Eth Lab: Predicting the Price of Ethereum Using RNNs

CS230 Project Final Report

Michael Zhu - mzhu25

March 23, 2018

1 Introduction

Cryptocurrencies, despite their recent growth in value and mainstream attention, are notoriously volatile. More than other commonly traded assets, the prices of cryptocurrencies are thought to be especially unpredictable. This project uses recurrent neural networks (RNNs) trained on cryptocurrency trading data (e.g. prices, market cap, volume) to predict the price of Ethereum.

2 Data

2.1 Historical Price Data

I use a Kaggle dataset for the daily price and price-related features (e.g. high, low, market cap, volume) of Ethereum and five other major cryptocurrencies (Bitcoin, Dash, Litecoin, Monero, Ripple). Each (cryptocurrency, price feature) pair is used as a feature, so there are 36 price features per day:

$$\text{Price features} = \begin{bmatrix} \text{open} \\ \text{close} \\ \text{volume} \\ \text{high} \\ \text{low} \\ \text{market cap} \end{bmatrix} \times \begin{bmatrix} \text{Bitcoin} \\ \text{Dash} \\ \text{Ethereum} \\ \text{Litecoin} \\ \text{Monero} \\ \text{Ripple} \end{bmatrix}$$

Let $p \in \mathbb{R}^{36}$ denote these features. Let $e \in \mathbb{R}$ denote the closing price of Ethereum (a single feature).

2.2 Twitter

The prices of cryptocurrencies are thought to be vulnerable to shifts in public attitudes as influenced by media. This project hoped to leverage this largely unexplored relationship prices by extracting natural language features from new headlines about cryptocurrencies. To obtain these features, I scraped Twitter for news tweets matching the queries "ethereum" and "crypto OR cryptocurrency" for every day since August 7, 2015 (the start of Ethereum in the Kaggle dataset). Each of the two queries thus generates a corpus of tweets for any given day. I tried two different methods of converting these corpora into vectors: word embeddings and sentiment analysis.

For the word embedding method, I used Numberbatch to obtain 300-dimensional embeddings of the individual words in each corpus. I then concatenate the element-wise max and the element-wise min over all the embeddings to obtain a 600-dimensional vector representation of the whole corpus, as suggested by this paper. Let $\tau_{\text{eth}}, \tau_{\text{crypto}} \in \mathbb{R}^{600}$ denote the word embedding features for the "ethereum" query and "crypto OR cryptocurrency" query, respectively.

For the sentiment analysis method, I used VADER Sentiment Analysis to obtain four sentiment scores (positive, neutral, negative, compound) for each tweet. Then I compute the max, min, and average of these four scores over the entire corpus. These twelve values are used as features. Let $s_{\text{eth}}, s_{\text{crypto}} \in \mathbb{R}^{12}$ denote the sentiment features for the "ethereum" query and "crypto OR cryptocurrency" query, respectively.

2.3 Feature Exploration

Training neural networks on different combinations of the above features yields significantly different results. The combinations I tested (and the names I will refer to them as) are:
- Baseline: $\{e\}$
- Price: $\{p\}$
- Embedding: $\{p, \tau_{\text{eth}}, \tau_{\text{crypto}}\}$
- Sentiment: $\{p, s_{\text{eth}}, s_{\text{crypto}}\}$

Further, for each combination (except Embedding, for computational efficiency) I tested window sizes (i.e. sequence lengths to feed into the RNN) of 5 days and 10 days. An example input of window size 5 would have the following structure:

$$X^{(t)} = \begin{bmatrix} x_{t-5} \\ x_{t-4} \\ x_{t-3} \\ x_{t-2} \\ x_{t-1} \end{bmatrix}$$

where x_{t-i} is a row vector of features for day $t - i$.

2.4 Data Augmentation

Because Ethereum was launched in 2015, there are only a few hundred data points in the dataset. The Kaggle dataset (at the time of model training) contained price data for Ethereum between August 7, 2015 and February 21, 2018, a total of 930 days. I used a 830/50/50 train/dev/test split, where the test set contains the last 50 days in the dataset and the dev set contains the penultimate 50 days. The problem here is twofold: (i) there is not enough training data, and (ii) the price and price fluctuations of Ethereum have increased significantly in the last year, so the dev and test sets represent a very different distribution from the training set.

To address these issues, I augmented the dataset with data for different cryptocurrencies. I generated the equivalents of $e, \tau_{\text{eth}}, \tau_{\text{crypto}}, s_{\text{eth}}$, and s_{crypto} for Bitcoin, Litecoin, and Monero. Note that the price features p can be kept the same. I chose these three cyptocurrencies because "dash" and "ripple" are English words which may appear in tweets unrelated to their respective cryptocurrencies. For these Bitcoin, Litecoin, and Monero, I only used data from the last 300 days of the dataset, for a total of 900 additional data points. This aims to address the train/test mismatch described above, since these cryptocurrencies have seen similar levels of growth as Ethereum in the last year.

I tested two augmentation methods. In the first method, the model is trained and evaluated (250/50 train/dev split) on the Bitcoin/Litecoin/Monero data (shuffled together), then the parameters with the best validation loss are used to warm start training on the Ethereum data. In the second method, all of the Bitcoin/Litecoin/Monero data is shuffled into the Ethereum training set. The model is trained on this set and evaluated on the Ethereum dev set. I will refer to these two methods as “warm start” and “shuffle”, respectively.

3 Methods

3.1 Prediction Tasks

Although daily closing price is perhaps the most natural task to train a neural network to predict, it is not necessarily the best. I trained models to perform three different prediction tasks, which I will refer to as “regression”, “relative”, and “binary”. Let e_t denote the closing price of Ethereum on day t. The data point labels for each prediction task are as follows:
- Regression: daily closing price, e_t.
- Relative: relative price change, $(e_t - e_{t-1})/e_t$
- Binary: direction of price change, $\mathbb{1}\{e_t > e_{t-1}\}$

For the regression and relative tasks, models were trained using mean squared error as the loss function. For the binary task, binary cross-entropy was used as the loss function.

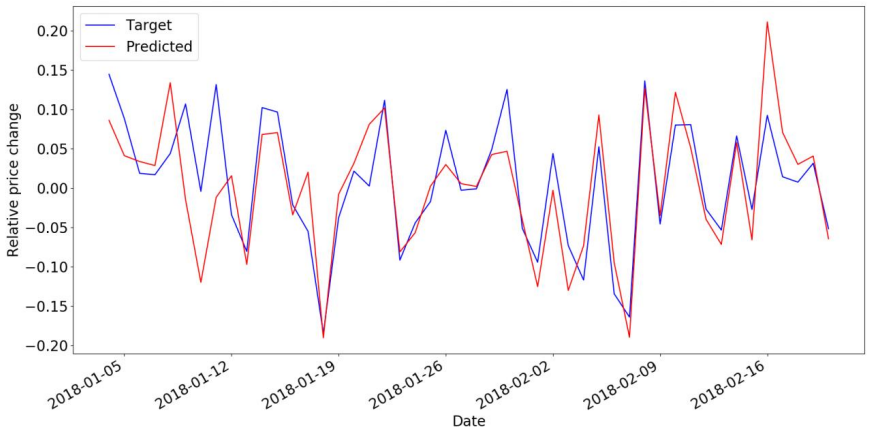

Figure 1: Test set predictions and labels for the relative task

Table 1: Relative task test results

	Mean Squared Error
Best model	0.12441
Same model, no data augmentation	0.28236
Always 0 baseline	0.30834
Baseline NN	0.32275
Lag* baseline	0.46616
Lag baseline	0.57645

3.2 Baselines

The following (non-neural network) baseline models were implemented. Note that some models are only applicable to one or two of the above prediction tasks, so the range of application is included in parentheses. Let y_t denote the target value (for a given prediction task) for day t, and let $\hat{y}_t$ denote the baseline's prediction.

- Lag: $\hat{y}_t = y_{t-1}$ (regression, relative, binary)
- Lag*: $\hat{y}_t = \frac{y_{t-2}+y_{t-1}}{2}$ (regression, relative)
- All 0: $\hat{y}_t = 0$ (relative)
- Majority: $\hat{y}_t = \text{mode}(y_{t-3}, y_{t-2}, y_{t-1})$ (binary)

Another baseline model is obtained by training a neural network for a given task on only the "Baseline" features (see Section 2.3).

3.3 Searching Configurations and Hyperparameters

To summarize, we have the following independent variables to choose from when training a model:
- Prediction task: {regression, relative, binary}
- Features: {baseline, price, embedding, sentiment}
- Window size (I tested 5 and 10)
- Data augmentation: {None, warm start, shuffle}
- Network architecture: e.g. number/type of layers, layer dimensions, etc.
- Other hyperparameters: e.g. learning rate, regularization, etc.

Clearly it would be intractable to do a grid search over these variables, so instead I adopted a coordinate-descent type approach for experimentation, i.e. testing values for a single variable while holding the others constant. For the sake of brevity, I will omit the results for most of these experiments. Some notable findings and the best configurations are discussed in Section 4.1.

Table 2: Regression task test results

	Mean Squared Error
Lag baseline	316862.6
Lag* baseline	410308.9
Best model	12635989
Baseline NN	27150004

Table 3: Binary task test results

	Cross-entropy loss	Accuracy
Best model	30.59149	0.60416
Baseline NN	33.26995	0.52083
Lag baseline	635.5135	0.52083
Majority baseline	663.1446	0.50000

4 Results

4.1 Best Configurations

From preliminary testing, the best network architecture was two LSTM layers, followed by a fully-connected linear layer and a linear output layer (for the binary prediction task, a sigmoid output layer is appended). The two LSTM layer model outperformed: (i) a feed-forward neural network, (ii) one LSTM layer, and (iii) three LSTM layers. Note that the feed-forward neural network was fed a flattened version of $X^{(t)}$ (see Section 2.3) as input. I held the network architecture constant for all of the experiments described below.

A window size of 10 consistently outperformed a window size of 5. The "shuffle" data augmentation method consistently outperformed the "warm start" strategy, and both strategies performed better than using the non-augmented dataset. A learning rate of `1e-4` performed better than `1e-3` and `1e-5`. For regularization, I experimented with dropout layers and weight decay, but neither improved performance. Models trained on "price" outperformed those trained on the other features ("baseline", "embedding", and "sentiment").

To summarize:
- Prediction task: {regression, **relative**, binary}
- Features: {baseline, **price**, embedding, sentiment}
- Window size: {5, **10**}
- Data augmentation: {None, warm start, **shuffle**}
- Network architecture: 2 LSTM layers + fully-connected linear layer + linear output
- Other hyperparameters: learning rate `1e-4`, no regularization

To choose the number of units in each layer, I performed a coarse grid search, trying values of 150, 200, 250 units for each layer (prior to the grid search, I had done some preliminary testing and had been using 200 units for each layer). The grid search found that the best values were 200 units for the first LSTM layer, 250 units for the second LSTM layer, and 250 units for the fully-connected linear layer. Note that the search was performed on the relative prediction task, but the found parameters were used for the final models of the other two tasks as well.

5 Conclusion & Future Work

This project has shown that recurrent neural networks are surprisingly effective in predicting the price of Ethereum. Exploration of various feature and model configurations yielded two important insights.

First, natural language features extracted from Twitter did not improve performance. This suggests that the extracted features do not have predictive value, which could mean that Ethereum prices are less susceptible to media influence than one might assume. Second, augmenting training data with other cryptocurrencies improves performances significantly. This shows that different cryptocurrency markets move in similar patterns.

Despite the inefficacy of the natural language features used in this project, there are likely other, price-unrelated features that one could use to help price predictions. Further work is needed

to discover what these features may be, and the extent to which they can help performance.

With regards to the three prediction tasks tested in this project – the models' performance on the binary and regression task left much to be desired. For the binary task, it would be valuable to compare the performance of a model trained using an accuracy-based loss function, as opposed to cross-entropy.