Machine Learning Nanodegree

Capstone Project

Michael Holberger
10/20/2018

I. Definition

Project Overview

Since the invention of computers, people have been engineering ways to use them to create an edge in investment markets. Being able to do calculations quickly, using current data enables traders to make educated trading decisions on when to enter or exit a market. Algorithmic trading has been a growing practice in recent decades as computational technologies become capable of employing high frequency trading strategies. Today, high frequency trading dominates public market trading.

Cryptocurrency markets democratize and accelerate this practice, in that many of the barriers for high-frequency trading in stock markets do not exist in the popular “alternative-coin” exchanges hosted on internet. Using the programmatic interfaces for these exchanges thousands of trades can be made daily, incurring minimal fees. Traders that employ systems (bots) to make split-second decisions have permeated these markets.

Ideally, an automated system can be created to gather current data, formulate predictions on market direction on which to trade, and execute those trades, all with minimal intervention required by the user. However, a system which is perfectly profitable in all conditions is highly unlikely, as the stochastic movement of price line data his highly unpredictable, and is the equivalent of what is known as a “random walk”. Many techniques for analyzing a market based on price data currently exist but no single technical analysis (TA) indicator can be relied upon alone. However, it may be possible to use some combination of indicators to supplement traders’ judgement. New, machine learning algorithms offer the possibility of dramatic improvements by analyzing multiple TA indicators in combination with market price and volume data. This project explores that possibility.

Problem Statement

This project uses machine learning techniques to attempt to predict the short term movement of certain cryptocurrency trading prices using multiple technical indicators, so that short term

trades can realize profits in volatile market conditions. A Long/Short Term Memory layer based recurrent neural network model was trained to make feed-forward predictions to forecast price movement during a specified number of timesteps in the future. The input dataset consisted of 5 minute grain price data that was preprocessed for several additional features including popular technical analysis indicators. This data was then divided into sequences of time frame windows which were normalized so that predictions could be made.

Metrics

Mean Squared Error (MSE) is used to compute loss for one-step-ahead predictions within each batch of time-sequences. MSE was chosen over Mean Absolute Error as it penalizes larger distances from the true value at a greater rate. MAE is more robust to outliers since it does not make use of the square, but for this application we would like to give greater weight to those outliers. For our model we are less concerned with smaller errors, as we do not expect to be able to predict the true data exactly. Other For each training epoch, the model trains through 40 separate sets of market data, divided into training and cross-validation sets. After training using data from a specific market, the training MSE score for that market is recorded. Then one-step-ahead predictions are made using the cross-validation data for that market. These are also evaluated for MSE and the score is recorded. The successive MSE results for both training and cross-validation are then plotted against each other for analysis. After the completion of several epochs, a final set of data reserved for testing is used to make final predictions to be evaluated against a theoretical buy-and-hold strategy.

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

II. Analysis

Data Exploration

	DateTime	Open	Close	Low	High	Volume	BaseVolume
0	2017-11-22 01:10:00	0.004316	0.004305	0.004305	0.004319	2004.199567	8.644211
1	2017-11-22 01:15:00	0.004305	0.004288	0.004282	0.004306	1011.973133	4.345338
2	2017-11-22 01:20:00	0.004288	0.004288	0.004282	0.004297	1990.877709	8.538191
3	2017-11-22 01:25:00	0.004288	0.004303	0.004288	0.004310	2116.678468	9.097914
4	2017-11-22 01:30:00	0.004303	0.004285	0.004283	0.004303	1343.086967	5.766947

From a preliminary examination of our dataset, we can see that it is made up 7 columns. The data is listed for 5 minute intervals.

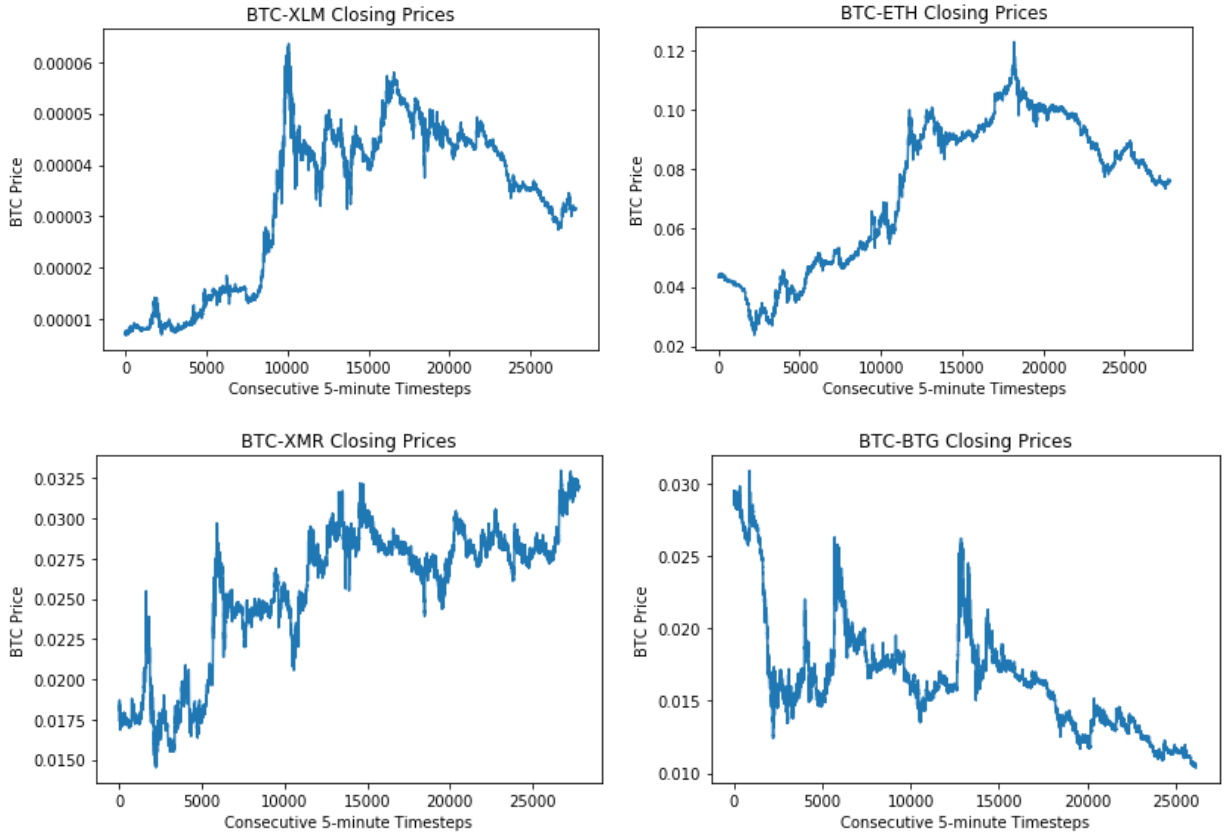
There are 2 separate Volume columns listed. The ‘Volume’ column lists volume data in units of the asset being traded against BTC. The ‘BaseVolume’ column lists the same volume data, but in units of BTC. This is the Volume data that will be used, while the asset-unit volume column will be dropped as it is redundant after scaling.

There is a very large amount of data collected, and there are points at which, due to network/server downage, there may be gaps of time missing from each set of market data. It will be necessary to “smooth” out these time gaps by using Last-Observation-Carried-Forward (LOCF) rows inserted, or simply making a selection from each dataset that contains a desirable length of consecutive data points.

It is undesirable to train on data collected from markets with extremely low trading volume. The top 45 markets by average trading volume were selected to be used, and the rest were discarded. After processing, five of these were also discarded due to time gaps in the data, leaving a total of 40 markets/data segments for training.

Exploratory Visualization

The predicted feature is the closing price (last traded price), of each 5-minute time interval.



After extracting the longest consecutive sequences from each market's full data, sequences that are between ~10,000 and ~28,0000 points in length remain. To provide some insight into the variety of data that is contained, the averages and standard deviation were calculated from the 'Close' prices for a few of the market datasets.

```
BTC-ETH Data:
(2017-11-30 08:50:00 to 2018-03-06 21:30:00)

count    27801.000000
mean         0.073255
std          0.024783
min          0.023810
25%          0.048508
50%          0.081585
75%          0.094000
max          0.123000
```

```
BTC-VTC Data:
(2017-11-30 08:40:00 to 2018-01-06 20:55:00)

count    10804.000000
mean         0.000526
std          0.000089
min          0.000369
25%          0.000476
50%          0.000500
75%          0.000547
max          0.000876
```

```
BTC-ZEC Data:
(2017-11-30 09:05:00 to 2018-03-06 21:30:00)

count    27798.000000
mean         0.038485
std          0.008337
min          0.016590
25%          0.034199
50%          0.039674
75%          0.043971
max          0.058839
```

```
BTC-DOGE Data:
(2017-12-07 06:35:00 to 2018-03-06 21:30:00)

count    2.581200e+04
mean     5.750477e-07
std      1.879658e-07
min      1.500000e-07
25%      5.100000e-07
50%      6.100000e-07
75%      6.500000e-07
max      1.130000e-06
```

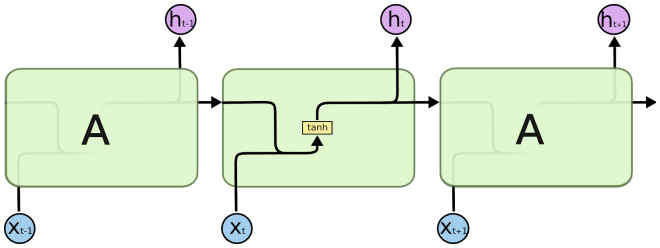
It can be seen that each different segment of data encompases differing windows of time. The ranges of 'Close' prices in each market are far removed from one another, signaling a need for value normalization.

From the plots of closing prices, we can see that the nature of price movement is non-stationary. This implies that the range of values that occur will vary widely over time. In order for the MSE loss function to provide values that bear any significance to objective price intervals, the data must first be normalized/scaled so that all values fall within a narrow expected range, while still accurately representing the raw data. Normalization methods will also compress noise and aid in avoiding issues with vanishing/expanding gradient.

Algorithms and Techniques

This project employs an LSTM based model which allows us to make predictions that are one time step ahead, and then feed that prediction back into the model as input for the next prediction. This enables us to make predictions for a sequence of future time steps.

Recurrent neural networks (RNN) can be thought of as multiple copies of the same network, each passing a message to a successor.

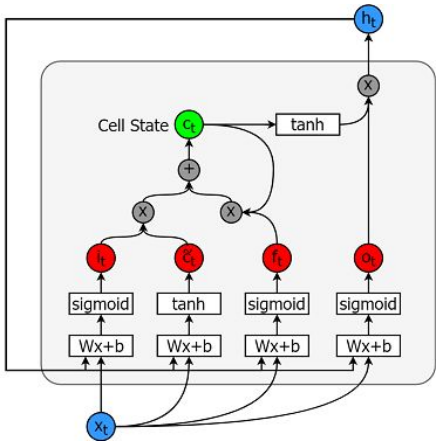


(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

The node in a *standard RNN* contains a single layer.

Long Short Term Memory model neural networks take this further by implementing a more complex architecture within each node, or "cell". Each cell contains a few essential components that allow it to effectively model time-sequenced data:

- Cell State - The LSTM has the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Forget Gate - This decides how much information from the current input and the previous cell state flows into the current cell state
- Hidden State - This contains output state information calculated from current input, previous hidden state and current cell input.



(https://www.datacamp.com/community/tutorials/lstm-python-stock-market)

The cell in an *LSTM* contains four interacting layers.

The final model consists of three LSTM module layers each with 100 units (cells). By default, tahn activation is used for these layers, with hard sigmoid for recurrent activation. Dropout layers set to a rate of 20% are placed after each LSTM layer. These help to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time. There is one final fully connected layer consisting of a single node with a linear activation function. This model uses ADAM optimization and mean squared error to compute loss.

Benchmark

The completed model is used to make (feedforward) predictions several steps ahead in time. These can then be compared to a buy and hold strategy. It can be assumed that entering and exiting a market arbitrarily will generally result in losses. Using the model to predict movement should allow us to make trading decisions that result in greater profit than a buy and hold strategy. By plotting the predicted price line against the true price data, it will then be apparent whether or not the predictions are useful.

III. Methodology

Data Processing

After dropping the (asset unit) 'Volume' column, the (BTC unit) 'BaseVolume' column is renamed to 'Volume' for clarity. Each market's dataset was then examined programmatically for time gaps. In many cases, the time gap is one single missing entry, where the trade volume was so low that nothing was recorded. In these instances we used LOCF to carry over the last closing price for the single missing entry's Open, Close, High and Low values, with 0 entered for Volume. In cases where the time gap is more than one row, we segmented the dataset. The longest segment of consecutive data points was then extracted to be used for training. A minimum threshold of 10,000 consecutive points was placed, and markets that did not meet this requirement were not used. Data segments from 40 of the top-volume markets remained after this process.

After extracting the longest segment of consecutive data points from each market, each segment was then processed for several popularly used technical (TA) indicators. The following indicators were chosen in order to offer a broad description of different features from the price data.

- Exponential Moving Average (EMA) (20 period) - A rolling average of the closing price for the past 20 periods. This moving average is weighted to place more significance on recent data points. The true closing price being significantly above/below the EMA can indicate a movement trend.
- Percent difference EMA (12 period) / EMA (26 period) - A value that is significantly above or below 1 may indicate a strong trend in price movement.
- Commodity Channel Index (CCI) - Similar to Relative Strength Index, CCI indicates whether an asset is considered overbought/oversold. A value of +100/-100 is considered overbought/oversold (respectively), and indicates a coming trend reversal.
- Movement Average Convergence/Divergence - The Moving Average Convergence Divergence (MACD) is the difference between two Exponential Moving Averages. The

Signal line is an Exponential Moving Average of the MACD. The MACD signals trend changes and indicates the start of new trend direction.

- Average True Range (14 period) - This indicates the average distance between the highest and lowest price for the past 14 periods. This is a good indicator of how much price volatility to expect.
- Bollinger Bands (5 period) - Bollinger Bands consist of three lines. The middle band is a simple moving average (generally 20 periods) of the typical price (TP). The upper and lower bands are F standard deviations (generally 2) above and below the middle band. The bands widen and narrow when the volatility of the price is higher or lower, respectively.
- Rate-of-Change (10 period) - Represents the percent difference between the last recorded price, and the price 10 periods ago, and can indicate trends.
- Momentum (20 period) - The Momentum is a measurement of the acceleration and deceleration of prices, also indicating trends.

Each market's dataset was processed for these indicators, and this new data was added as new columns/features.

Close	Low	High	Volume	EMA_20	EMA_12_26_PDIFF	CCI	MACD	MACD_SIG	MACD_HIST	ATR	BOLBAND_UP	BOLBAND_M
0.031000	0.030700	0.031360	4.830614	0.031028	1.000167	-14.999413	0.000013	3.413286e-05	-2.102981e-05	0.000331	0.031042	0.0310
0.031390	0.030950	0.031390	0.758724	0.031063	1.001082	142.202345	0.000040	3.536495e-05	4.928342e-06	0.000338	0.031391	0.0310
0.031425	0.031300	0.031425	2.592691	0.031097	1.001873	204.014250	0.000064	4.107138e-05	2.282573e-05	0.000323	0.031556	0.0311
0.031302	0.030700	0.031566	10.781046	0.031117	1.002158	92.422913	0.000072	4.724099e-05	2.467842e-05	0.000362	0.031589	0.0312

(clipped image of full-featured data preview)

Implementation

Two classes were created:

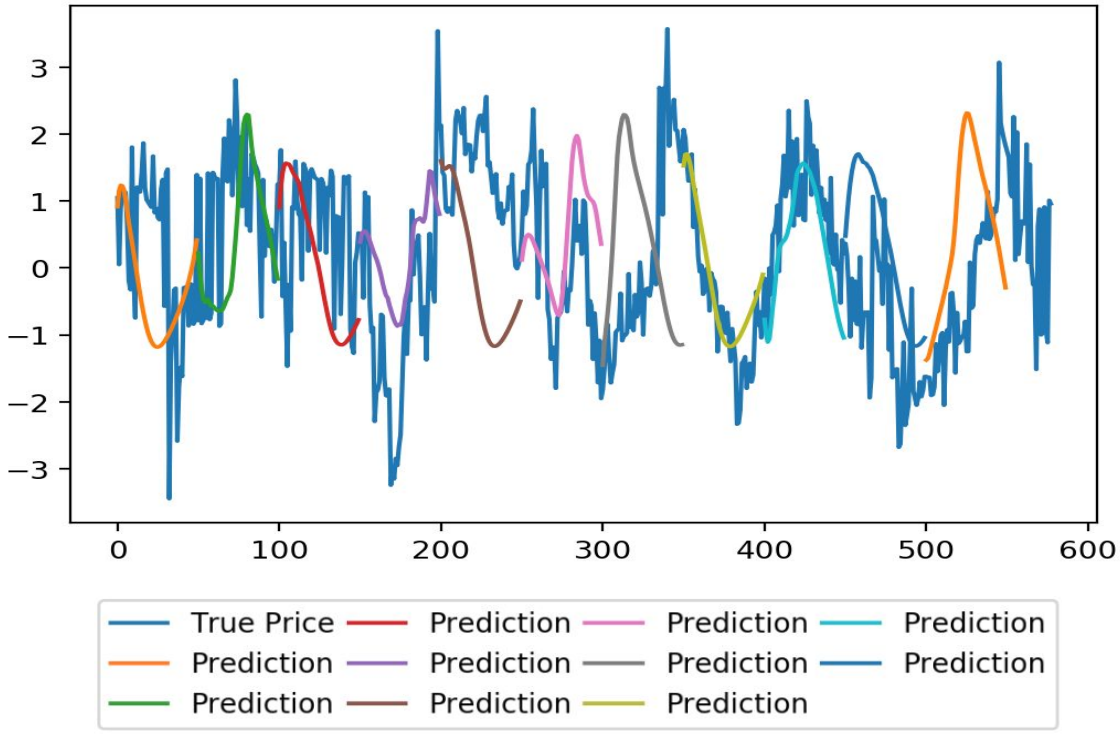
- A Data loader class imports data, creates the training and cross validation segments, generates batch/window data sequences, and also normalizes each data window upon generation.
- A Model class contains the build process for the models, where nn-layers are specified with parameters including input shape and node length. The model class is fed data by the data loader class and conducts processes related to training and testing.

Markets were selected in random order for training, and each is trained and cross-validated individually within each epoch. The MSE loss metric score is recorded by the Keras model object during training, and is easily retrieved by referencing its attribute. The cross-validation MSE score must be calculated using the arrays of predicted values and true values. This was done by applying the Scikit learn method for MSE to this data.

After the completion of training and cross-validation on each single market, a supplemental chart was created from predictions created based on 50 timesteps of data. Each prediction was fed back into the model as input for the next prediction (feed forward). This was done to blindly create predictions for the next 50 timesteps. This was done iteratively using the cross-validation data and displayed in a chart. This was useful to supplement the MSE scores in evaluating the progression of the model's prediction behavior.

Below is an example of the supplemental predictions chart. The main blue line depicts the True Price ('Close') for each timestep. Each other colored line is a prediction formed from a sequence of one-step-ahead predictions, each formed with the previous 'Close' value fed back in to form the next prediction, for 50 continuous timesteps. The chart below displays 11 sequential predictions in this manner.

Epoch 1, BTC-WAVES: 50-timesteps-ahead predictions, based on previous 50 timesteps



Normalization Refinement

Before lengthy training courses began, tests were executed using two popular methods of data scaling, and wavelet transform was applied for signal decomposition.

MinMax feature normalization scales data into a specified range (generally between 0 and 1) based on that features minimum and maximum values. MinMax scaling tends to work well in cases where the distribution of the data is not Gaussian or the standard deviation is very small.

$$\frac{x_i - min(x)}{max(x) - min(x)}$$

Standard scaling assumes the data is normally distributed and scales each feature such that the distribution is now centred around 0.

$$\frac{x_i - mean(x)}{stdev(x)}$$

In many machine learning applications, MinMax scaling is generally preferred. However, Standard scaling works well in normalizing data that is non-stationary. Additionally, the application of (Haar) wavelet transform was tested after each method of scaling. Applying wavelet transform splits (decomposes) each feature into a mother and father wavelet. This doubles our feature column length, and is considered useful for providing more resolution for our data.

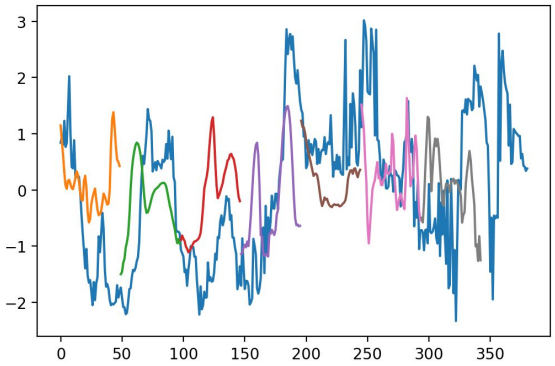
Each normalization method was tested by executing a single training epoch on a partial or full dataset. The MSE progression and sample predictions made from CV data were then analyzed for learning progress and predictive behavior respectively. From the results it was clear that models using the MinMax scaling method tended to make predictions that pointed towards a local average every time. Results using the Standard scaler seemed to produce extremely varied predictive behavior, displaying highly curved prediction lines. For the baseline, a test was run with no scaler applied, and the prediction lines bore no resemblance to the true price data, and no learning progress was being made.

Tests were also executed with and without wavelet transform applied for signal decomposition. With wavelet transformation, learning progress was being made slowly, but the prediction lines did not seem to be making as accurate predictions as tests using Standard scaler alone. The predictive behavior seemed to display large initial jumps/dips in value pointing towards a local average, then a tight jagged wave. It may be possible that with a significant amount of training, this model could improve greatly with many more iterations. For the purposes of this project, a decision was made to omit the application of wavelet transform in the normalization process.

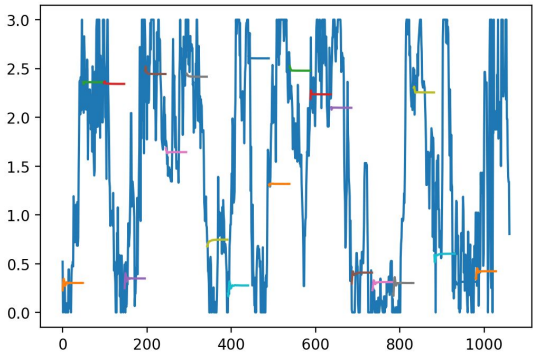
The results from tests performed with Standard scaling alone were the most impressive. Standard scaling was used for training our final model.

The following charts show the prediction results of the described scaling options. Again the blue jagged lines are the actual price data and the alternate colored lines are predictions.

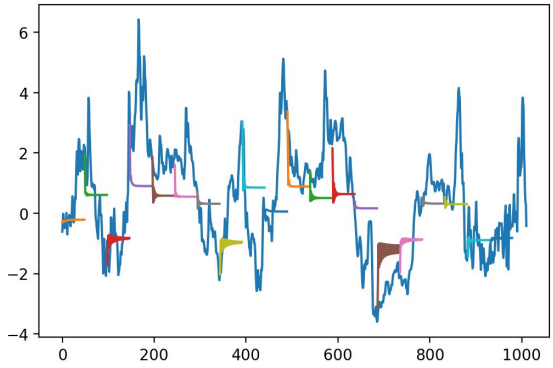
Partial Dataset Tests: 50-step-ahead prediction sequences with different normalizations



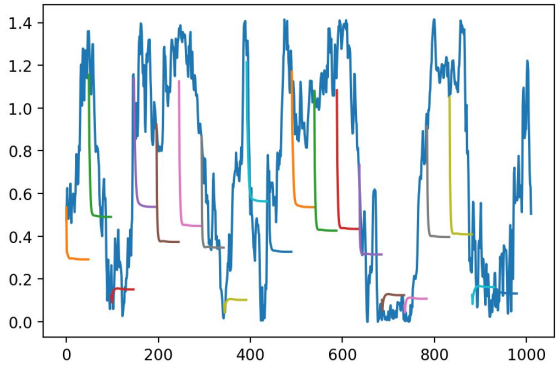
Standard Scaling Alone



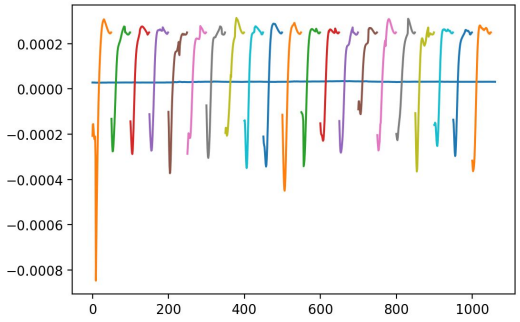
MinMax Scaling Alone



Standard Scaling with Wavelet Transform



MinMax Scaling with Wavelet Transform



No scaling applied

Model Refinement

Three different model architectures were tested during the model tuning process. The models are saved and labeled for each full epoch inside the ‘Model_Tuning/saved_models’ folder’. As a baseline, the default model (3 LSTM layers with 100 cell units, with one, single-node, fully connected layer with linear activation function) was trained for 10 epochs total.

Two other layer configurations were then trained for one full epoch. First, a model with fully connected layer built in *after* the LSTM layer, consisting of 32 nodes and with RELU activation. Also the unit count for each of the three LSTM layers was adjusted to the sequence: 128, 64, 32.

A second model was built with two fully connected layers coming *before* the default LSTM (100 cell unit) layers. The first fully connected layer contains 64 nodes and the second with 32 nodes. Each utilized the RELU activation function.

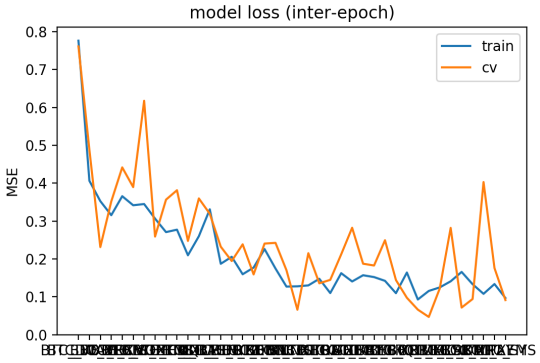
The model with fully connected layers coming after the LSTM layers seemed to perform well, but offered no significant improvement over the first model. The model with fully connected layers built before the LSTM layers performed far worse than our first model, with predictions constantly pointing towards a local average. It takes about 4 hours to complete a single training epoch with all models. Given an unlimited time frame, many more model architectures could be tested.

IV. Results

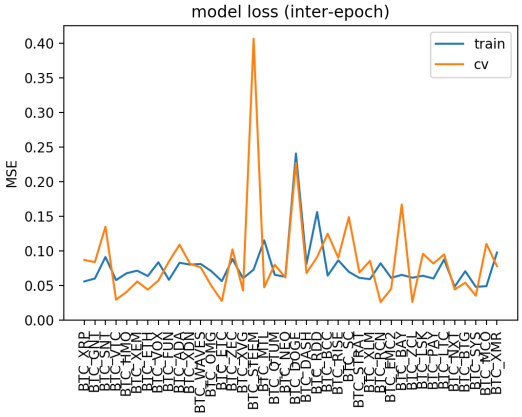
Model Evaluation and Validation

Mean squared error (MSE) measurements were taken after each training batch within each dataset. After the training was completed using data from a specific market, one-step-ahead predictions are made from the pre-designated cross-validation data segment for that market. The MSE was then calculated for these prediction results against the true data. A full training epoch consists one epoch trained in this way for each market. Both the training and cross-validation MSE scores were recorded for each market within each training epoch. The learning progress was then evaluated from the succession MSE plots from each full epoch.

The following is a plots of MSE loss progression within the 1st Epoch. MSE begins high and reduces quickly as training begins:



5th epoch: MSE begins localizing around 0.05-0.1:



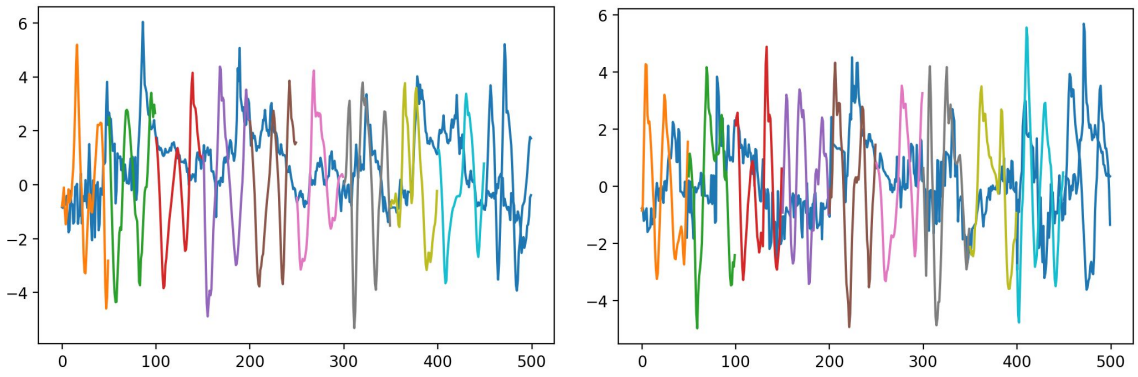
9th Epoch: Note that both lines seem to have converged at ~0.035-0.04. The cross-validation scores are becoming more erratic, while the training scores are smooth in comparison, and are not improving greatly. This may indicate that we have completed training, and training more epochs beyond this may result in overfitting.

Test Data Score & Predictions

The data collected for testing consists of market data collected from the same cryptocurrency exchange. This testing data consists of data from the same exchange as our training dataset, collected from September - October 2018. In the time since the training data was collected, trading volume on the Bittrex exchange has decreased sharply, due to decreased interest in cryptocurrencies in general, and also more competitive fee pricing on competing exchanges. It will be interesting to see how reduced volume, and therefore volatility, will affect the accuracy of the models predictions.

After running one-step-ahead predictions on our processed test data, the accumulated mean squared error for this testing set was 0.0392. This displays a marked improvement over this models initial training scores: ~0.1 to ~0.5+.

These are some supplemental prediction plots using the test data:



It can be seen from the prediction plots that the predicted price movement is generally more volatile (having greater deviation) than the true price line. This is representative of the non-stationarity, trending, and seasonality of investment market data, which makes it difficult to predict. The data that this model was trained on contained trading volumes significantly higher than contained in the test data set. This does not mean the predictions are not useful, but that the prediction results may benefit from more training on current data.

Justification

Predicting the full 50 steps ahead, as was done to monitor training, is an extreme use case, but it is useful to gauge whether or not the predictions the model is making are within the range that we would expect from normal price movement. In many cases, the 50 step prediction very closely matches the true price data, or the last point in the prediction line ends up being very

close to the corresponding true data point. It is reasonable to expect an even higher degree of accuracy from predictions made with fewer feedforward steps.

We kept track of our cross-validation MSE scores in relation to the training MSE scores to ensure that this is not simply a case of overfitting the training data. After 10 epochs the MSE of both training and cross-validation had still not seemed to diverge significantly. This indicates a possibility that several more epochs could be trained given more time, a larger data set, and/or improved hardware configuration.

The main problem with the buy-and-hold strategy is that one arbitrarily picks price points at which to enter and exit the market. This, of course, results in a random chance of making profit, with a high probability of losing money in a volatile market. Using our combination indicator for our theoretical trading strategy will prove quite useful when compared with a blind, buy-and-hold strategy. The prediction plots have displayed an ability for the model to predict a direction of price movement with degree of precision that is higher than random chance.

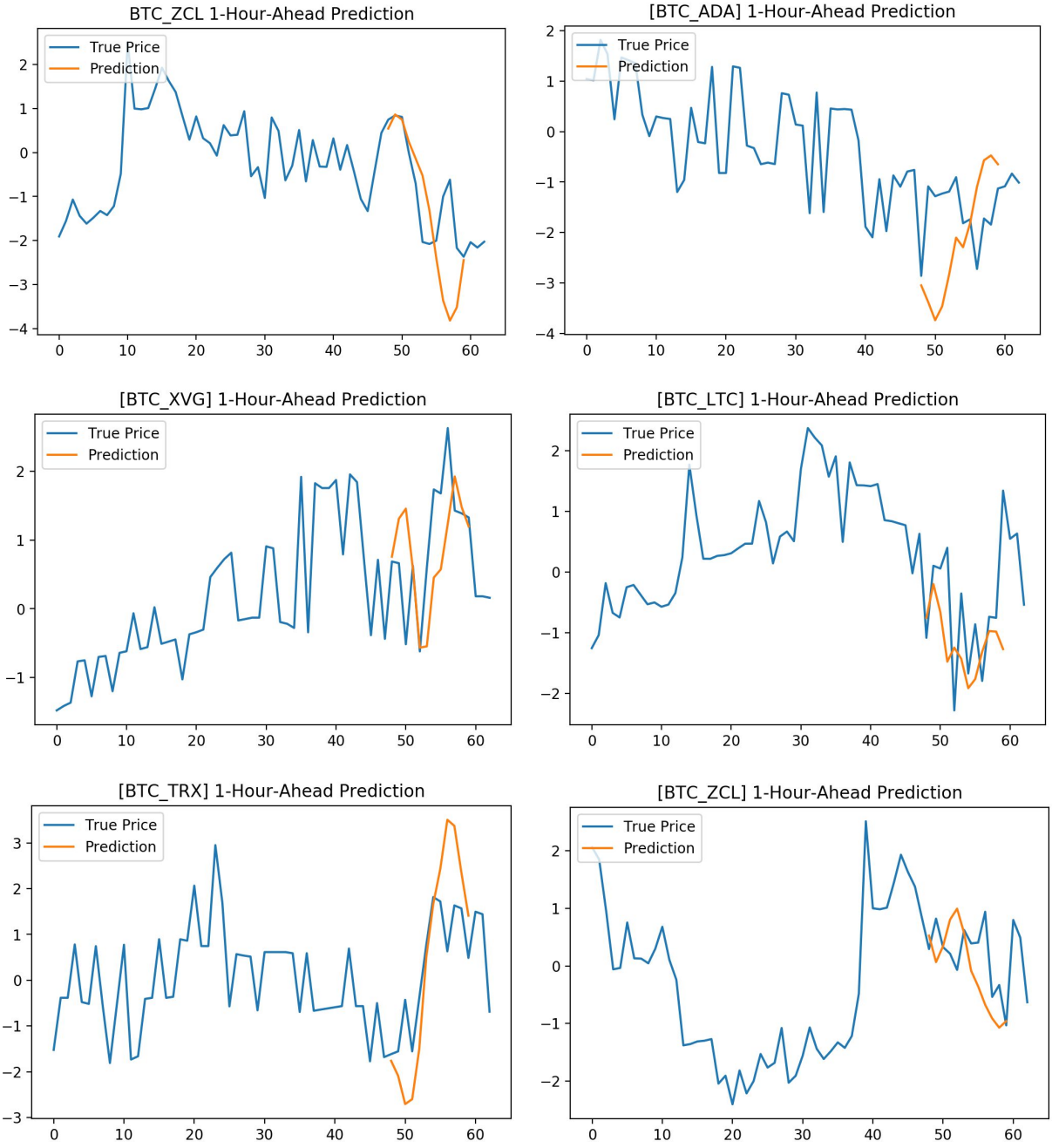
It would be unwise to rely on this model alone for an automated trading algorithm. As with other indicators, it is impossible to perfectly predict the stochastic “random walk” nature of investment market price movement.

There are many other practical applications for this type of model. It should be quite useful in the creation of an infographic which would display *possible* future price movement which could be generated and used on a website that displays and analyzes cryptocurrency prices. Similarly, it may be useful by an analyst when trying to call a prospective price top/bottom within a given period that is within the expected range. It could also prove to be a practical tool to be used in forecasting price movement for the average trader on a day-to-day basis.

V. Conclusion

Free-Form Visualization

These charts use the true price data segment (blue), randomly selected from within the testing data, to form a feedforward prediction line (orange) of 12 timesteps (12 x 5 minutes = 1 hour) into the future. The prediction line is plotted overlaying the true price data in order to visually gauge accuracy.



Reflection

The work in this project consisted of three major parts: preprocessing, normalization, and model tuning. Since I had initially collected such a large amount of price data, I ended up discarding much of it.

During preprocessing, I mainly focused on extracting one large consecutive segment from each market in the top average trading volume shortlist. I believe this was a good decision because it offers many samples from a variety of market conditions, as opposed to training on data from one specific market.

I spent a large portion of time trying out different techniques for normalization. I was initially excited by the prospect of feature decomposition using wavelet transform but I was unable to apply this to my data in a way that provided any predictive benefit. If I had just went ahead with the application of wavelet transform without first testing it, the model may have suffered severely.

I feel that I have only brushed the surface as far as trying out different model layer architectures and hyper-parameter configurations. I experimented by trying out different combinations for a total of over 65+ training hours.

I did not expect a resulting model to perfectly predict short term price movement. I hoped that the model would at least predict a general direction, and be correct at least half the time. Admittedly, I cherry picked the Free-Form Visualizations in order to display positive examples, but I am honestly impressed that the predictions are able to generally follow the actual price movement, at least some of the time. In cases where the prediction line diverges from the true price data, it is still predicting values that are within a reasonable expectation for that future period.

Improvement

In order to make the feedforward prediction results ‘production ready’, the normalized predictions must be adapted back to the actual raw price movement through a process of inverse scaling.

New data could be collected continuously from various cryptocurrency exchanges. A data pipeline should be created to process current data in order to further improve the model.

The error rate of the predictions might be improved through the engineering and testing of more complex model architectures. Hardware improvements would accelerate the learning process, allowing more trials of different hyper-parameter configurations to be conducted in a smaller amount of time.

Online Resources:

“Understanding LSTM Networks”. August 27, 2015.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

“A deep learning framework for financial time series using stacked autoencoders and long-short term memory”. July 14, 2017.
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0180944

“Stock Market Predictions with LSTM in Python”. May 3, 2018.
https://www.datacamp.com/community/tutorials/lstm-python-stock-market

“Time Series Prediction Using Lstm Deep Neural Networks”. September 1, 2018.
https://www.altumintelligence.com/articles/a/Time-Series-Prediction-Using-LSTM-Deep-Neural-Networks

“A comprehensive beginner’s guide to create a Time Series Forecast”. February 6, 2016.
https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/