User Guide for Bitcoin Prediction Investigation

By Michael Gallagher

Contents

Data Collection:

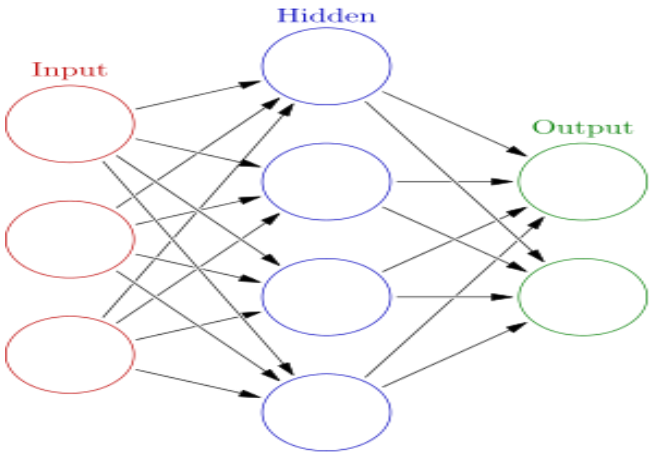
The following is a brief user guide of how the Bitcoin predictions came about. All data that was obtained was stored in a database located on Azure.

The first step in this process is to obtain the necessary data for training and runtime. I gathered Bitcoin data first through the Cryptocompare API. Next, I gathered Sentiment data from tweets. The Sentiment data was based around the search term Bitcoin and tweets were continuously pulled down from twitter based on this term. As the tweets were pulled down they were ran through a sentiment analyser. Sentiment analysis is a term used to determine if a piece of text is positive, negative or neutral.

The sentiment analyser for this project was called Vader-Sentiment-Analysis (Hutto & Gilbert, 2018). This was chosen based on my own research into the subject and a research paper by the creators of Vader Sentiment Analysis where they found that found Vader had a 96% accuracy in predicting the sentiment value of Tweets. (Hutto & Gilbert, VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text, 2018).

Recurrent Neural Network(RNN):

After collecting the data next, I had to decide how I was going to use it. This meant deciding on a neural network. A Neural network is used when you want to recognise patterns in data. You give the network inputs demonstrated below on the left side and outputs on the right-hand side. There is a hidden layer in between where the network try's out combinations of the data.



The neural network I ended up choosing for my project was a recurrent neural network. The difference between a RNN and a standard neural network is that a recurrent neural allows the edges to loop back they don't all just flow in a single direction as demonstrated in *fig 3*. RNN were designed in order to use information in the past in order to predict what will happen in the future. However, an issue arises the larger the distance between the past and the present information becomes, as the RNN becomes unable to learn to connect the relevant information. This is also known as the vanishing gradient problem.

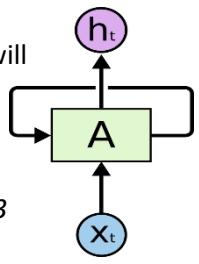


Fig 3

LSTM:

More specifically the RNN I chose was a Long Short Term Memory Network also known as LSTM's. These networks are designed to learn long-term dependencies and thus eliminating the vanishing gradient problem. They are widely used in language modelling, image captioning, question answering and learning to predict stock prices. Eliminating the vanishing gradient problem was a key reason why I chose LSTM as my RNN.

Time Series Forecasting:

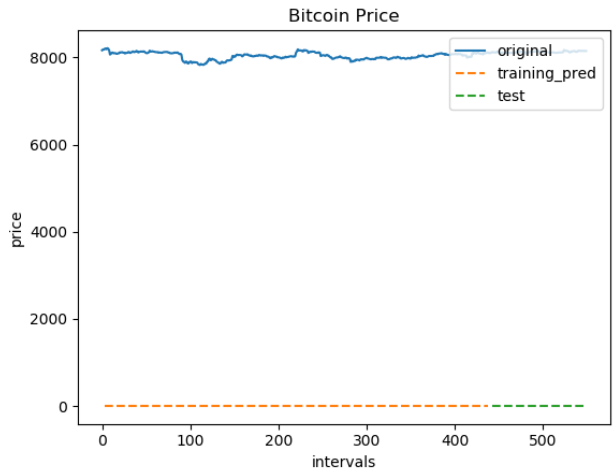
In a normal dataset time does not play a role in the outcome or prediction. Instead all prior observations are treated equally for the most part. However, in a timeseries dataset it is different. Time adds a specific order between the data: a time dimension. This dimension acts as both a constraint and a structure to provide additional information. A time-series uses previous observation values in in order to predict future values. So, for example given the given the price of bitcoin now what will be the price of bitcoin in 5 minutes? To find this, I used a sliding window, meaning if I set the window to the size of 3 then it will use the first 3 data points as output to predict the next 3. To do this I use a function that will shift the specified column forward for example below. All Nan values will be then converted to 0 in order for the data to be processed.

Original data	shift of -1	shift of -2	shift of -3
1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7
5	6	7	8
6	7	8	9
7	8	9	10
8	9	10	Nan
9	10	Nan	Nan
10	Nan	Nan	Nan

Pre-process Data:

Normalize the Data

Once the data input data has been decided on it must be normalized. The data gets normalised due to fact that LSTM's are sensitive to the scale of the input. Therefore, it is a good idea to normalise data before inputting it into the LSTM network.



Example of unnormalized data.

Train & Test Data:

To train and test our model we need to split up the data into two parts to determine how well our model does on unseen data. For this model I chose the split to be 80:20.

Reshape Data:

After these steps have been performed the train and test data is run through a function that takes in the size of the window the data will take for forecasting and the data you wish to forecast. It returns 2 datasets in the from of numpy arrays where Value X is the set at a given time and Y is t+1.

After creating values X and Y the data must be transformed once again to fit the sequential model. The Sequential model expects the input shape in (Samples(Our dataset), Timesteps(the window size), features(the different inputs from our data).

Model:

Features:

I decided I would create two models based on the features I obtained. Bitcoin is a currency which is heavily driven by speculation and thus it makes sense sentiment would be influential in driving the price up and down. For the first model I decided to create a model based around Sentiment and the closing price. This meant I had 3 inputs or features; Sentiment, Total Tweets and Closing price. As the model is sequential there is no need to include time. It is assumed to be 5 minutes.

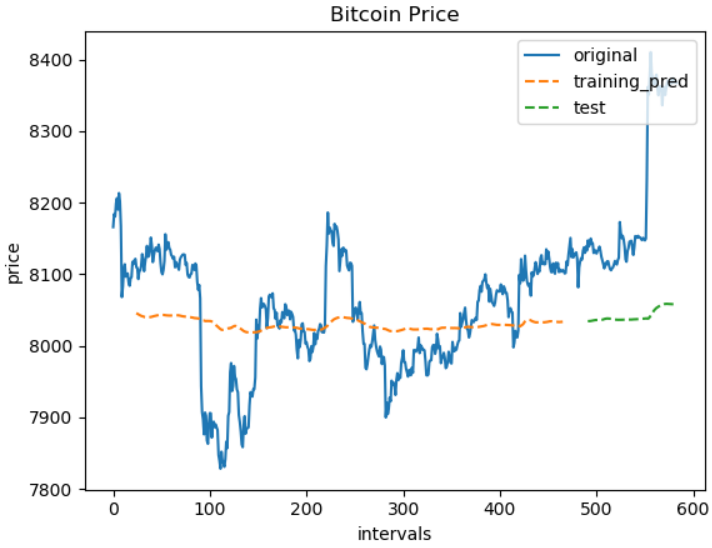
The second model took in 10 features taking in the model one data plus; opening price, high & low price, upper & lower Bollinger bands (measures volatility), volume of bitcoin sold and total price it was sold for.

These features are the ran through the pre-process data stage above and are fed into the model.

LSTM:

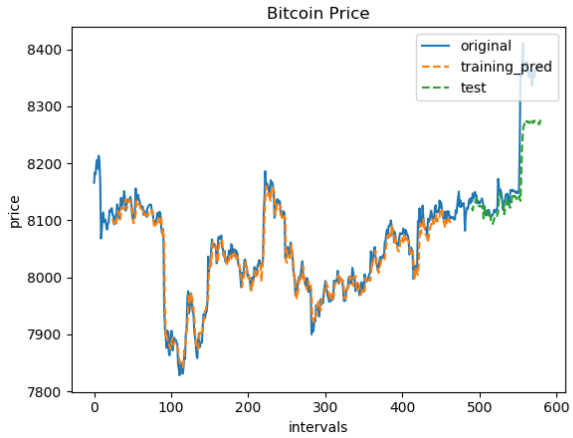
The model I ended up choosing had 3 layers. Two Lstm layers and 1 dense layer which is the output layer. After looking into optimizer's I decided on the Adam optimizer while choosing the Tanh activation function. I had one layer originally but I found two gave more accurate results.

My selection of epoch's (the number of forward and backwards passes of the training examples) was 205 due to fact when it was low it didn't predict my model very well as seen below.

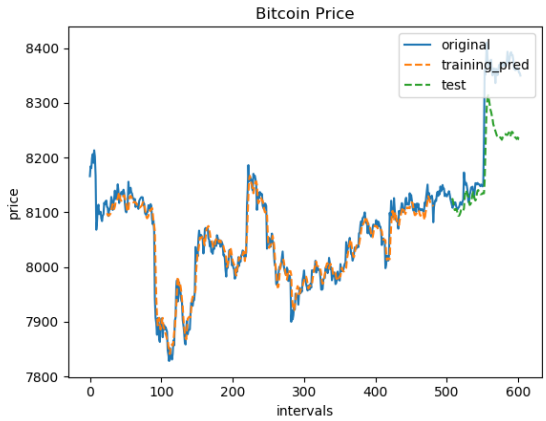


Conclusion:

My final models shown below both had problems with lagging behind which can be expected dealing when trying to deal with time-series. Lagging was more noticeable in the 10 input model compared to three input model.

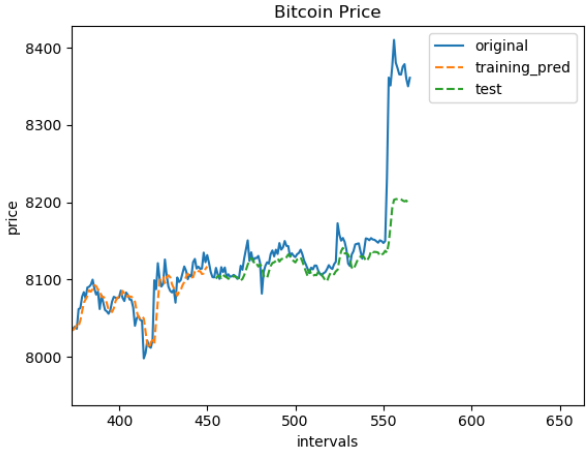


10 Input model



3 Input model

Another issue was that if the model hadn't seen an unexpected jump before it struggled to match the values when predicting.



Bibliography

Cryptocompare. (2018, April 13). *https://www.cryptocompare.com/api/#*. Retrieved from cryptocompare api: https://www.cryptocompare.com/api/#

Hutto, C., & Gilbert, E. (2018, April 13). *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text.* Retrieved from http://comp.social.gatech.edu: http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf

Hutto, C., & Gilbert, E. (2018, April 13). *VADER-Sentiment-Analysis*. Retrieved from Github.com: https://github.com/cjhutto/vaderSentiment