A Project Report

On

Predicting Bitcoin Price Movements using Recurrent Neural Networks

Submitted in the partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Submitted By

16621A0522	**B. Saikiran**
16621A0532	**Ch. Deepak Yogi**
16621A0543	**D. Sai Siva**
16621A0544	**D. Om Prakash**

Under the Guidance of

Dr. T.M. USHA

PROFESSOR

Department of Computer Science Engineering & Information Technology

AURORA'S ENGINEERING COLLEGE

(Affiliated to JNTU-H, approved by AICTE New Delhi and Accredited by NBA)

2016-20

AURORA'S ENGINEERING COLLEGE
Department of Computer Science Engineering

CERTIFICATE

This is to certify that this project entitled **"Predicting Bitcoin Price Movements using Recurrent Neural Networks"** is a bonafide for the work carried out by **B. Saikiran, Ch. Deepak Yogi, D. Sai Siva, D. Om Prakash** bearing Hall Ticket numbers **16621A0522, 16621A0532, 16621A0543, 16621A0544** in the partial fulfilment of requirement for the award of the Degree of Bachelor of Technology in **COMPUTER SCIENCE AND ENGINEERING** JNTU, Hyderabad is a record of bonafide work carried by them.

The results presented in this thesis have been verified and are found to be satisfactory.

Dr. T.M. USHA
Professor

Dr.Arun Singh Chouhan
HoD of CSE Department

External Examiner

ACKNOWLEDGEMENT

We take this opportunity to remember and acknowledge the cooperation, goodwill and support, both moral and technical extended by several individuals out of which this project evolved. We shall always cherish our associate on with them.

We have immense pleasure in expressing my thanks and deep sense of gratitude to my project guide **Dr T.M. USHA, Professor, Department of Computer Science Engineering, Aurora's Engineering College**, for the guidance and help throughout the Development of this project work by providing us with the required information.

We express our profound gratitude to Prof**. Dr. Arun Singh Chouhan, HOD of Computer Science Engineering, Aurora's Engineering College** for his support and encouragement in completing our project. We would like to thank him for encouragement and valuable guidance in bringing to this dissertation.

We are also thankful to **Prof. K. CHANDRA SHEKAR, Principal of AURORA'S ENGINEERING COLLEGE**, for his continuous help and support during the project development.

A lot, thanks to other faculty members of the department who gave their valuable suggestions at different stages of our project.

We are very much thankful to our parents who helped us with the utmost friendliness and warmth always. They kept our spirit flying high and persistently encouraged us to undertake and complete this project.

B. Saikiran	**16621A0522**
Ch. Yogi Deepak	**16621A0532**
D. Sai Siva	**16621A0543**
D. Om Prakash	**16621A0544**

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

AURORA'S ENGINEERING COLLEGE

2016-2020

CERTIFICATE

We B. Saikiran, Ch. Deepak Yogi, D. Sai Siva, D. Om Prakash bearing Hall ticket numbers 16621A0522, 16621A0532, 16621A0543, 16621A0544 hereby declare that the Project entitled **"Predicting Bitcoin Price Movements using Recurrent Neural Networks" is** being submitted by us in partial fulfillment of the requirement for the award of the Degree of Bachelor of **Technology in COMPUTER SCIENCE & ENGINEERING**, JNTU, Hyderabad and is a record of bonafide work carried out by us under the guidance of **Dr. T.M. Usha Professor**, Department of Computer Science and Engineering, Aurora's Engineering College, Abids.

The results embodied in the dissertation have not been submitted to any other University or institute for the award of any degree or diploma.

B. Saikiran	16621A0522
Ch. Yogi Deepak	16621A0532
D. Sai Siva	16621A0543
D. Om Prakash	16621A0544

Contents

ABSTRACT

In the past eleven years of Bitcoin's history, the economy has seen the price of Bitcoin rapidly grow due to its promising outlook on the future for cryptocurrencies. Investors have taken note of several advantages Bitcoin provides over the traditional banking system. One such trait is that Bitcoin allows for decentralized banking, meaning that Bitcoin cannot be regulated by powerful banks. There is also a market cap of 21 million Bitcoins that can be in circulation, therefore a surplus of Bitcoins cannot be "printed" which would result in inflation. Bitcoin resolves the issue with transaction security by using a blockchain, or a ledger, which records the history of every transaction ever made into one long hexadecimal "chain" of anonymous transactions, which keeps transaction history transparent, but also confidential. Bitcoin as a result has become a very bullish investing opportunity, and due to the huge volatility of the Bitcoin market price, this paper attempts to aid in investment decision making by providing Bitcoin market price prediction. Our team explored several Machine Learning algorithms by using Supervised Learning to train a prediction model and provide informative analysis of future market prices. We start with Linear Regression models, and train on several important features, then build a more efficient Polynomial Regression model. We proceed with the implementation of Recurrent Neural Networks (RNN) with Long-Short-Term-Memory (LSTM) cells. All code is written in Python using Google's TensorFlow software library. We show that the price of Bitcoin can be predicted with Machine Learning. Further on this investigation, we show the usage of LSTM architecture with the aforementioned time series. To conclude, we outline the results of predicting Bitcoin price for upto 180 days ahead.

LIST OF FIGURES

LIST OF TABLES

LIST OF SCREENS

CHAPTER 1

INTRODUCTION

1.1 Motivation and Project Overview:

Cryptocurrency has become increasingly popular over the few years. Bitcoin is one of the most popular and widely known cryptocurrencies. It is mainly designed to remove the need of any third-party entities or financial institutions and thereby eliminate the possibility of fraud during the transactions. One significant and unique characteristic of Bitcoin is the multi-version concurrency control that allows safe concurrent transactions without any significant delay. Bitcoin has a total market cap of around $800 billion USD at a time in January 2018 and Bitcoin having a market cap of around $260 billion USD share in the cryptocurrency market as of today (i.e. 21 – 05 -2020). Not only the investors but also brokers and private investors are finding cryptocurrency as an investment tool. In this regard, it is very much necessary to predict the future values of cryptocurrency so as to take correct trading decisions. In the past years, traditional statistical methods such as linear regression were popular. However, due to uncertainty in the trends of financial markets, computing techniques such as neural networks have become quite popular. Artificial Neural networks can adjust by itself based on the information given to it. They have the capability to capture the non-linear trends of the financial markets. More complex deep learning methods such as RNNs and LSTM techniques should yield substantially higher prediction accuracy. Some studies have used RNN's and LSTM to forecast Bitcoin pricing in comparison with traditional ARIMA models. McNally 2018 show that RNN and LSTM neural networks predict prices better than the traditional multilayer perceptron (MLP) due to the temporal nature of the more advanced algorithms.

1.2 Problem Definition:

Bitcoin is the longest running and most well-known cryptocurrency. Cryptocurrencies are relatively unpredictable compared to traditional financial instruments. The increase/decrease in Bitcoin's price with large percentages over short periods of time is an interesting phenomenon which is difficult to be predicted at times. This Project aims to predict the price of these Cryptocurrencies with Deep Learning using Bitcoin as an example so as to provide insight into the future trend of Bitcoin. These predictions could be used as the foundation of a bitcoin trading strategy. To make these predictions, first take we will have to familiarize yourself with a machine learning technique Recurrent Neural Network (RNN), Long Short Term Memory (LSTM) cells. Regardless of the speculative bubbles in Bitcoin price, this project aims to give a hint on the future closing price values of this cryptocurrency.

1.3 Objective of The Project:

This work presents a model to predict Bitcoin Price using Recurrent Neural Network(RNN), Long Short Term Memory (LSTM) cells by using TensorFlow and Keras in Python. This project aims to give a hint on the movement of future Bitcoin prices and future closing price values. The predicted closing price using these neural networks are presented as outcomes. To conclude, we outline the results of predicting Bitcoin price for upto 180 days ahead. At the end, the report discusses possible improvements that can be made to increase the scope of the experiment.

CHAPTER 2

LITERATURE SURVEY

This chapter discusses related work done in the area of predicting the Bitcoin price using various approaches. Also, we will look at how the current approach evolved based on past motivating use cases.

2.1 Introduction:

Due to the high volatility associated with the market price of Bitcoin, prediction models have become more and more challenging for investors to accurately forecast investment decisions. Thinking in regards to a stock market investment, a typical approach many investors would use as a prediction metric, is called Fibonacci retracement. Fibonacci retracement helps identify the opportune moments for traders to buy or sell stock at the optimal price. Fibonacci retracement utilizes the volatility of the stock, meaning that the two extreme trends in a graph are found, and the calculated vertical distance is then divided by key Fibonacci ratios, most commonly the golden ratio of 0.618. These values are then used to help identify the most prominent support and resistance levels. The problem with such prediction models however, is that the Fibonacci retracement levels are static, and are constant calculated values that are only good metrics for simple identification. In regards to Bitcoin, time series predictions is not a novel idea for brokerage companies, and many brokers have invested heavily to improve performance of forecasting stock prices and foreign exchange rates by way of Machine Learning. Bitcoin, a cryptocurrency first launched in 2009 by Satoshi Nakamoto, has parallels with many of the stock markets such as the New York Stock Exchange (NYSE) and can be modelled in similar ways. However, there are key differences between Bitcoin and the stock market which makes our prediction model much different in nature. For one, Bitcoin commands a much smaller transaction volume as compared to many of the major stock exchanges. The average daily volume for Bitcoin at the present day is $10 billion, whereas trillions are seen for the latter. As a result of the small transaction volume, lower liquidity, and huge peak in investment turnover of Bitcoin as the future of decentralized banking increases, the market price of Bitcoin has become highly volatile and as a result, challenging to make successful predictions While there is a great amount of scholarly literature and practical guides on Machine Learning applied to stock market, there is a lack of information about predicting movement of the value of cryptocurrencies. Namely, there is no consensus on the features that are to be used for training a prediction model. Our research primarily focuses on blockchain size as an important feature of the market price trend, using various Machine Learning models, as well as expanding our input variables into multi featured data. Previous attempts at time series forecasting have been well-documented and there are a lot of resources and tools available to use for free which our team plans to utilize in order to solve the problem at hand efficiently rather than reinventing the wheel at each stage. Specifically, and Google's TensorFlow library for Python allows for function calls to complex Machine Learning algorithms such as Gradient Descent. Other libraries such as Keras will also be implemented for more complex modelling.

2.2 Bitcoin:

Bitcoin is a crypto currency which is used worldwide for digital payment or simply for investment purposes. Bitcoin is decentralized i.e. it is not owned by anyone. Transactions made by Bitcoins are easy as they are not tied to any country investment can be done through various marketplaces known as "bitcoin exchanges". These allow people to sell/buy Bitcoins using different currencies. The largest Bitcoin exchange was Mt Gox & HCoin, Coinbits and BitForex are present largest primary Bitcoin exchanges as of January 2020. Bitcoins are stored in a digital wallet which is basically like a virtual bank account. The record of all the transactions, the timestamp data is stored in a place called Blockchain. Each record in a blockchain is called a block. Each block contains a pointer to a previous block of data. The data on blockchain is encrypted. During transactions the user's name is not revealed, but only their wallet ID is made public.

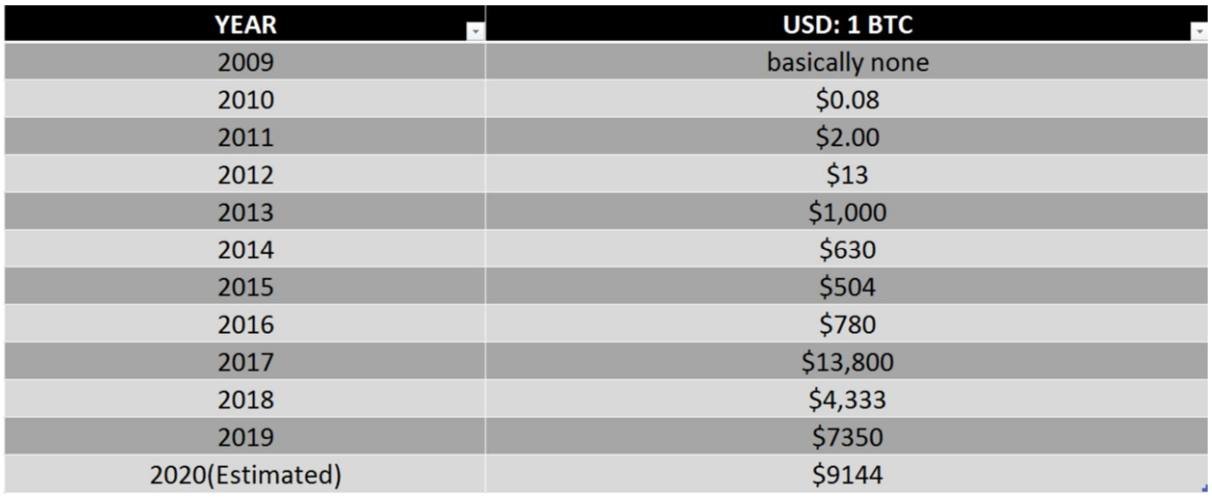
2.3 Bitcoin Price:

Many endorse Bitcoin, while other are sceptic. Regardless, the price of Bitcoin is a topic discussed from an economic angle, computer science, financial, and psychological perspective. In the time of writing this article, the price seems to be getting stable. In some way, this may be normal given that many investors are waiting to see regulations, the scaling problems is not seeing any improvement, the fact that the first hype around 2017 is now over and it is normal for a market to get stable for some time. At present (i.e. January 2020) the

Bitcoin is averaging at 9000 United States Dollars and is equivalent to nearly 7 Lakh Indian Rupees for one Bitcoin. Nevertheless, given the features of Bitcoin like:

- Tax free & Anonymity
- Unforgeable
- Decentralized
- Verifiable and secure
- Negligible transaction fees
- Cannot be counterfeited

It is a plausible solution for countries with developing economies and financial systems to improve their economical position, while struggling to access the best technology of its time. However, for a country to have an aptitude towards this monetary system, a techsavvy population should be present to adopt the system, along with regulations from financial institutions that support it. Both of these are absent for the moment, but the future prospect is very potent. Another factor why we believe Bitcoin should be studied, is the fast paced advancements in technology, that favour Bitcoin, considering its qualities as a software and a decentralized system, which cannot be beaten by banking systems, or a likes.



YEAR	USD: 1 BTC
2009	basically none
2010	$0.08
2011	$2.00
2012	$13
2013	$1,000
2014	$630
2015	$504
2016	$780
2017	$13,800
2018	$4,333
2019	$7350
2020(Estimated)	$9144

Table 2.3 Bitcoin Closing Price by Year.

2.4 Bitcoin Inflation:

Bitcoin is not debt based, and no artificial money can be issued. Additionally, because of the fixed supply mentioned above no more Bitcoins than predicted can be created, unlike the economy system of today. Bitcoin's deflationary attribute, stems from imitating gold, in that a currency must be scarce (or with a finite supply in case of BTC), consequently, no one can increase the supply and inflate the value of goods. Nonetheless, the bitcoin has also its dark side which sometimes makes users quite sceptical on its possession and usage. Several of these problems have been reported (and may not be limited to) by Kaspersky Lab.

We may include the facts that: blockchain nodes do exactly the same thing (no paralleling, no synergy, no mutual assistance), growing size of storage used (currently 100 GB for the Bitcoin for each high-grade Bitcoin network client), transaction confirmation needs 10 50 minutes, etc. While we will try to build a predictive model for the Bitcoin prospect value calculation, we are aware in advance that price may differ greatly because of internal and external factors to Bitcoin. By internal factors we are presuming factors inside the Bitcoin security (some breach). By external we are referring to agents which influence indirectly the price of Bitcoin, exchange closures, replacing cryptocurrencies, speculation markets, the fact that as its believed widely over 80% of Bitcoins in circulation is concentrated in a limited number of investors etc. Bitcoin's supply is predetermined *by design*, and represented by this geometric series:

$$S_n = \frac{a(1-r^n)}{1-r} = 210000 \times \frac{50(1-0.5)}{1-0.5} \approx 21 \times 10^6.$$

As it is clearly stated in Bitcoin's wiki, the decrease of supply, resembles the rate at which commodities like gold are mined. This makes many consider Bitcoin as deflationary, but this currency is infinitely divisible, not only because 1 BTC = 10^8 *satoshis*, and in turn, no one would run out that fast of every satoshi, but also because the protocol could be updated allowing for satoshis to be more divisible (have more zeros). As a result, deflation, does not have to occur.

2.5 Established Methods and Approaches:

The technical details and approaches previously used for prediction of the Bitcoin closing and the highest price are discussed in this section.

2.5.1 Artificial Neural Networks for Prediction:

Artificial neural networks (ANNs) are inspired by the working of human nervous system, that is, the neurons of a human brain. It consists of multiple nodes, each of which takes input data and performs some basic operations and results are passed to the other neurons. Neurons can transmit information among themselves using links between them. Each of the links has a weight (synaptic weight) associated with it. An activation function is used to calculate whether the neuron fires. This should be a non-linear function in order to be able to express complex patterns in the data. The following diagram illustrates a simple ANN.

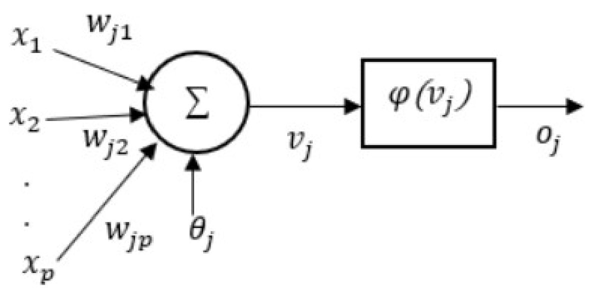


Figure 2.5.1 Representation of a simple neuron

input signals = $\{x1, x2, . . , xp\}$ synaptic
weights = $\{wj1, wj2, . . , wjp\}$
bias value = θj
activation potential = vj activation function (generally non-linear sigmoid function) = $\varphi(vj\ )$
oj is the Output signal defined as follows:

$$\boldsymbol{oj} = \boldsymbol{\varphi}(\boldsymbol{\theta j\ p\ i}=\mathbf{1}\ \boldsymbol{wji\ xi})$$

A typical neural network has an input layer (leftmost layer) which sends stimulus to the network, output layer (rightmost layer) and all the layers between input and output layers are called hidden layers. Once the neural network is trained with data, it the weights are adjusted, and knowledge is stored. When a new input signal arrives that is not used for testing, based on the pre-calculated weights, output will be generated. The current work focuses on two temporal neural networks, that is – Time-Delay Neural Networks (TDNN) and Recurrent Neural Networks (RNN).

2.5.2 Time-Delay Neural Network(TDNN):

Time-delay neural networks is one popular and potential methods for prediction of time series data. A simple time-delay neural network with one hidden layer can be illustrated as follows:

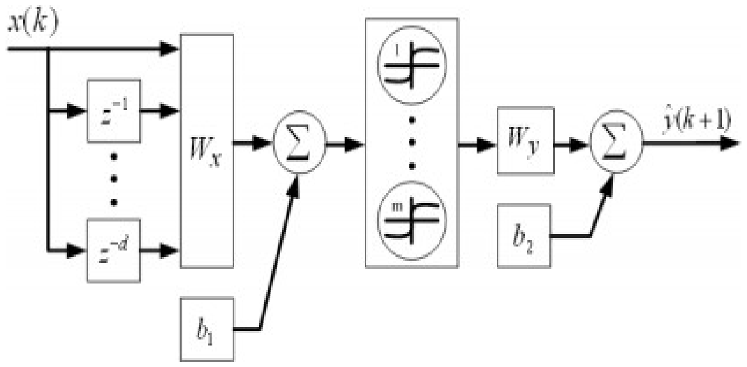


Figure 2.5.3 Simple TDNN model with one hidden layer

In this study, we have used a simple time-delay neural network with one hidden layer. Past observations of a variable serve as input data, which will be used to perform one-step ahead prediction. Assuming ŷ(k+1) is the output from TDNN, it can be calculated as follows:

$$\hat{y}\,(\boldsymbol{k}+\mathbf{1}) = \boldsymbol{g}(\boldsymbol{Wx} * [\boldsymbol{x}(\boldsymbol{k})\ \boldsymbol{x}(\boldsymbol{k}-\mathbf{1}) \ldots \boldsymbol{x}(\boldsymbol{k}-\boldsymbol{d})]\boldsymbol{T} + \boldsymbol{b}\mathbf{1}) * \ \boldsymbol{Wy} + \boldsymbol{b}\mathbf{2}$$

where Wx and Wy are weight vectors, $b1$ and $b2$ represents bias value, d denotes the time delays applied to input vector $x(k)$, g represents the activation function.

2.5.3 ARIMA:

Autoregressive integrated moving average (ARIMA) is a statistical regression model, which can be utilized in time series forecasting applications, such as finance. ARIMA makes predictions while considering the lagged values of a time series, while accommodating for non-stationarity. The model, which is one of the most popular linear models for time series forecasting, originates from the autoregressive (AR) and moving average (MA) models, as well as their combination, also known as ARMA. For making predictions with the ARIMA model, we had to follow a step by step procedure to be able to feed the data to the ARIMA model. This first involved Visualizing the time series data: It is essential to analyse the trends prior to building any kind of time series model. The details we are interested in pertains to any kind of trend, seasonality or random behaviour in the series.

2.5.4 Recurrent Neural Network (RNN):

Recurrent neural networks make use of one or feedback connections. The idea behind recurrent neural networks is that to find the next value that might occur in the sequence, better we have an idea of which ones come before it. RNN are called “recurrent” because they perform same task on each element of the sequence and also the output of this step depends on the previous computations. The input to an RNN is based on all the previous inputs, that are used for feedback connections. From this it can be stated that the output of an RNN is function of current external input along with inputs and outputs of the previous state.

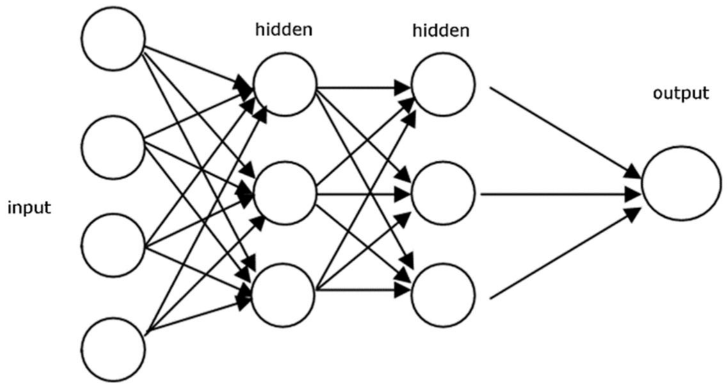


Figure 2.5.4. Simple RNN model with one hidden layer

Recurrent networks are the state of the art in nonlinear time series prediction, temporal pattern classification. The idea behind RNNs is to make use of sequential information. In a traditional neural network, we assume that all inputs (and outputs) are independent of each other. But for many tasks that’s a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a memory which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps (more on this later). Here is what a typical RNN looks like:

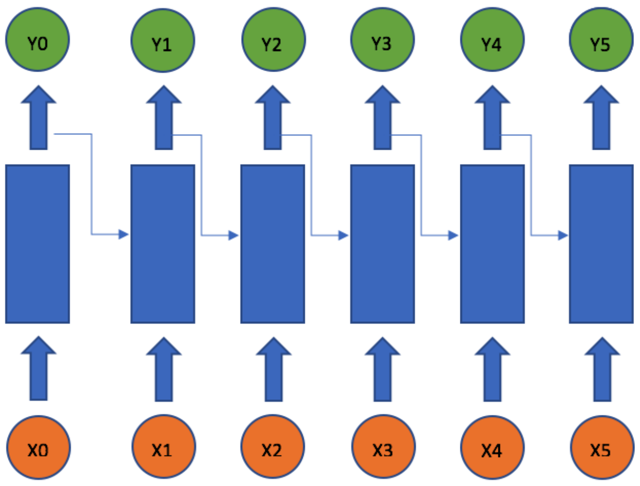


Fig 2.5.4.i Simple representation of RNN Structure

The above diagram shows a RNN being unrolled (or unfolded) into a full network. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network.

2.5.5 Bidirectional RNN:

Based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also future elements. For example, to predict a missing word in a sequence you want to look at both the left and the right context. Bidirectional RNNs are quite simple. They are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs

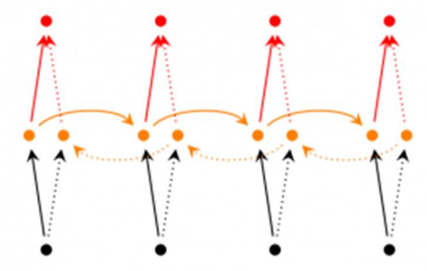


Fig 2.5.5 Bidirectional RNN structure

Traditional time series prediction methods include univariate Autoregressive (AR), Univariate Moving Average (MA), Simple Exponential Smoothing (SES), and Autoregressive Integrated Moving Average (ARIMA). Due to the lack of seasonality in the cryptocurrencies market and its high volatility, these methods are not very effective for this task. Machine learning methods seem to be promising in this regard. Machine learning methods have been applied for asset price/return prediction in recent years by incorporating non-linearity in prediction models to deal with non-stationary financial time-series. Machine learning techniques have been successfully applied for stock markets prediction (Enke 2005, Huang 2005, Sheta 2015, Chang 2009). However, there is a dearth of machine learning application in the cryptocurrency price prediction literature. In contrast to traditional linear statistical models such as ARMA, the artificial intelligence approach enables us to capture the non-linear property of the high volatile crypto-currency prices. Examples of machine learning studies to predict Bitcoin prices include random forests (Madan 2015), Bayesian neural network (Jang 2017), and neural networks (McNally 2018). Deep learning techniques developed by Hinton et. al 2006 have been used in literature to approximate non-linear functions with high accuracy (Cybenko 1989). There are a number of previous works that have applied artificial neural networks to financial investment problems (Chong 2017, Huck 2010). However, Pichl and Kaizoji 2017 conclude that although neural networks are successful in approximating bitcoin log return distribution, more complex deep learning methods such as RNNs and LSTM techniques should yield substantially higher prediction accuracy. Some studies have used RNN's and LSTM to forecast Bitcoin pricing in comparison with traditional ARIMA models (McNally 2018, Guo 2018). McNally 2018 show that RNN and LSTM neural networks predict prices better than the traditional multilayer perceptron (MLP) due to the temporal nature of the more advanced algorithms. Karakoyun 2018 in comparing the ARIMA time series model to the LSTM deep learning algorithm in estimating the future price of Bitcoin, find significantly lower mean absolute error in LSTM prediction. In this paper, we focus on two aspects to predict Bitcoin price. We consider a set of exogenous and endogenous variables to predict Bitcoin price. Some of these variables have not been investigated in previous research studies on Bitcoin price prediction. This holistic approach should explain whether Bitcoin is a financial asset. Additionally, we also study and compare RNN models with traditional machine learning models and propose a GRU architecture to predict Bitcoin price. GRU's train faster than traditional RNN or LSTM and have not been investigated in the past for cryptocurrency price prediction. In particular, we developed a LSTM architecture which can learn the Bitcoin price fluctuations more efficiently than the traditional models. We compare our model to both the traditional neural networks and time series models with different lookback periods to check the robustness of the architecture. For application purposes in algorithmic trading, we have implemented our proposed architecture to test two simple trading strategies for profitability.

2.6 LSTM:

A survey of the current literature on neural networks, reveals that traditional neural networks have shortcomings in effectively using prior information for future predictions (Wang, 2015). RNN is a class of neural networks which uses their internal state memory for processing sequences. However, RNNs' on their own are not capable of learning long-term dependencies and they often suffer from short-term memory. With long sequences, especially in time series modelling and textual analysis, RNNs' suffer from vanishing gradient problem during back propagation. If the gradient value shrinks to a very small value, then the RNNs' fail to learn longer past sequences, thus having short-term memory. LSTM, the Long Short-Term Memory is an RNN architecture with feedback connections, designed to regulate the flow of information. LSTMs' are a variant of the RNN which are explicitly designed to learn long-term dependencies. A single LSTM unit is composed of an input gate, a cell, a forget gate (sigmoid layer and a tan h layer) and an output gate (Figure 1). The gates control the flow of information in and out the LSTM cell. LSTMs' are best suited for time-series forecasting. In the forget gate, the input from the previous hidden state is passed through a sigmoid function along with the input from the current state to generate forget gate output ft. The sigmoid function regulates values between 0 and 1; values closer to 0 are discarded and only values closer to 1 are considered. The input gate is used to update the cell state. Values from the previous hidden state and current state are simultaneously passed through a sigmoid function and a tan h function and the output

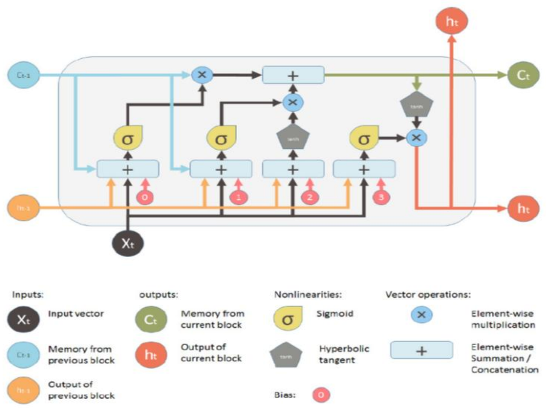
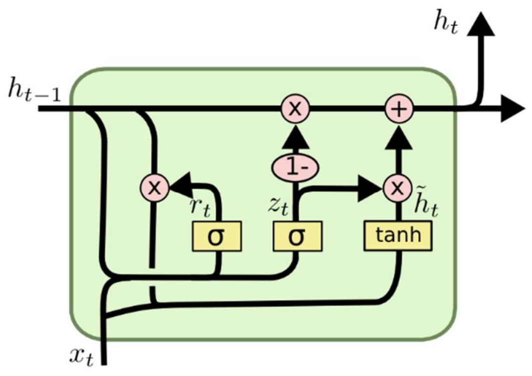


Fig 2.6 LSTM cell circuit

The previous cell state value is multiplied with the forget gate output and then added pointwise with the output from the input gate to generate the new cell state *c* as shown in equation 1. The output gate operation consists of two steps: first the previous hidden state and current input values are passed through a sigmoid function and secondly the last obtained cell state values are passed through a tan h function. Finally, the tanh output and the sigmoid output are multiplied to produce the new hidden state, which is carried over to the next step. Thus, the forget gate, input gate and output gate decide what information to forget, what information to add from the current step and what information to carry forward respectively. GRU introduced by Cho et. al (2014) solves the problem of the vanishing gradient with a standard RNN. The GRU is similar to LSTM but it combines the forget and the input gates of the LSTM into a single update gate. The GRU further merges the cell state and the hidden state. A GRU unit consists of a cell containing multiple operations which are repeated and each of the operations could be a neural, reset gate and a current memory content. These gates enable a GRU unit to store values in the memory for a certain amount of time and use these values to carry information forward, when required, to the current state to update at a future date, the update gate is represented by z where at each step the input x and the output from the previous unit $ht1$ are multiplied by the weight Wnd added together, and a sigmoid function is applied to get an output between 0 and 1. The update gate addresses the vanishing gradient problem as the model learns how much information to pass forward. The reset gate is represented by r in equation 2, where a similar operation as input gate is carried out but this gate in the model is used to determine how much of the past information to forget. The current memory content is denoted by ht where x is multiplied by W and r is multiplied by $ht1$ element wise (Hadamard product operation) to pass only the relevant information. Finally, a tan h activation function is applied to the summation. The final memory in the GRU unit is denoted by h which holds the information for the current unit and passes it on to the network. The computation in the final step is given in equation 2 below. As shown in equation 2, if z is close to 0 ((1- z close to 1), then most of the current content will be irrelevant and the network will pass majority of the past information and vice versa. Both LSTM and GRU are efficient at addressing the problem of vanishing gradient that occurs in long sequence models. GRU's have fewer tensor operations and are speedier to train than LSTM's.

If the weights in this matrix are small (or, more formally, if the leading eigenvalue of the weight matrix is smaller than 1.0), it can lead to a situation called vanishing gradients where the gradient signal gets so small that learning either becomes very slow or stops working altogether. It can also make more difficult the task of learning long-term dependencies in the data. Conversely, if the weights in this matrix are large (or, again, more formally, if the leading eigenvalue of the weight matrix is larger than 1.0), it can lead to a situation where the gradient signal is so large that it can cause learning to diverge. This is often referred to as exploding gradients.

LSTM networks are quite popular these days and we briefly talked about them above. LSTMs don’t have a fundamentally different architecture from RNNs, but they use a different function to compute the hidden state. The memory in LSTMs are called cells and you can think of them as black boxes that take as input the previous state h_{t-1} and current input x_t. Internally these cells decide what to keep in (and what to erase from) memory. They then combine the previous state, the current memory, and the input. It turns out that these types of units are very efficient at capturing long-term dependencies. The repeating module in an LSTM contains four interacting layers.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Fig 2.6.i LSTM circuit with operation notation

2.7 LSTM Implementation:

A chief feature of feedforward Networks, is that they don't retain any memory. So each input is processed independently, with no state being saved between inputs. Given that we are dealing with time series where information from previous Bitcoin price are needed, we should maintain some information to predict the future. An architecture providing this is the Recurrent neural network (RNN) which along with the output has a self-directing loop. So the window we provide as input gets processed in a sequence rather than in a single step. However, when the time step (size of window) is large (which is often the case) the gradient gets too small/large, which leads to the phenomenon known as vanishing/exploding gradient respectively [Chollet2017]. This problem occurs while the optimizer back propagates, and will make the algorithm run, while the weights almost do not change at all. RNN variations mitigate the problem, namely LSTM. The LSTM layer adds some cells that carry information across many timesteps. The cell state is the horizontal line from Ct-1 to Ct, and its importance lies in holding the long-term or short term memory. The output of LSTM is modulated by the state of these cells. And this is important when it comes to predict based on historic context, rather than only the last input. LSTM networks manage to remember inputs by making use of a loop. These loops are absent in RNN. On the other hand, as more time passes, the less likely it becomes that the next output depends on a very old input, therefore forgetting is necessary. LSTM achieves this by learning when to remember and when to forget, through their forget-gates. We mention them shortly to not consider LSTM just as a black box model [Olah2015].

- Forget gate: $f_t = \sigma(W_f S_{t-1} + W_f S_t)$
- Input gate: $i_t = \sigma(W_i S_{t-1} + W_i S_t)$
- Output gate: $o_t = \sigma(W_o S_{t-1} + W_o S_t)$

- Intermediate Cell State: $\tilde{C} = tanh(W_c S_{t-1} + W_c X_t)$
- Cell state (memory for next input): $c_t = (i_t * \tilde{C_t}) + (f_t * c_{t-1})$
- Calculating new state: $h_t = o_t * tanh(c_t)$

Figure 2.7 LSTM Gate Notation

As it can be seen from the equations, each gate has different sets of weights. In the last equation, the input gate and intermediate cell state are added with the old cell state and the forget gate. Output of this operation is then used to calculate the new state. So, this advanced cell with four interacting layers instead of just one tan h layer in RNN, make LSTM perfect for sequence prediction.

2.8 Conclusion:

The neural network models considered for the Bitcoin price prediction are simple neural network (NN), LSTM. The neural networks are trained with optimized hyper parameters and tested on the test set. Finally, the best performing model is considered for portfolio strategy execution. this project aims to give a hint on the future closing price values of this cryptocurrency

CHAPTER 3

DATA

3.1 Data Gathering: API – Crypto compare:

Several Bitcoin data sets are available online to download for free. Most of them provide the data related to price of Bitcoin on a minute to minute basis. However, the end goal of the project is to make prediction ahead of closing price of Bitcoin. So, we will need data such as closing price and Timestamp of Bitcoin for each day over period of several years. The Cryptocompare API provides the Bitcoin price data set, starting from past 5 years since present date.

This API gives access to Bitcoin exchanges and daily Bitcoin values. It allows users to customize the query while using the interface to use the bitcoin historical prices. This API returns the current price of any cryptocurrency and all the trading info (price, vol, open, high, low etc.) for the requested pairs, In particular we choose closing price and timestamp. Get open, high, low, close, volume from and volume to from the daily historical data. The values are based on 00:00 GMT time. This API is accessed using following address

https://min-api.cryptocompare.com

CRYPTO
COMPARE

3.2 Data Cleansing:

From exchange data we consider relevant only the Date, Close prices. For all data sets if NaN values are found to be existent, they are replaced with the mean of the respective attribute. After this, all datasets are merged into one, along the time dimension. We considered best to get rid of data points which are NA/empty, hence the data which will be passed to the neural network will be 5 years past today i.e., 2015 to 2020.

3.3 Normalisation:

Deciding on the method for normalizing a time series, especially financial ones is never easy. What's more, as a rule of thumb a neural network should load data that take relatively large values, or data that is heterogeneous (referring to time-series that have different scales, like exchange price, with Google Trends). Doing so can trigger large gradient updates that will prevent the network from converging. To make learning easier for the network, data should have the following characteristics [Geron2017]:

- Take small values - Typically, most values should be in the range 0-1 range
- Be homogeneous - That is, all features should take values at roughly the same range. The most common normalization methods used during data transformation include:
- Min-Max Scaling, where the data inputs are mapped on a number from 0 to 1:

$$x\ 0 = x - \min(X)$$
$$\max(X) - \min(X)$$

- Mean Normalization, which makes data to have a value between -1 and 1 with a mean of 0:

$$x\ 0 = x - \text{mean}(X)$$
$$\max(X) - \min(X)$$

- Z-Score (Standardization), where the features are redistributed with their mean of 0 and standard deviation of 1:

$$x\ 0 = x - \text{mean}(x)\ \rho$$

- For our problem, we use Min-Max Scaling and adjust features on a scale from 0 to 1 given that most of our time-series have a peek, therefore we might argue we know the maximum of the series, in which case Min-Max Scaling does a good job.

3.4 Data Pre-processing:

Data from different sources are merged and certain assumptions are made. Since, cryptocurrencies get traded twenty-four hours a day and seven days a week. The data-set values are normalized by first demeaning each data-series and then dividing it by its standard deviation. After normalizing the data, the dataset is divided into a training set.

In other words, the process of Data Gathering from online databases, Cleansing the obtained data (Data Cleansing) and Normalizing the cleaned data through min-max scalar and converting data in range of [0,1] (Data Normalization) are combined and are considered as Data pre-processing in the present model.

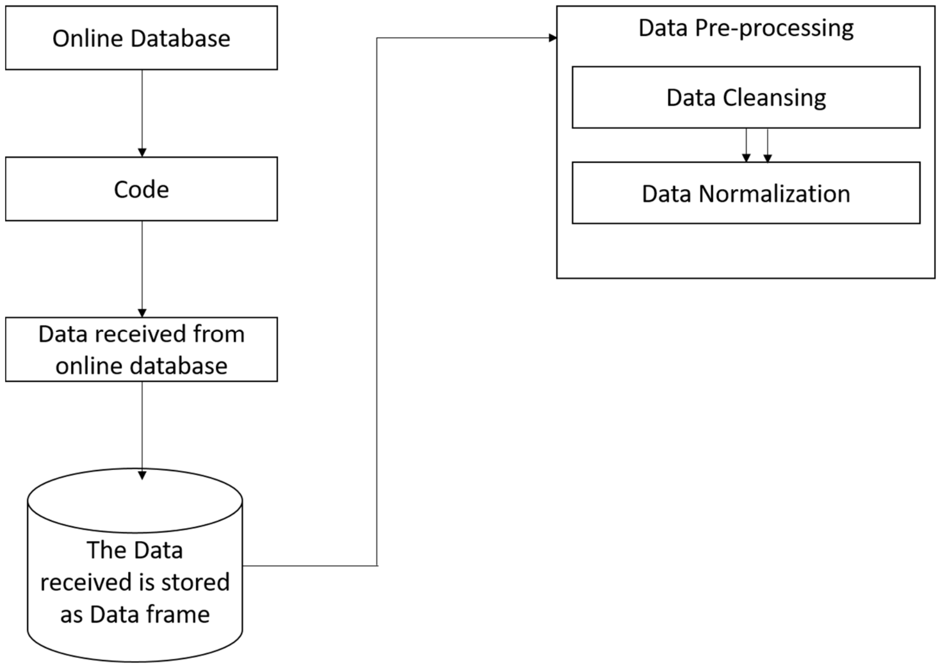

Online Database
Code
Data received from online database
The Data received is stored as Data frame
Data Pre-processing
Data Cleansing
Data Normalization

Fig 3.4 Flow diagram to represent Data Pre-processing

CHAPTER 4

ANALYSIS

4.1 Introduction:

In this section we describe how to make time series data adaptable for supervised machine learning problems. The price prediction is treated as regression rather than classification, and we show how LSTM can be used in such cases. We then, discuss hyper parameters.

4.2 Software Requirement Specification:

For Deep Learning backend system, we choose Tensor flow, and Keras as the front-end layer of building neural networks fast. Pandas is used extensively for data related tasks, Numpy is utilized for matrix/vector operations and for storing training and test data sets, Scikit-learn (also known as: sklearn) is used for per forming the min-max normalization. Lastly, Matplotlib is used for displaying the charts.

4.2.1 Software Requirements:

- **Python**, is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.
- Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.
- All the code is written in python using googles TensorFlow library as backend with Keras as frontend.

- **Microsoft Visual Studio Code,** Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS. It includes embedded Git and support for debugging, syntax highlighting, intelligent code completion, snippets, and code refactoring Working with Python in Visual Studio Code, using the Microsoft Visual Studio Code.
- Python extension in VS Code, is simple, fun, and productive. The extension makes VS Code an excellent Python editor, and works on any operating system with a variety of Python interpreters. It leverages all of VS Code's power to provide auto complete and IntelliSense, linting, debugging, and unit testing, along with the ability to easily switch between Python environments, including virtual and conda environments.

- **Anaconda,** Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is a cloud-based repository to find and install over 7,500 data science and machine learning packages.

4.2.2 Environment Setup:

There are number of packages and methods are used in this project. New libraries/packages can be installed using pip. pip is a de facto standard package-management system used to install and manage software packages written in Python. Many packages can be found in the default source for packages and their dependencies — Python Package Index. Most distributions of Python come with pip preinstalled.

4.2.2.1 Pandas:

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labelled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal. Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.
- Any other form of observational / statistical data sets. The data actually need not be labelled at all to be placed into a pandas data structure.

4.2.2.2 Numpy

Numpy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

  Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

4.2.2.3 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension Numpy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

4.2.2.4 Scikit-Learn

Scikit-learn (formerly scikits learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries Numpy and SciPy.

- Scikit-learn is largely written in Python, and uses Numpy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Python to improve performance. Support vector machines are implemented by a Python wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.
- Scikit-learn integrates well with many other Python libraries, such as Matplotlib for plotting, Numpy for array vectorization, pandas data frames, SciPy, and many more.

scikits
learn

4.2.2.5 TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

- TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization to conduct machine learning and deep neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well.

- TensorFlow provides stable Python and C++ APIs, as well as non-guaranteed backward compatible API for other languages.

4.2.2.6 Keras

The Keras is TensorFlow's high-level API for building and training deep learning models. It's used for fast prototyping, state-of-the-art research, and production, with three key advantages:

- User-friendly
  Keras has a simple, consistent interface optimized for common use cases. It provides clear and actionable feedback for user errors.
- Modular and compostable
  Keras models are made by connecting configurable building blocks together, with few restrictions.
- Easy to extend
  Write custom building blocks to express new ideas for research. Create new layers, metrics, loss functions, and develop state-of-the-art models.

4.2.2.7 JSON:

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

{JSON}

4.3 Hardware Requirements:

4.3.1 Minimum System Requirements:

- Operating Systems: 64-bit versions of Microsoft Windows 7, Windows 8.1, and Windows 10, Linux 16.04 or Latest, MacOS.
- Hard Disk Size: 2GB+, after the installation of prerequisites.
- RAM: 1 GB minimum
- Robust Internet Connection

4.3.2 Recommended System Requirements:

- Operating Systems: 64-bit versions of Microsoft Windows 7, Windows 8.1, and Windows 10. Linux 19.04 or 20.04, MacOS.
- Hard Disk Space: 250GB or more (SSD Recommended).
- RAM: 4GB+
- Robust Internet Connection.

CHAPTER 5

DESIGN

5.1 Introduction:

We used the Sequential API of Keras, rather than the functional one. The overall architecture is as follows:

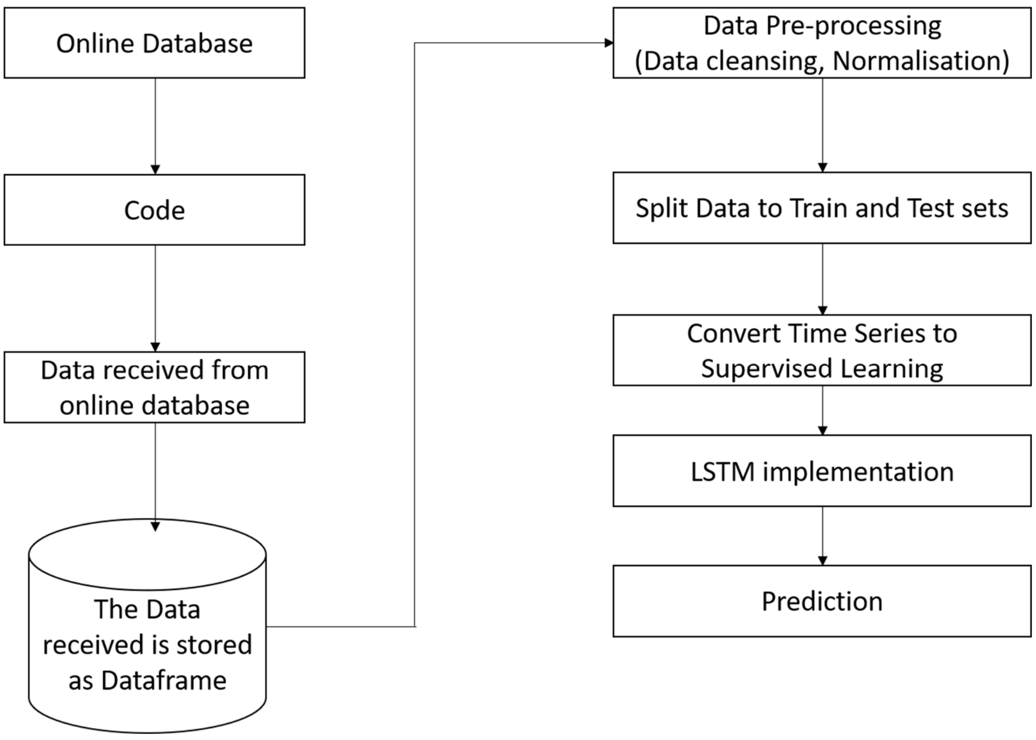

Online Database
Code
Data received from online database
The Data received is stored as Dataframe
Data Pre-processing (Data cleansing, Normalisation)
Split Data to Train and Test sets
Convert Time Series to Supervised Learning
LSTM implementation
Prediction

Fig 5.1 Flow of this Project

5.1.1 LSTM Layer:

The LSTM layer is the inner one, and all the gates, mentioned at the very beginning are already implemented by Keras, with a default activation of hard-sigmoid [Keras2015]. The LSTM parameters are the number of neurons, and the input shape as discussed above.

5.1.2 Dropout Layer:

Typically, this is used before the Dense layer. As for Keras, a dropout can be added after any hidden layer, in our case it is after the LSTM.

5.1.3 Dense Layer:

The **LSTM** recurrent **layer** comprised of memory units is called **LSTM**(). A fully connected **layer** that often follows **LSTM layers** and is used for outputting a prediction is called **Dense**()

5.1.4 Activation Layer:

Because we are solving a regression problem, the last layer should give the linear combination of the activations of the previous layer with the weight vectors. Therefore, this activation is a linear one. Alternatively, it could be passed as a parameter to the previous Dense layer.

5.2 Time Series Data:

Normally a time series is a sequence of numbers along time. LSTM for sequence prediction acts as a supervised algorithm unlike its auto encoder version. As such, the overall dataset should be split into inputs and outputs. Moreover, LSTM is great in comparison with c las sic statistics linear models, since it can easier handle multiple input forecasting problems. In our approach, the LSTM will use previous data to predict 30 days ahead of closing price. First, we should decide on how many previous days one forecast will have access to. This number we refer as the window size. We have opted for 60 days in case of monthly prediction, and 180 days in that of 2 months prediction, therefore the input data set will b e a tensor comprising of matrices with dimension 35x12/65x12 respectively, such that we have 12 features, and 35 rows in each window. So, the first window will consist of 0 to the 34 rows (python is zero indexed), the second from 1 to 35 and so on. Another reason for choosing this window length is that a small window leaves out patterns which may app ear in a longer sequence. The output data takes into account not only the window size but also the prediction range which in our case is 30 -180 days. The output dataset starts from row 35 up until the end, and is made of chunks of length 30-180. The prediction range also determines the output size for the LSTM network.

5.3 LSTM Internals:

A chief feature of feedforward Networks, is that they don’t retain any memory. So, each input is processed independently, with no state being saved between inputs. Given that we are dealing with time series where information from previous Bitcoin price are needed, we should maintain some information to predict the future. An architecture providing this is the Recurrent neural network (RNN) which along with the output has a self-directing loop. So, the window we provide as input gets processed in a sequence rather than in a single step. However, when the time step (size of window) is large (which is often the case) the gradient gets too small/large, which leads to the phenomenon known as vanishing/exploding gradient respectively [Chollet2017]. This problem occurs while the optimizer back propagates, and will make the algorithm run, while the weights almost do not change at all. RNN variations mitigate the problem, namely LSTM and GRU. The LSTM layer adds some cells that carry information across many timesteps. The cell state is the horizontal line from Ct − 1 to Ct, and its importance lies in holding the long-term or short-term memory. The output of LSTM is modulated by the state of these cells. And this is important when it comes to predict based on historic context, rather than only the last input. LSTM networks manage to remember inputs by making use of a loop. These loops are absent in RNN. On the other hand, as more time passes, the less likely it becomes that the next output depends on a very old input, therefore forgetting is necessary. LSTM achieves this by learning when to remember and when to forget, through their forget-gates. We mention them shortly to not consider LSTM just as a black box model.

5.4 Turn Data into Tensors:

LSTM expects that the input is given in the form of a 2-dimensional vector of float values. A key feature of tensors is their shape, which in Python is a tuple of integers representing the dimensions of it along the 2 axis. For instance, in our testing data of Bitcoin, the shape of training inputs is: (1611, 35), so we have 1600+ samples, a window size (timestep) of 35 values, and 12 features. In overall the idea is simple, in that we separate the data into chunks of 35, and push these small windows of data into a Numpy array. Each window is a 35x12 matrix, so all windows will create the tensor. Furthermore, in LSTM the input layer is by design, specified from the input shape argument on the first hidden, these three dimensions of input shape:

- Samples
- Window size

- Number of features

5.5 Hyper parameters:

5.5.1 Optimizer:

While Stochastic Gradient Descent is used in many Neural Network problems, it has the problem of converging to a local minimum. This of course presents a problem considering Bitcoin price. Some other nice optimizers are variations of adaptive learning algorithms, like Adam, Adagrad, and RMSProp. Adam was found to work slightly better than the rest, and that’s why we go for it. (All of these come packed with Keras.)

5.5.2 Loss Function:

The performance measure for regression problems, will typically be either RMSE (Root Mean Square Error) or MAE (Mean Absolute Error).

- RMSE(X, h) = r n1 Σni =1h(xi) − yi2
- MAE(X, h) = n1 Σni =1 h(xi) − yi

RMSE is generally used when distribution resembles a bell-shaped curve, but given the Bitcoin price spikes we chose to go MAE, since it deals better with outliers.

5.5.3 Activation Function:

The choice for activation function was not very difficult. The most popular are sigmoid, tanh, and ReLu. Sigmoid suffers from vanishing gradient, therefore almost no signal flows from the neuron to its weight, moreover it is not centered around zero, as a result the gradient might be to high or to low a number. By contrast, tanh makes the output zero centered, and in practice is almost always preferred to sigmoid. ReLu is also widely used, and since it was invented later, it should be better. Nevertheless, for predicting Bitcoin price that was not the case, and we chose tanh due to better results.

5.5.4 Dropout Layer:

Regularization is the technique for constraining the weights of the network. While in simple neural networks, l 1 and l 2 regularization is used, in multi-layer networks, drop out regularization takes place. It randomly sets some input units to 0 in order to prevent overfitting.

Hence, its value represents the percentage of disabled neurons in the preceding layer and ranges from 0 to 1. We have tried 0.25 and 0.3 and lastly, we decided for 0.3.

5.5.5 Number of Neurons in Hidden Layers:

We opted for 10 neurons in the hidden layers, it actually costs a lot to have more neurons, as the training process will last longer. Also, trying a larger number did not give improved results.

5.5.6 Epochs:

Rather arbitrarily, we decided for 30 epochs, after trying other values, like 50, or 30. As with the number of hidden layer neurons, the more epochs, the more time it takes for training to finish, since one epoch is a full iteration over the training data. Also, it may overfit the model.

5.5.7 Batch Size:

The **batch** size is a number of samples processed before the model is updated. The number of **epochs** is the number of complete passes through the training dataset. The size of a **batch** must be more than or equal to one and less than or equal to the number of samples in the training dataset. We decided to feed the network, with batches of 120 data (again this number is a guess).

5.6 Architecture of Network

We used the Sequential API of Keras, rather than the functional one. The overall architecture is as follows:

• **1 LSTM Layer**: The LSTM layer is the inner one, and all the gates, mentioned at the very beginning are already implemented by Keras, with a default activation of hard-sigmoid [Keras2015]. The LSTM parameters are the number of neurons, and the input shape as discussed above.

• **1 Dropout Layer:** Typically, this is used before the Dense layer. As for Keras, a dropout can be added after any hidden layer, in our case it is after the LSTM.

• **1 Dense Layer:** This is the regular fully connected layer.

• **1 Activation Layer:** Because we are solving a regression problem, the last layer should give the linear combination of the activations of the previous layer with the weight vectors. Therefore, this activation is a linear one. Alternatively, it could be passed as a parameter to the previous Dense layer.

5.7 Plotting of Predicted Data:

The data predicted is then plotted on a legend which consists of 2- Axis I.e. X- Axis which is independent and denotes Time and Y- Axis which is dependent and denotes the predicted data in form of Time series and Date axes plots. This plots are final outcome of this project and describes the future bitcoin movement which consists of 3 scenarios i.e. Increase if curve moves upward direction, static if the plot is horizontal to X – Axis and decreases if the plot moves in downward direction.

Additional features like storing the prediction model, printing and saving predicted values are also added in version 2 of our project code.

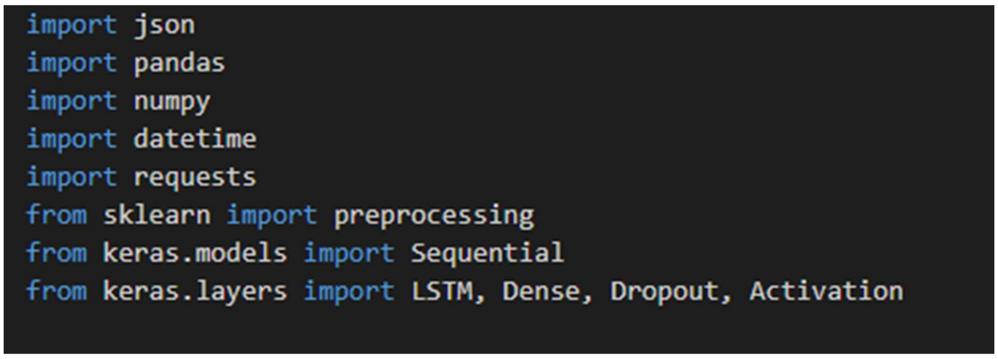
CHAPTER 6

IMPLEMENTATION AND RESULT

6.1 Introduction:

In this section we show the results of our LSTM model It was noted during training that the higher the batch size the worst the prediction on the test set. Of course, this is no wonder, since the more training, the more prone to overfitting the model becomes. While it is difficult to predict the price of Bit coin, we see that features are critical to the algorithm future work includes trying out the Gated Recurrent Unit version of RNN, as well as tuning, on existing hyper-parameters. Below we show the loss from the Mean Absolute Error function, when using the mode to predict the training and test data.

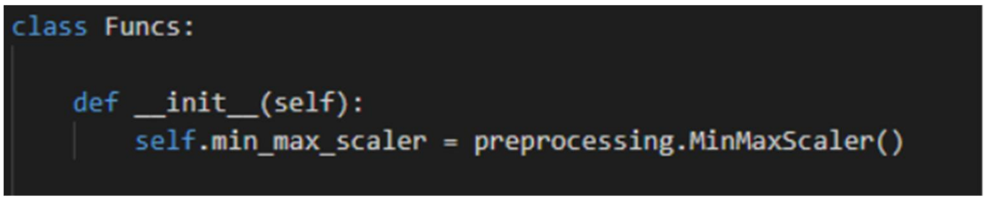
6.2 Code Snippets:



```
import json
import pandas
import numpy
import datetime
import requests
from sklearn import preprocessing
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, Activation
```

Fig 6.2.1 Import Libraries

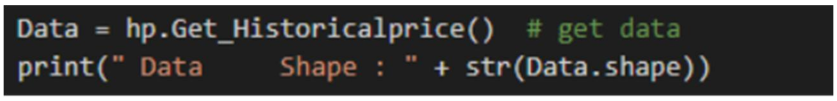
Import all the libraries used for this project, to install any new package – try the following command "pip install <package_name>" in command prompt/terminal.



```
class Funcs:

    def __init__(self):
        self.min_max_scaler = preprocessing.MinMaxScaler()
```

Fig 6.2.2 User defined function initialization

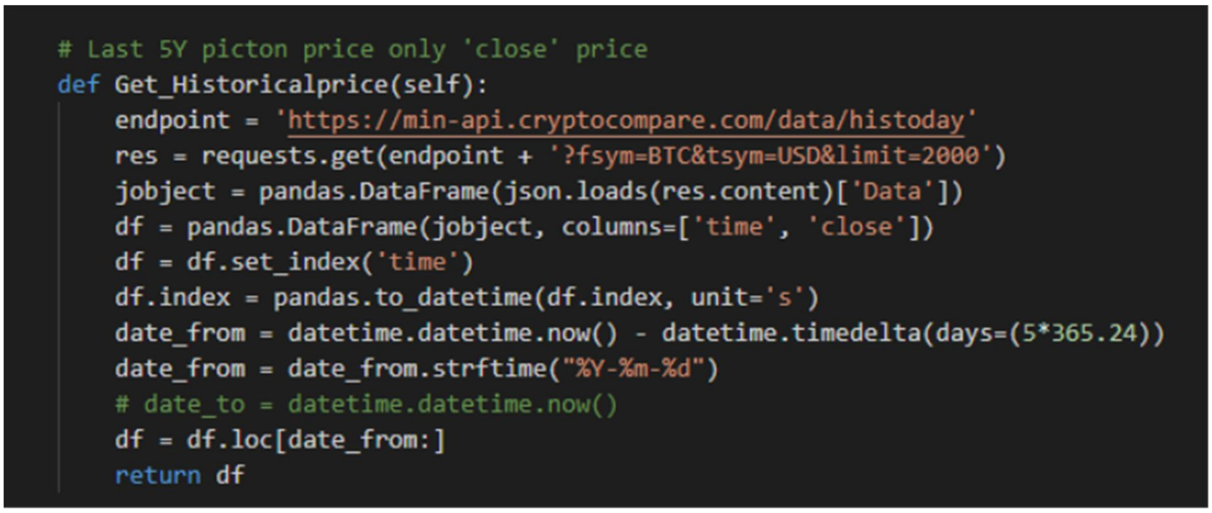
A user defined function named "Funcs" is created and all the user full methods are written in it.



```python
Data = hp.Get_Historicalprice()  # get data
print(" Data      Shape : " + str(Data.shape))
```

Fig 6.2.3 Retrive data from online database

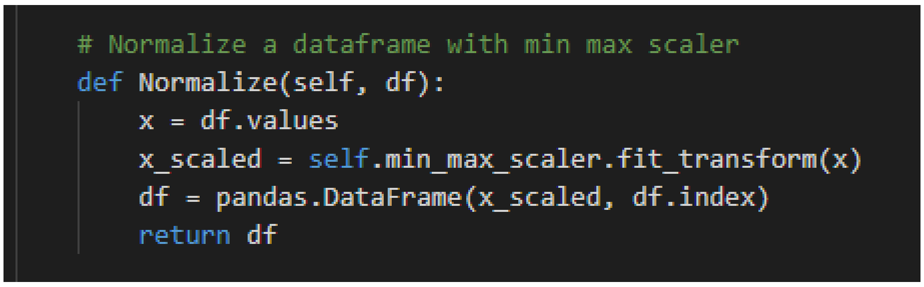
The Data is requested from the Cryptocompare API by calling the Get_Historicprice() and is stored into Data variable



```python
# Last 5Y picton price only 'close' price
def Get_Historicalprice(self):
    endpoint = 'https://min-api.cryptocompare.com/data/histoday'
    res = requests.get(endpoint + '?fsym=BTC&tsym=USD&limit=2000')
    jobject = pandas.DataFrame(json.loads(res.content)['Data'])
    df = pandas.DataFrame(jobject, columns=['time', 'close'])
    df = df.set_index('time')
    df.index = pandas.to_datetime(df.index, unit='s')
    date_from = datetime.datetime.now() - datetime.timedelta(days=(5*365.24))
    date_from = date_from.strftime("%Y-%m-%d")
    # date_to = datetime.datetime.now()
    df = df.loc[date_from:]
    return df
```

Fig 6.2.4 Request historic data through Cryptocompare API

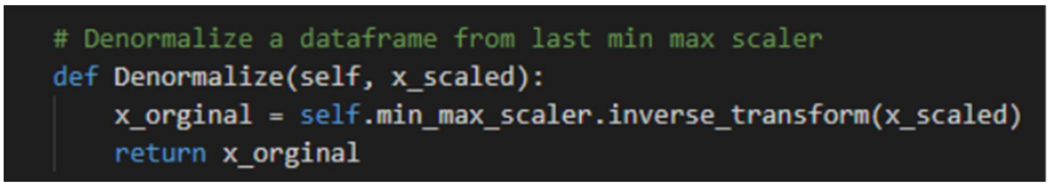
The above code snippet is used to connect with Cryptocompare API and request Bitcoin historic data in USD- denomination, for past 5 years from present date. The above method returns data in form of pandas data frame “df”.



```python
# Normalize a dataframe with min max scaler
def Normalize(self, df):
    x = df.values
    x_scaled = self.min_max_scaler.fit_transform(x)
    df = pandas.DataFrame(x_scaled, df.index)
    return df
```

Fig 6.2.5 Normalization of data

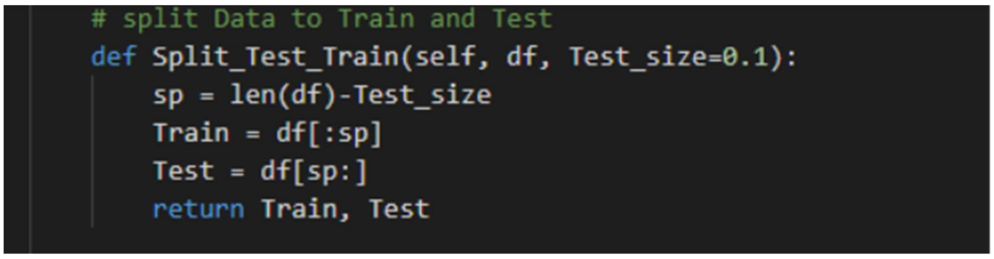
The above code snippet is used to normalize the data in range of 0 to 1, we use min max scalar normalization.



```python
# Denormalize a dataframe from last min max scaler
def Denormalize(self, x_scaled):
    x_orginal = self.min_max_scaler.inverse_transform(x_scaled)
    return x_orginal
```

Fig 6.2.6 De-normalization method

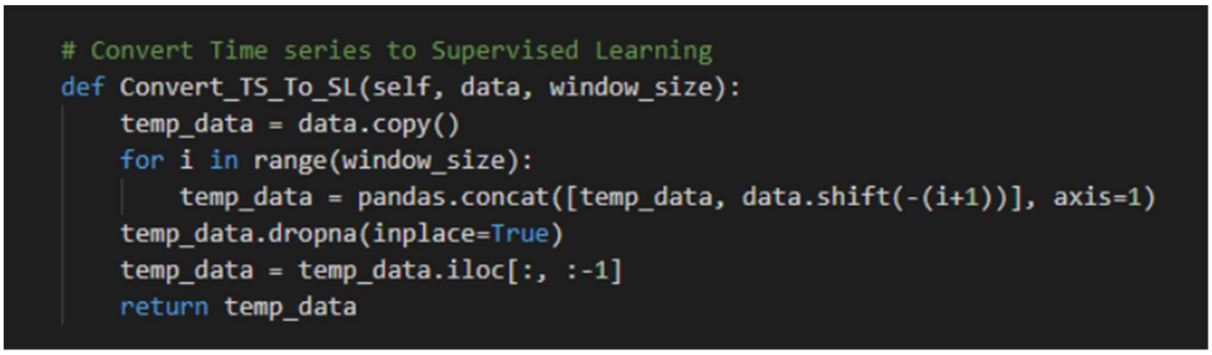
The above method is used to obtain original data values from Normalized data



```python
# split Data to Train and Test
def Split_Test_Train(self, df, Test_size=0.1):
    sp = len(df)-Test_size
    Train = df[:sp]
    Test = df[sp:]
    return Train, Test
```

Fig 6.2.7 Split Data to Test and Train

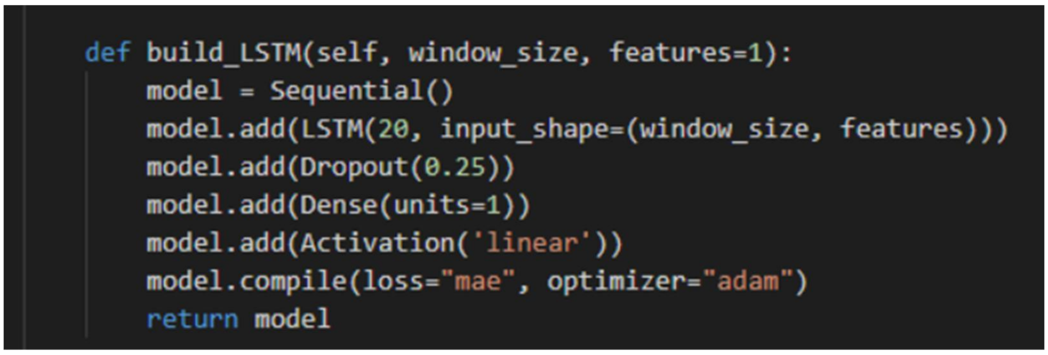
The method is used to Split the whole data into Train data set and rest to Test data set.



```python
# Convert Time series to Supervised Learning
def Convert_TS_To_SL(self, data, window_size):
    temp_data = data.copy()
    for i in range(window_size):
        temp_data = pandas.concat([temp_data, data.shift(-(i+1))], axis=1)
    temp_data.dropna(inplace=True)
    temp_data = temp_data.iloc[:, :-1]
    return temp_data
```

Fig 6.2.8 Time series to supervised learning

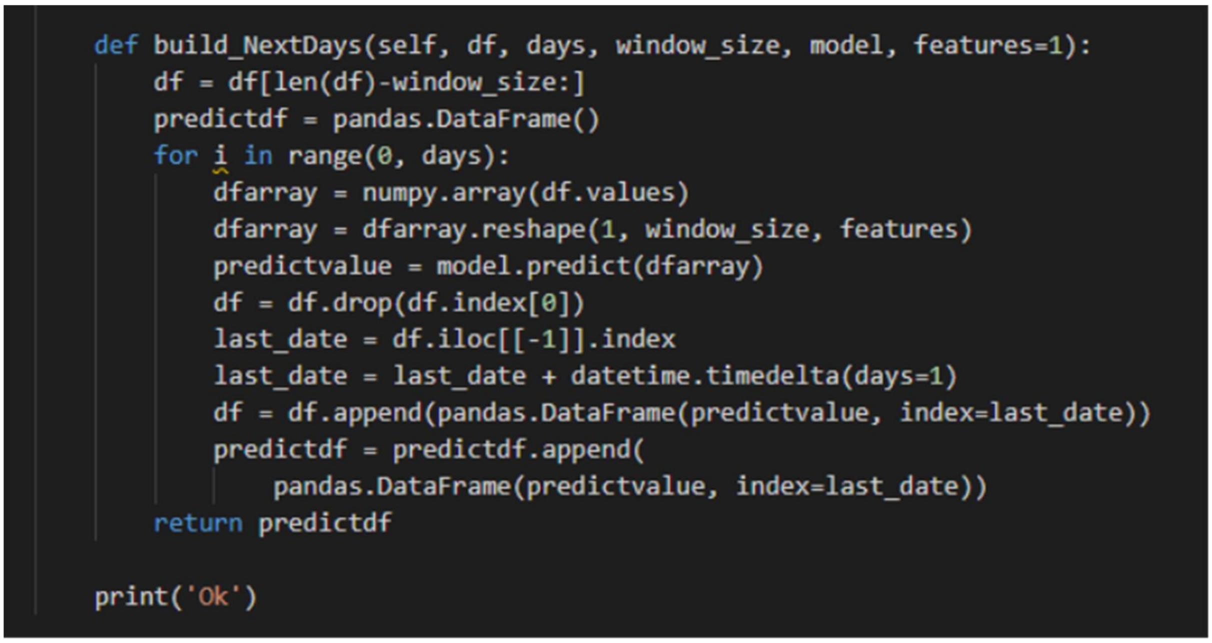
This is where the Time series data is converted into supervised learning suitable for Neural networks.



```python
def build_LSTM(self, window_size, features=1):
    model = Sequential()
    model.add(LSTM(20, input_shape=(window_size, features)))
    model.add(Dropout(0.25))
    model.add(Dense(units=1))
    model.add(Activation('linear'))
    model.compile(loss="mae", optimizer="adam")
    return model
```

Fig 6.2.9 Build LSTM model

The Long Shirt Term Model is built through above code, the LSTM takes window size and features as arguments.



```python
def build_NextDays(self, df, days, window_size, model, features=1):
    df = df[len(df)-window_size:]
    predictdf = pandas.DataFrame()
    for i in range(0, days):
        dfarray = numpy.array(df.values)
        dfarray = dfarray.reshape(1, window_size, features)
        predictvalue = model.predict(dfarray)
        df = df.drop(df.index[0])
        last_date = df.iloc[[-1]].index
        last_date = last_date + datetime.timedelta(days=1)
        df = df.append(pandas.DataFrame(predictvalue, index=last_date))
        predictdf = predictdf.append(
            pandas.DataFrame(predictvalue, index=last_date))
    return predictdf

print('Ok')
```

Fig 6.2.10 Building prediction data

The prediction is done and the future data is created with above code and it uses days to predict, LSTM model, features, window size and data frame as input arguments and outputs prediction data frame.

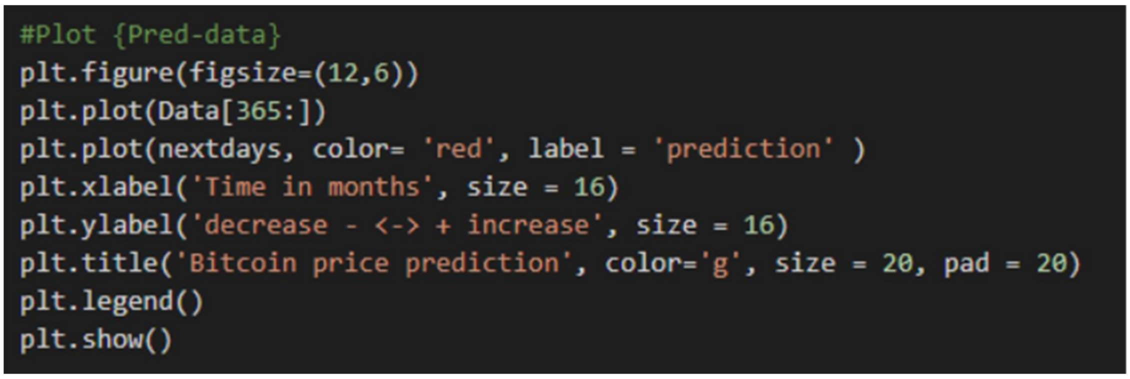

```
#Plot {Pred-data}
plt.figure(figsize=(12,6))
plt.plot(Data[365:])
plt.plot(nextdays, color= 'red', label = 'prediction' )
plt.xlabel('Time in months', size = 16)
plt.ylabel('decrease - <-> + increase', size = 16)
plt.title('Bitcoin price prediction', color='g', size = 20, pad = 20)
plt.legend()
plt.show()
```

Fig 6.2.11 Plot Prediction Graph

The above code snippet is used to plot the prediction graph which is final outcome of this project.

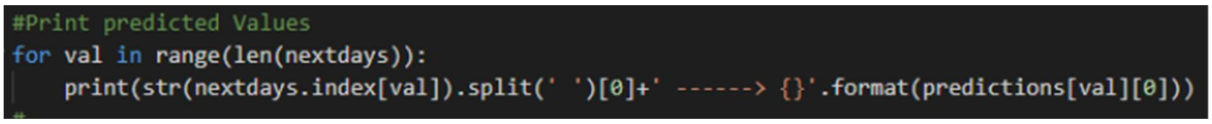

```
#Print predicted Values
for val in range(len(nextdays)):
    print(str(nextdays.index[val]).split(' ')[0]+' ------> {}'.format(predictions[val][0]))
```

Fig 6.2.12 Print Prediction Values

The prediction valued along with Timestamp as index are outputted through above piece of code.

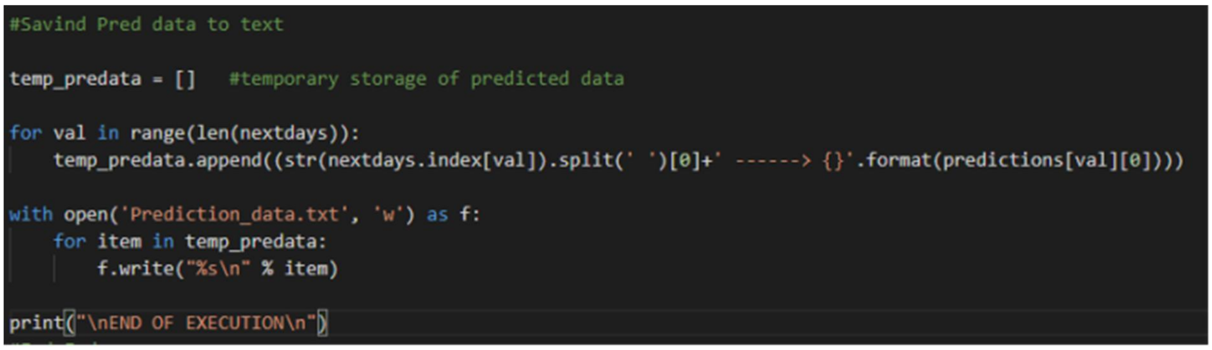
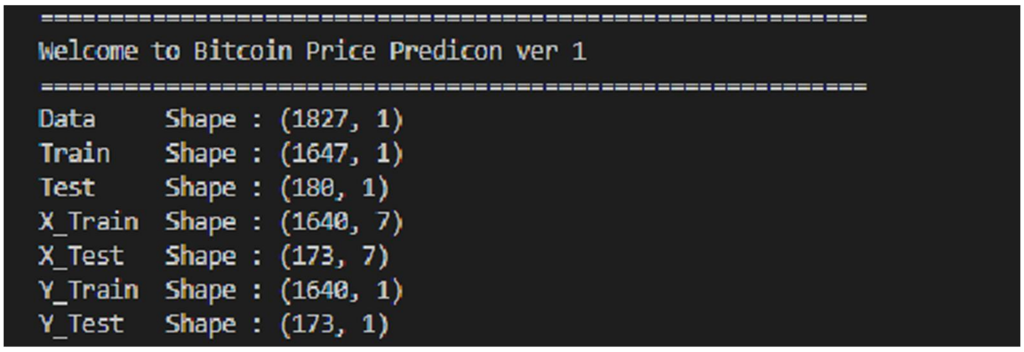

```
#Savind Pred data to text

temp_predata = []   #temporary storage of predicted data

for val in range(len(nextdays)):
    temp_predata.append((str(nextdays.index[val]).split(' ')[0]+' ------> {}'.format(predictions[val][0])))

with open('Prediction_data.txt', 'w') as f:
    for item in temp_predata:
        f.write("%s\n" % item)

print("\nEND OF EXECUTION\n")
```

Fig 6.2.13 Store prediction values to .txt file

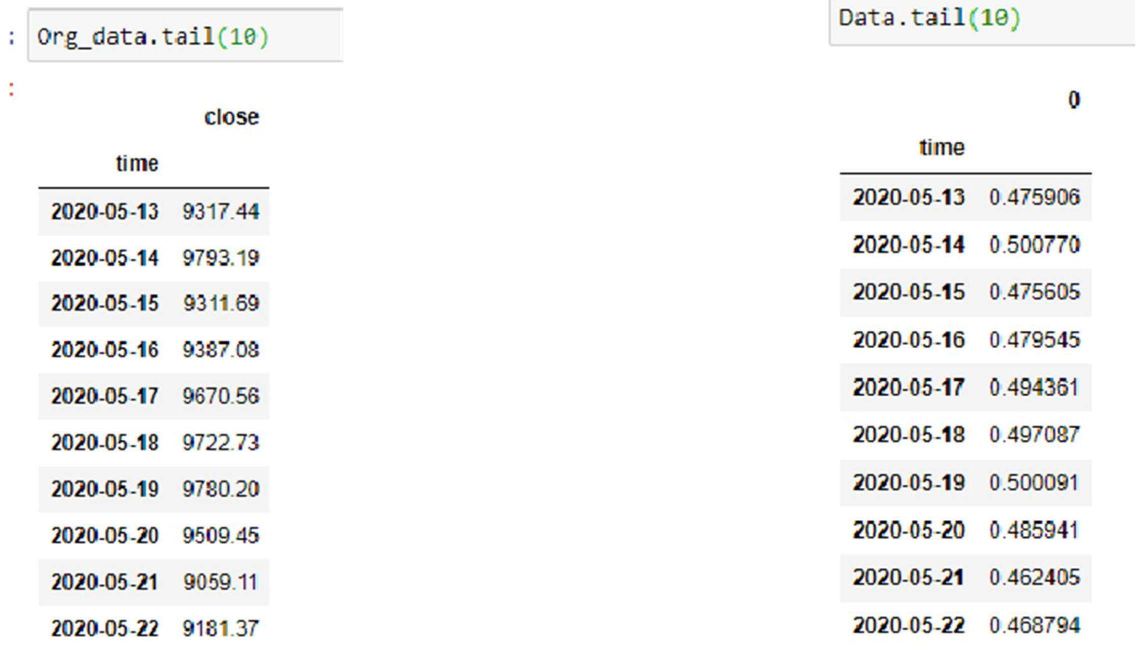
The predicted data can be saved automatically into a text file using above peace of code.

6.3 Output:



```
============================================================
Welcome to Bitcoin Price Predicon ver 1
============================================================
Data     Shape : (1827, 1)
Train    Shape : (1647, 1)
Test     Shape : (180, 1)
X_Train  Shape : (1640, 7)
X_Test   Shape : (173, 7)
Y_Train  Shape : (1640, 1)
Y_Test   Shape : (173, 1)
```

Screen 6.3.1 Start of program execution & splitting Test and Train datasets



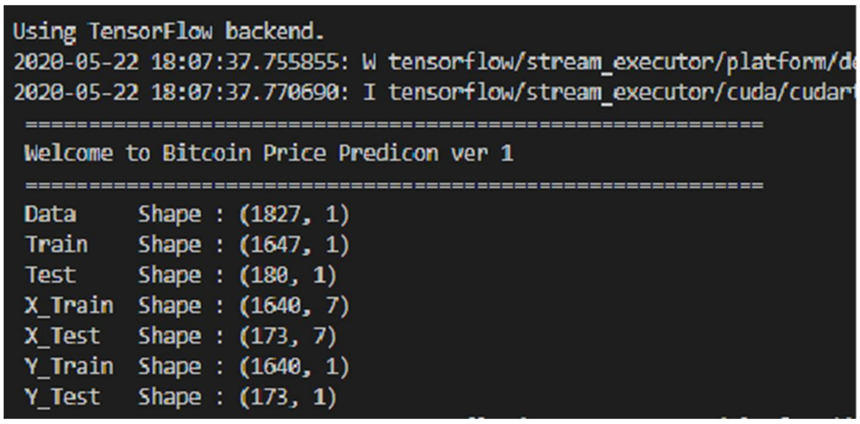
```
: Org_data.tail(10)
```

time	close
2020-05-13	9317.44
2020-05-14	9793.19
2020-05-15	9311.69
2020-05-16	9387.08
2020-05-17	9670.56
2020-05-18	9722.73
2020-05-19	9780.20
2020-05-20	9509.45
2020-05-21	9059.11
2020-05-22	9181.37

```
Data.tail(10)
```

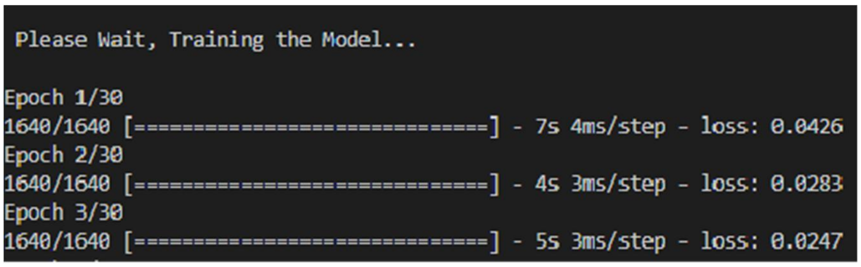
time	0
2020-05-13	0.475906
2020-05-14	0.500770
2020-05-15	0.475605
2020-05-16	0.479545
2020-05-17	0.494361
2020-05-18	0.497087
2020-05-19	0.500091
2020-05-20	0.485941
2020-05-21	0.462405
2020-05-22	0.468794

Screen 6.3.2 The data before normalization and after normalization using min-max scalar method



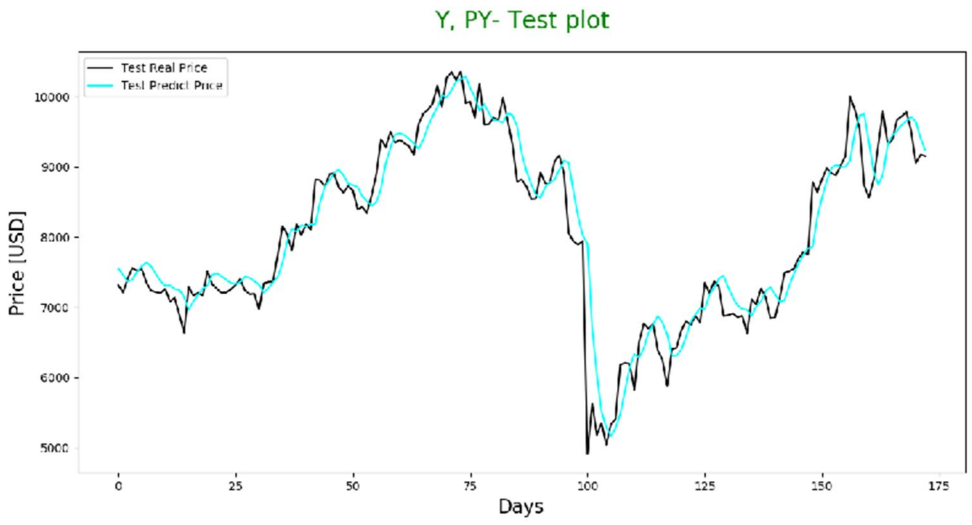
```
Using TensorFlow backend.
2020-05-22 18:07:37.755855: W tensorflow/stream_executor/platform/d
2020-05-22 18:07:37.770690: I tensorflow/stream_executor/cuda/cudar
 ==========================================================
 Welcome to Bitcoin Price Predicon ver 1
 ==========================================================
 Data     Shape : (1827, 1)
 Train    Shape : (1647, 1)
 Test     Shape : (180, 1)
 X_Train  Shape : (1640, 7)
 X_Test   Shape : (173, 7)
 Y_Train  Shape : (1640, 1)
 Y_Test   Shape : (173, 1)
```

Screen 6.2.3 The data is obtained from Cryptocompare API and Normalized, later the Normalized data is split into Test set and Train set as shown

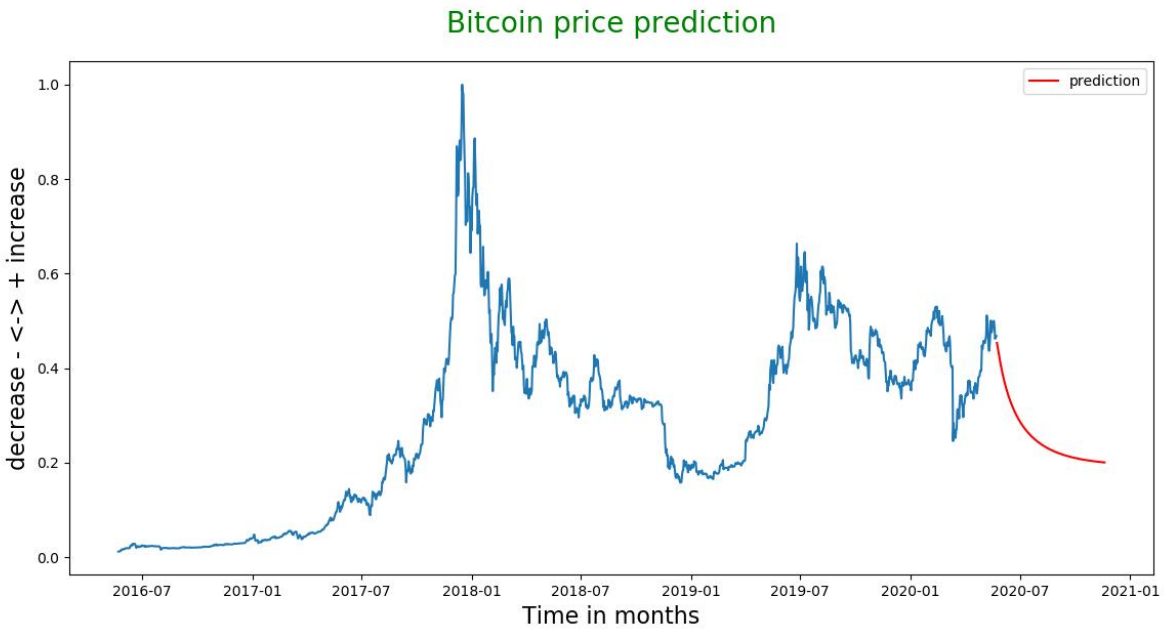


```
 Please Wait, Training the Model...

Epoch 1/30
1640/1640 [==============================] - 7s 4ms/step - loss: 0.0426
Epoch 2/30
1640/1640 [==============================] - 4s 3ms/step - loss: 0.0283
Epoch 3/30
1640/1640 [==============================] - 5s 3ms/step - loss: 0.0247
```

Screen 6.2.4 The data frame is then passed into the Neural network to Train the Model. Thus a model is built after this step and the same is used for prediction



Screen 6.4.5 The above screen snip is a plot of Test predict prices – Test Real prices – represented with black and cyan colours respectively.



Screen 6.4.6 the plotting of past 5 year historic prices of bitcoin in blue and the red plot shows the prediction future prediction which is the ultimately the outcome of this project, the predicted data can also be stored in a text file automatically.

Note : The Above Outputs are based on Model Trained on 22 May, 2020.

CHAPTER 7

CONCLUSION

7.1 Project Conclusion:

All in all, predicting a price-related variable is difficult given the multitude of forces impacting the market. Add to that, the fact that prices are by a large extent depended on future prospect rather than historic data. However, using deep neural networks, has provided us with a better understanding of Bitcoin, and LSTM architecture. The work in progress, includes implementing hyperparameter tuning, in order to get a more ac curate network architecture. Also, other features can be considered (although from our experiments with Bitcoin, more features have not always led to better results). Microeconomic factors might be included in the model for a better predictive result. Anyway, maybe the data we gathered for Bitcoin, even though has been collected through years, might have become interesting, producing historic interpretation only in the last couple of years. Furthermore, a breakthrough evolution in peer-to-peer transactions is ongoing and transforming the landscape of payment services. While it seems, all doubts have not been settled, time might be perfect to act. We think it's difficult to give a mature thought on Bitcoin for the future.

7.2 Future Projections:

Current work focuses on one-day ahead and partly on three-day ahead prediction of Bitcoin price. However, we can follow a similar procedure to perform multi-step ahead prediction of Bitcoin price thereby increasing the scope of experiments. In addition to cryptocurrency Bitcoin, there are other cryptocurrencies such as Ethereum, Litecoin, Ripple, etc. that gained attention in the trading markets. Analysis on these cryptocurrencies can help investors decide which one to buy or sell so as to end up with profitable trades. Also, I have plans on using the GPU based accelerator that can significantly reduce the training time of the models. Deep learning techniques such as Long Short-Term Memory (LSTM) networks (Hochreiter & 34 Schmidhuber, 1997) can also boost the predictions, thereby helping to make better trading decisions. The scope of this project can be extended to use some deep learning techniques.

REFERENCES

1. Hochreiter, S. & Schmidhuber, J. (1997). *Long Short-Term Memory.* Institute of Bioinformatics, Johannes Kepler University Linz.

2. Learning to predict cryptocurrency price using artificial neural network models of by Sneha Gullapalli, B.Tech., Jawaharlal Nehru Technological University,Hyderabad India, 2016.

3. Project Based Learning: Predicting Bitcoin Prices using Deep Learning, Maninder Jeet Kaur Department of Engineering Amity University Dubai Dubai, UAE mkaur@amityuniversity.ae Piyush Maheshwari Department of Engineering Amity University Dubai Dubai, UAE pmaheshwari@amityuniversity.ae

4. A Gated Recurrent Unit Approach to Bitcoin Price Prediction, 2019, Aniruddha Dutta 1, Saket Kumar 1 and Meheli Basu 2 1 Haas School of Business, University of California Berkeley, CA 94720, USA 2 Joseph M. Katz Graduate School of Business, University of Pittsburgh, PA 15260, USA. Correspondence: aniruddha_dutta@berkeley.edu

5. Bitcoin Price Prediction using Machine Learning Siddhi Velankar, Sakshi Valecha, Shreya Maji, Department of Electronics & Telecommunication, Pune Institute of Computer Technology, Pune, Maharashtra, India. For International Conference on Advanced Communications Technology (ICACT) February 2018 velankar.siddhi@gmail.com,sakshivalecha.96@gmail.com, shreyamaji50@gmail.com

6. Prediction of Crypto Currency Returns using Machine Learning Erdinc Akyildirim, Ahmet Goncu, Ahmet Sensoy November, 2018. University of Zurich, Departme nt of Banking and Finance, Zurich, Switzerland.

7. Bitcoin Price Prediction with Neural Networks, 2018, Paper 06, Olti Qirici & Kejsi Struga kejsi.struga@fshnstudent.info olti.qirici@fshn.edu.al University of Tirana, Tirana.

8. [Nakamoto2008] Satoshi Nakamoto. Bitcoin: A Peerto-Peer Electronic Cash System

9. [BitcoinWiki2017] Bitcoin Wiki. Deep Learning with Python. https://en.bitcoin.it/wiki/ Controlled_supply

10. [Chollet2017] Deep Learning with Python. https://www.manning.com/books/ deep-learning-with-python

11. [Batnick2018] M. Batnick. The market cap of the top 5 S&P 500 companies. https://theirrelevantinvestor.com/ 2018/07/19/pareto/

12. [Geron2017] Bitcoin Wiki. Hands on Machine Learning with scikit-learn and tensorflow https://www.oreilly.com/library/ view/hands-on-machine-learning/ 9781491962282/

13. [Olah2015] Understanding LSTMs https://colah.github.io/posts/ 2015-08-Understanding-LSTMs/

14. [Kasp ersky2017] , K1 Kaspersky Lab Daily, Six Myths about blockchain and Bitcoin: De bunking the effectiveness of the technology https://www.kaspersky.com/blog/ bitcoin-blockchain-issues/18019/

15. [Keras2015] Keras https://keras.io/ getting-started

16. Molina, D., Liang, J., Harley, R., & Venayagamoorthy, G. K. (2011). "Comparison of TDNN and RNN Performances for Neuro-Identification on Small to Medium-Sized Power Systems". IEEE Symp. on Comp. Intelligence in Smart Grids, Proc. of the 2011.

17. Scheier, C. & Tschacher, W. (1996). Appropriate algorithms for non-linear time series analysis in psychology.

18. Shumway, R. H. & Stoffer, D. S. (2017). Time Series Analysis and It's Applications (3 ed.).Springer International Publishing AG 2017.

19. Thomas, A. (2013). API for Bitcoin Data. Retrieved March 19, 2018, from Quandl Blog: https://blog.quandl.com/api-for-bitcoin-data

20. Ugiliweneza, B. (2007). User of ARIMA time series and Regressors to forecast the sale of electricity. SESUG Proceedings.