CS230

Predicting Bitcoin Price Trends

Cindy Jiang
Department of Computer Science
Stanford University
cindyj@stanford.edu

Lucas Lin
Department of Computer Science
Stanford University
lucaslin@stanford.edu

Orien Zeng
Department of Computer Science
Stanford University
ozeng@stanford.edu

Abstract

Bitcoin algorithms that help predict the direction of price trends can help investors make important financial decisions. This paper tries to predict the direction of Bitcoin prices in the near future. The problem is simplified into a binary classification problem of whether the price increases or decreases. The three component model used consists of a Fourier transform, stacked autoencoder (SAE) and long-short term memory (LSTM). This model achieves an accuracy of 69.8%.

1 Introduction

Financial markets have grown more prominent online and thus, produce a great amount of data for analysis. With so much data, the application of neural networks to search for patterns have become more common. As one of the most well-known cryptocurrency, Bitcoin has garnered much attention in the past decade. The problem of predicting Bitcoin price trends can aid buyers and sellers of Bitcoin in making decisions about whether to act.

For this project, the task is simplified to predicting the direction that the Bitcoin price trending. The input to our algorithm is a time series of Bitcoin data at 1-minute intervals, each data point containing two features: the weighted price and volume of exchanges. We then use a Fourier transform to denoise the input, feed it into a stacked autoencoder to generate features and finally, run it through an LSTM to make the final binary classification prediction. The output is a prediction of whether the price will increase or decrease in the next 100 minutes.

The described algorithm is called WSAEs-LSTM because the original paper by Bao *et al.* [1] used Wavelet transforms, stacked autoencoders (SAE) and LSTMs. Since we used a Fourier transform insted of a Wavelet transform, this paper will be calling the architecture FSAEs-LSTM.

2 Related work

Our project is based on the WSAEs-LSTM architecture described in Bao *et al.* [1], which aims to predict stock market prices. Instead of using a wavelet transform for denoising, we use a Fourier transform instead. We also referenced a paper with a similar classification task by McNally *et al.* [2]. McNally *et al.* compares the accuracy of an RNN and LSTM in order to predict the direction of Bitcoin price and their results were an informative benchmark.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Dataset and Features

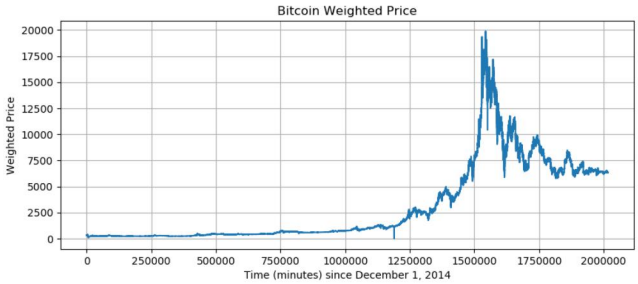


Figure 1: Visualization of the Bitcoin Historical Data Kaggle dataset

We use data from the Bitcoin Historical Data Kaggle dataset [3], which contains bitcoin pricing data at 1-minute intervals and spans from December 1, 2014 to January 9, 2019. In total, there are around 2 million data points where data point consists of seven features, but for our project we only use the weighted price and volume. Figure 1 shows the weighted price of bitcoin over the entire dataset timeseries. There is a large left skew in the graph, with the rising and falling popularity of Bitcoin in the recent years.

For intervals during which no bitcoin was traded, the dataset is populated with NaN values. These values consisted of approximately 5.4% of the dataset. We preprocessed the dataset by discarding densely populated NaN areas and replacing NaN values in sparsely populated areas with interpolated prices between the previous and next known price values. For interpolated prices, we also set the volume to 0. The prices in the dataset are then rescaled by subtracting the mean price across the whole dataset and then downscaling by a constant factor.

The dataset was split 70%-15%-15% to make our training, dev and test sets. Originally, we randomly shuffled the training data right before splitting the train/dev/test sets for the LSTM. However, this does not capture the temporal nature of future test data in real life and might make it easier to memorize the dataset and interpolate values onto the test set. Thus, the splits are done sequentially to mimic training on "past" data and testing on "future" data. This also causes an imbalance in the training, dev and test distributions, which might hinder the learning process.

4 Methods

We used a CNN for our baseline and combined a Fourier transform, stacked autoencoder and LSTM for our model. For both architectures, we use Adam optimizer and binary cross entropy loss for optimization and chose our metric to be accuracy.

4.1 Baseline

We adapted our baseline CNN model and parameters from the architecture described by Tsantekidis *et al.* [4] and used Algorithmia [5] as a Pytorch reference. The architecture is a three layer CNN with hidden layer sizes 16, 16 and 32. This feeds into a maxpool layer and two linear layers that outputs 16 values and 1 value, respectively. The final output is put through a sigmoid function.

Simple hyperparameter tuning for learning rate gives a value of around 4.6e-5. After 100 epochs, the accuracy is around 60%, demonstrating that a neural network can beat random guessing. It is possible that more rigorous tuning including architecture hyperparameters could increase the performance of the baseline.

4.2 Fourier transform + stacked autoencoder + LSTM

The first procedure in this model is the Fourier transform. The Fourier transform computes the frequency content at various bins, which assists in denoising. In addition to the Fourier transform output, we include phase information by applying the arctan function to the real and imaginary components of the Fourier transform. This phase information may be used to determine whether a sine wave is increasing or decreasing at a given time segment.

Then, we trained a three-layer stacked autoencoder to extract high-level features that represent the data in a more condensed vector. Each layer's output dimension has half the dimensionality of the input, with sigmoid activations. Each layer was trained individually, with the loss corresponding to that layer's reconstruction loss.

Finally, the dataset has a timeseries component, so using an LSTM captures temporal patterns in the data. The input to the LSTM is ten consecutive data points stacked to create ten timesteps in a series. The output is whether the price goes up or down after one minute from the end of the time series. The LSTM feeds into a linear layer that outputs one value which gets put through a sigmoid function for classification. An output value above 0.5 was considered a positive label (increasing price), whereas a lower value was considered a negative label (decreasing price).

5 Experiments/Results/Discussion

5.1 Hyperparameter tuning

Hyperparameters	Dev Accuracy	Dev Loss	Training Accuracy	Training Loss
LR: 3.2e-5, Dropout: 0, Epochs: 200, # layers: 1	72%	0.64	77%	0.61
LR: 1.3e-4, Dropout: 0, Epochs: 100, # layers: 2	71%	0.65	81%	0.61
LR: 3.6e-4, Dropout: 0.2, Epochs: 100, # layers: 2	69%	0.61	88%	0.39
LR: 1e-3, Dropout: 0, Epochs: 100, # layers: 2	69%	0.73	93%	0.26
LR: 2.8e-5, Dropout: 0, Epochs: 400, # layers: 1	68%	0.64	82%	0.56

Table 1: Hyperparameter tuning: learning rate (LR), dropout, # of epochs, # of LSTM layers

The hyperparameters in the Table 1 are the five that yielded the highest dev accuracies. We automated the hyperparameter search for learning rate and dropout with a grid search strategy. The learning rate search tried ten values in logspace from 1e-1 to 1e-6, then narrowed down to search ten more values from 1e-3 to 1e-5. The dropout value was picked between the values 0, 0.2 and 0.5.

The number of LSTM layers (1, 2, or 3) and number of hidden units per layer (3 or 5) were manually tuned, then the learning rate and dropout were re-tuned for the top configurations. We also used early stopping for each run to train until the epoch when the accuracy is approximately the highest. The mini-batch size was 50, which is based on the mini-batch size of 60 in Bao *et al.*'s architecture [1].

5.2 Results

Each model is run five times and the average metrics are calculated. The average accuracy, precision, recall and F1 scores are displayed in Table 2.

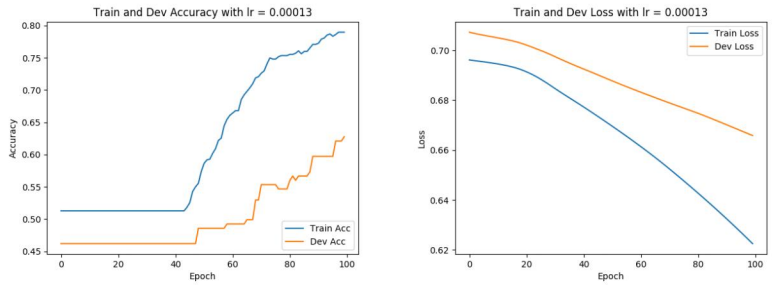
The top two hyperparameter sets found during tuning are used.

- Hyperparameter set #1 is 3.2e-5 learning rate, no dropout, 200 epochs and 1 LSTM layer.
- Hyperparameter set #2 is 1.3e-4 learning rate, no dropout, 100 epochs and 2 LSTM layers.

Model	Accuracy	Precision	Recall	F1
Baseline CNN	60.8%	66.7%	**72.82%**	64.78%
FSAEs-LSTM with hyperparameter set #1	69.4%	**74.24%**	63.64%	67.22%
FSAEs-LSTM with hyperparameter set #2	**69.8%**	70.88%	65.44%	**67.82%**

Table 2: Results from baseline and LSTM models

The best performing model is FSAEs-LSTM with hyperparameter set #2, achieving an accuracy of 69.8%. Figure 2 shows the accuracy and loss curves for a run of this model.



(a) Accuracy curves

(b) Loss curves

Figure 2: Accuracy and loss curves for the training and dev sets.

Overall, FSAEs-LSTM performs better than the baseline, though the baseline CNN seems to do better when it comes to recall. The two hyperparameter sets achieve pretty similar results and the minor differences can be attributed to noise.

5.3 Discussion

Both the baseline and FSAEs-LSTM were able to exceed the 50% expected accuracy from random guessing, and FSAEs-LSTM achieved about 10% higher accuracy than the baseline model. It is important to keep in mind that there might be a bias because more effort was spent on optimizing the performance of the FSAEs-LSTM model.

Our results are not comparable with the results found by Bao *et al.* because we are not making price predictions, rather we are solving a classification problem. Therefore, it would be better to align our results with a more similar problem, like the one posed by McNally *et al.* [2]. Their problem slightly differs from ours in that there are three classes; in addition to price increasing and decreasing, they also have a no change in price class. We expect our accuracy to be better since our task is slightly easier and we are applying a more complex model to solve the problem. Indeed, we find that the 69.8% FSAEs-LSTM accuracy beats the 52.78% vanilla LSTM accuracy presented in the paper.

The main problem faced by FSAEs-LSTM is overfitting. Without any regularization, the dev set tends to perform much worse than the training set and the dev loss starts to increase after the first hundred epochs. The reason for the overfitting is most likely that the dataset is very small. We make the dataset even smaller by grouping together one hundred data points as an input to the Fourier transform, then later stacking groups of ten data points to make a time series input for the LSTM.

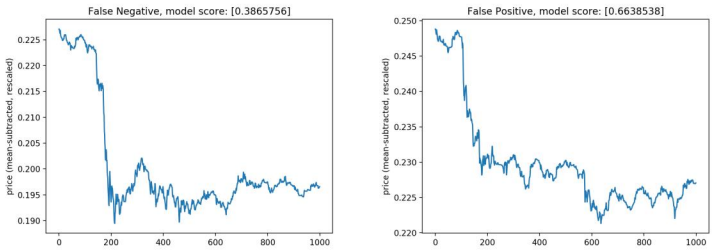
We attempted to decrease the overfitting by adding dropout and L2 regularization as well as adjusting the model architecture by decreasing the number of hidden layers or LSTM layers. These techniques helped to an extent, but they tended to decrease the training accuracy and did not increase the dev accuracy as much.

5.3.1 Analysis

To start, we analyzed whether the training/dev/test split has a good distribution. The training set is split into 48.689% 1's and 51.311% 0's labels. Both the dev and test sets are split into 48.246% 1's and 51.754% 0's labels. Thus, the labels are similar and relatively balanced, with slightly more 0 labels than 1 labels.

Next, we looked at the confusion matrices for multiple runs. Out of 10 runs, we found that the model was predicting false negatives more 7 out of 10 times. This could be due to the slight bias in the labels, since there are more negative than positive labels.

Unfortunately, closer inspection into the dataset in Figure 1 reveals that there is a long tail in the first half of the dataset where the price is relatively low. Due to the way we split our dataset, this data is all allocated into the training set. In addition, there is a peak in price that is primarily split into the dev set, making the dev and test distribution uneven. We decided to proceed with the sequential split of



(a) Actual label: increasing, Predicted label: decreasing

(b) Actual label: decreasing, Predicted label: increasing

Figure 3: Examples of similar inputs with incorrectly predicted outputs, where negative is decreasing and positive is increasing.

data in order to portray real world conditions, but realize that the results would benefit considerably with more data, especially in the past year.

We also attempted to perform error analysis by visualizing the input on a normalized price vs. timestep together with the predicted and actual label. However, it is very difficult to tell if there are patterns that influence the error by analyzing the graph. This is most likely because the input to our FSAEs-LSTM model receives many features outputted by the Fourier transform and stacked autoencoder in addition to the price, but visualizing all of the features together was uninterpretable as well.

Figure 3 shows two similar inputs that our model predicted incorrectly. In both figures, the bitcoin price decreases over time. There are no clear differences between the false positive and false negatives in this batch.

6 Conclusion/Future Work

Two architectures were implemented: a CNN baseline and a FSAEs-LSTM model that consists of a Fourier transform, stacked autoencoder and LSTM. The CNN baseline achieves 60.8% accuracy and the three component LSTM model achieves 69.8% accuracy. FSAEs-LSTM is more complex, utilizing a Fourier transform to denoise the input and an autoencoder to condense the data to get features. FSAEs-LSTM probably performs better due to the extraction the data's most informative features. However, the small size of our dataset led to overfitting by the FSAEs-LSTM model so addressing the overfitting issue better can improve the performance.

6.1 Future work

A major limitation of the dataset used is the number of training examples given. There are about 2 million rows in the dataset, some of which contain NaN values. During hyperparameter training, the model was prone to overfitting on the training dataset, and had runs that achieved up to 99% accuracy on the training dataset compared to $\sim$68% on validation when no dropout was used. Although we attempted to add regularization, additional methods to alleviate this are to use more features from the original dataset, augment the dataset with data from other bitcoin exchanges, or apply transfer learning from a model trained on a larger financial market.

Another avenue for future work is to incorporate the wavelet transform as an alternative to Fourier transform. The wavelet transform does not assume a stationary signal and also incorporates time-series information, which may allow a model to better understand fluctuations in the financial signal [6]. Bao *et al.* applied wavelet transform and it worked well for denoising their dataset.

Finally, the results would benefit from profitability analysis. In order to interpret the value of the model, it is important to understand whether the model is useful in a buy vs. sell strategy. This might also help with interpretation during error analysis to pinpoint possible patterns in the data.

7 Code and Contributions

Github repo: https://github.com/ofzeng/cs230earthquake
Orien: Preprocessing data, Fourier transform, stacked autoencoder
Cindy: Baseline CNN and LSTM implementation, manual hyperparameter tuning
Lucas: Automate hyperparameter tuning, dataset analysis

References

[1] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[2] Sean McNally, Jason Roche, and Simon Caton. Predicting the price of bitcoin using machine learning. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 339–343. IEEE, 2018.

[3] Zielak. Bitcoin historical data, Mar 2019.

[4] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Forecasting stock prices from the limit order book using convolutional neural networks. In *2017 IEEE 19th Conference on Business Informatics (CBI)*, volume 1, pages 7–12. IEEE, 2017.

[5] Algorithmia. Convolutional neural nets in pytorch, Dec 2018.

[6] Philippe Masset. Analysis of financial time-series using fourier and wavelet methods. 2008.

[7] Jessica Yung. Lstms for time series in pytorch, Sep 2018.