Predicting Cryptocurrency Prices Using Technical Indicators and Neural Networks

Bryan W. Nonni
Bachelor of Computer Science
Georgia State University
Atlanta, GA 30309, USA

Abstract*—Cryptocurrency is a digital innovation on traditional, fiat currency. Cryptocurrencies establish a chain of cryptographically signed blocks containing transaction data. Coins are issued as an incentive to miners who maintain a network node, which is a digital copy of the entire transaction history of the currency. Coins are traded on a peer-to-peer monetary exchange network similar to traditional exchanges like the NYSE. As cryptocurrencies continue to grow, industry advocates expect to see a stabilization of the price of these coins. In an effort to better understand what influences the major peaks and valleys of a cryptocurrency, an Artificial Neural Network (ANN) will be leveraged to predict price fluctuations—specifically a form of a Recurrent Neural Network (RNN) called a Long Short-Term Memory (LSTM). The cryptocurrency examined will be Bitcoin (BTC). Historical price data, including High and Low, will be used to calculate the following technical indicators, which will also be used to predict price changes: Relative Strength Index (RSI), On-Balance Volume (OBV) and Accumulation/Distribution Line (A/D).

Index Terms*—Cryptocurrency, Digital Currency, Artificial Neural Networks, Long Short-Term Memory, LSTM, Recurrent Neural Network, RNN, Decentralized Currency, Blockchain Technology, Technical Indicators, Cryptocurrency Price Prediction, Relative Strength Index, RSI, On-Balance Volume, OBV, Accumulation/ Distribution Line, A/D, Bitcoin.

I. Introduction

Digital banking has become increasingly popular in recent years and is continuing to grow. Users of digital banking have services at their disposal that allow them to send, receive, store and access their money via website or device apps. A key feature of this digital system is its reliance on customers keeping their money in the bank. If everyone who holds an account with a bank withdrew all of their money at the same time, the bank would not have enough physical money to pay all customers–especially if each customer has hundreds of thousands of dollars in their account. The bank is betting that this will not happen, and customers will continue to keep a majority of their funds in the bank. The numbers reflected on an online banking statement do not directly reflect the physical money in the bank thus making money, as we know it, a form of digital currency. The rise of digital banking and onset of currency as “digital” has resulted in the creation of peer-to-peer (P2P) payment services such as PayPal or Venmo; however, P2P systems are not truly P2P. Users are still required to attached a bank account in order to move money from peer-to-peer.

Within the P2P system, banks act as arbitrary third parties or “middle-men” aiding the process of money exchange. When viewed from this lens, it becomes clear that banks add no tangible value to the transaction, and in fact, a bank is susceptible to a single-point-of-failure flaw.

A potential solution is Cryptocurrency. While the idea of cryptocurrency is new, the practice of digital payments, digital banking and digital money is not. Cryptocurrency is not without its flaws, which is why it has not been adopted. The price of a "coin" is considerably unstable, and the overall system is misunderstood by the public. The rapid trading of these cryptocurrencies combined with rampant speculation and lack of knowledge is part of what makes them so volatile.

The goal of this research is to better understand these fluctuations using a new method that takes into account new features. If successful, the new method will bring deeper insight and broader knowledge to a widely misunderstood topic with the hopes of helping pave a path towards adoption into the market.

II. Previous work

Predicting the future of a security is a time-series analysis. Traditional methods for time-series forecasting vary greatly. Simple methods include Mean—predictions are equal to the mean of the time-series; Naïve—predictions are equal to the last value of the time-series; Seasonal Naïve—the forecasts for a given season are equal to the value of the same season a full period before (i.e. January 2020 predictions based on January 2019 data) [4]. More complex strategies include autoregressive integrated moving average (ARIMA)—the AR takes into account the influence of the previous values on the predicted one, and the MA models the influence of noise on the future values; Seasonal Autoregressive Integrated Moving Average (SARIMA)—same as ARIMA, but the S mitigates the lack of seasonality in the ARIMA model [4].

Some previous research done on cryptocurrency price prediction has involved different types of machine learning and deep learning models such as: Random Forest Classifier (RFC) [2], Feed-Forward Multi-Layer Perceptron (MLP) [1], and a Convolutional Neural Network (CNN) [3]. E. Almasri and E. Arslan, saw promising results using a feed forward MLP with accuracies ranging from 75% to 97.3% using daily close price and volume values for Bitcoin [1]. Z. Jiang and J. Liang found that their CNN was the second most successful at increasing portfolio values but also found that the longer the delay (of time) between the training set and testing set, the lower the accuracy [3]. This suggests a “forgetting” behavior from the model or a lack of data for the model to generalize short-term trends over a long-term series. Results from J. Sun, Y. Zhou and J. Lin [2] using RFC were mixed. The conclusion suggests trying deep learning in the future to improve accuracy. Another challenge with the previous works cited here is in the data used. All three papers focused exclusively on open, high, low, close and volume [1][2][3]. While useful, these data points alone are overly myopic.

Learnings from these cited prior works suggest the need for a model with a longer memory of past patterns, more data and new data points that are more indicative of price/market fluctuations.

III. Methodology

To address these problems, this research will use an Artificial Neural Network (ANN). Neural networks (NN) are well-suited for this type of problem because they do not require stationary data. Fundamentally, neural networks are effective at finding relationships between data and using those relationships to classify new data; however, some fundamental flaws with specific types of ANNs need to be addressed and mitigated in the following sections.

The five main steps of this research are data collection, feature identification, data preprocessing, neural network selection and neural network design & implementation.

A. Data Collection

There are varying sources available for Bitcoin (BTC) data including repositories on GitHub and APIs from various trading platforms; however, data in this research was gathered over the course of more than 6 months starting in late May 2019 and is still being collected. For this research, the dataset starts on June 15, 2019 and goes until August 26, 2019. The Coinbase.com API was leveraged to pull prices for BTC each minute and save them to a cloud-based MongoDB server. The server runs a Node.js based application that makes the API call, stores the latest price, calculates technical indicators and stores those data points as well. Upon API call, the app uses Keras to quickly calculate RSI, OBV, A/D and the slopes of the latter two indicators (OBV and A/D). All of this is saved in the Mongo instance and can be easily pulled down to a remote machine using the Mongo API.

From here, a CSV file was created using roughly 105,000 samples of data, which equates to roughly 2 to 3 months’ worth of data. Each sample represents a minute and has the following features: Datetime, Price, Volume, High, Low, RSI, A/D, A/D slope, OBV and OBV slope.

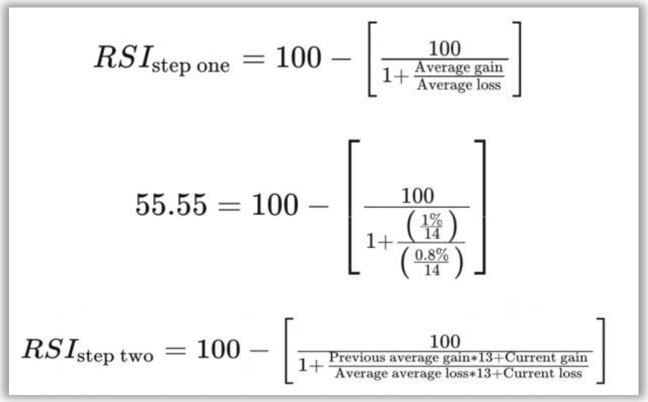
B. Feature Identification

This research takes a new approach to feature selection as part of the strategy that distinguishes it from the research mentioned in the Previous work section.

The features used in this research can be classified into two groups: traditional features and technical features. Traditional features are the classically used data points: High, Low and Volume. Technical features are defined as the technical indicators used widely across practical trading but not used in previous research. Technical features include Relative Strength Index (RSI), On-Balance Volume (OBV) and Accumulation/Distribution Line (A/D).

RSI is a measure of the buying and selling of the current market. It is computed with a two-part calculation:

Figure 1. Formula for RSI Calculation [5]



$$RSI_{\text{step one}} = 100 - \left[\frac{100}{1+\frac{\text{Average gain}}{\text{Average loss}}}\right]$$

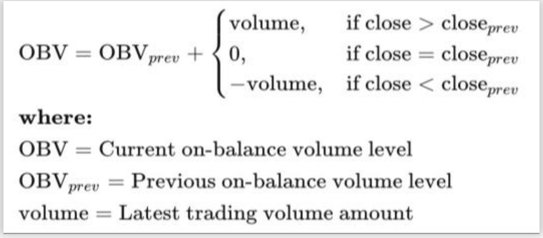
$$55.55 = 100 - \left[\frac{100}{1+\frac{\left(\frac{1\%}{14}\right)}{\left(\frac{0.8\%}{14}\right)}}\right]$$

$$RSI_{\text{step two}} = 100 - \left[\frac{100}{1+\frac{\text{Previous average gain}*13+\text{Current gain}}{\text{Average average loss}*13+\text{Current loss}}}\right]$$

Typically, if RSI is above 70, the market is seen as over-bought, which means that the price is high due to a previous period of excessive buying. RSI trending up to 70 and hanging there indicates rising prices and possibly a good time to sell. If RSI is below 30, the market is the opposite: over-sold, which means the price is low due to a previous period of excessive selling. RSI trending down to 30 and hanging there indicates falling prices and possibly a good time to buy.

OBV is a measure of momentum in the market. When looking at a market, it is typical to see volume fluctuate heavily in a minute, hour or day. However, when viewed over a longer timeframe, volume starts to produce a clear trend up or down. OBV normalizes short-term volume fluctuations to better understand them in relation to the market as a whole. In other words, OBV corrects for heavy buying or heavy selling, which would create volatile fluctuations in the volume of a coin. From this normalized volume, a slope or trend line can be generated to understand how the overall market is trending, which is why OBV Slope is used. If OBV slope is positive, volume is increasing, the market is selling more frequently, and prices are dropping. If OBV slope is negative, the opposite is the case: market volume is decreasing, the market is buying more frequently, and prices are rising.

Figure 2. Formula for OBV Calculation [5]



$$\text{OBV} = \text{OBV}_{prev} + \begin{cases} \text{volume}, & \text{if close} > \text{close}_{prev} \\ 0, & \text{if close} = \text{close}_{prev} \\ -\text{volume}, & \text{if close} < \text{close}_{prev} \end{cases}$$

where:

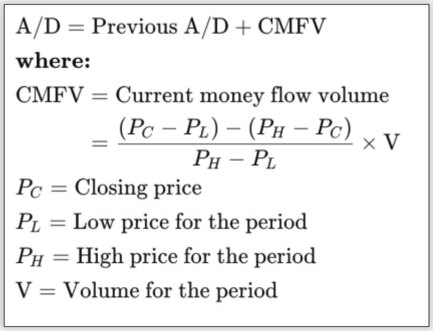
$\text{OBV} = \text{Current on-balance volume level}$

$\text{OBV}_{prev} = \text{Previous on-balance volume level}$

$\text{volume} = \text{Latest trading volume amount}$

A/D is a measure of the supply and demand in the market by looking at where the price closed within a time range and multiplies that by the current volume. A/D trend is used to understand how much supply or demand is in the market, which provides insight into whether a price is rising or falling. If A/D is positive, supply is rising and demand is falling, which indicates that prices are rising. If A/D is negative, supply is falling and demand is rising, which indicates that prices are falling.

Figure 3. Formula for A/D Calculation [5]



$$\mathrm{A/D} = \mathrm{Previous\ A/D} + \mathrm{CMFV}$$

where:

$$\mathrm{CMFV} = \mathrm{Current\ money\ flow\ volume}$$

$$= \frac{(P_C - P_L) - (P_H - P_C)}{P_H - P_L} \times \mathrm{V}$$

P_C = Closing price

P_L = Low price for the period

P_H = High price for the period

V = Volume for the period

C. Data Preprocessing

The data for this research was collected using 1st party means as a way to mitigate the common problems with puling data from 3rd parties such as Github or Kaggle, which tends to include various errors and missing values. This was done in an attempt to expedite the process of data collection and mitigate data cleaning and; however, despite best efforts, the data still required some cleaning prior to preprocessing.

1. Missing Values

The data collection process required testing and refactoring code in the server app—both app code and database code. Different methods were tested to find the most optimal key, value pairings for structured JSON objects inside the different collections. As a result, the database contained data objects with missing or bad values (i.e. 0, None, null or NaN). Table I shows how this data was handled.

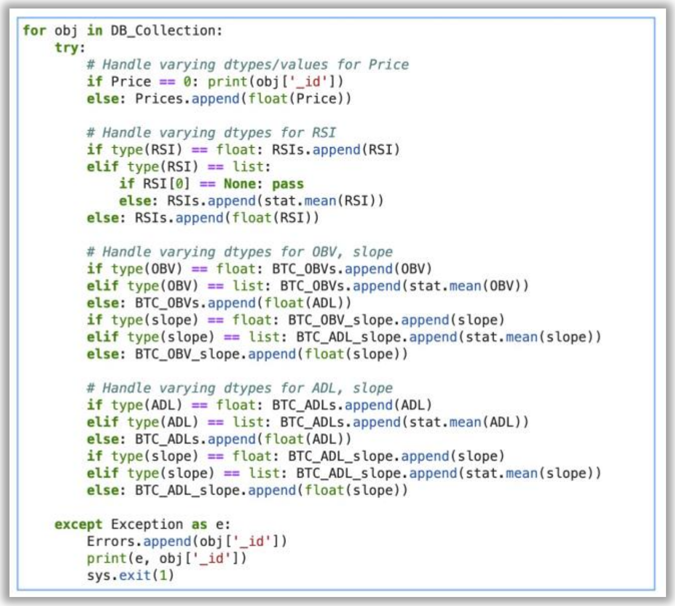
Table I. Remove missing values

Feature	Unexpected Behavior	Action
Price	0	print id of object for review; replace with proper price point from 3rd party source
RSI	None or null	Imputed via calculation using prices @ corresponding datetime
OBV, slope	None or null	Impute via calculation using volumes @ corresponding datetime
A/D, slope	None or null	Impute via calculation using volumes and prices @ corresponding datetime

2. Unexpected Data Structures/Data Types

An implication of changing data structures is unexpected structure or type upon pulling from the database. Certain data was expected to be a key, value pair that contained a single value; however, there were objects with keys that pointed to arrays of values or another JSON object with its own key, value pairs. In addition to unexpected data structures, data types for certain values varied as a result of both the API and the 'technical indicators' package that was used to quickly compute technical feature data. Figure. 4 shows how this unexpected behavior was handled.

Figure 4. Handling Unexpected Data Structures/Types



```python
for obj in DB_Collection:
    try:
        # Handle varying dtypes/values for Price
        if Price == 0: print(obj['_id'])
        else: Prices.append(float(Price))

        # Handle varying dtypes for RSI
        if type(RSI) == float: RSIs.append(RSI)
        elif type(RSI) == list:
            if RSI[0] == None: pass
            else: RSIs.append(stat.mean(RSI))
        else: RSIs.append(float(RSI))

        # Handle varying dtypes for OBV, slope
        if type(OBV) == float: BTC_OBVs.append(OBV)
        elif type(OBV) == list: BTC_OBVs.append(stat.mean(OBV))
        else: BTC_OBVs.append(float(ADL))
        if type(slope) == float: BTC_OBV_slope.append(slope)
        elif type(slope) == list: BTC_ADL_slope.append(stat.mean(slope))
        else: BTC_OBV_slope.append(float(slope))

        # Handle varying dtypes for ADL, slope
        if type(ADL) == float: BTC_ADLs.append(ADL)
        elif type(ADL) == list: BTC_ADLs.append(stat.mean(ADL))
        else: BTC_ADLs.append(float(ADL))
        if type(slope) == float: BTC_ADL_slope.append(slope)
        elif type(slope) == list: BTC_ADL_slope.append(stat.mean(slope))
        else: BTC_ADL_slope.append(float(slope))

    except Exception as e:
        Errors.append(obj['_id'])
        print(e, obj['_id'])
        sys.exit(1)
```

Prices were rarely missing from data objects, but in the event of a missing price or a 0 price, the object that contained the missing data was logged for review and that object was later replaced with accurate data. Otherwise, it was added to the dataset casting it from a string into a float.

RSI, OBV and A/D all varied between a float, an array with multiple values or an array with null values. Floats were added, array data was used to impute a mean of all array values into a single value (Table I), null arrays were identified, and the object id was logged for later cleaning.

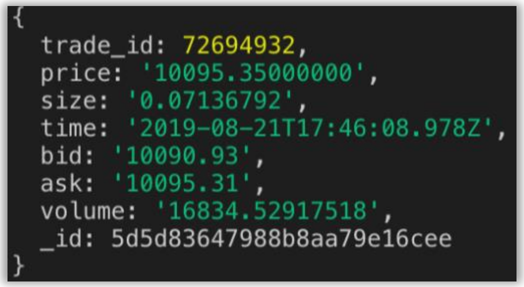
3. Database Object Conversion and Resetting

In an effort to be more semantic, all dates were initially logged as formatted datetime objects instead of something more suited for analysis like epochs. To handle this conversion, a combination of a custom function and the time/ datetime python libraries were used.

Table II. Datetime conversion

Datetime Object in DB	Epoch Conversion
2019-08-21T17:46:08.978Z	1576318321

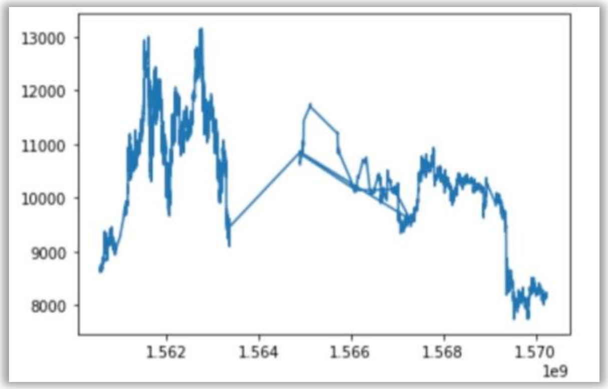
Figure 5. Date object conversion and resetting



```
{
  trade_id: 72694932,
  price: '10095.35000000',
  size: '0.07136792',
  time: '2019-08-21T17:46:08.978Z',
  bid: '10090.93',
  ask: '10095.31',
  volume: '16834.52917518',
  _id: 5d5d83647988b8aa79e16cee
}
```

The datetime was stripped of all formatting and converted to an epoch time integer for use in modeling. Upon further analysis, it was discovered that the dates in the database were considerably flawed.

Figure 6. Date Series Inconsistencies
(x-axis: epoch date, y-axis: BTC price)



To mitigate this error, simple looping was used to detect inconsistent time steps from object to object and the object id was used to programmatically fix the dates.

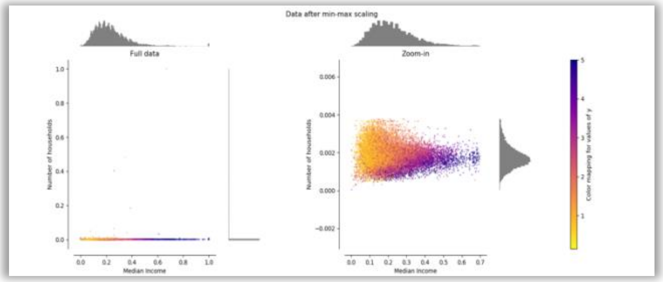
4. MinMaxScalar Transformation

A key component of the preprocessing step for NNs is to fit and transform the data set. The MinMaxScalar was used in this research. This scalar is found within the sci-kit learn python package, and it applies the following formula for each feature:

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

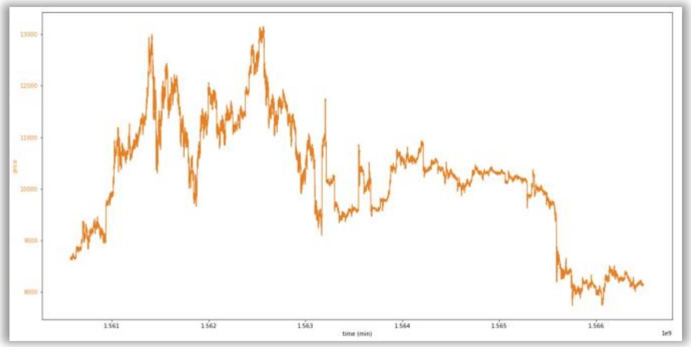
where x_i is each feature in the set of features, $\{x_i, x_{i+1}, \cdots, x_n\}$. Here, i is the first feature starting at 1, and n is the total number of features. The MinMaxScalar shrinks the range of each feature, so that it falls somewhere between 0 to 1 for positive values and -1 to 1 for negative values. Scaling is done to normalize the features and allow the NN to find patterns more easily.

Figure 7. Transforming Dataset Using MinMaxScalar [7]



MinMaxScalar is best suited for datasets that do not fit a normal Gaussian distribution or for datasets with a very small standard deviation. Given the volatile nature of BTC (fig. 8), these attributes apply to the research dataset.

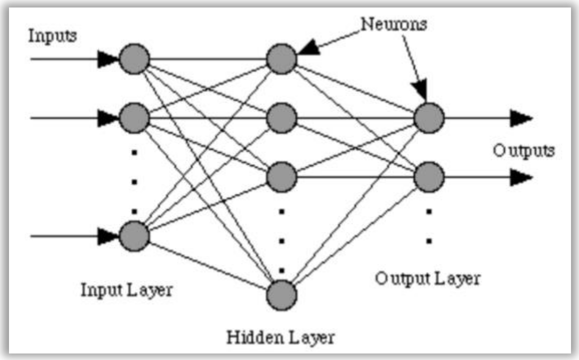
Figure 8. BTC Prices by Minute Over 2 Months



D. Neural Network Selection

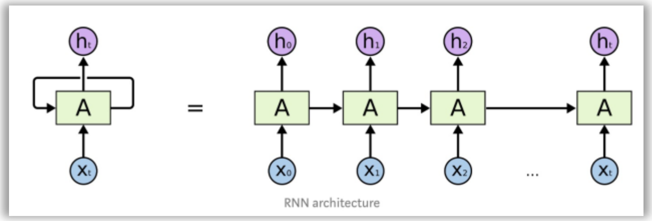
Another distinguishing feature of this research is the ANN model used. Three possibilities were entertained for model selection: Multi-Layer Perceptron (MLP), Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM).

Figure 9. Feed Forward MLP architecture [1]



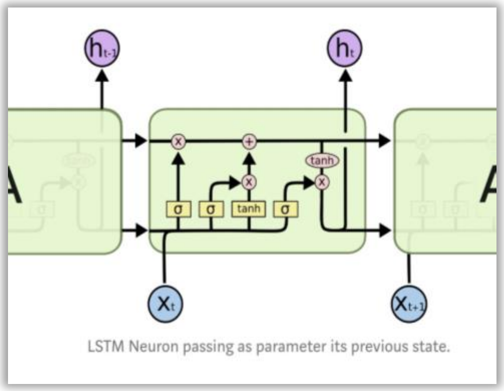
MLPs are the simplest form of neural networks. Inputs are fed into an input layer of nodes and assigned random weights. The values are multiplied by the random weights and are fed forward through the hidden layers to produce the output. The learning comes from backpropagating through the hidden layers to change the value of the weights between each neuron. One issue with MLPs is the lack of memory: learnings from each layer cannot be passed to subsequent layers for later use in predictions.

Figure 10. RNN architecture [6]



RNNs mitigate this problem. The cells of an RNN are able to store past learnings in the cell's state, which is particularly useful when data order matters such as in time-series forecasting. The RNN node uses the previous cell's state and inputs new data to make a prediction feeding forward previous states and new predictions. However, the problem with an RNN is its memory loss. In deep learning, layers and nodes start to add up quickly. As the layers deepen and nodes thicken, the network's older states are fast forgotten. This leads to another common problem with RNNs. They suffer from a vanishing gradient. NNs assign weights to input data and pass it through a summation function. As errors occur, those weights are changed to reduce that error. Over time, the NN begins to multiply smaller and smaller decimals and the gradient of change becomes vanishingly small effectively preventing the weight from changing its value. Thus, no learning can take place and the model's accuracy plateaus. This is also a common issue within NNs in general due to their very nature—a NN uses weights to determine the importance of certain data.

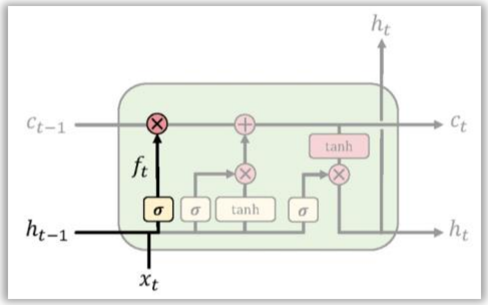
Figure 11. LSTM Cell Architecture [6]



LSTMs solve both the memory problem and the vanishing gradient problem. The LSTM is a type of Recurrent Neural Network (RNN) that possesses the ability to store information about the input data for later use in the model. This storage capacity extends the neural network's ability to analyze the complex relationship structures between data points. The cells of a LSTM create a connection between a forget gate and the gradients computed by the cell. This connection creates a path for information flow through the forget gate into the next cell's state. In other words, the gating structure inside of an LSTM cell allows important information that the LSTM should not forget to pass from cell to cell propagating learnings from layer to layer.

The LSTM cell has 3 gates, each with a different goal: Forget Gate, Update Gate, Output Gate.

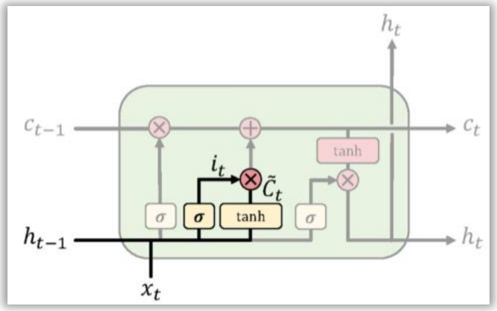
Figure 12. LSTM Forget Gate [6]



$$f_t = \sigma(W_i[h_{t-1}, x_t] + b_f)$$

The Forget Gate (σ) takes input feature data, x_t along with previous cell state data, h_{t-1} applies a weight W_i plus a bias b_f and puts it into a Sigmoid function (σ) to compute a value between 0 and 1. This function determines how much of the previous data should be kept and passed forward.

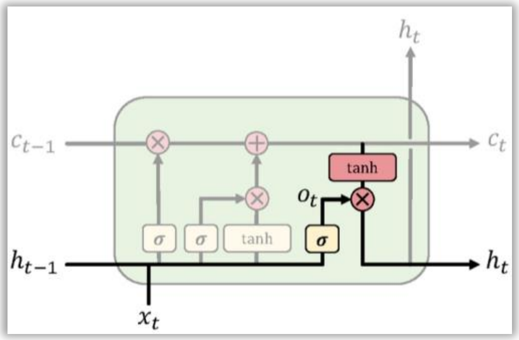
Figure 13. LSTM Update Gate [6]



$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$\widetilde{C_t} = tanh(W_c[h_{t-1}, x_t] + b_C)$$

The Update Gate (σ, tanh) is a combination of both a Sigmoid function (σ) and a Hyperbolic Tangent function (tanh), where i_t is the input data. The sigmoid function takes the input x_t and determines which values to update from that input. Tanh creates a new set of values, where $\widetilde{C_t}$ is a new vector of "candidate data" to be added to the cell's state. The net of this gate is a forgetting of previous data deemed to be not useful and an adding of new candidate data, which is passed along to the next gate.

Figure 14. LSTM Output Gate [6]



The Output Gate (σ, tanh) is another Sigmoid function that decides what data from the Update Gate will be output to the next cell. It multiples that data and the tanh function's

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

output. Here, o_t is the output data. The tanh function scales the output values between -1 and 1 to ensure only the most relevant and predictive data is sent onward.

Given the unique and complex structure of the LSTM, it is well suited for time-series predictions and therefore, well-suited for this research.

E. Neural Network Design & Implementation

NNs are a newer addition to the AI and Machine Learning realm. Looking for research that provides insight into the optimal implementation of a NN with an optimal number of layers and nodes is a spurious pursuit because even if said research exists and is successful, every new research hypothesis has nuances. The topic of predicting cryptocurrencies using technical features falls into this category of nuanced problems.

Numerous designs were implemented using research and brute force combinations of varying numbers of layers, nodes, dropout percentages, layer parameters, etc. However, for the given hypothesis, the following model provided the best results to date:

- 5 LSTM layers with 200 nodes per layer
- 5 interleaved Dropout layers, 20% dropout per layer
- 1 LSTM layer with 200 nodes
- 1 Following Dropout layer, 50% dropout
- 1 Dense output layer with 1 node

Table III. LSTM Design

Model Layer Break Down			
Type	**Detail**	**Nodes**	**Parameters**

LSTM	Input	200	return_sequences=True,
			input_shape=(#features, 1)

Dropout	20%	--	--
LSTM	Hidden	200	return_ sequences =True
Dropout	20%	--	--
LSTM	Hidden	200	return_ sequences =True
Dropout	20%	--	--
LSTM	Hidden	200	return_ sequences =True
Dropout	20%	--	--
LSTM	Hidden	200	return_ sequences =True
Dropout	20%	--	--
LSTM	Hidden	--	--
Dropout	50%	--	--
Dense	Output	1	--

IV. EXPERIMENTS & RESULTS

The main experiments focused on two areas: running the model on each individual feature and running the model on all possible permutations of the features. Each permutation is listed below:

- [High, Low], [High, Volume], [High, RSI], [High, ADL], [High, ADL_slope], [High, OBV], [High, OBV_slope]
- [Low, Volume], [Low, RSI], [Low, ADL], [Low, ADL_slope], [Low, OBV], [Low, OBV_slope]
- [Volumes, RSI], [Volumes, ADL], [Volumes, ADL_slope], [Volumes, OBV], [Volumes, OBV_slope]
- [RSI, ADL], [RSI, ADL_slope], [RSI, OBV], [RSI, OBV_slope]
- [ADL, ADL_slope], [ADL, OBV], [ADL, OBV_slope]
- [OBV, ADL_slope], [OBV, OBV_slope]
- [ADL_slope, OBV_slope]

The following simple loop was implemented to run this iteratively:

```
for i, feature in enumerate(features):
        x = dataframe[[feature]].values
        y = dataframe[['Price']].values
        // split into train, test data
        // use MinMaxScalar to fit/transform
        // reshape train, test into 3D datasets
        // add layers to model, compile, fit, evaluate
        Repeat loop until end of features
End loop
```

A. Accuracy Metrics

Accuracy for a NN using continuous data is not measured with an accuracy score similar to a NN using categorical data. Its measured using loss. This model used mean squared error (MSE) as its loss function to understand the amount of error in the model. MSE calculates the average of the squared difference between the predicted and the actual data points:

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

Here, *n* is the number of data points, *i* is the particular predicted data point in the set $\{Y_i, Y_{i+1}, \cdots, Y_n\}$, and $\hat{Y}_i$ is the predicted data point for the corresponding actual data point Y_i. In addition to MSE, mean absolute percent error was used as a way to understand calculate a basic accuracy score.

$$\text{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

Here, *n* is the number of data points, *t* is time, A_t is the actual value at time *t* and F_t is the forecasted value at time *t*. To obtain an accuracy score, simply subtract MAPE from 100:

$$Accuracy = 100 - MAPE$$

Similar to MSE, MAPE is the average of the errors (i.e. actual – predicted) taken as an absolute value and multiplied by 100% over the number of data points.

B. Single Feature Analysis Minutely Data

The best single feature results were High and Low. However, odd behavior was exhibited. After epoch 75/100, the model' training loss plateaus resulting in unreasonably high training MAPE, but the validation and testing continues to reduce. The suggested thought process around this behavior is one of three possibilities:

1. Underfitting
2. Too Much Dropout
3. Bad Validation & Test Data

Table IV. Single Feature Results

Single Feature Results			
Feature	**Train MAPE**	**Test/Val MAPE**	**Test/Val Accuracy**
High	287%	34%	66%
Low	290%	53%	47%
ADL	1072%	1026%	0%
OBV	1072%	1026%	0%
OBV_Slope	1071%	1026%	0%
ADL Slope	1072%	1026%	0%
RSI	1072%	1026%	0%
Volume	1072%	1026%	0%

C. *Multi-Feature Analysis Minutely Data*

There were several top performers from the multi-feature sets. The top two include [High, Low] with 79% accuracy and [RSI, Prices] with 74%. Do note the use of "Prices" as part of the training feature set. While realizing the difficulties with this strategy, it was done for purely experimental reasons. Considering the strange single feature results, a baseline was needing to be established and this seemed well suited for that purpose. The process of including Prices in the training feature set was conducted in a sound manner. Prices were isolated based on a datetime cutoff point. The feature training set included prices prior to that date, and the feature testing set included prices after that cutoff datetime. However, casting doubt on those results, there are still a number of other multi-feature sets that include both traditional features and technical features with promising validation and testing accuracies. Again, its noteworthy to point out the similar phenomenon here as in the single feature results. Multi-feature training accuracies were equally poor with promising validation and testing accuracies.

Table V. Multi-Feature Results

Multi-Feature Results			
Feature Set	**Train MAPE**	**Test/Val MAPE**	**Test/Val Accuracy**
High, Low	280%	21%	79%
RSI, Prices	284%	26%	74%
High, Low, RSI	283%	28%	72%
OBV, Prices	281%	29%	71%
High, OBV Slope	285%	35%	65%
High, ADL	281%	38%	62%
High, RSI	282%	44%	56%
ADL, Prices	280%	44%	56%

V. Conclusion

The aim of this research is to identify short-term trends of BTC using both traditional and technical features in an effort to better understand and document the instable nature of the cryptocurrency market. The inevitable goal is to educate the public at large leading to eventual adoption and mitigation of glaring flaws in our current monetary system. While a full, widespread adoption is unlikely, a future certainly exists where fiat currency and cryptocurrency work along side one another in some form. This result would be the most ideal as it would maintain the massive infrastructure of the current system while providing more control and security to the public.

The results of this research are inconclusive due to the resulting mismatch of training and testing MAPE and Accuracy. As mentioned in the previous section, the possible reasons for this behavior are as follows:

1. Underfitting

Underfitting could be the root cause only if dropout is also a root cause, as these work in conjunction. Underfitting occurs when a model is not complex enough to accurately capture relationships between a dataset's features and a target variable. While not ideal, an underfit model is somewhat encouraging because it suggests more room for improving the model as opposed to a model with poor training, validation and testing accuracy or an overfit model requiring more in-depth feature engineering.

2. Too Much Dropout

When considering the model's implementation, this possibility becomes more likely. The model interleaves a 20% dropout with each LSTM layer and a final 50% dropout layer. This was done in an effort to avoid overfitting; however, it appears to have worked against the model. When considering how an LSTM cell works, it becomes clearer that too much dropout would then lead to an underfit model. Dropout layers are used to define a percent of data that should be dropped prior to being fed into the next layer. This only occurs in the training phase. During validation and testing, the model has access to all data thus leading to a higher accuracy score than training.

3. Bad Validation & Test Data

The term "bad data" suggests that there may be some incongruence between the training and validation/testing data sets. This is also a liklier possibility given the vast number of issues presented in the data gathering and cleaning steps. If the data in the training set is less representative of the desired time-series while the data in validation and testing is more representative, this behavior may be exhibited.

The results of this research are inconclusive yet still useful. The results indicate that the hypothesis is viable and reinforces that traditional features alone are a good predictor while technical features alone are not. However, the research indicates that using technical features with traditional features may lead to improved accuracy predictions. With testing and validation scores at such high levels for the multi-feature sets, but incongeunce between that and training scores, it warrants further exploration and experimentation. If all scores were poor, then there may be a need to abandon the hypothesis entirely. However, that is not the case here.

An immediate next step for this research is to determine the source of the mismatching accuracy scores. Armed with the information about possible best feature mixes, the model can be tested in various ways to seek improvement. If the model is underfit due to too much dropout, analysis can be conducted to determine the optimal dropout, as well as optimal number of layers and nodes. Finally, a deep dive into the dataset will mitigate the possibility of bad data. A simple, yet effective strategy for this will be to plot out the various data points in different ways to look for inconsistencies.

References

[1] E. Almasri and E. Arslan, "Predicting Cryptocurrencies Prices with Neural Networks," *2018 6th International Conference on Control Engineering & Information Technology (CEIT).* http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8751939&isnumber=8751702

[2] J. Sun, Y. Zhou and J. Lin, "Using machine learning for cryptocurrency trading," *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8780358&isnumber=8780112

[3] Z. Jiang and J. Liang, "Cryptocurrency portfolio management with deep reinforcement learning," *2017 Intelligent Systems Conference (IntelliSys).* http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8324237&isnumber=8324208

[4] https://towardsdatascience.com/the-best-forecast-techniques-or-how-to-predict-from-time-series-data-967221811981

[5] https://www.investopedia.com/

[6] https://medium.com/neuronio/predicting-stock-prices-with-lstm-349f5a0974d4

[7] https://scikit-learn.org/