Bitcoin Price Prediction and Trading Strategy with Deep Learning

Xuejiao Li (xjli1013), Holly Liang (xuejiao), Taiming Zhang (tzhang55)

Abstract—**This project aims to build an accurate and automated system that is capable of predicting Bitcoin Price Percentage Change. Using 622,641 pieces of data from Meltwater, we have implemented different architectures using different machine learning algorithms. Our models for price prediction include: LSTM and CNN. Our Model for Bitcoin Trading Strategy include: Approximate Q Learning and different LSTM-based models. Our Variable model with 8-class LSTM Model proves to outperform the other models and can achieve an average of 14.5% profit annually.**

Keywords-Bitcoin, Price Prediction, Trading Strategy, LSTM, CNN, Approximate Q Learning

I. Introduction

Bitcoin is the leading cryptocurrency which is used worldwide for digital payment or simply for investment purposes. Transactions made by Bitcoins are easy, since they are not tied to any country. Bitcoins can be 'mined' by solving/computing complex cryptographic/mathematical puzzles. One of the most interesting topic with regard to Bitcoin is its price prediction [1]. The goal of price prediction is to ascertain with what accuracy can the direction of Bitcoin price in USD be predicted. With the huge trading volume, it makes sense to think of it as a proper financial instrument as part of any reasonable quantitative trading strategy. And based on the price prediction method, we can further devise profitable strategy for trading Bitcoin.

In this report, our interest is to understand whether there is "information" in the historical data related to Bitcoin that can help to predict future price in Bitcoin and thus, to use our prediction to develop profitable quantitative Bitcoin trading strategy.

We describe related work in Section 2, our Bitcoin Dataset in Section 3, Bitcoin Price Prediction in Section 4. The Trading Strategy follows in Section 5. Section 6 closes with our conclusions.

II. Related Work

Time series prediction is not a new phenomenon and prediction of mature financial markets such as the stock market has been researched at length [1,2]. Bitcoin presents an interesting parallel to this as it is a time series prediction problem in a market still in its transient stage. Traditional time series prediction methods such as Holt-Winters exponential smoothing models rely on linear assumptions and require data that can be broken down into seasonal trend [3]. This type of methodology is more suitable for a task such as forecasting sales where seasonal effects are available. Due to the lack of seasonality in the Bitcoin market, this method is not very effective for this task. Shah et al. [4] implemented a latent source model as developed by Chen et al. [5] to predict the price of Bitcoin. The model received an impressive 89 percent return in 50 days. Matta et al. [6] investigated the relationship between Bitcoin price, tweets and views for Bitcoin on Google Trends. The author found a weak to moderate correlation between Bitcoin price and both positive tweets on Twitter and Google Trends views. The author found this to be proof that they can be used as predictors. However, one limitation of this study is that the sample has only 60 days in size.

Considering the complexity of our task, deep learning makes for a potentially more powerful technological solution. Tasks such as natural language processing which are also sequential in nature have shown promising results [7]. This type of task uses data of a sequential nature and as a result is similar to a price prediction task. The recurrent neural network (RNN) and the long short term memory (LSTM) flavor of artificial neural networks are favored over the traditional multilayer perceptron (MLP) due to the temporal nature of the more advanced algorithms [8]. Thus, we are going to use Long Short Term Model (LSTM) as our model for predicting Bitcoin Price. And based on our price prediction model, we will further dive into building our Bitcoin trading strategy.

III. Dataset

In this project, to perform experiments, we have used data related to price and other information from Meltwater. The dataset contains time period ranging from Jan 1, 2016 to May 29, 2018 and information about bitcoin such as open, close, high, low prices and volumes for each minute. We have also considered features from other cryptocurrency such as ETH, BCH, and LTC that might help improve our performance. For the purpose of computational ease (we are interested in predicting bitcoin close price percent change for next minute), and other issues such as breakpoints (time discontinuity) and outliers, we have constructed new continuous time series with time interval of minutes. We will address those issues during data preprocessing in Section 4a and 5a.

IV. Price Prediction

Our goal for this price prediction task is to predict the next minute close price percent change of Bitcoin based on the previous 60-minute historical information. Since price percentage change is a continuous feature, our first intuition is to treat this task as a regression problem; and after some careful study and some experiments (described in details 4b and 4b), we decide to continue to change the problem into a classification task to gain better performance (described in details 4b and 4c).

A. Data Handling and Preprocess

We first sort the feature information in ascending time order and we maintain a 60-minute sliding window that crops the data into examples. For each examples, we used the first 60-minute information as our training input and the last minute close price percent change as our label. The final training data structure is shown in Fig.1. The sliding window went through the whole dataset and we generated 622,641 samples.

For each sample, instead of keeping the actual bitcoin prices as input, we have tried several different normalization methods, such as Log Normalization, Standard Deviation Normalization, Sliding Window Normalization, etc. The Sliding Window Normalization proves to outperform the other methods and we finalized to normalize the price and volume feature as described in Equation [1], where pi is the actual price or volume at time i. We convert the actual prices in consecutive 60 minute as percent changes with respect to the information in the first minute. As a result, the model becomes invariant to the magnitude of actual prices and will be able to capture the percent changes information within the hour. We split samples into training and validation sets (60% and 20% for the first 80% data), and the last 20% data are used as test set.

$$n_i = \frac{p_i}{p_0} - 1$$

Equation [1]: Feature normalization

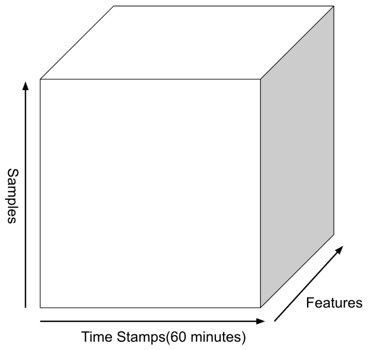

Figure 1: Data Input Structure for LSTM

B. Model

i) Baseline (Moving Average)

Since our goal is to use previous 60-minute information to predict the next minute close price percent change, in order to make baseline comparable, one reasonable baseline is to firstly average the previous 60 minutes' close prices and use this averaged price to calculate the price percentage change.

ii) LSTM

One commonly used method in deep learning that address the sequential data problems is Long Short Term Memory (LSTM) whose inner structure is shown in Fig. 2, an variant of RNN. LSTM is best suited for time-series prediction problems that solves the long-term dependencies issue, and cryptocurrency price prediction falls into that category. LSTM is a special gated structured RNN in which a "forget gate" controls information to forget, and the output then loops back to itself that updates its state. In our price prediction model, we applied a 2-layer LSTM both with 512 nodes followed by an output layer. For regression task, the activation function is linear with mean squared error as loss function; for classification task, the activation function is softmax with categorical cross entropy loss. The LSTM model schematics is shown in Fig. 3. We applied both Adam and SGD with learning rate 0.001 as our optimizer and results showed that SGD optimizer outperforms Adam optimizer.

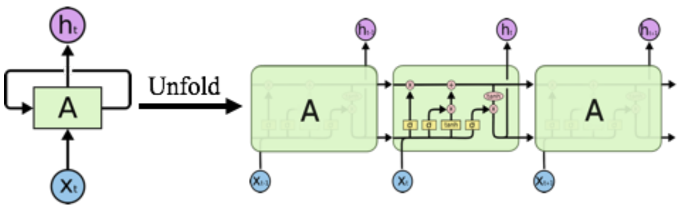

Fig. 2: LSTM inner structure [9]

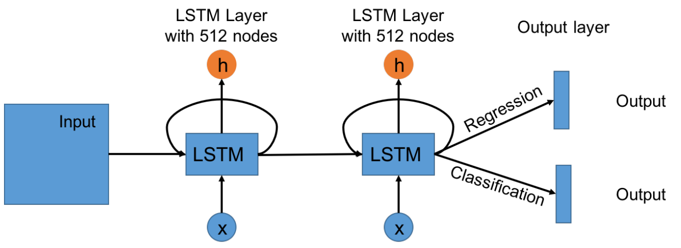

Fig. 3: LSTM Model Architecture

iii) CNN

As an alternative to LSTM, we've attempted to use Convolutional Neural Networks (CNN) to generate a model. The topology that we used consists of a 3-layer CNN with 64, 128 and 256 nodes respectively, followed by a 2-layer fully-connected neural network with 256 and 8 nodes respectively. The detailed model architecture is illustrated in Fig. 4 below.

We initialize the weights using Xavier initialization, a commonly used method that ensures that the weights are randomly initialized to values that are neither too large nor too small.

We then compared the performance of using Adam optimizer vs stochastic gradient descent (SGD) and it turns out that SGD performs better. Over 90 epochs of training, the test accuracy rises from 0.2828 to 0.3315, making the CNN model comparable to that of LSTM.

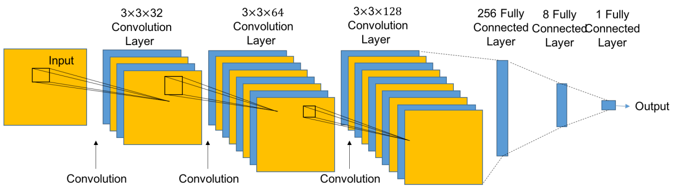

Fig. 4 CNN Model Architecture

C. Result and Discussion

For regression, we calculated the MSE over validation set, and performed a single prediction on our test set. Table 1 to illustrates the final MSE for test set using different models.

Table 1: Comparison between different models for regression

Model	MSE
Baseline	3943.74
LSTM	450.95
CNN	500.87

Our first observation is that LSTM and CNN models outperformed our baseline, by 774.54% and 687.38% respectively. It proves that both LSTM and CNN models are suitable for time series prediction tasks.

The second observation is that treating price prediction as a regression task is challenging. When we tried to decrease MSE with more complexed models (by adding more LSTM layers, fine-tuning hyperparameters, training more epochs), the overall MSE stayed high, indicating that we need to consider changing the regression task into a classification task. With this mindset, we then modified our dataset labels and separated data into different classes according to train data distribution shown in Fig. 5. We split data into different classes according to their price percent change, and we also made sure that the number of examples for each class is relative the same in order to prevent from overfitting due to class imbalance. For each class, we take the middle point as the predict percent change to calculate the relative MSE with respect to the actual percent change.

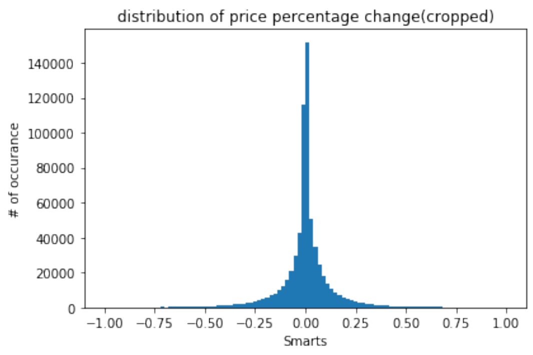

Fig. 5 Training Data Distribution

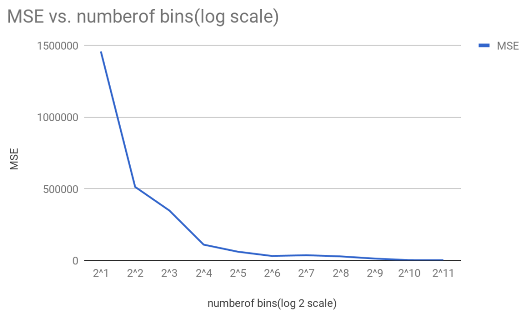

Fig. 6 MSE vs Number of Class

We began with the simplest case: 3-class model to predict whether the price will increase, decrease or stay the same based on the predicted price percent change. The experiment shows that we can achieve an accuracy of 47.9%. We continued to increase the number of classes and try to find an optimum number of classes to achieve the maximum performance possible.

As the number of classes increases, we observed a decrease in test-accuracy and a increase in MSE as illustrated in Fig. 6. This result is reasonable, since per-minute percentage change is relatively small, adding the number of classes will make the difference between each class becomes really hard to recognize, and at the same time, will decrease quantization error.

We also noticed the trade-off between bias and variance. If the number of class is too small, a complicated LSTM will make it easy to become overfit, creating too much variance; if the number of class is too large, it becomes really hard for LSTM model to accurately predict, creating too much bias. Nevertheless, after some careful tuning and experiments, we finalize to use 5-class, 9-class and 17-class as our model to further develop our trading strategy. And a detailed analysis to compare performance using different number of class models will be shown in Section 5c.

An example of our final price prediction using 9-class LSTM model is illustrated in Fig. 7. Apart from a few kinks, our predicted price broadly tracks the actual closing price for each

Bitcoin. This indicates that our Model for price prediction can perform relatively good performance.

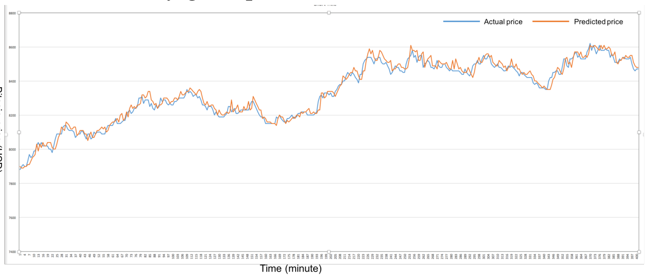

Fig. 7 Final Predicted Price vs Actual Price

V. TRADING STRATEGY

Assume we have 1 million initial cash (maximum amount of money we can spend), and our goal is to maximize our portfolios within an hour (it is based on our data preprocessing strategy described in section 5a). We use the last 1-hour per minute Bitcoin price as our "historical information" and based on different models, to make our decision to develop maximum profitable quantitative strategy for Bitcoin.

A. DATA HANDLING AND PREPROCESS

Our goal now moves to building a trading strategy; we would like to maximize our portfolio within one hour. Different from the preprocessing procedure in price prediction, we maintain a 120-minute sliding window that crops over the dataset without data overlapping as our samples. For each sample with 120-minute price and volume information, we maintain a 60-minute sliding window that starts from time 0 and moves to time 59 where we will generate 60 samples that will be fed into our LSTM price prediction model to predict prices for each minute in the last 60 minutes. The following sections will discuss in detail the methods we used to generate the optimal trading strategy.

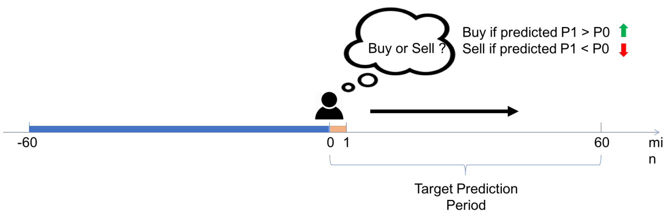

Fig.8 Trading procedure illustration

B. METHODS

i) Baseline

In order to make our baseline as simple and intuitive as possible, we buy equal value of bitcoin (1/60 * initial cash value) at each minute within one hour. And at the end of this hour, we evaluate our in-hand Bitcoin value versus our initials to see if this strategy can profit or not. Due to the existence of transaction fees, this method may not work so well, given that the per-minute Bitcoin price won't increase that much, compared with the per-minute transaction fees. We present our experiment result and discussions in section 5c.

ii) Approximate Q-Learning

We implemented approximate Q-learning, a reinforcement learning technique in machine learning, to learn the trading strategy for Bitcoin. Q-learning involves the determination of Q-values, which can be thought of as the "quality" of an action for a given state, in order to produce the optimal policy. However, with approximate Q-Learning, Q-values are approximated by a linear combination of features, rather than exact definition of states as in standard Q-Learning. By approximating the Q-values with a weight vector w, we do not need to quantize the analog price results as was done with the other methods. Although this process will introduce some errors, it consumes much less memory and computation time, and can potentially perform better than quantizing action space into buckets and introducing quantization error. The algorithm for performing approximate Q-Learning is detailed below.

```
Approximate Q-Learning Algorithm

Initialize w arbitrarily
Repeat (for each epoch):
  Initialize s using bootstrap
  Repeat (for each step of epoch):
  Choose a from s using ε-greedy
  Take action a, observe r and s'
  difference = (r + γ *
  max_a'(Q(s',a')) - Q(s,a)
  Repeat (for each i in len(w)):
       w[i] <— w[i] + η * difference
  * f(s, a)[i]
  s <- s'
  until isTerminal(s)
```

In this method, we prepared our data the same way as in Section 5a. Then, we divided our dataset into training and validation sets by randomly sampling 10,000 data points. During training, we applied bootstrapping to obtain a random starting point from the training set for each of 2,000,000 epochs. Since we have 622,641 data points, we set the number of epochs much larger than the number of data points in order to increase the likelihood of learning as much as possible from the data set. However, since we cannot leverage the parallel computing power nor the GPU computing during training with our Python code, 2,000,000 epochs took more than 72 hours to complete. Thus, we did not train our model for more epochs.

During training, we tested our model on the validation set data and tune some hyperparameters. Here are the results with the different parameters:

We first tuned learning rate:

Learning rate η	1.00E-04	1.00e-3	2.00e-1
Average Profit in USD	6.02	5.79	NaN

When we set learning rate too high, such as η=0.2, it suffers from the exploding gradient problem. We also tried learning rate decay, and set the decay function such as $\eta = 1/\sqrt{N}$, but the results did not improve much. Thus, we settled on η=0.0001.

Next, we tuned the exploration factor ε:

Exploration factor ε	0.8	0.7	0.6	0.5
Average Profit in USD	5.19	6.01	2.23	-10.8

We also tuned the action space. By increasing the possible amounts, the investor can buy or sell, we're effectively increasing the action space. Though at the cost of more computation, we found that increasing the action space increased the average profit on the validation dataset. Figure 9 shows the profit we made during validation and test phases. In the validation set we made an average of around $10 with 3 actions, and $48 with 7 actions. However, the test set does not show as promising results. Our guess is the data discrepancy. Since bitcoin price varied much more this year than previous years, it's possible that our training does not fit the test set as well.

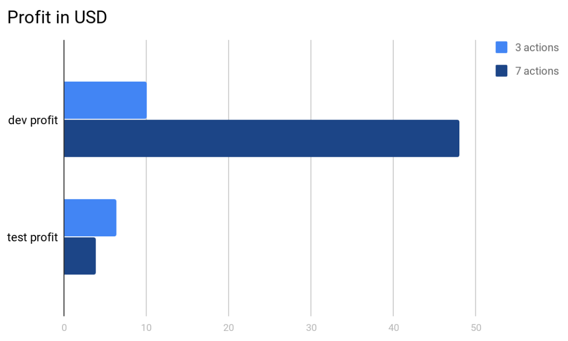

Fig. 9 Q-Learning Profit Validation set vs. test set

iii) LSTM based Strategy

The LSTM based trading strategy is intuitive: at each minute, we make a decision of either buy some amount of Bitcoin or sell some amount of Bitcoin. While training the LSTM model for classification, as we increase the number of classes, the number of examples in each class decreases except the middle one (percent change equals 0). Consequently, we saw strong overfitting issue due to the class imbalance. To resolve this issue, we decide to remove the middle class when percent change equals 0 and force our model to predict either rise or fall. At each time instance, here in our case, at each minute, we predict the next-minute price percentage change movement, say Δp, using our LSTM model (precise details explained in section 4b). If Δp > t, here t is a threshold, then we buy some bitcoin if our current cash amount still has money; if Δp < −t, then we sell some amount of bitcoin; else do nothing. The choice of time steps when we make trading decisions as mentioned above are chosen as per minute, since per hour/per day strategies have been carefully studied by many people and institutes, and our team want to take the challenge to dive into the per-minute case, to see if this more precise case can make better result. And we have carefully made different choices of threshold t and different amount to buy/sell Bitcoin to see which one works the best. The detailed cases are illustrated in Fig. 10. We present the simulated result and discussions regarding different hyperparameters in section 5c.

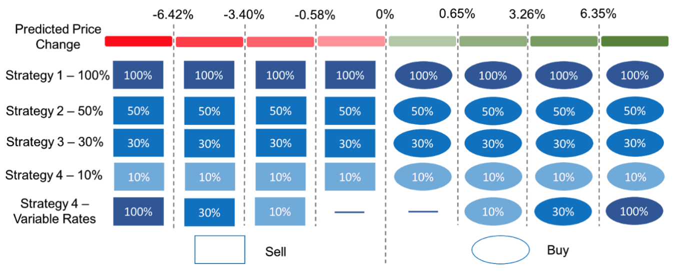

Fig. 10 Different Trading Strategy Actions and Decision Boundaries

C. *Result and Discussion*

We compare the performance of different methods against baseline with various transaction fees. Table 2 shows the maximum portfolio profit we can gain using different methods. And we illustrated our averaged result in Fig. 11, 12 and 13. For the standardized transaction fee, we see that our Q-Learning model, 10% model with 8-class, 30% model with 8-class, variable model with 8-class can all profit. Our best performance comes from our variable model with 8-class, which is able to averagely gain money of 18 USD within an hour. The 4-class LSTM model and 16-class LSTM model will lose money and cannot perform well. The reason is probably the trade-off between model complexity and accuracy. 4-class LSTM model is too simple, and our LSTM model has probably already overfit, thus cannot perform well on our test-set. 16-class LSTM model has too many number of classes and thus, our LSTM model cannot predict accurately enough to perform our trading strategy. And 8-class LSTM model is in the middle and can achieve the best result.

Fig. 14 shows a detailed behavior of our strategy within an hour. From the image we can find the model catches the point at time 29, where there is a price increase in reality, and decides to buy Bitcoin.

As the transaction fee decreases, we can observe in Fig. 12 and Fig. 13 a clear tendency of more benefit. Our final result shows that, our LSTM model can benefit with 14.5% Profit Annually, which is really awesome, considering the little price change within one minute.

Table 2: Maximum Profit Percentage within an hour using different models

Model	0.25% Transaction Fee	0.1% Transaction Fee	0% Transaction Fee
10% LSTM model	0.98%	1.43%	1.89%
30% LSTM model	1.57%	2.87%	3.98%
50% LSTM model	2.86%	3.42%	4.86%
100% LSTM	2.84%	3.41%	4.58%
Variable LSTM	9.59%	9.68%	10.57%

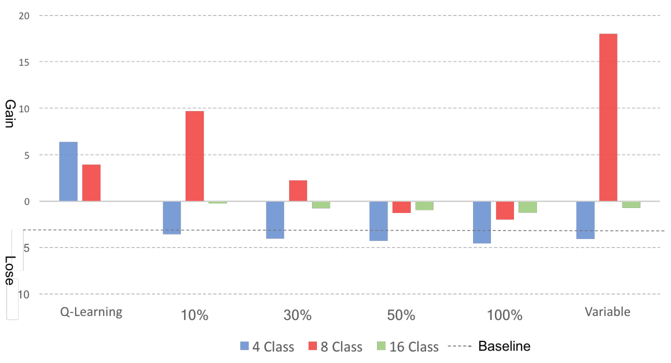

Fig. 11 Comparison Between Different Models with Transaction fee = 0.25%

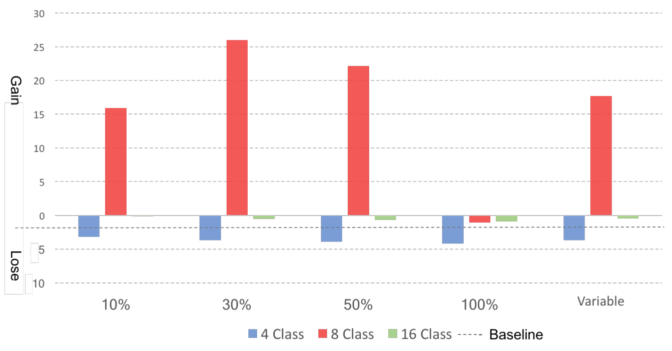

Fig. 12 Comparison Between Different Models with Transaction fee = 0.1%

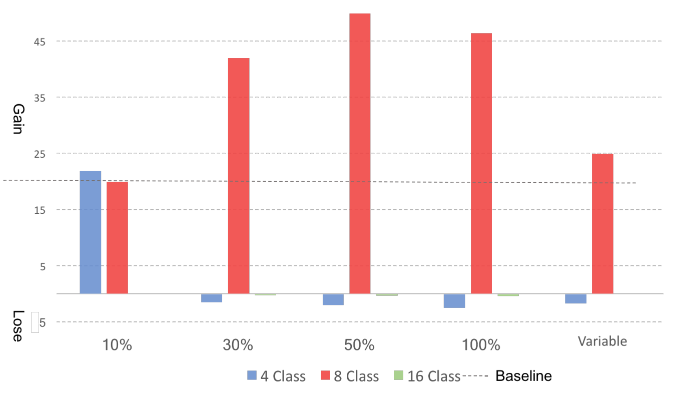

Fig. 13 Comparison Between Different Models with Transaction fee = 0%.

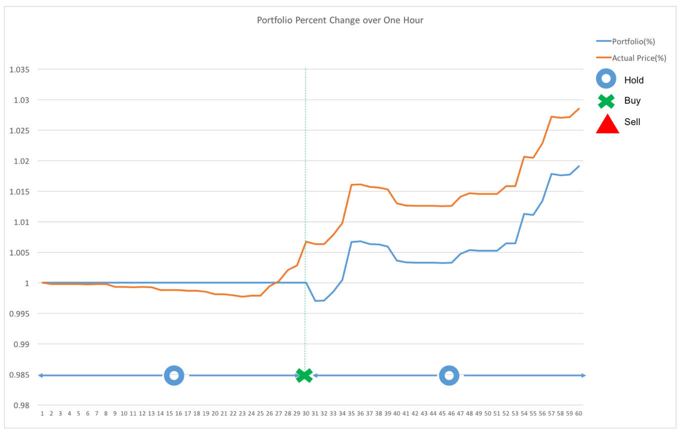

Fig. 14 Example of Portfolio and Actual Price Percent Change

VI. Conclusion and Future Work

We learned that Bitcoin price prediction is a very challenging task, especially using low frequency data to predict relatively high frequency price. Bitcoin price is very volatile, and very unpredictable given sparse information. Trading strategy is also hard to come up with due to the same reason. However, with our best model using LSTM, we can make 14.5% profit annually.

One thing we noticed while training our model is that there might be more things we can do in feature engineering. We need to extract more useful patterns from our data to make it easier for the machine learning models to perform its prediction. From an investigation of the behavior of consistent top performers in Kaggle data mining competitions, feature engineering is often the most important part leading to a good result. It is quite a subjective process requiring domain knowledge to be effective, and it is also considered an art. Engineered features should represent what one is trying to teach the network. For us, a few experiments have been done, but probably not enough.

REFERENCES

[1] Shah D, Zhang K. Bayesian regression and Bitcoin[C]//Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on. IEEE, 2014: 409-414.

[2] Kaastra I, Boyd M. Designing a neural network for forecasting financial and economic time series[J]. Neurocomputing, 1996, 10(3): 215-236.

[3] White H. Economic prediction using neural networks: The case of IBM daily stock returns[J]. 1988.

[4] Chatfield C, Yar M. Holt-Winters forecasting: some practical issues[J]. The Statistician, 1988: 129-140.

[5] Shah D, Zhang K. Bayesian regression and Bitcoin[C]//Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on. IEEE, 2014: 409-414.

[6] Chen G H, Nikolov S, Shah D. A latent source model for nonparametric time series classification[C]//Advances in Neural Information Processing Systems. 2013: 1088-1096.

[7] Matta M, Lunesu I, Marchesi M. Bitcoin Spread Prediction Using Social and Web Search Media[C]//UMAP Workshops. 2015.

[8] Karpathy A. The unreasonable effectiveness of recurrent neural networks[J]. Andrej Karpathy blog, 2015.

[9] Elman J L. Finding structure in time[J]. Cognitive science, 1990, 14(2): 179-211.

[10] Understanding LSTM Networks. (2015, August 27). Retrieved March 16, 2018, from http://colah.github.io/posts/2015-08-Understanding-LSTMs/