn wrote the introduction and related work sections of the paper, Rayan worked on the dataset and features and methods, while Nate focused on the results and conclusion section. All team members contributed to the bibliography.

8 References/Bibliography

[1] Dutta, Aniruddha et al. "A Gated Recurrent Unit Approach to Bitcoin Price Prediction". Journal of Risk and Financial Management 13. (2020): 23.

[2] Jaquart, Patrick et al. "Short-Term Bitcoin Market Prediction via Machine Learning". The Journal of Finance and Data Science 7. (2021).

[3] Hamayel, Mohammad et al. "A Novel Cryptocurrency Price Prediction Model Using GRU, LSTM and bi-LSTM Machine Learning Algorithms". AI 2. (2021): 477-496.

[4] Koker, Thomas et al. "Cryptocurrency Trading Using Machine Learning". Journal of Risk and Financial Management 13. (2020): 178.

[5] Lahmiri, Salim et al. "Cryptocurrency forecasting with deep learning chaotic neural networks". Chaos, Solitons Fractals 118. (2019): 35-40.

[6] Stein N Brito, Carlos, et al. "Tutorial to the G-Research Crypto Competition." Kaggle, Kaggle, 3 Nov. 2021, https://www.kaggle.com/cstein06/tutorial-to-the-g-research-crypto-competition.

[7] Ng, Andrew. "CS229 Lecture Notes." CS229: Machine Learning , https://cs229.stanford.edu/notes2021fall/cs229-notes1.pdf.

[8] Brownlee, Jason. "How Do Convolutional Layers Work in Deep Learning Neural Networks?" Machine Learning Mastery, 16 Apr. 2020, https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[9] Olah, Christopher. "Understanding LSTM Networks." Understanding LSTM Networks – Colah's Blog, 27 Aug. 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[10] Singhal, Gaurav. "Introduction to LSTM Units in RNN." Pluralsight, 9 Sept. 2020, https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn.

[11] "Tanh Activation." Explained | Papers With Code, https://paperswithcode.com/method/tanh-activation.

[12] Phi, Michael. "Illustrated Guide to LSTM's and GRU's: A Step by Step Explanation." Medium, Towards Data Science, 24 June 2018, https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[13] Wes McKinney, . " Data Structures for Statistical Computing in Python ." Proceedings of the 9th Python in Science Conference.

[14] Charles R. Harris, et al. "Array programming with NumPy". Nature 585. 7825(2020): 357–362.

[15] Hunter, J. D.. "Matplotlib: A 2D graphics environment". Computing in Science Engineering 9. 3(2007): 90–95.

[16] Paszke, Adam et al. PyTorch: An imperative style, high-performance deep learning library. 2019.

[17] Virtanen, Pauli et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". Nature Methods 17. (2020): 261–272.