CS230

Predicting Cryptocurrency Prices with LSTMs

Pascal Cevaer-Corey, *pascalcc@stanford.edu*
Geza Kovacs, *gkovacs@stanford.edu*
Jonathon Walters, *waltersj@stanford.edu*

Abstract

In a world where cryptocurrencies have ballooned in value over the past couple years, investors have been scrambling to find ways to capitalize on this incredibly volatile and unpredictable market. Algorithmic trading is already at the core of many investment firms and is a relatively well documented space. However, due to the novelty of the crypto craze, this technology has not seen much application to trading cryptocurrencies. We hope to use a long short-term memory, LSTM, network trained on 7 months worth of BTC orders to predict future BTC prices, allowing traders to build a strategy around the model's output.

1 Introduction

The general problem we are investigating is algorithmic management of cryptocurrency portfolios. This is interesting from the perspective of algorithmic trading, as cryptocurrency markets are unregulated and highly volatile, with low liquidity on certain trading pairs making them subject to market manipulation tactics [3], so traditional algorithms may perform poorly on cryptocurrency markets. Existing algorithms for cryptocurrency trading have only used inputs such as historic prices [2], whereas APIs for cryptocurrency exchanges provide a wider breadth of data such as complete order book information and a history of executed buy/sell orders [4], which we believe can be invaluable for making more accurate predictions (particularly with regards to trends caused by market manipulation).

The challenges of this project are outperforming existing, more traditional trading algorithms, which due to the highly profitable nature of the problem is an extremely well explored space. An additional challenge is that cryptocurrency markets are currently in the midst of a downturn (compared to the bull market in 2017) and are increasingly subject to external market forces such as government regulation and futures markets, which may cause cryptocurrency trading algorithms that have historically performed well to no longer perform as well.

2 Related Work

Due to the novel nature of cryptocurrency trading, there is very little publicly available work on the applications of deep learning to cryptocurrency trading. There is however a substantial amount of research around how deep learning can be applied to other financial exchanges such as equities and commodities. This is due to the massive amount of historical data available as well as the financial incentives aligned with building a model that can correctly predict market movements. Work in this space ranges from relatively simple two hidden layer neural nets [5] to deep reinforcement learning [6].

Research attempting to predict price movement predates the year 2000 [7]. Until recently, the majority of published papers attempted to train relatively shallow neural nets with historical pricing data to accurately predict whether a security's price will increase or decrease. The success of this research was less so reliant on the ability of their models and more so reliant on whether or not the researchers decided to test their models on highly volatile and historically unpredictable securities or more stable ones (such as commodities like copper and gold). This is not to say that this work was unsuccessful, it did product wonderful results for certain sleepier securities, but when thinking about applying deep learning to cryptocurrencies, looking to the architecture of models that do not handle volatility well was not an option.

Much more recently, teams have been focused on using recurrent neural networks, RNNs, and more specifically long short-term memory networks, LSTMs, to successfully predict more volatile price action [8], [9]. These models are trained on similar data to that of the older models but the new architecture allows for the models to better keep track of long term trends and cycles which tend to dictate economic action and therefore price action. These models can make use of a relatively short, recent window of information to predict security prices in the next time step, which has allowed for actionable trading strategies and even real-time day trading based on model output. Due to their ability to better handle market trends and therefore more volatile securities, we based our model on literature about LSTM applications in algorithmic trading rather than the application of simpler neural network architectures.

3 Dataset and Features

We used the CBOE Gemini Order Books dataset over the period of 10/08/2015 to 02/20/2018 [1]. This dataset includes over 13,000,000 entries which each represent anytime a BTC order was placed, filled, or canceled. The fields included in this data are: Event ID, Event Date, Event Time, Event Millis, Order ID, Execution Options, Event Type, Symbol, Order Type, Side, Limit Price (CCY2), Original Quantity (CCY1), Gross Notional Value (CCY2), Fill Price (CCY2), Fill Quantity (CCY1), Total Exec Quantity (CCY1), Remaining Quantity (CCY1), Avg Price (CCY2).

Given high volatility in trading volume, the density of order book events that occurs within some time frame is also highly variable. To simplify this data, we preprocessed the order book to exist in minute-by-minute resolution, whereby various metrics (see Table 1) were computed from all order book events that occurred within a particular minute (see Appendix for example time series of each feature over a single day). Once this had been done, we derived five classes of features: price, buy, sell, buy vs sell, and time. Time features are all hot-one encoded and buy, sell, and buy vs sell features are calculated as $x/(x+y)$.

Price	Time
-Percent change in price	-Month of year -Day of week -Hour of day
Buy & Sell	**Buy vs. Sell**
Buys: -Volume of filled markets vs filled limits -Volume of placed limits vs filled limits -Frequency of filled markets vs filled limits -Frequency of placed limits vs filled limits for each category **Sells:** -Volume of filled markets vs filled limits -Volume of placed limits vs filled limits -Frequency of filled markets vs filled limits -Frequency of placed limits vs filled limits for each category	-Volume of filled market sells vs volume of filled market buys -Volume of placed limit sells vs volume of placed limit buys -Frequency of filled market sells vs frequency of filled market buys -Frequency of placed limit sells vs frequency of placed limit buys -Volume of cancelled limit sells vs volume cancelled limit buys -Frequency of cancelled limit sells vs frequency of cancelled limit buys

Table 1. Extracted minute-by-minute order book features.

We split the dataset chronologically, by placing the first 90% into the training set and having the next 5% be the dev set, and the final 5% be the test set. We chose to split between train and dev/test chronologically instead of randomly because that is how the model would operate in the real world (training on what has already happened and predicting the future), and chose not to randomize the splits to preserve the chronological order of the data (price histories are sequential in nature -- just like we do not want to randomize the order of words in a text corpus, we would not want to randomize the order of prices for our problem).

4 Methods

We constructed two baseline models:

1. Predict the next price as the current price (i.e., 0% change in price)
2. Predict the next percentage change in price as the current percentage change.

As for our model, to determine the architecture and other hyperparameters we performed a grid search over the following sets of values:

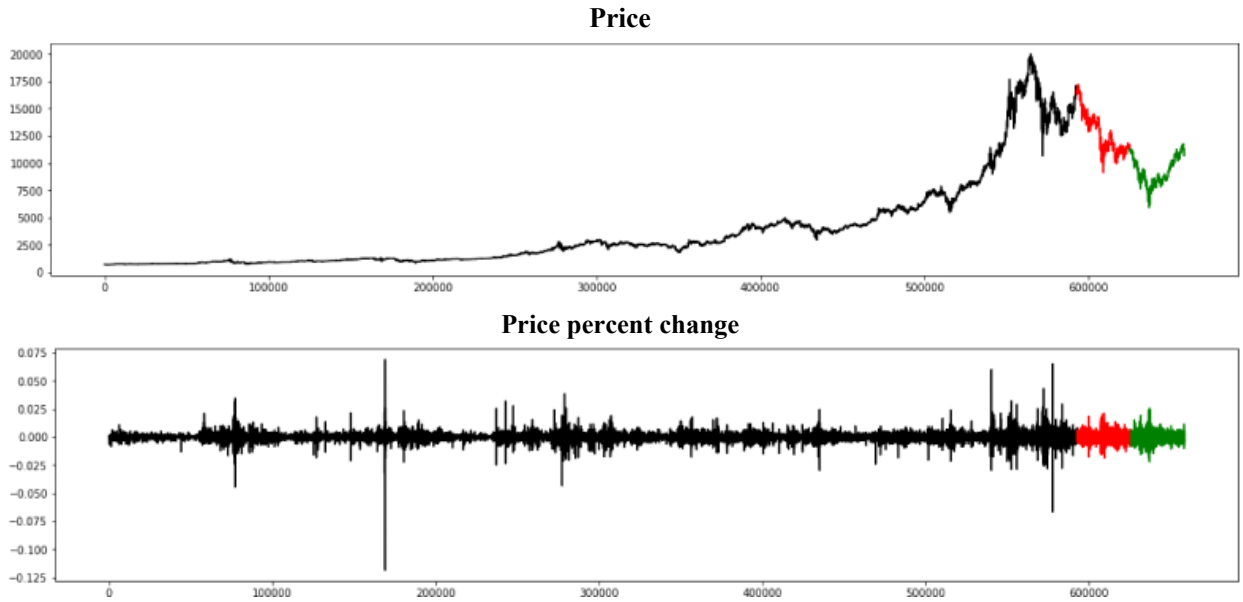
optimizers = ['adagrad', 'adam', 'rmsprop'],
batch_size = [1024, 4096, 8192],
num_LSTMs = [2, 3],
num_units_2 = [[128, 256], [256, 256]],
num_units_3 = [[128, 256, 256], [256, 256, 256], [256, 512, 512]],
window_size = [30, 60] ,
temporal_features = [True, False],
And predicting the next percent change in price at :
each timestep within the window or *at the end of the window.*

The variables we held constant were:

loss = 'mean_squared_error'
learning_rate = 0.01

num_epochs = 30
dropout = 0.1
window_step = 1

Examples from the grid search is shown in Table 2. Our final model is a 3-layer LSTM with 128, 256, and 256 units in each layer and a final fully connected layer, with a window size of 60 minutes, no temporal features (i.e., month, day of week, hour of day), and Adam optimized. The shapes of our train, dev, and test data were, in the format of [number of minutes, number of minutes within sliding window, number of features], the following: X_train: (592823, 60, 15), X_dev: (32879, 60, 15), X_trest: (32879, 60, 15), Y_train:(592823, 60, 1), Y_dev: (32879, 60, 1), Y_test: (32879, 60, 1).



Figs 1 and 2. Splits of training data (black), dev data (red), test data (green).

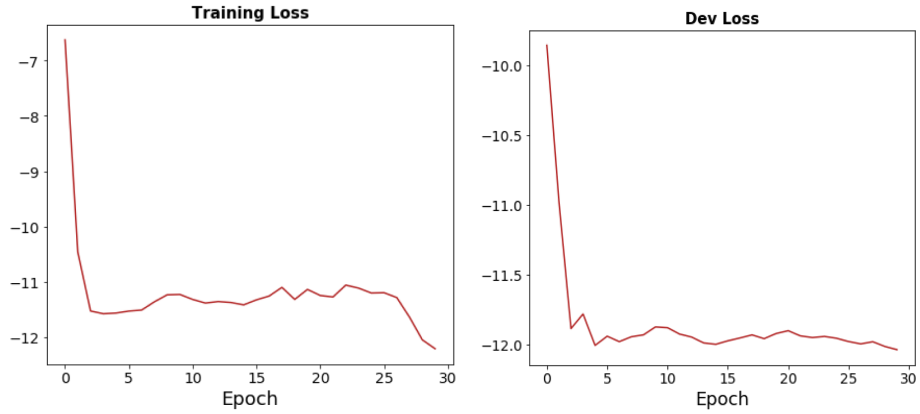
We used mean squared error (MSE) as our loss function:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

,

with RMSE and proportion of correct direction prediction as additional evaluation metrics.

5 Results



Figs 3 and 4. Training Loss and Dev Loss

Predict throughout or end of window	Temporal features	Window Size	Num LSTMs	Loss (Train)	Loss (Dev)	Loss (Test)	Direction Accuracy (Train)	Direction Accuracy (Dev)	Direction Accuracy (Test)
Throughout	No	30	Two	1.690e-6	5.907e-6	5.893e-6	0.256	0.468	0.470
Throughout	No	60	Three	1.673e-6	5.79e-6	5.865e-6	0.250	0.465	0.458
Throughout	Yes	30	Two	1.925e-6	1.043e-5	1.18e-6	0.240	0.456	0.453
Throughout	Yes	60	Three	4.959e-6	5.909e-6	5.97e-6	0.243	0.460	0.458
End of window	No	30	Two	1.694e-6	6.255e-6	6.31e-6	0.246	0.460	0.457
End of window	No	60	Three	1.799e-6	5.791e-6	5.94e-6	0.259	0.472	0.461
End of window	Yes	30	Two	1.822e-6	1.00e-5	1.09e-5	0.240	0.456	0.453
End of window	Yes	60	Three	1.718e-5	6.934e-5	6.988e-5	0.251	0.458	0.460

Table 2. Example results from our hyperparameter search.

The RMSE of our baseline models (see Appendix) was 0.0276 for predicting the same percent change in price and 0.0190 for predicting no percent change in price. The RMSE of our best model (row 2) was 0.0024, placing the performance of this model between the two baselines.

6 Conclusion/Future Work

In this paper we describe the implementation and training of an LSTM network for the purpose of predicting percentage change in BTC price from one timestep to the next. We found that the most effective method was to use minute-by-minute order data with input features in five categories (price, buys, sells, buys vs sells, and time). However, despite our model's ability to beat the baseline, the task of accurately predicting cryptocurrency price movement proved incredibly challenging and ultimately left us with a model that would most likely not be an effective tool for day trading cryptocurrencies.

Given more time we would have liked to experiment with other output schemes such as a classification that tells the user whether to buy if the price is expected to increase above a certain threshold without risk of it dipping below a certain threshold or vice versa for selling. We would have also liked to explore other possible inputs, either from a different dataset or the one we are currently using. Finally, we would have liked to look at how well this model does on other cryptocurrencies to understand the robustness of our model the key differences between how cryptocurrencies are traded, if any. Once achieving better results, our model could be directly connected to APIs on existing markets to generate profits.

Contributions

Geza: identified datasets, brainstormed architectures, determined baselines, and wrote keras code
Jon: identified datasets, brainstormed architectures, determines baselines, and wrote keras code
Pascal: brainstormed architectures, made poster, and wrote report

Code Repository

https://github.com/gkovacs/crypocurrency-trading

References

[1] https://datashop.cboe.com/cryptocurrency-gemini-order-book-data
[2] Jiang, Zhengyao, Dixing Xu, and Jinjun Liang. "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem." *arXiv preprint arXiv:1706.10059*(2017).
[3] Gandal, Neil, et al. "Price manipulation in the Bitcoin ecosystem." *Journal of Monetary Economics* (2018).
[4] https://docs.coinapi.io/#historical-data39
[5] Dixon, M., Klabjan, D., & Bang, J. (2017). Classification-based financial markets prediction using deep neural networks. *Algorithmic Finance*, *6*(3-4), 67-77.
[6] Maknickienė, N., & Maknickas, A. (2012, May), "Application of Neural Network for Forecasting of Exchange Rates and Forex Trading," in *The 7th international scientific conference Business and Management*, (pp. 10-11).
[7] Faraway J. , Chatfield C. , (1998) . Time series forecasting with neural networks: A comparative study using the air line data, Journal of the Royal Statistical Society: Series C (Applied Statistics) 47: (2), 231–250.
[8] Y. Deng, F. Bao, Y. Kong, Z. Ren and Q. Dai, "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 653-664, March 2017.
[9] Di Persio, Luca, and Oleksandr Honchar. "Artificial neural networks approach to the forecast of stock market price movements." *International Journal of Economics and Management Systems* 1 (2016): 158-162.
[10] Chollet, Francois, et al (2015). “Keras.” *GitHub.* https://github.com/fchollet/keras.

Appendix

========== BASELINES ==========
========= Training Data =========
Predict NO Percent Change
MSE: 0.000122
RMSE: 0.011056
Predict SAME Percent Change
MSE: 0.000241
RMSE: 0.015516
========= Dev Data =========
Predict NO Percent Change
MSE: 0.000366
RMSE: 0.019134
Predict SAME Percent Change
MSE: 0.000706
RMSE: 0.026566
========= Test Data =========
Predict NO Percent Change
MSE: 0.000363
RMSE: 0.019056
Predict SAME Percent Change
MSE: 0.000763
RMSE: 0.027616

Visualization of minute-by-minute order book features for 02/20/2018: