# Machine Learning for Bitcoin Price Prediction

**Gabriel Belmont (yutab@bu.edu)**
**Hyunsoo Kim (hkim42@bu.edu)**
**Vikram Varma (vvarma@bu.edu)**
**Wenxuan Yan (yanwx@bu.edu)**

## Abstract

Using machine learning techniques for asset price prediction is often considered difficult because although models can be trained to fit price data closely, the ability to predict accurately is yet to be shown. In order to determine if a different approach to asset price prediction could be used that relies on sentiment data, we implemented a Long Short Term Memory Recurrent Neural Net, a Simple Recurrent Neural Net, Linear Regression, and a Random Walk price prediction model for Bitcoin. By using the machine learning techniques and comparing them to a set of baseline strategies, the ultimate aim is to try to construct a trading strategy based around a machine learning technique that is capable of beating more simplistic strategies such as buying and holding. It was found that despite being able to beat baseline strategies under certain conditions, no machine learning based strategy was capable of beating buying and holding during the recent bull market in Bitcoin.

## 1 Introduction

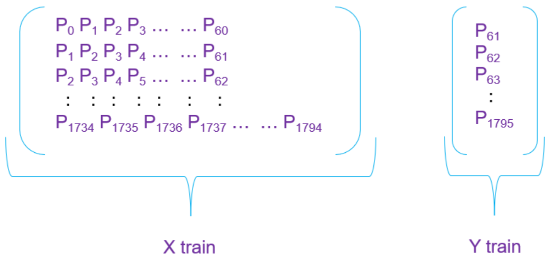
Recurrent neural networks (RNNs) have gained popular use for tasks such as machine translation, time series prediction, speech recognition, and machine control amongst others. Despite seeing some use and utility in trading, it has been shown that RNNs have performed relatively poorly at predicting the prices of assets and securities such as stocks. In spite of being able to fit time series data very accurately, RNNs are not great at consistently predicting trend changes in stock prices due to the sheer complexity of price discovery in a free and ever changing market. The ability for a market actor to predict price better than other market participants is an extremely lucrative endeavor and has been the core pursuit of market participants since the inception of markets. The intention of this paper is to examine the difficulty of predicting prices through several different machine learning techniques and to find a good machine learning predictor model.

*Novelty and Significance*: 8 variables pertaining to Bitcoin[1] price and sentiment were collected over a period of 5 years and 4 months from 2016-01-01 to 2021-04-03. In order to get a better informed model, not only is market data aggregated for training, but also sentiment data is collected from Google search trends data with the volume of keyword searches for “Bitcoin”, “Bitcoin energy”, and “Bitcoin near me”.[2] Additionally, the accuracy of the model was determined through 2 main metrics: 1. the standard deviation between predicted price and actual price and 2. the correctness of predicted price direction. Standard deviation alone is a weak metric for determining the usefulness of a model because a trailing predictor that predicts next day's price based on the previous day’s price like such: $P_d = A_{d-1}$, where $P_d$ is the predicted price on day $d$ and $A_{d-1}$ is the actual price on the previous day, is not generating information that is useful for price prediction despite a low standard deviation. In order to get around this, the predicted direction of price was taken into account. In other words, if the model correctly predicted the direction of price movement for the following day ($P_d > A_{d-1}$ is predicting positive price movement on day $d$, $P_d < A_{d-1}$ is predicting negative price movement on day $d$) with a probability significantly different than 0.5, the model is deemed useful. The prediction is checked against the actual price movements ($A_d - A_{d-1}$), where positive values are positive price movements.

A model that predicts price direction correctly less than half the time can simply be inverted to become a useful model. Thus a useful model would allow a market participant to buy, sell, or short Bitcoin with a greater than 0.5 probability of profiting by the next 24 hour increment. We'll call this metric $C$, the probability of correctly predicting the direction of price movement.

## 2 Creating the Predictive Models

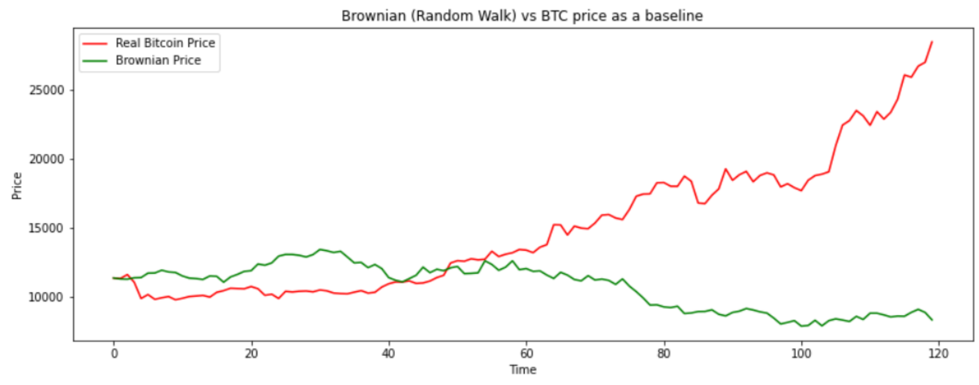
Before running our models, we had to prepare our data. First the Bitcoin data is aggregated into a collection of 9 variables by column: date, open price, high price, low price, closing price, volume, "Bitcoin" Google keyword search volume, "Bitcoin near me" Google keyword search volume, and "Bitcoin energy" Google keyword search volume. The date column was removed because each row is taken at a constant time increment of 24 hours. The data was then scaled using the default MinMaxScaler function which scales each feature to be between 0 and 1 while maintaining correct proportionality. The data was then aggregated into a 3 dimensional array of shape ($p$, $q$, $r$) whereby $p$ denotes the 8 variables described above, $q$ denotes the previous 60 days used to determine the 61st day, and $r$ denotes the number of 60 day groupings, visualized by the following:


$$\underbrace{\begin{pmatrix} P_0\ P_1\ P_2\ P_3 \dots \dots P_{60} \\ P_1\ P_2\ P_3\ P_4 \dots \dots P_{61} \\ P_2\ P_3\ P_4\ P_5 \dots \dots P_{62} \\ \vdots \\ P_{1734}\ P_{1735}\ P_{1736}\ P_{1737} \dots \dots P_{1794} \end{pmatrix}}_{\text{X train}} \quad \underbrace{\begin{pmatrix} P_{61} \\ P_{62} \\ P_{63} \\ \vdots \\ P_{1795} \end{pmatrix}}_{\text{Y train}}$$

Where $P_i$ represents a single grouping of the 8 training variables, X train is the procured training data, and Y train is the output values for predicted price.

### 2.1 Random walk price prediction

As a baseline price prediction model, we used a random walk.[3] As a general overview, the random walk model starts at a given price value and predicts in increments of 1 day. The only required input is the initial price $P_1$. From there the next day's price is predicted through a Gaussian distribution $g$ from [-1, 1] and a scalar value $a$ such that $P_n = g*a + P_{n-1}$, where $P_n$ is the predicted price on day $n$. On average the random walk price fluctuates but remains around the initial value. By comparing the random walk baseline price prediction to the price of Bitcoin over a 120 day period, the following graph was generated:

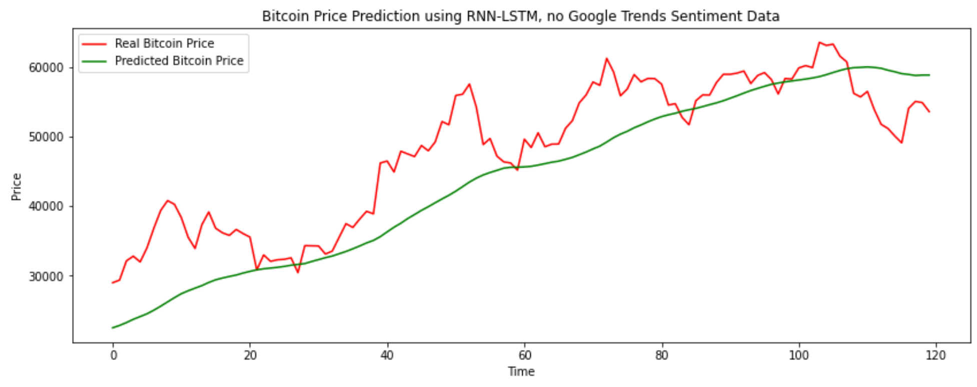


The standard deviation for this particular graph is σ = 1917.93 and the probability of predicting the price direction for the next day was $C = 0.471$.

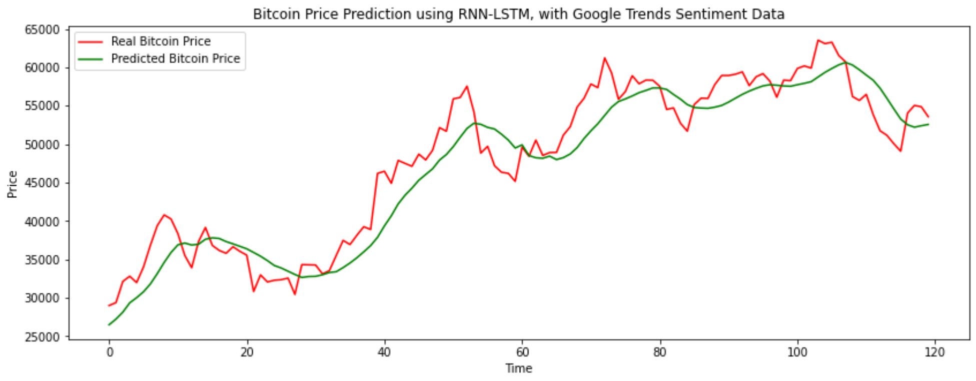
### 2.2 Linear regression

The linear regression model implements a coefficient for each feature that minimizes the cumulated error over the entire input dataset. Meaning the coefficients may not necessarily minimize the error for a particular datapoint. While there are many different implementations for measuring the cumulative error, least squares error is chosen for the model. The coefficients are transformed using the gradient descent principle by adjusting the individual coefficients up or down, whichever reduces the error. This is implemented manually in the code, which has the advantage of making the process transparent. One of the practical benefits of the transparency is the ability to tune the magnitude in which the coefficients are adjusted.

#### 2.2.2 Linear regression results



The figure above shows two plots: green for the predicted prices and red for the actual prices. This plot is obtained using the linear regression model trained without the sentiment data with σ = 6508.72 and C = 0.487. In practical terms, the model seems good for setting an expected lower bound for the future prices. The predicted prices curve can be seen matching the price dips on around the 25th, 60th, and 80th days. However it fails to predict the bigger dip around the 110th day. Depending on how the actual prices trend going forward, significant price dips below the predicted curve may be interpreted as a specific event instead of a simple failure.



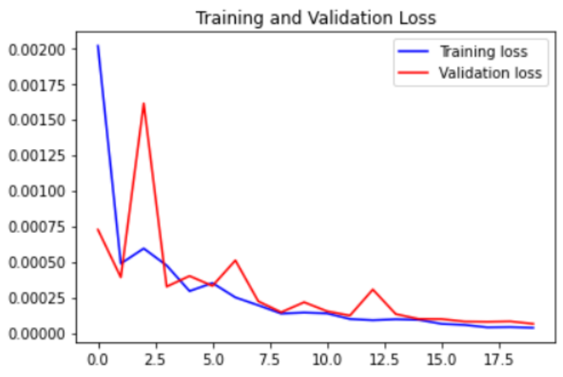
The figure above is obtained using the linear regression model trained with the sentiment data with σ = 3303.65 and C = 0.445. Compared to the model without the sentiment data, it can be seen that the predicted prices curve follows the actual prices curve more closely. Under close inspection, however, it appears that the predicted curve is simply lagging behind the actual curve. This makes the model practically useless.

## 2.3 Simple recursive neural net

A simple RNN remembers its previous input using an internal memory. This internal memory structure makes RNNs ideal for dealing with sequential data, hence its use for price prediction with time series data.

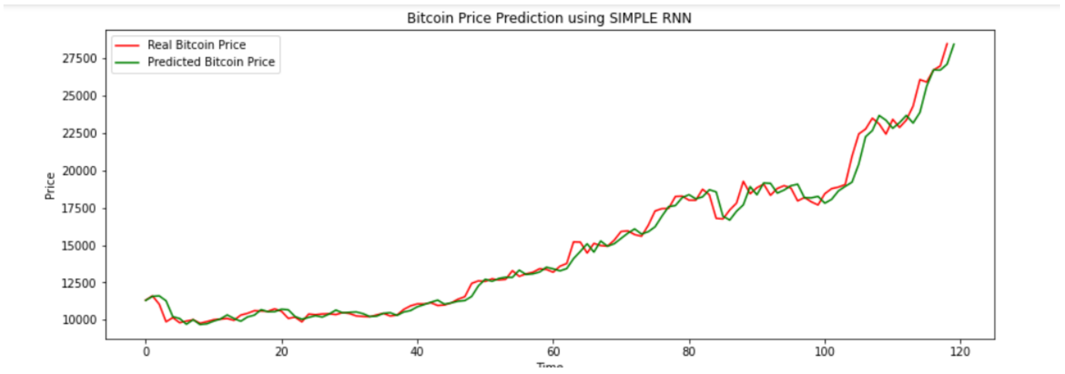
### 2.3.1 Simple RNN setup

Using 4 Simple RNN layers, a batch size of 50, a validation split of 0.1, and 20 epochs, the model was trained on 4 years worth of data resulting in the following training and loss validation:

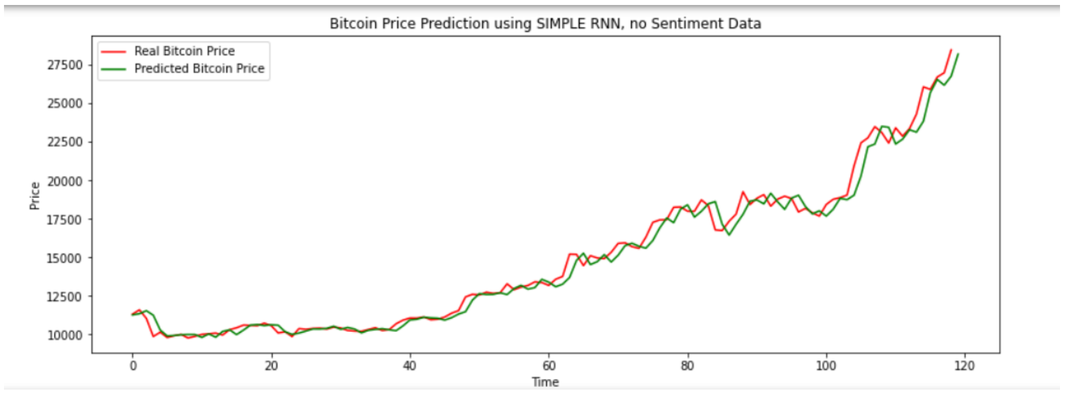


### 2.3.2 Simple RNN results

Using the last 120 days of data as a test to check the predictive power of this model, the Simple RNN model was capable of generating a closely fitted prediction model with σ = 733.56 and $C$ = 0.454, where $C$ is the probability of the model correctly predicting the direction of price movement in the next 24 hours. For comparison, a unidirectional strategy where the model always predicts upward price movement day after day, following the general market trend, generates $C = 0.6167$. This indicates that the model is not useful for its predictive power. The following graph was generated using the Simple RNN prediction model over 120 days:

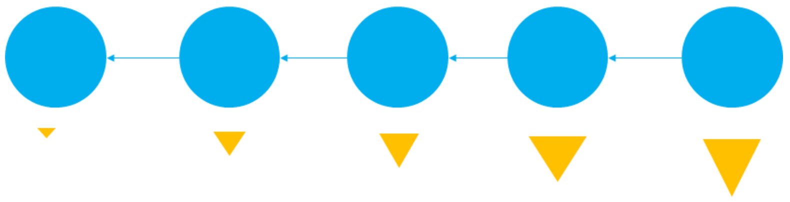


For comparison, the Simple RNN model was also trained without Google sentiment data, resulting in $\sigma = 559.14$ and $C = 0.521$. Thus without sentiment data the model was not capable of getting a significantly better fit, yet was slightly better at predicting the directional price movements of the following day. The following graph shows the Bitcoin price vs the predicted price for a Simple RNN with no sentiment data.



### 2.4 Long short term memory (LSTM) recursive neural net

An LSTM model is a subclass of an RNN capable of learning long-term dependencies. Simple RNN models have an issue retaining long-term information as during back-propagation in a simple RNN, each node in a layer calculates its gradient with respect to the effects of the gradient in the prior layer. If adjustments to a prior layer are small, the adjustments to the current layer will be even smaller. This “vanishing gradient” problem limits the capability of a Simple RNN to incorporate long-term dependencies.

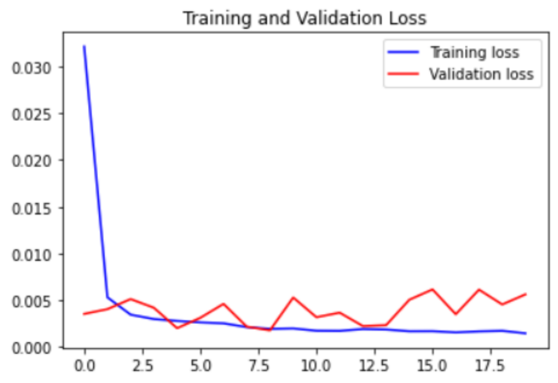


“Vanishing Gradient”

The ability of an LSTM to capture long-term dependencies has made LSTMs  more effective at predicting time series data and due to this may show promise for our purposes.

### 2.4.1 LSTM RNN setup

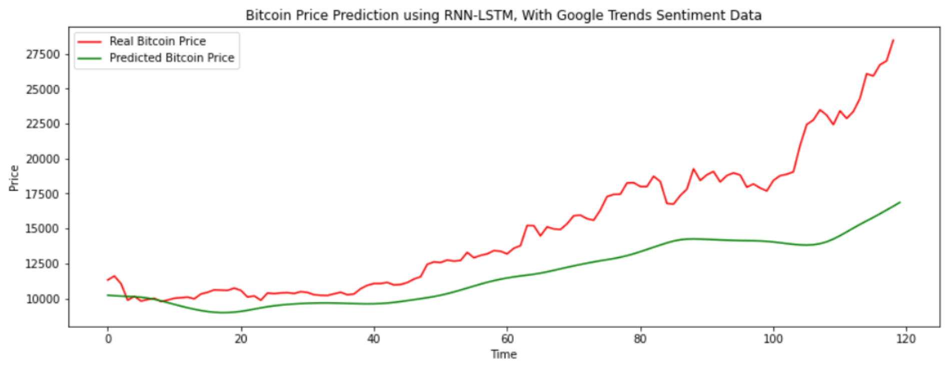
The LSTM RNN data was procured in the same manner as described in section (2.3.1). The model itself consists of 4 LSTM layers which have a dropout at each layer of 0.2, 0.1, 0.1, and 0.1 respectively. The model also trained on 20 epochs with a batch size of 50 and a validation split of 0.1. Despite changing the fit, toggling the dropout parameters were not found to have any significant effect on the predictive power of the model. The following training and validation loss for each epoch is generated:



Although we utilized 20 epochs in the model, we determined that the model performed well after 10 epochs and that training and validation losses could be minimized with a lower number of epochs and less training.

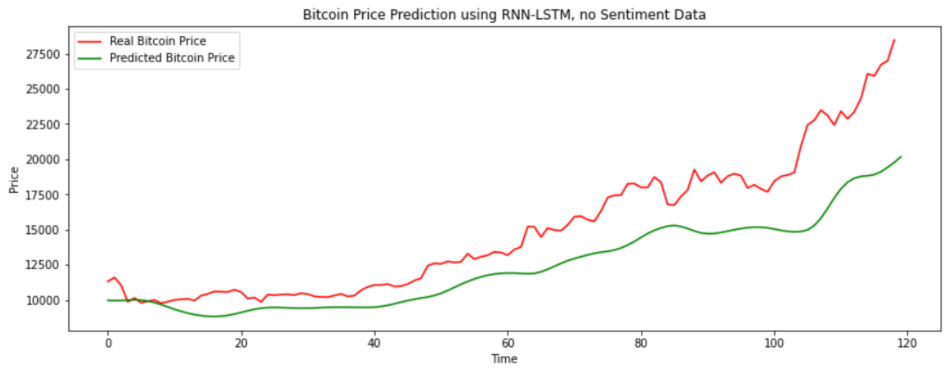
### 2.4.2 LSTM RNN results

Using the last 120 days of data as a test to check the predictive power of this LSTM model, the model was capable of generating a prediction with $\sigma = 4081.74$ and $C = 0.411$, where $C$ is the probability of the model correctly predicting the direction of price movement in the next 24 hours. This indicates that the model is not useful because although it could get a relatively close fit to the actual data, its $C$ value was almost exactly the same as a random walk model. A unidirectional strategy model where the model predicts only upward price movement would have done better with $C = 0.617$. The following graph was generated using the LSTM RNN prediction model over the last 120 days:



Notice that for the 120 days, the model predicts a lower price almost every day. This is because the market moved rapidly up over 100% in the 120 days. This demonstrates that the model, although it is better at predicting more stabilized price movements, consistently underpredicts rapid upward price movements. The opposite is also true, when the price drops rapidly, the model will consistently predict higher prices than actual prices.

For comparison, an LSTM with the same parameters was trained without Google sentiment data. This model resulted in $\sigma =$ 2214.12 and $C = 0.490$. The following graph was generating using the model trained with no sentiment data comparing the Bitcoin price vs the predicted price:



## 3 Experimental Results Comparison

The best machine learning model turned out to be the Simple RNN with no sentiment data, although no model was capable of beating a "buy and hold" strategy, indicating that the models hold no significant predictive power. Below is the order of the models from highest $C$ value to lowest $C$ value:

1. Always predict upward price movement (buy and hold): $C = 0.617$
2. Simple RNN with no Sentiment Data: $C = 0.521$
3. LSTM without Sentiment Data: $C = 0.490$
4. Brownian: $C = 0.471$
5. Simple RNN with Sentiment Data: $C = 0.454$
6. Linear Reg. $C = 0.445$
7. LSTM with Sentiment Data: $C = 0.411$

## 4 Discussion

Although the LSTM and Simple RNN models failed to yield any predictive power above a random brownian model, for the sake of curiosity, a trading strategy was constructed around the models to determine their efficacy in the market.

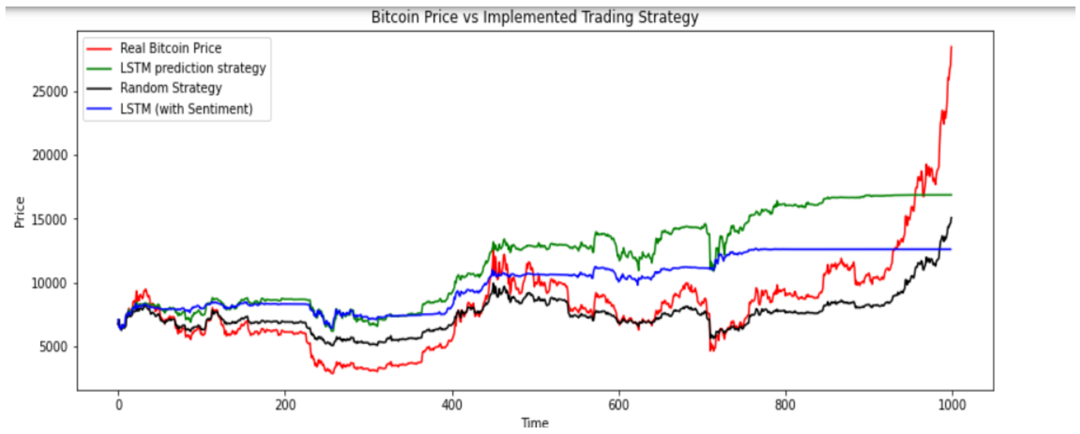
In order to see the actual effects of implementing a trading strategy around one of these models, a test was run over a 120 and 1000 day period.

### 4.1 Trading strategy

Suppose $A_i$ is the actual price of Bitcoin and $P_{i+1}$ is the prediction of tomorrow's price on day *i*.

If the predicted price gradient is positive for the next day $(P_{i+1} - A_i > 0)$, the trading algorithm would allocate a percentage of its portfolio to Bitcoin if possible. If $P_{i+1} - A_i < 0$, the algorithm would trade out of a percentage of its Bitcoin into US dollars. By running this strategy over the initial allotted 120 day period, all algorithms were incapable of beating a "buy and hold" strategy. This is clearly in part due to the high appreciation during the 120 day test data, where the price of Bitcoin appreciated over 100% in 120 days. For a period of mostly unidirectional positive price movement like this of course it is hard to beat a buy and hold strategy.

In order to detect if the model would perform any better during a sideways price movement period, the model was tested against a larger set of test data. The following are the results of using an LSTM model and applying the strategy described above where every day 10% of either the US dollar holdings or 10% of the Bitcoin holdings were traded to the other asset based on whether or not Bitcoin was predicted to go up or down. In order to see the results during non-bull market conditions, the test was run over a 1000 day period. All tests start with 1 Bitcoin and no USD reserves, although as time goes on the trading strategies can adjust their holdings.



The figure above compares the price of Bitcoin (red), LSTM prediction using the strategy described above (green and blue), and a random strategy where buying and selling was chosen randomly (black) with 0.5 probabilities. As the graph shows, the LSTM strategy was actually capable of beating the returns of both other strategies until the final bull run, where it ended up losing to both. The LSTM model without sentiment data started with 1 Bitcoin or $6737.18, and ended with 0.562 Bitcoin or $16017.51, losing nearly half its value to a "buy and hold" strategy despite gaining 137% in US dollar terms. The LSTM with sentiment data included performed worse than without sentiment data.

The appearance of flat regions in the portfolios using LSTM prediction is due to the model consistently predicting a lower Bitcoin price than the actual price day after day (fig. in section 2.4.2). This happens when price moves parabolically, something Bitcoin is known for.

## 5. Conclusion

Bitcoin price prediction is a complex model to solve as Bitcoin is significantly more volatile than traditional fiat currencies. Our LSTM and RNN models underperform relative to a "buy and hold" strategy based on recent Bitcoin market performance with sentiment data as these models tend to "undershoot" actual prices. We believe this was due to the difficulty of modeling negative versus positive sentiment within Google trends keyword searches. This is an area that could be improved with additional work on key words with negative and positive sentiment impacts on Bitcoin.

References:
[1] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2008.
[2] L. Kristoufek, *What are the main drivers of the Bitcoin price? Evidence from wavelet coherence analysis,* PloS one, vol. 10, no. 4, 2015.
[3] E. Fama, *Random walks in stock market prices*, Financial analysts journal, 1995
[4] E Altay and M. Satman, *Stock Market Forecasting: Artificial Neural Network and Linear Regression Comparison in an Emerging Market,* Journal of financial management and analysis, Vol 18, Issue 2, 2005