CRYPTOCURRENCY PRICE PREDICTION USING TRANSFORMER: A DEEP LEARNING ARCHITECTURE

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTER OF TECHNOLOGY

IN

INFORMATION SYSTEMS

Submitted By:

DEEPAKSHI JAIN

(2K17/ISY/06)

Under the supervision of
Prof. Kapil Sharma

DEPARTMENT OF INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

JULY, 2019

CANDIDATE'S DECLARATION

I, Deepakshi Jain, Roll No. 2K17/ISY/06 student of M.Tech Information Systems, hereby declare that the project Dissertation titled "Cryptocurrency Price Prediction Using Transformer: A Deep Learning Architecture" which is submitted by me to the Department of Information Technology, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi Deepakshi Jain

Date:

CERTIFICATE

I hereby certify that the Project Dissertation titled “Cryptocurrency Price Prediction Using Transformer: A Deep Learning Architecture” which is submitted by Deepakshi Jain, Roll No 2K17/ISY/06 Information Technology, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi **Prof. Kapil Sharma**

Date: **SUPERVISOR**

ACKNOWLEDGEMENT

I express my gratitude to my major project guide Prof. Kapil Sharma, Professor, lT Dept., Delhi Technological University, for the valuable support and guidance she provided in making this major project. It is my pleasure to record my sincere thanks to my respected guide for his constructive criticism and insight without which the project would not have shaped as it has.

I humbly extend my words of gratitude to other faculty members of this department for providing their valuable help and time whenever it was required.

Deepakshi Jain

Roll No. 2K17/ISY/06

M.Tech (Information Systems)

E-mail: j.deepakshi55@gmail.com

ABSTRACT

In recent years, price prediction for cryptocurrency has gained so much attention as there are a lot of investors, spectators and consumers in the market and are interested to know where to spend their money for profitable trades. Cryptocurrency is attracting customer in reshaping the finance structure because of its fame and acceptance. Bitcoin is the most popular and transparent cryptocurrency over the internet. Due to the rapid fluctuation in bitcoin prices, uncertainty, dynamic features, etc. there comes a need to predict its price as it varies rapidly in short interval of time. Many deep learning model has been implemented for price prediction but attention mechanism is proved to be more efficient and has sought focus in recent years. The aim of our work is to develop a trained and efficient machine model which can predict bitcoin price if we input huge amount of data with a good computational time and power. In our study, we proposed transformer architecture which implements attention layer. Our dataset contains features which are related to bitcoin price and has data of over ten years recorded daily. Features in the dataset are dependent on each other which can affect the price level for the upcoming time. The architecture is used with two types of deep learning models which are LSTM and GRU. Both these model are used as LSTM-with-attention and GRU-with-attention. Finally, we compared both these models on the basis of four error metrics which are mean absolute error (MAE), mean square error (RMSE), maximum error (ME), and root mean squared error (RMSE). After comparing results, it has been observed that the transformer model GRU-with-attention gives less error rate i.e. better estimation than LSTM-with-attention. The results obtained might have inferences on implementation of more complex time series problems with deep neural networks.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Symbols, Abbreviations and Nomenclature

ReLu - Rectified Linear Unit	

ReLu - Rectified Linear Unit

RNN – Recurrent Neural Network

FNN – Feed Forward Network

LSTM – Long Short Term Memory

GRU – Gated Recurrent Unit

MAE – Mean Absolute Error

MSE – Mean Squared Error

RMSE – Root Mean Squares Error

ME – Mean Error

SVM – Support Vector Machine

PCA – Principal Component Analysis

LSM -- Latent Source Model

ARIMA -- Auto-Regressive Integrated Moving Average Model

NARX – Non-Linear Auto-Regressive with Exogenous Input

ANN – Artificial Neural Network

RBM -- Restricted Boltzmann Machine

DBN – Deep Belief Network

μ – Mean

σ – Variance

Σ – Summation

CHAPTER 1 INTRODUCTION

1.1 BLOCKCHAIN AND CRYPTOCURRENCY

Blockchain is an immutable, transparent, public ledger that is distributed among the nodes in the network. It is a decentralized system in which transactions run on untrusted devices. It is like a chain of nodes which are connected through hash values. If any of the block in the chain is hacked or interrupted, the whole chain after that block becomes invalid and is informed to the whole network of the peers. Cryptocurrency is a blockchain platform which is critically used for confirmation of the transactions. Various consensus algorithms are used for making transactions valid among the peers in the network.

1.1.1 Blockchain Architecture

Blockchain is an immutable and distributed system. It keeps a record of all transactions and this record is referred to as 'ledger'. It is a common ledger of data exchanges or transactions which is shared among all the participants in the network. Blocks in these systems are made up of transactions. A block can refer to its previous block using a hash value [2]. The transactions among the peers are verified by the participating nodes in the network. Once verified and signed, the block is added to the chain and it cannot be altered. Blockchains ensure reliable transfer of data in a decentralized way with third party involvement. [5].

The blocks are arranged in a chronological order. The first block in the chain is referred to as genesis block and does not has a parent block [1]. Basically, a block consist of the block header, hash value of its previous and current block, transaction counter, timestamp and merkle root as shown in figure 1. Hash is a one-way function that takes input of any size and the resultant output is of fixed length [4]. A transaction is started by a node and is digitally signed by its private key. A block is created which represents these transactions. This block is now shared with every node in the network. A group of nodes is assigned to validate the block and in return, they receive the reward for the Proof of Work. This validated block is added to the existing blockchain. All the processes are done based on a set of rules which are known as the consensus protocol. Depending on the type of blockchain implementation (permissioned or permissionless), the consensus is applied.

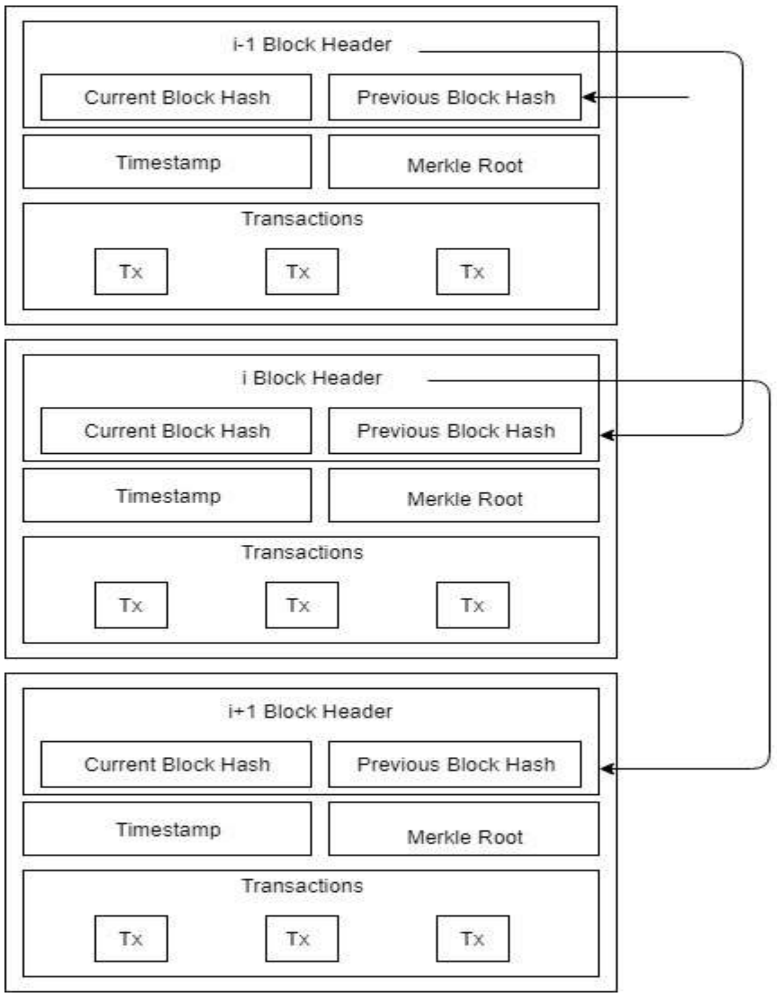


Figure 1. 1 Blockchain Framework

Broadly, blockchain are classified into three categories. They are public, consortium and fully private blockchain [3]. A public blockchain is permissionless blockchain whereas consortium and fully private blockchain come under the category of permissioned blockchain. In a public blockchain system, no centralized party is there. Everyone in public blockchain possesses equal power and has the right to validate the transaction. Anyone can join or leave the system according to their wish as these blockchains are open for all. The best and well-known example of a public blockchain is Bitcoin. In consortium blockchain, not everyone holds the power to validate transactions. Only a few people have the privilege to validate. The rest of the people can validate but they have to agree on consensus first before implementation. The fully private blockchain is a little bit different from them. They have a centralized structure. The system runs based on the consensus proposed by the centralized head. The head possesses the power to make all decisions and controls the whole validation process. Compared to permissionless blockchains, permissioned blockchains are faster, easy to implement and more energy efficient.

1.1.2 Blockchain Applications

The decentralized and verifiable nature of blockchain results in many applications. Blockchains are implemented in various industrial, business and research areas. Major applications of blockchain include Internet of Things (IoT), Big Data, Identity Management, Smart Contracts, Identity Management, Supply Chains, Medical Informatics, Cloud and Edge Computing, Communication and Networking and many others as shown in the figure 2.

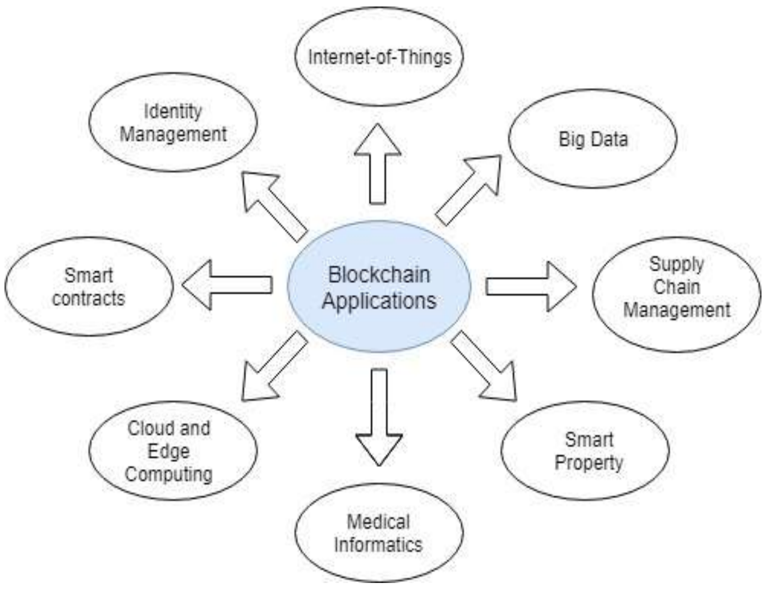


Figure 1. 2 Blockchain Applications

1.1.3 Blockchain Consensus Algorithms

Consensus algorithms are the root of blockchain technology. As blockchain systems are decentralized that runs on untrusted devices, algorithms are required to reach on an agreement among the nodes to append a new block. Consensus algorithms are like the decision-making process for a group of individuals to construct and support decisions that work best for all in the chain. Consensus models are the methods to provide fairness and equality in the network. There are so many algorithms like Proof-of-Work (PoW), Proof-of-Stake (PoS), Proof-of-Burn (PoB), Proof-of-Capacity (PoC), Practical Byzantine fault Tolerance Protocol (PBFT), Proof-of-Elasped Time (PoET), Proof-of-Weight (Pow) and many more. Out of these, we have discussed three major consensus algorithms that are being used in blockchain transactions and compared them.

A. Proof-of-Work (PoW)

PoW is a widely used mechanism in blockchain systems. PoW consensus algorithm is used by many blockchain systems to validate all their transactions and to introduce relevant blocks to the chain. Participating miners have to solve a mathematical puzzle. The miner who wins the race is rewarded. The peer who creates a block successfully receives a reward as coins for their work [10]. A lot of computational power is required to solve this puzzle. An example is: calculating a hash function or reaching an output without knowing input. This algorithm depends on power, number of users and overall load on the network. New blocks are added to the chain like a node is added to the linked list. Instead of an address, blocks contain the hash functions of the previous block. The transaction is confirmed when the miner solves the puzzle and the block is added to the chain. PoW is used in bitcoin technology. In Bitcoin, the node who first solves the mathematical puzzle wins the race. After winning, the node broadcasts this block and the rest of the nodes vote to accept the block [8]. PoW is more likely prone to 51% attack. The attacker can alter or reshuffle the order of the transactions if he takes control of more than half of the computational power of the system [9]. Greater energy consumption and the centralization of miners are the main limitations of this algorithm.

B. Proof-of-Stake (PoS)

As there was a greater energy consumption issue in PoW, PoS is introduced to overcome this drawback. Basically, the nodes having a high stake will be getting a chance to add blocks to the network more often as compared to other nodes having less stake. Random selection based on the amount of stake of miners is done to elect a new block. Because of this, the power consumption in mining is reduced as compared to PoW. Two well-known PoS based consensus protocols are Oroboros and Casper [6]. PIVX, NavCoin, Startis are three most popular cryptocurrencies using Proof-of-Stake as a base in blockchain technology. This algorithm does not need any heavy amount of hardware for its backup, the threat of 51% attack is reduced, and less power consumption is required. As the system is fully decentralized, nodes having larger coins can take control of the network. This also makes double spending attacks easier for the attacker [7].

C. Practical Byzantine Fault Tolerance (PBFT)

This algorithm has been designed to solve issues of Byzantine Fault Tolerance solutions and to work effectively in asynchronous systems. PBFT can work even if there are malicious nodes in the system. Nodes are sequentially ordered. One node is the leader node or the primary node and others are backup nodes or the secondary nodes. In the case of a primary node failure, any eligible secondary node can be assigned as the primary node. PBFT works on the principle that a number of malicious nodes should not be greater than or equal to one-third of the total nodes in the system. This consensus is done in four phases. The leader node is replaced in every phase. If required, honest nodes can vote and replace the current leader node by the next node in the line. Transactions in PBFT do not require multiple confirmations. This algorithm is energy efficient and every node in the network leads to low reward variance which is good for decision making. The disadvantage of PBFT is that it is prone to Sybil attacks and Scaling. This algorithm does not scale well because of the communication gap.

We have compared these three consensus on the basis of their type, energy consumption, power tolerance and blockchain platform as shown in table 1.1.

Property	**PoW**	**PoS**	**PBFT**
Type of blockchain	Permissionless	Permissionless	Permissioned
Energy Consumption	Higher	Partial	Lower
Power Tolerance	Less than 25% computing power	Less than 51% stake	Less than 33% Defective replicas
Blockchain Platform	Bitcoin	Ethereum	Hyperledger Fabric

Table 1. 1 Comparison between PoW, PoS and PBFT

1.2 CRYPTOCURRENCY – A BLOCKCHAIN PLATFORM

Cryptocurrency is a digital currency which is introduced to work as a mode of exchange of assets between various users in the network. This digital currency is critically used for confirmation of the transactions. Strong cryptography algorithms are used to make secure transactions and to control additional creation of nodes in the network. They are fully decentralised that is there is no role of any third party involvement in the whole process. This decentralised control works with the help of distributed ledger. Major advantages of cryptocurrency include security, permissionless, faster, global, pseudonymous etc. Some of the well-known cryptocurrency are “Bitcoin” and “Ethereum”. Bitcoin was first released in 2009 as an open source software. It is basically a decentralised cryptocurrency. This makes it different from standard financial models. After seven years of its foundation, its value raised from 0 to 650 dollars and achieved a daily transaction volume of around 200 transactions per day. Some well-known cryptocurrency are shown in the figure below.

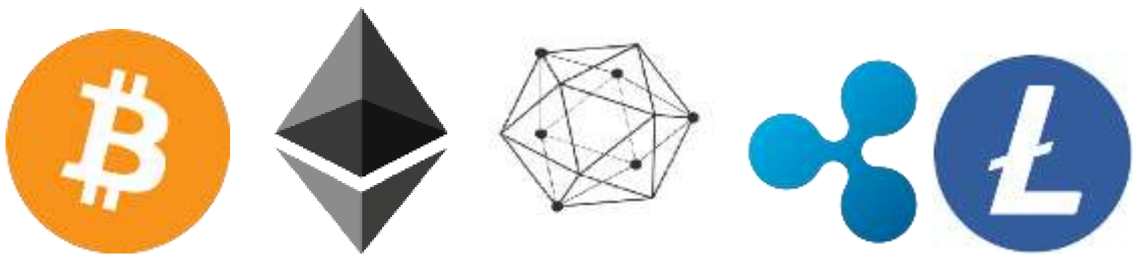


Figure 1. 3 Bitcoin, Ethereum, Hyperledger Fabric, Ripple, Litecoin

1.3 PREDICTION – MACHINE LEARNING

Prediction may be defined as the estimation making of values. Prediction is the outcome of an algorithm after the algorithm is trained with some kind of historical dataset and then employed on new data. Prediction can be made for many outputs like for wheather forecasting, for estimating stock price, and many more. Along with data, predictive analysis uses statics, machine learning techniques to construct predictive models. Supervised learning is used for predictive models as they can compute future values or can make probability estimations. Some of the area of application for prediction are healthcare, finance, aerospace, and manufacturing etc. Predictive analysis can be done in the foloowing steps as shown in figure 1.4.

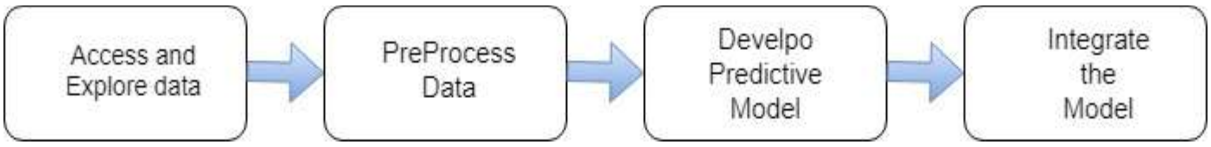


Figure 1. 4 Predictive Analysis Workflow

In machine learning, every algorithm is based on either supervised or unsupervised learning. They both are the base for deciding which type of model we want to build. Both can be brieflt described as follows:

1.3.1 Supervised Learning

In supervised learning, we have input variables as ‘x’ and an output variable as ‘Y’ and an algorithm to map function from input to output. It can be shown in equation below. As the name suggests, the machine is supervised with training data which is labelled. Machine is provided with new data and supervised learning algorithms gives output according to labelled data.

$$Y = f(x) \tag{1.1}$$

The aim is to produce correct output ‘Y’ on providing set of new data ‘x’. Classification and Regression uses supervised learning. Some of the supervised learning techniques are Support Vector Machine, Linear and Logistic Regression, Decision Trees, and more.

1.3.2 Unsupervised Learning

In unsupervised learning, we have input data ‘x’ only and no respective output data. The machine is trained without any classified or labelled data. The machine itself learns to group the information with the help of similarities, patterns or differences in the given data. Unlike supervised learning, there is no supervisor to provide correct labels to the input data and algorithm tries to find hidden structures in the data. Clustering and Association are the examples of unsupervised learning. Some of the unsupervised learning approaches involve K-means clustering, Neural Networks, Principle Component Analysis (PCA), Hierarchal Clustering, etc.

1.4 RECURRENT NEURAL NETWORK (RNN)

In Recurrent Neural Network, the output of previous stage act as input for the next state. Basically in a deep neural network, inputs are independent of each other. But it might be possible that relationship between inputs may provide better results. For example, if we want to predict next word in the sentence, it is necessary to have the knowledge of the word which comes before it. Here comes the need of connection between the inputs themselves. RNN consist of three types of layers which are input, hidden and output layer. A basic RNN with only one hidden layer is presented in figure 1.5. The hidden layers in the network may be of any number. Hidden layer in the network plays the most important role because they have memory to remember information about the sequence.

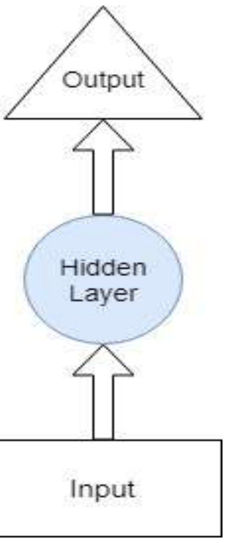


Figure 1. 5 A basic Recurrent Neural Network

1.4.1 LSTM

Long Short-Term Memory networks are the type of RNN which are used to predict sequence prediction problems. They have capacity to learn order dependencies and because of this behaviour they are used in the field of speech recognition, machine translation, handwriting recognition, and many more. LSTM uses feedback connections unlike standard feed-forward neural network. A single unit of LSTM is made up of a cell, an input gate, an output gate and a forget gate as presented in figure 1.6.

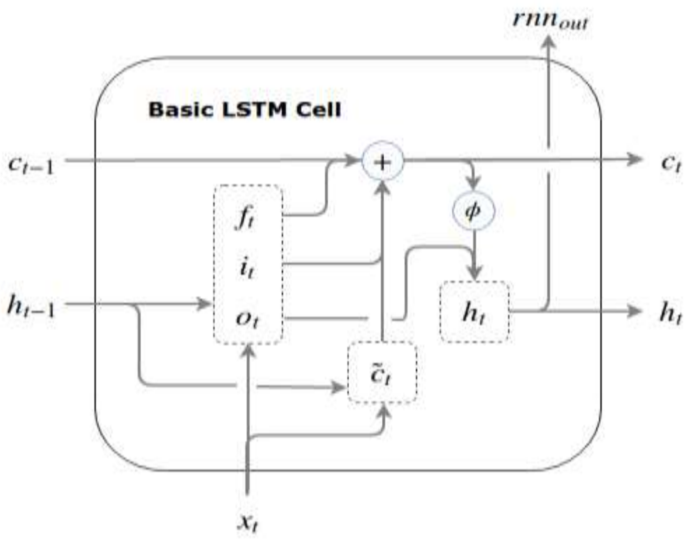


Figure 1. 6 Long Short-Term Memory [24]

The work of the cell is to keep magnitude over the time interval and these three gates are used to process the information flow in and out of the cell. The responsibility of the cell is to keep track of the dependencies between input elements. Input gate keeps controlling flow of new values to the cell, forget gate controls the limit that the value should remain in the cell and the output gate ensures the limit to which the value is used to process the activation of the whole LSTM unit. The logistic function is employed for calculating activation function. Connections are there in and out of the LSTM gates and some of them are recurrent. During in phase, the weights of these connections are required to be learned that examines operation of these gates. LSTM models works well for classification, making predictions on the basis of time series data. LSTMs are programmed to solve exploding and vanishing gradient problems which comes while training standard RNNs. One of the advantage of LSTM network is relative insensitivity to gap the length. Mathematic implementation can be shown as below. Output hidden state (h_t) in LSTM is calculated as:

$$i_t = \sigma (x_tU^i + h_{t-1}W^i) \tag{1.2}$$

$$f_t = \sigma (x_tU^f + h_{t-1}W^f) \tag{1.3}$$

$$o_t = \sigma (x_tU^o + h_{t-1}W^o) \tag{1.4}$$

$$C_t' = \tanh (x_tU^g + h_{t-1}W^g) \tag{1.5}$$

$$C_t = \sigma (f_t * C_{t-1} + i_t * C_t') \tag{1.6}$$

$$h_t = \tanh (C_t) * o_t \tag{1.7}$$

where ‘i’, ‘f’ and ‘o’ are input, forget and output gates respectively. ‘W’ is the recurrent connection for previous hidden layer. ‘U’ is weight matrix which connects input layer to hidden layer. C_t’ is candidate hidden state and ‘C’ is the internal memory of the unit.

1.4.1.1 Bidirectional LSTM

Bidirectional LSTM is the improvement of basic LSTM. It is the combination of Bidirectional RNNs (BRNNs) and LSTMS. Unlike standard LSTM, the idea behind bidirectional RNN is to process every training sequence in both forward and backward direction to two different recurrent nets. These both recurrent nets are connected to the same output layer which indicates that at each point in the sequence, the bidirectional recurrent neural network (BRNN) possesses complete and sequential information before and after it. There is no necessity to find a time window as the network is free to use as much as required. The disadvantage of basic RNNs is that they can use the information of previous context. BRNNs resolves this issue by processing the data in two directions i.e. forward and backward with two separate layers leading to same output layer. One of the hidden layer processes the input in forward direction while the other hidden layer processes the sequences in the reverse direction.

1.4.2 GRU

Gated Recurrent Units (GRUs) are the improvement on LSTM recurrent neural networks. Both these models consists of additional parameters to control memory updating process. They both are able to capture long and short term dependencies in the sequences but there are less parameters in GRU as compared to LSTM and hence training process is faster in GRUs. In GRU, there are two types of gates reset and forget gate. There is no output gate unlike LSTMs. These gates helps in assuring that memory is not taken over when short term dependencies are tracked. The network itself is able to learn how to use its gates to guard its memory so as to make long term predictions. Some of the applications of GRU which showed better performance than LSTM involves polyphonic music modelling, speech signal modelling, etc. Even on smaller datasets, they shows better results. A basic GRU cell can be shown in figure 1.7.

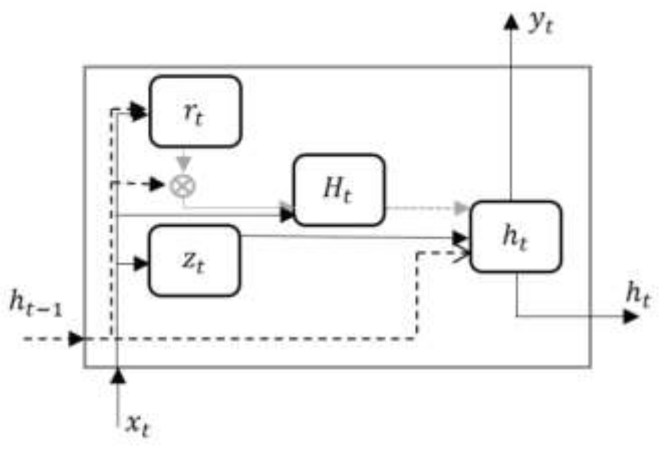


Figure 1. 7 Gated Recurrent Unit [31]

In GRU, the hidden state is calculated as:

$$z_t = \sigma (x_tU^z + h_{t-1}W^z) \quad (1.7)$$

$$r_t = \sigma (x_tU^r + h_{t-1}W^r) \quad (1.8)$$

$$H_t = \tanh (x_tU^h + (r_t * h_{t-1}) W^h) \quad (1.9)$$

$$h_t = (1 – z_t) * h_{t-1} + z_t * H_t \quad (1.10)$$

where ‘r’ is the reset gate and ‘z’ is an update gate. Rest of the symbols are same as in LSTM.

1.4.2.1 Bidirectional GRU

Bidirectional GRU are a type of bidirectional recurrent neural networks. Bidirectional GRU or BGRU consists of input and forget gates. It uses information from previous and later both time intervals for predictions about the current state. The model performs fairly well on classification tasks. Same as bidirectional LSTM, in bidirectional GRU, the idea is to process every training sequence in both forward and backward direction to two different recurrent nets. These both recurrent nets are connected to the same output layer. It is the combination of Bidirectional RNNs (BRNNs) and GRUs.

CHAPTER 2 RELATED WORK

In this section, we reviewed what work has been done before in the field of price prediction. We studied the type of techniques has been employed for the same and how much they are efficient.

Bitcoin is widely used and valuable cryptocurrency in the market. Because of high fluctuation in its price, prediction becomes difficult [18]. In this paper, the authors discovered accurate and efficient methods for predicting price using machine learning. Regression model is chosen because of the continuous values of prices. Four models are selected; Theil-Sen Regression Huber Regression, LSTM and GRU. In Theil-Sen model, slope median of lines with pairs of data points is used. Due to this, outliers grows to 29% for 2-D data. On increasing dimensions, outlier's robustness decreases [19]. Theil-Sen method took parameters – maximum step size, loss function and epsilon. Huber Regression uses linear loss for separating outlier and inlier data. Data is outlier if it has weight less than weight of the inlier. Parameters involved in this method are epsilon and alpha, which is a parameter for regularization. For LSTM and GRU, same parameters are taken as epochs, batch size, level, neurons, activation and kernel initializer. Metrics evaluated for the comparison of all four methods are Mean-Squared error and R-square (R^2). Results showed that out of all methods, GRU model gives best values of MSE at 0.0002 and R^2 at 99.2%, whereas computational time for predicting values is much less for Huber Regression as compared to LSTM and GRU.

Various other methods are also computed to achieve high accuracy for price prediction. The task is accomplished by implementing Bayesian optimized recurrent neural network and LSTM network [20]. Among both these methods, LSTM acquired better accuracy of 52% and RMSE of 8%. To compare these deep learning methods, the authors built an optimised ARIMA model as the model is used for price prediction problems [24] [25]. Now the authors compared accuracy and RMSE value for all these models and achieved highest accuracy and lowest RMSE with LSTM model. To evaluate model training, performance of RNN and LSTM are measured on the basis of CPU and GPU computational time. In the results, RNN model runs faster with less execution time on both processors. The authors also analysed bitcoin price by using Support Vector

Machine (SVM) and Artificial Neural Networks (ANN) and obtained an accuracy around 55% [21] with the standard ANN. However, implementation of SVM, Random Forest and Binomial Generalised Linear model (GLM) gives an accuracy of 97% with no cross-validation of the model limiting generalisability. [22] [23] wavelets has been also applied for bitcoin price prediction having positive correlation between views from search engines, hash rate of the network and mining complexity in bitcoin prices.

Some of the established methods for price prediction are Auto-Regressive Integrated Moving Average Model (ARIMA), Latent Source Model, Multi-Layer Perceptron Model and Non-Linear Auto-Regressive with Exogenous Input Model (NARX) [26]. ARIMA can predict time-series data even with less term time period. It is a parametric method for predicting time series individually. Prices are accurately predicted by using threshold for estimation [27]. Some of the disadvantages of this model are: it cannot work parallel for more than one time series, it is not guaranteed that values estimated are closer to actual ones and lastly, the model fails when accuracy, RMSE etc. parameters are considered. Latent Source Model was developed for binary classification. There is a method called Bayesian Regression for predicting variation in bitcoin price. In combination with Bayesian Regression, LSM determines the patterns in the system. Using trading strategy for making decision either to buy or sell bitcoin, mid-price is predicted. A threshold value is set for this. If average price is less than the threshold, then bitcoin will be sold and if it is equal to greater than bitcoin is purchased. Value for threshold is updated from time to time according to trading strategy. Another method which is multi-layer perceptron is a Deep Artificial Neural Network containing greater than one perceptron. It contains one input and output layer, and an arbitrary number of hidden layers. MLP uses backpropagation and are implemented for supervised learning problems. MLP uses binary classification with two classes as 0 or 1 [28] where 1 indicates increment in bitcoin price for the upcoming day and 0 indicates there will drop in price for the same. One more model i.e. NARX is a dynamic recurrent network having feedback connections with many network layers [29]. This model can make predictions even on dynamic data of time series. The attraction of the model is that this model can accept continuous as well as discreet values for making predictions. It is implemented from linear ARX model. Advantages of this model is that it is faster, gives good accuracy, understand system behaviour better and can generalise effectively as compared to other neural networks.

Comparing all these models it was found that NARX model gave highest accuracy in predicting prices.

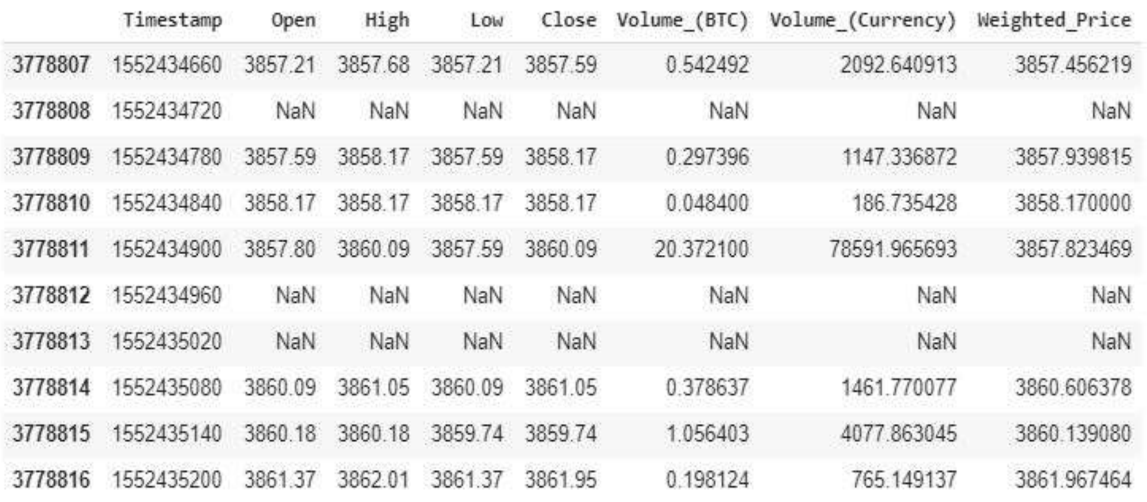
Having a lot of methods and models for price prediction, there is a need to design methods which can improve performance of prediction [30]. In this study, the authors proposed techniques for closing stock price prediction using deep beliefs networks (DBNs). Intrinsic Plasticity (IP) has been also applied to make the network more adaptive. Results show that employment of intrinsic plasticity improved the performance. DBN with IP performed better as compared to Triesch's IP and without IP. Deep Belief Networks are multilayer probabilistic generative models implemented with various Restricted Boltzmann Machine (RBM) layers and BP layer. A RBM has two layers, one is visible and another is hidden layer. The two layers are not connected by themselves but there is a connection between their random units. The training sample is first given to the visible units which computes binary state of hidden layers. Now all nodes of hidden layers are examined to compute binary state of visible units and obtains updated visible layer. IP rule is used as it keeps the neuron in discriminative state. The employment of IP in DBNs not only increased prediction ability but also lead to have less computational time for the test.

In our study, we introduced a new deep learning architecture which is known as transformer. Transformer uses the concept of attention layer. To best of our knowledge, this work gives better results which are closer to actual values with a fair computational time.

CHAPTER 3 - EXPERIMENTAL APPROACH

3.1 DATASET

We downloaded the historical bitcoin dataset from kaggle. The dataset consists of bitcoin price of last 10 years from 2009 to 2019 and is of size 118 Mb. It has columns as Timestamp, Open value, High value, Low value, Close value for the bitcoin price, Volume of bitcoin, Volume of currency and Weighted Price. A sample image for the taken dataset is shown in figure 3.1. We have shown last few columns in our image. Our dataset consist of 3778816 rows and 8 columns.



	Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
3778807	1552434660	3857.21	3857.68	3857.21	3857.59	0.542492	2092.640913	3857.456219
3778808	1552434720	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3778809	1552434780	3857.59	3858.17	3857.59	3858.17	0.297396	1147.336872	3857.939815
3778810	1552434840	3858.17	3858.17	3858.17	3858.17	0.048400	186.735428	3858.170000
3778811	1552434900	3857.80	3860.09	3857.59	3860.09	20.372100	78591.965693	3857.823469
3778812	1552434960	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3778813	1552435020	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3778814	1552435080	3860.09	3861.05	3860.09	3861.05	0.378637	1461.770077	3860.606378
3778815	1552435140	3860.18	3860.18	3859.74	3859.74	1.056403	4077.863045	3860.139080
3778816	1552435200	3861.37	3862.01	3861.37	3861.95	0.198124	765.149137	3861.967464

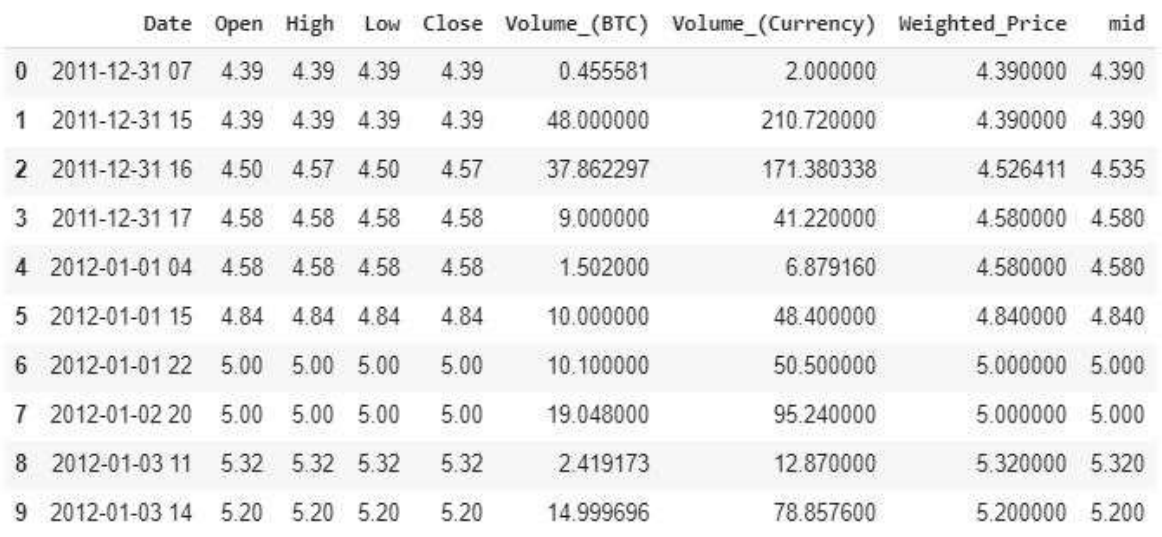
Figure 3. 1 Bitcoin price dataset

3.2. DATA PRE-PROCESSING

Most of the time, the real world data or the raw data is not complete, not consistent, or is full of so many errors. To transform the data into an understandable form, data pre-processing is done. Data pre-processing is the method to resolve the unwanted issues before applying algorithms. Some of the data pre-processing steps involves cleaning, integration, transformation, reduction, etc. We have pre-processed our data for further feeding into algorithms in the following steps:

- As from figure 3.1, we can see there are so many NaN (not a number) cells. To get rid of NaN values, we have used the function - df.dropna(), where df is the variable name of our dataset, to drop the rows containing NaN values. This is done to make the dataset more efficient for processing. Now, the size of our dataset became 59562, 9.

- To get the time in more abstracted form, we convert the value of timestamp in to the date format as yyyy-mm-dd-hh. After converting, our dataset looks like as shown in figure 3.2. This gives our dataset a better look rather than timestamp.
- Also, as we have four different type of values for bitcoin price in our dataset which are open, close, high and low, it is required to manipulate them in a single type of value. To do this, we selected high and low values of the price and divided it by 2 (basically we are taking average of high and low values) and hence calculated mid value for the same. The obtained dataset sample is shown in figure 3.2. We have shown starting ten values in the figure of our dataset.



	Date	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price	mid
0	2011-12-31 07	4.39	4.39	4.39	4.39	0.455581	2.000000	4.390000	4.390
1	2011-12-31 15	4.39	4.39	4.39	4.39	48.000000	210.720000	4.390000	4.390
2	2011-12-31 16	4.50	4.57	4.50	4.57	37.862297	171.380338	4.526411	4.535
3	2011-12-31 17	4.58	4.58	4.58	4.58	9.000000	41.220000	4.580000	4.580
4	2012-01-01 04	4.58	4.58	4.58	4.58	1.502000	6.879160	4.580000	4.580
5	2012-01-01 15	4.84	4.84	4.84	4.84	10.000000	48.400000	4.840000	4.840
6	2012-01-01 22	5.00	5.00	5.00	5.00	10.100000	50.500000	5.000000	5.000
7	2012-01-02 20	5.00	5.00	5.00	5.00	19.048000	95.240000	5.000000	5.000
8	2012-01-03 11	5.32	5.32	5.32	5.32	2.419173	12.870000	5.320000	5.320
9	2012-01-03 14	5.20	5.20	5.20	5.20	14.999696	78.857600	5.200000	5.200

Figure 3. 2 Dataset after processing

- After getting mid values, we now plot the mid values with corresponding date to see the pattern of the fluctuation of bitcoin price as shown in figure 3.3. This give us insight about how the bitcoin price varied in those ten years. As we found that the minimum value of ‘mid’ is 4.14 and maximum value is 19565.24208 approximately, we set a range for middle price values. The range we put is >=500 to <=20000. The sample figure shows the ‘mid’ values in the same range. Now, our dataset contains 29974 rows.

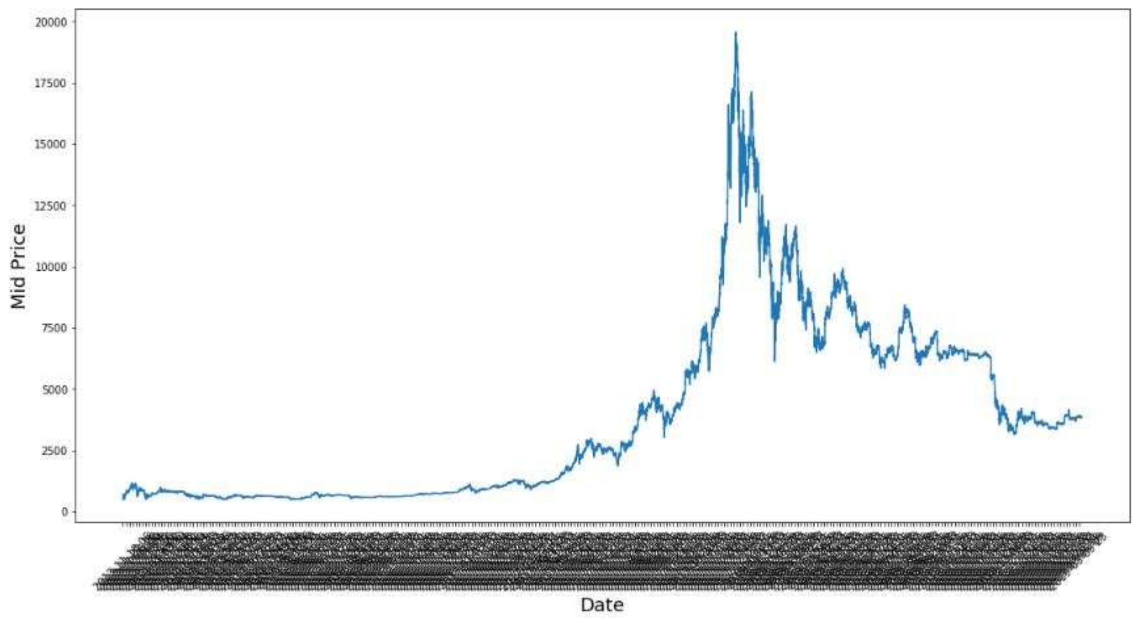
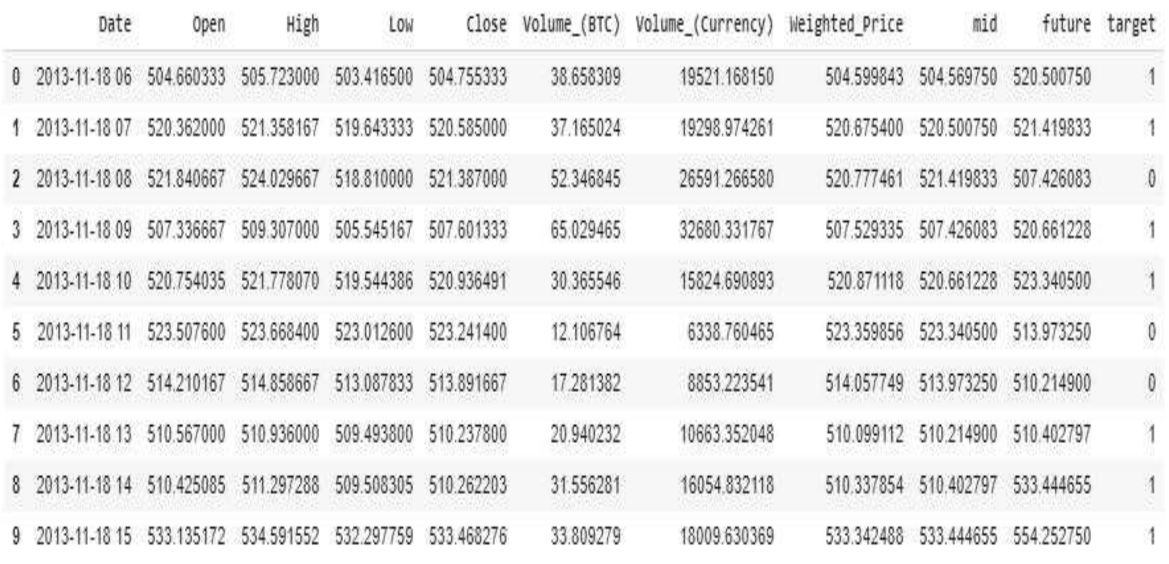


Figure 3. 3 Plotting average values from ‘low’ and ‘high’ values

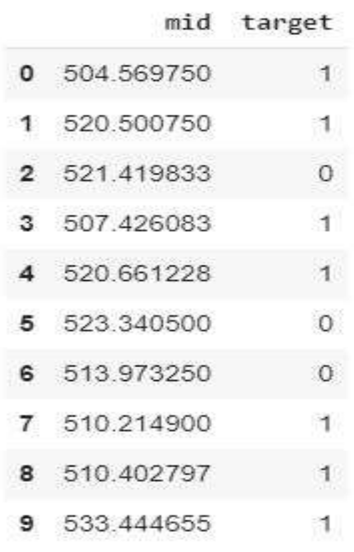
- A column named as ‘future’ is set. We made another column as ‘target’. We distributed the value in this column as - if the price of future value is greater than price of mid, than target is set to ‘1’ otherwise it is set to ‘0’. We have displayed first ten rows of the dataset as shown in figure 3.4.



	Date	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price	mid	future	target
0	2013-11-18 06	504.660333	505.723000	503.416500	504.755333	38.658309	19521.168150	504.599843	504.569750	520.500750	1
1	2013-11-18 07	520.362000	521.358167	519.643333	520.585000	37.165024	19298.974261	520.675400	520.500750	521.419833	1
2	2013-11-18 08	521.840667	524.029667	518.810000	521.387000	52.346845	26591.266580	520.777461	521.419833	507.426083	0
3	2013-11-18 09	507.336667	509.307000	505.545167	507.601333	65.029465	32680.331767	507.529335	507.426083	520.661228	1
4	2013-11-18 10	520.754035	521.778070	519.544386	520.936491	30.365546	15824.690893	520.871118	520.661228	523.340500	1
5	2013-11-18 11	523.507600	523.668400	523.012600	523.241400	12.106764	6338.760465	523.359856	523.340500	513.973250	0
6	2013-11-18 12	514.210167	514.858667	513.087833	513.891667	17.281382	8853.223541	514.057749	513.973250	510.214900	0
7	2013-11-18 13	510.567000	510.936000	509.493800	510.237800	20.940232	10663.352048	510.099112	510.214900	510.402797	1
8	2013-11-18 14	510.425085	511.297288	509.508305	510.262203	31.556281	16054.832118	510.337854	510.402797	533.444655	1
9	2013-11-18 15	533.135172	534.591552	532.297759	533.468276	33.809279	18009.630369	533.342488	533.444655	554.252750	1

Figure 3. 4 Dataset after setting target column

- We have divided the dataset into three different datasets which are train set which is used to train our model, validation set for validating our results and test set predict the value on the sample values. We have distributed our data in train, valid and test set as 70%, 20% and 10% of the whole dataset respectively. Now, for our processing, we only need two columns ‘mid’ and ‘target’. So, we dropped the rest of the columns. The dataset after dropping looks like as shown in figure 3.5.



	mid	target
0	504.569750	1
1	520.500750	1
2	521.419833	0
3	507.426083	1
4	520.661228	1
5	523.340500	0
6	513.973250	0
7	510.214900	1
8	510.402797	1
9	533.444655	1

Figure 3. 5 Dataset after dropping columns

- We reshape train and test data and normalise test and validation dataset. After that, we performed exponential moving average smoothing so that the data have smoother curves as compared to original data. Figure 3.6 shows the curve for train, valid and test dataset in blue, gray and black colour respectively. The graph shows how these three set are separated. The normalized predictions with respect to time is shown in the graph.

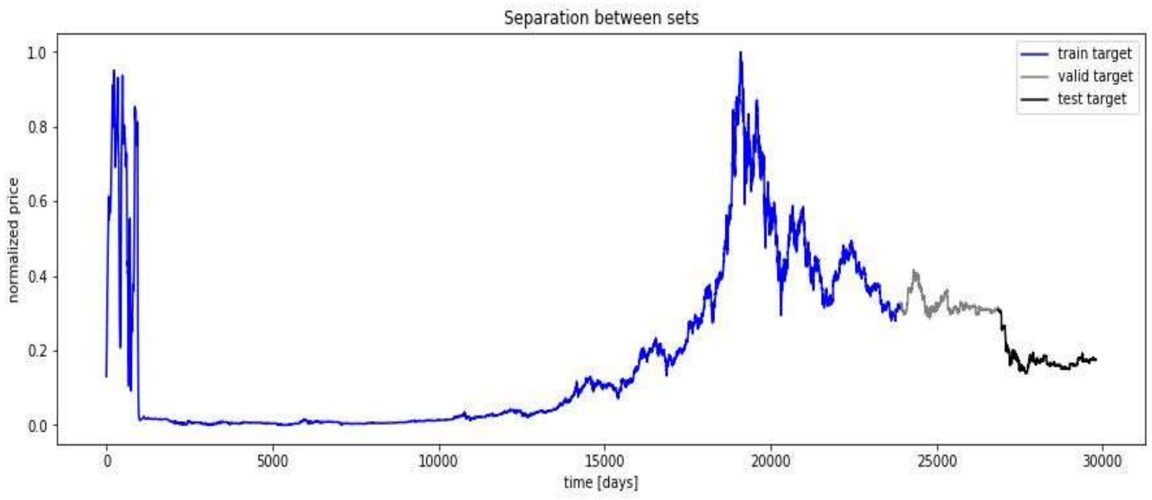


Figure 3. 6 Separation between train, valid and test set

3.3 METHODOLOGY

After pre-processing, we now move on to our transformer model architecture. We run the transformer model with two types of RNNs - transformer with LSTM and transformer with GRU.

3.3.1 Model Architecture

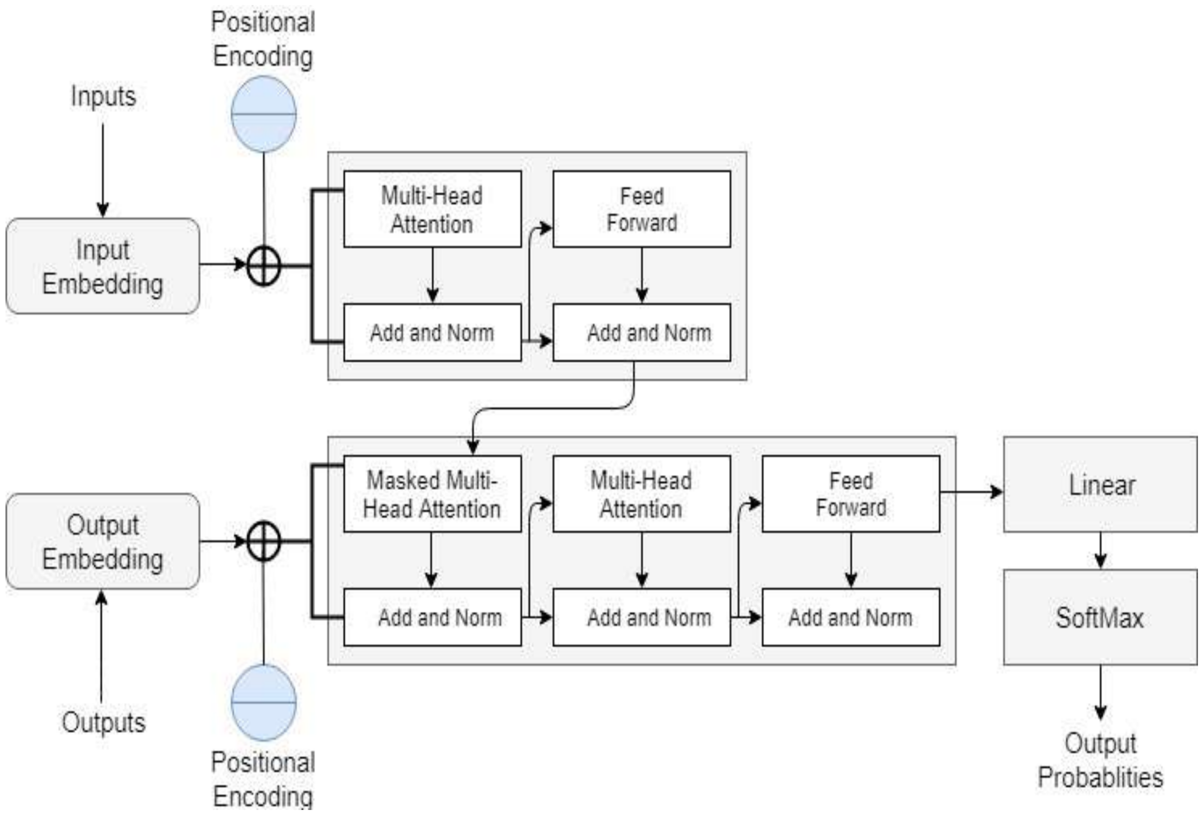


Figure 3. 7 The Transformer Model Architecture

Most of the good competitive neural networks have encoder decoder structure [11] [12] [13]. The function of encoder is to map the input sequence x = (x_1, x_2, ….. x_n) to the continuous sequence of z = (z_1, z_2, ….. z_n). Obtaining z, now comes the function of the decoder. The decoder now produces the sequence of output as y = (y_1, y_2, ….. y_n), generated as a single element once at a time. Model is auto-regressive at every step [14], which consumes the symbols that are generated previously as more inputs to generate the next one. Transformer model follows the above described process with the help of stacked self-attention and point-wise, fully-connected layers for both the encoder and the decoder as represented in the figure 3.7 respectively.

3.3.2 Encoder and Decoder Stacks

3.3.2.1 Encoder

The encoder in the model is made up of a stack of N similar layers and every layer is composed of two sub-layers. The first mechanism is the Multi-Head Attention and the second one is a basic, Position-Wise Fully-Connected Feed Forward network. Around each sub-layer, we applied a residual connection [16] following by layer normalization [15]. This means that output of every sub-layer is LayerNorm(x + SubLayer(x)), where SubLayer(x) is implemented by the sub-layer itself. And hence, the output of dimensional d_{model} is produced by all the sub-layers and embedding layers in the model just to facilitate the residual connections. A basic encoder-decoder mapping can be shown in figure 3.2.

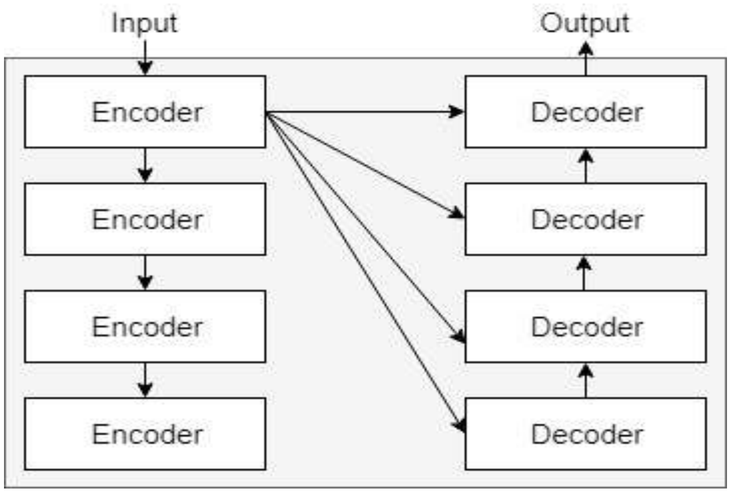


Figure 3. 8 A simple Encoder-Decoder mapping

The step-wise working of encoder may be defined as follows:

- The input to the encoder is produced by adding combination of input embedding and positional encoding.
- Residual connections are most vital part of encoding. Around each of the sub-layers, 'n' multi-head attention layers and position-wise feed forward along with these connections are applied and is followed by layer normalization.
- Lastly, before the normalization, for each of the sub-layers, dropout are applied to the output.

Input embedding and positional encoding in transformer first generates initial inputs in encoder phase for every word in the sequence of the input. For every single word, the role of self-attention is to aggregate information from all the pair-wise words of the sentence.

This leads to the creation of new representation of all words. This is an attended presentation of each word of the sequence. Lastly, the above process loops for every word and hence builds new representation on the top the previous ones for multiple times.

3.3.2.2 Decoder

Just like encoder, decoder is too made up of a stack of 'N' similar layers. Having same those two sub-layers in each encoder layer, decoder, in addition it has a third sub-layer. This additional layer performs the multi-head attention on the encoder output stack, same as encoder in which residual connections are applied upon each sub-layers and is following by layer normalization. In decoder stack, modifications on self-attention layer are made so as to avoid positions from attending to subsequent ones. The embedding of the output are the offsets by one position. This masking is combined with the above fact and it ensures that for a given position, predictions are dependent on the already known outputs by less than the given position. The steps can be summarised as shown below, they are much similar like in encoder:

- The input of the decoder is obtained by the combination of output embedding and positional encoding by one offset position ensuring that the prediction for the given position is dependent only on the position before the given one.
- Around each of the two sub-layers, 'n' masked multi-head attention layers, multi-head attention and position-wise feed forward along with residual connections are applied and is followed by layer normalization.
- To avoid words from future being the part of attention, masked multi-head attention is employed and followed by position-wise feed forward neural network.

Internal working of the encoder-decoder model can be visually shown in figure 3.3. Decoder produces a single word at an instance of time starting from left to right. The first word in the sequence is dependent on finalised presentation of encoder which is the offset by one position. Each predicted word attends to word produced at that layer of the decoder and multi-head representation of encoder which is identical to a typical encoder-decoder structure.

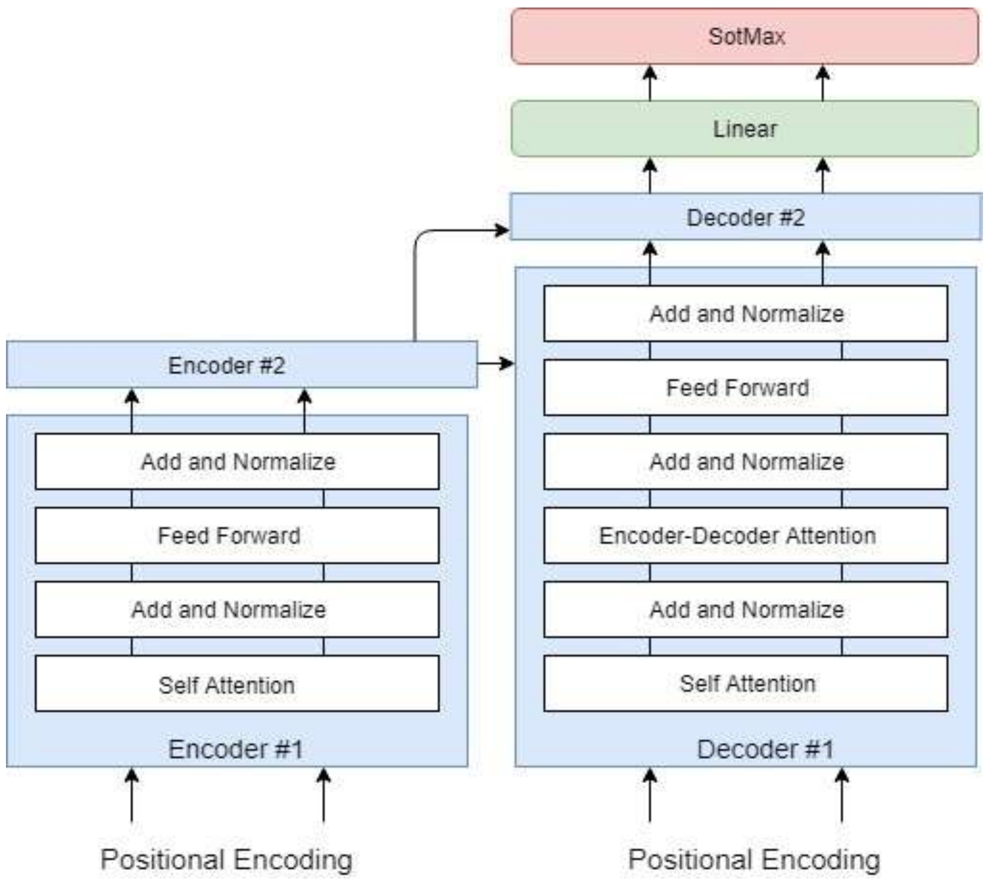


Figure 3. 9 Working of internal layers in Encoder-Decoder

3.3.3 Attention

3.3.3.1 Layer Normalization

One of the main reasons why deep learning progressed lot in recent years is due to batch normalization. Although batch normalization is an effective tool, but it also has various shortcomings too. The most important limitation of batch normalization is that it depends on the mini batch. In this method, mean and variance of each mini-batch is calculated and each feature is normalized according to mini-batch statics. Due to this reason, mean and variance of each mini-batch differs from each other. This leads in two main problems. First one is: Lower limit on batch size. It is obvious that batch normalization cannot be used when the batch size is ‘1’. But if the batch size is slightly greater, it may also cause problems. The process of calculating mean of the whole dataset after each network update is costly and due to this mean and variance are estimated using mini-batch statics. Now they will have error which varies from one mini-batch to another mini-batch. The second issue is that it is difficult to apply batch normalization to recurrent connections in RNN. In an RNN, the activation of each step has different statics. It means that it is necessary to fit a separate normalization layer for each step. And hence, all this makes the model more typical. In layer Normalization, mini-batch dependency is eliminated. Let us explain what layer normalization is. Let's consider that a mini batch with same number of features having multiple example. Mini-batches are matrices where one axis or axes refers to the batch dimensions and other axis or axes refers to the feature dimensions (if input is multi-dimensional). In batch normalization, normalization of inputs is done on batch dimension where as in layer normalization it is done on feature dimensions. This is the prime feature of layer normalization. The equations for both batch and layer normalization are so much identical as shown:

Equations for batch normalization:

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_{ij} \tag{3.1}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x_{ij} - \mu_j\right)^2 \tag{3.2}$$

$$\widehat{x_{ij}} = \frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\epsilon}} \tag{3.3}$$

Equations for layer normalization:

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_{ij} \tag{3.4}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x_{ij} - \mu_j\right)^2 \tag{3.5}$$

$$\widehat{x_{ij}} = \frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\epsilon}} \tag{3.6}$$

Where x_{ij} is the i-jth element of the input, the first one represents the batch and the second one represents the features. The difference can be made clear with help of the figure 3.10 shown below.

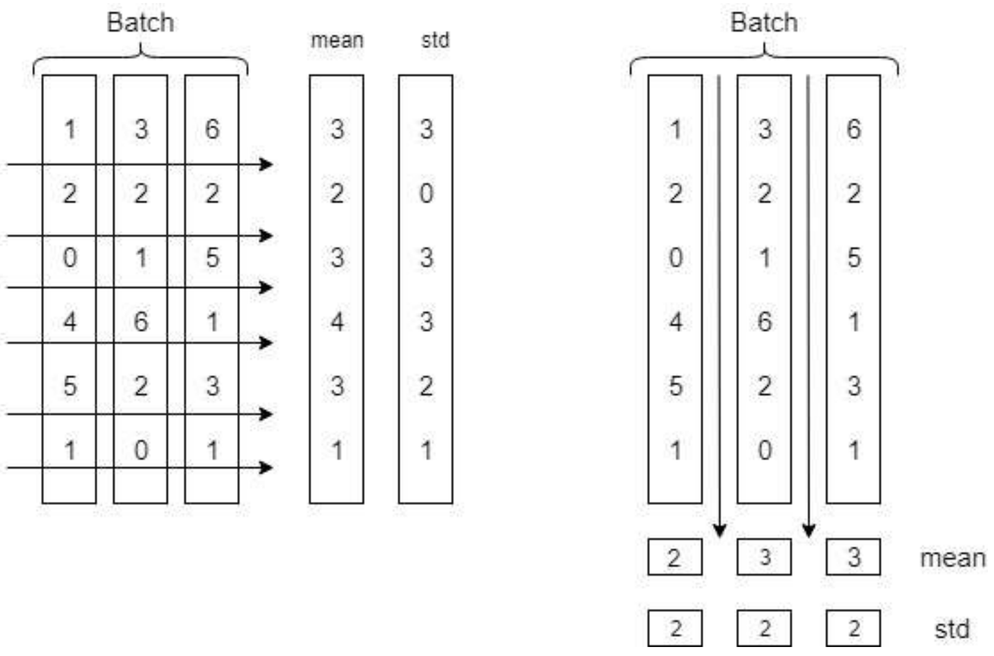


Figure 3. 10 Batch Normalization (left) and Layer Normalization (right)

As from the figure, it is easy to understand that in batch normalization, the statistics are made across batch and are same for each batch where as in layer normalization, it is computed across the features and there is no dependency on the examples. Mean and standard variance are same for all feature dimensions.

Advantages of layer normalization are:

1. Experimental results have shown that layer normalization have better performance on recurrent neural networks.

2. As there are independency between inputs, this indicates that each input has different normalization operation even if we are using mini-batch sizes.

3.3.3.2 Scaled Dot-Product Attention

The attention used in the model is called as Scaled Dot-Product Attention as shown in figure 3.3. In the model, the input is containing queries, keys and values, keys of dimension d_k and values of dimension d_v. Query dot-product with all the keys are computed and this multiplication is divided by $\sqrt{d_k}$. To get weights on the values, we applied 'Softmax' function. However, the queries Q, keys K and values V are packed together in the matrix, simultaneously we compute function for attention on the set of queries Q. Matrix for the output is calculated as shown below in the equation:

$$\text{Attention}(Q, K, V) = \text{Softmax}(QK^T/\sqrt{d_k})V \tag{3.7}$$

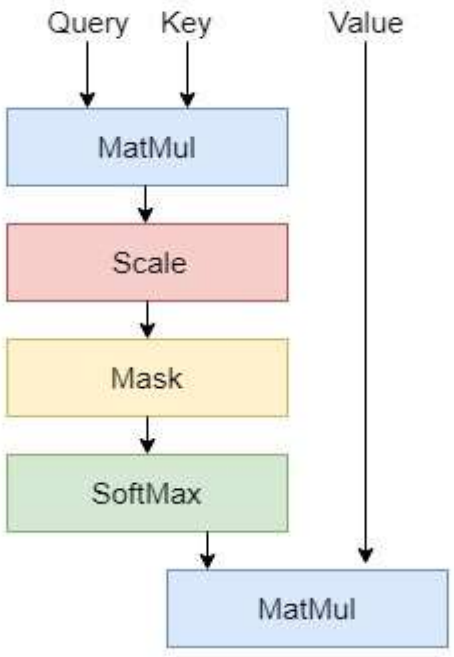


Figure 3. 11 Scaled Dot-Product Attention

There are two common functions for attention, they are additive attention and dot-product attention. In additive attention, compatibility function is calculated with the help of a feed-forward network with one hidden layer. In fact, the dot-product attention is similar to our algorithm, the only difference is the scaling factor of $1/\sqrt{d_{k.}}$ However, both the methods are same in theoretical complexity but dot-product attention is much efficient in space and faster too in use as it is implemented with high optimized matrices multiplication code.

In case of low values of d_k, both the above methods performs same. Without scaling for larger value of $d_{k,}$ additive attention method performs better than the dot-product attention method. For greater values of $d_{k,}$ we find that the dot-product grows large value and push the function ‘softmax’ into areas where it has very low gradients. And to overcome this effect, we scaled this dot-product by $1/\ d_{k.}$

3.3.3.3 Multi Head Attention

Instead of applying an individual attention function with d_{model}-dimensional on queries, keys and values, we observed that is fruitful to project linearly *h* times these queries, keys and values with various learning linear projections to d_k, d_k and d_v dimensions respectively. We now performed attention function in parallel on each projected version of queries, keys and values and yield d_v-dimensional output values.

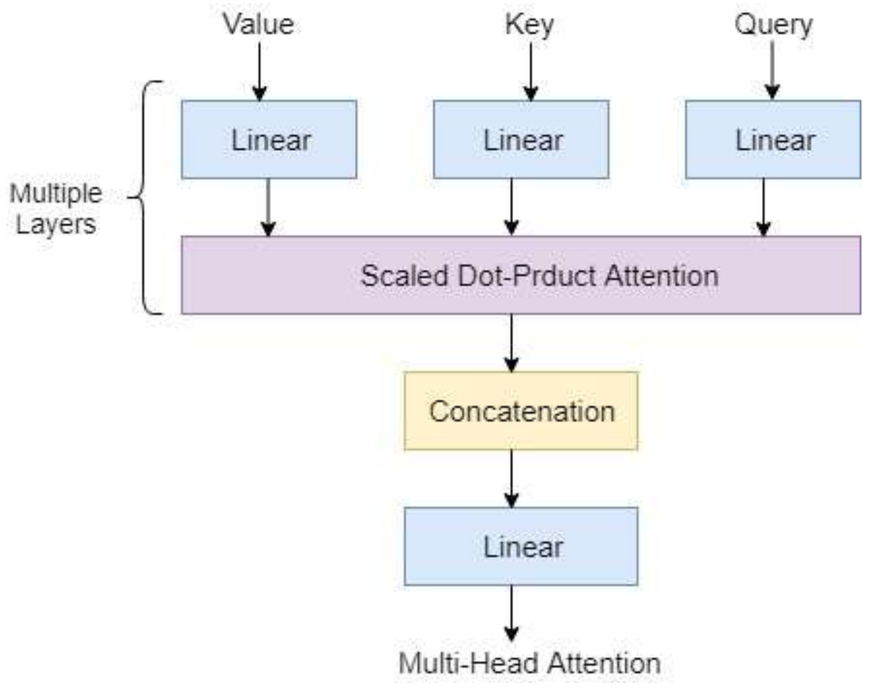


Figure 3. 12 Multi-Head Attention

They are now combined and projected again which results in final values as shown in the figure 3.4. This multi-head attention permits the model for attending information in combined form at different positions from different representation subspaces.

Multi Head Attention (Q, K, V) = Concat (head_1, head_2, …… head_h)W^O (3.8)

where, head_i = Attention (QW_i^Q, KW_i^K, VW_i^V)

where, the projections are parameter matrices $W_i^Q \in R^{d_{model} * d_k}$, $W_i^K \in R^{d_{model} * d_k}$, $W_i^V \in R^{d_{model} * d_v}$ and $W_i^O \in R^{d_{model} * hd_v}$

Transformer model applies multi-head attention in the following different forms:

1. The encoder has self-attention layers internally. All queries, keys and values comes from the same place in the self-attention layer i.e. the outcome of the previous encoder layer. The input sequence itself is the input to multi-head self-attention i.e. the queries, keys and values in linearly transform heads.
2. The queries comes from previous layer of decoder whereas the values and keys comes from outcome of the decoder in encoder-decoder attention layers. This helps the decoder in allowing each position for attending all positions in input sequence like in the basic encoder-decoder structure.

3. Masking support is employed in scaled dot-product attention. This is done by masking out the input values of multi-head attention softmax referring to wrong connections.

3.3.3.4 Position-Wise Feed Forward Network

Every layer in Encoder and Decoder consist a Fully-Connected Feed-Forward network (dense layers) in addition to attention sub-layers. They are applied to every position identically and in separation. And it contains two linear transformation in between with a ReLU activation function. However, these linear transformation are identical for different positions, different parameters are used which varies from layer to layer.

$$\text{FNN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{3.9}$$

3.3.3.5 Positional Encoding

As the model is containing no recurrent and convolution layer, then how then model will be able to know about the order of the sequence? Because of the absence of recurrent and convolution layer, we needed positional encodings. We are required to add few information for relative or absolute positions of the tokens in the sequence. With the help of positional encodings, we can input embedding at the end of the stacks of encoder and decoder. As the dimensions of both the positional encoding and d_{model} are same, they both can be summed up. The positional encoding tells the model that to which potion of the input the model is dealing with currently. There are various methods for positional encodings but in our work we have taken sin and cos functions of variable frequencies:

$$PE(pos, 2i) = \sin(pos/ 1000^{2i/d}{}_{model}) \tag{3.10}$$

$$PE(pos, 2i+1) = \cos(pos/ 1000^{2i/d}{}_{model}) \tag{3.11}$$

where 'pos' refers to the position and 'i' is the dimension, means dimension of positional encoding corresponding to a sinusoid and the wavelengths are in geometric progression of 2π to $1000*2\pi$. We choose this as it will permit the model to learn efficiently to attend on the basis of relative positions.

3.3.4 Add and Norm

In this phase, the output and input of position-wise feed forward network or multi-head attention layer are concatenated by a block which consists of residual structure a layer for layer normalization. Layer normalization is identical to batch normalization except the mean and variances are computed over the last dimension i.e. X_mean(axis=-1) instead of dimension of batch which is X_mean (axis=0). The block which is employed accepts inputs X and Y which are input and output of another block. Dropout is applied in this connection block. X and Y must have similar shape because of residual connections.

3.3.5. Masking

Input of variable length in recurrent neural networks are handle by masking. Although RNNs are efficient in handling inputs of variable length but still it is required that inputs should be of fixed length. To manage this issue we create mask per sample of length which is equal to the longest sequence in the dataset and is initialised to ‘0’. After that, the mask is filled with value ‘1’ where it was containing values in. In our model, we have used two type of masking sub-mask and pad-mask.

3.3.6 Pooling

Pooling is a way of down sampling the extracted feature map, through reducing the number of dimensions of the feature map but also keeping map features required for classifying which sufficiently decreases the processing time, while still keeping the important features for further processing. The aim of pooling is to achieve spatial invariance. This is done with the help of rotational invariance and translation. For training pooling operations, backpropagation is used. Pooling is mostly used for making input or features independent. There are so many types of pooling like sum pooling, max pooling, min pooling, average pooling, etc. In our model, we have applied global average and global max pooling.

3.3.6.1 Global Max Pooling

Max pooling or Maximum pooling operation is basically used to calculate the maximum value in each feature map. Like in normal pooling, patches of input map are down-sampled while in global pooling, the whole feature map is down-sampled to a one value. It means we are setting the pool size to the input feature map size. The outputs are down-sampled feature maps to highlight the existing features in that patch means not like the average presence of features in average pooling. It is found that max pooling works better than average pooling. Arguments required by Max Pooling are pool size, strides, padding, and data format whereas Global Max Pooling requires only data format argument. Sometimes, global pooling is used as a substitute for fully-connected layer to transform feature map to the output prediction in the model.

3.3.6.2 Global Average Pooling

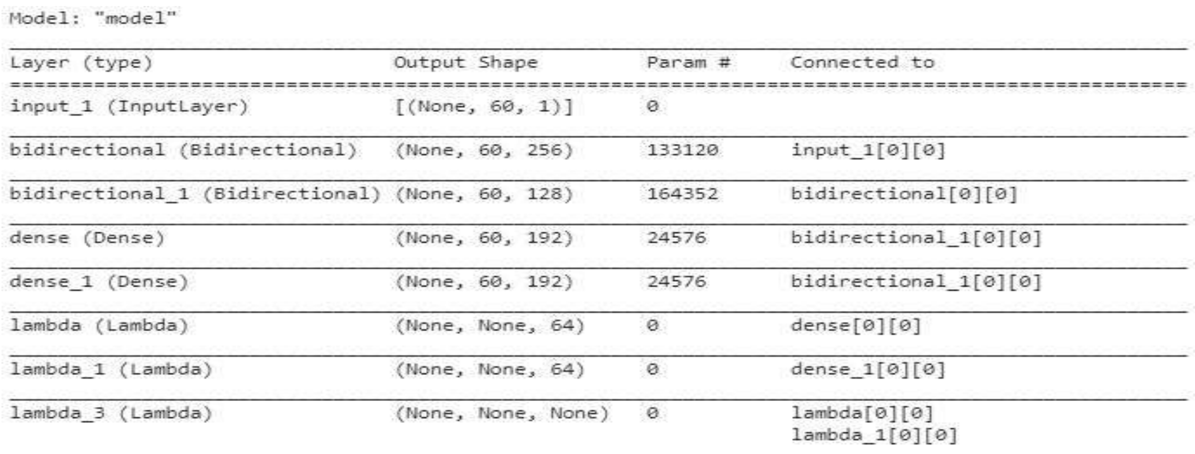
The global average pooling operation computes the average outcome of each feature map of previous employed layer. This operation is used to reduce the data sufficiently and to prepare the model for final stage of classification. It contains no trainable parameters which is same as in max pooling. The reduction of these trainable parameters eliminates the tendency of overfitting which are required in fully-connected layers with the help of dropout. Due to this, every feature map becomes a type of ‘confidence map’. Average pooling makes the model more efficient for spatial translation. Average Pooling takes arguments as pool size, strides, padding, and data format whereas Global Average Pooling requires only data format argument.

3.3.6.3 Max and Average Pooling Concatenation

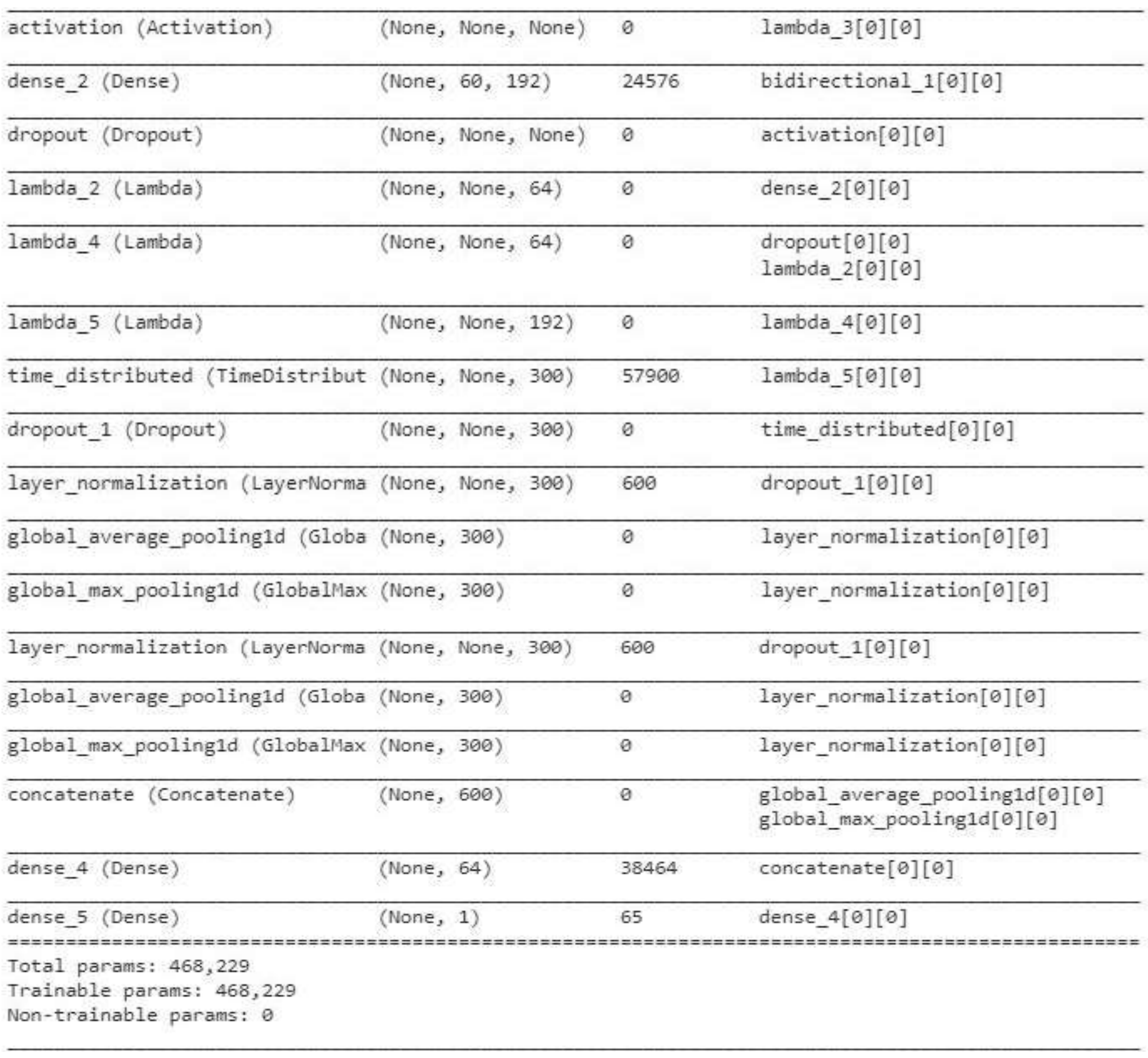
Hybrid of both global max pooling and global average pooling seems to give better and improved results. Although the operation is supposed to have higher computational cost but the outputs are more flexible. Applying this concatenation, one should make checks that the output has the same shape as that was of single pooling.

3.3.7 Building the Model

The summary of constructed model can be shown in figure 3.13.



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 60, 1)]      0
__________________________________________________________________________________________________
bidirectional (Bidirectional)   (None, 60, 256)      133120      input_1[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 60, 128)      164352      bidirectional[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 60, 192)      24576       bidirectional_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 60, 192)      24576       bidirectional_1[0][0]
__________________________________________________________________________________________________
lambda (Lambda)                 (None, None, 64)     0           dense[0][0]
__________________________________________________________________________________________________
lambda_1 (Lambda)               (None, None, 64)     0           dense_1[0][0]
__________________________________________________________________________________________________
lambda_3 (Lambda)               (None, None, None)   0           lambda[0][0]
                                                                 lambda_1[0][0]
```



```
________________________________________________________________________________________________
activation (Activation)         (None, None, None)   0           lambda_3[0][0]
________________________________________________________________________________________________
dense_2 (Dense)                 (None, 60, 192)      24576       bidirectional_1[0][0]
________________________________________________________________________________________________
dropout (Dropout)               (None, None, None)   0           activation[0][0]
________________________________________________________________________________________________
lambda_2 (Lambda)               (None, None, 64)     0           dense_2[0][0]
________________________________________________________________________________________________
lambda_4 (Lambda)               (None, None, 64)     0           dropout[0][0]
                                                                 lambda_2[0][0]
________________________________________________________________________________________________
lambda_5 (Lambda)               (None, None, 192)    0           lambda_4[0][0]
________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 300)    57900       lambda_5[0][0]
________________________________________________________________________________________________
dropout_1 (Dropout)             (None, None, 300)    0           time_distributed[0][0]
________________________________________________________________________________________________
layer_normalization (LayerNorma (None, None, 300)    600         dropout_1[0][0]
________________________________________________________________________________________________
global_average_pooling1d (Globa (None, 300)          0           layer_normalization[0][0]
________________________________________________________________________________________________
global_max_pooling1d (GlobalMax (None, 300)          0           layer_normalization[0][0]
________________________________________________________________________________________________
layer_normalization (LayerNorma (None, None, 300)    600         dropout_1[0][0]
________________________________________________________________________________________________
global_average_pooling1d (Globa (None, 300)          0           layer_normalization[0][0]
________________________________________________________________________________________________
global_max_pooling1d (GlobalMax (None, 300)          0           layer_normalization[0][0]
________________________________________________________________________________________________
concatenate (Concatenate)       (None, 600)          0           global_average_pooling1d[0][0]
                                                                 global_max_pooling1d[0][0]
________________________________________________________________________________________________
dense_4 (Dense)                 (None, 64)           38464       concatenate[0][0]
________________________________________________________________________________________________
dense_5 (Dense)                 (None, 1)            65          dense_4[0][0]
================================================================================================
Total params: 468,229
Trainable params: 468,229
Non-trainable params: 0
________________________________________________________________________________________________
```

Figure 3. 13 Model Summary

CHAPTER 4 RESULTS

After applying model on both LSTM-with-attention and GRU-with-attention, we obtain the predicted values as shown in the table 4.1, only first ten values of the dataset has been shown. Observing these values, we can see the difference between the actual and predicted values for both the models.

Sr. No.	**Actual Values**	**Predicted values from LSTM**	**Predicted values from GRU**
0	0.307744	0.375506	0.419975
1	0.307679	0.375485	0.419990
2	0.307167	0.375453	0.420010
3	0.307238	0.375363	0.420008
4	0.307348	0.375278	0.420015
5	0.307587	0.375220	0.420038
6	0.307496	0.375187	0.420077
7	0.307074	0.375169	0.420122
8	0.307216	0.375134	0.420154
9	0.307254	0.375129	0.420202

Table 4. 1 Comparison of predicted values with actual values of both models

We plot these actual and predicted values for both model as shown in figure 4.1 and 4.2. We plot around 3000 values from our test dataset to make predictions. From the figures, we can observe that the GRU model gives better results for predicting bitcoin values. The values predicted by GRU model are closer to actual values than in LSTM model.

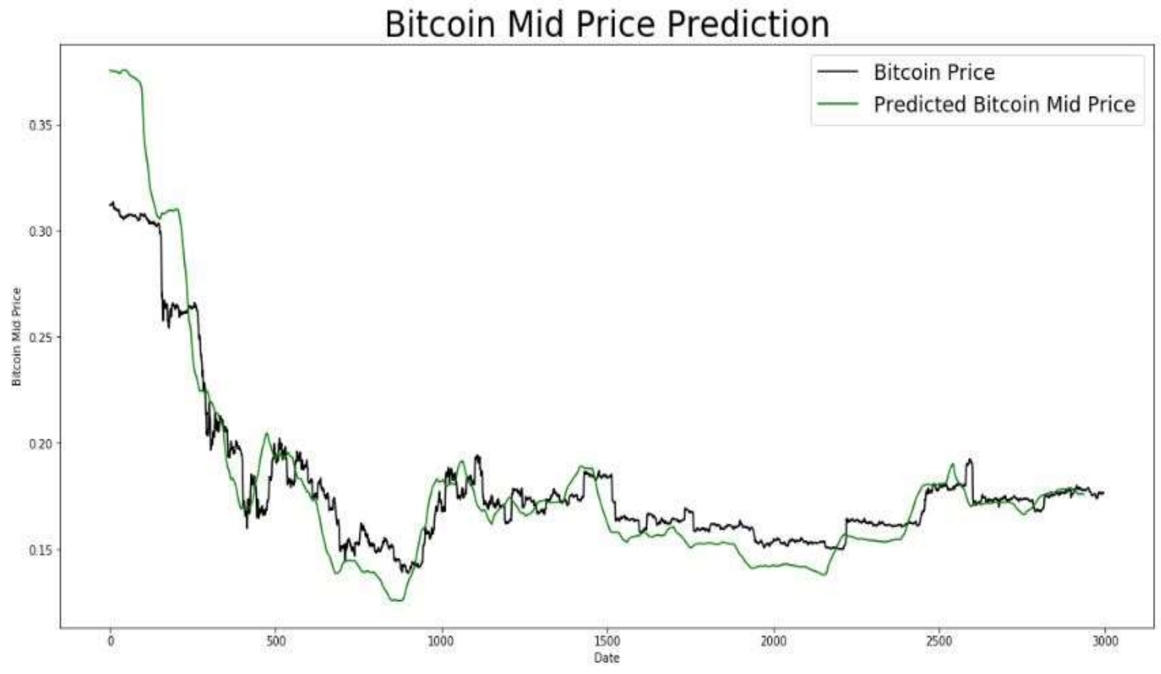


Figure 4. 1 Bitcoin Price Prediction using GRU-with-Attention

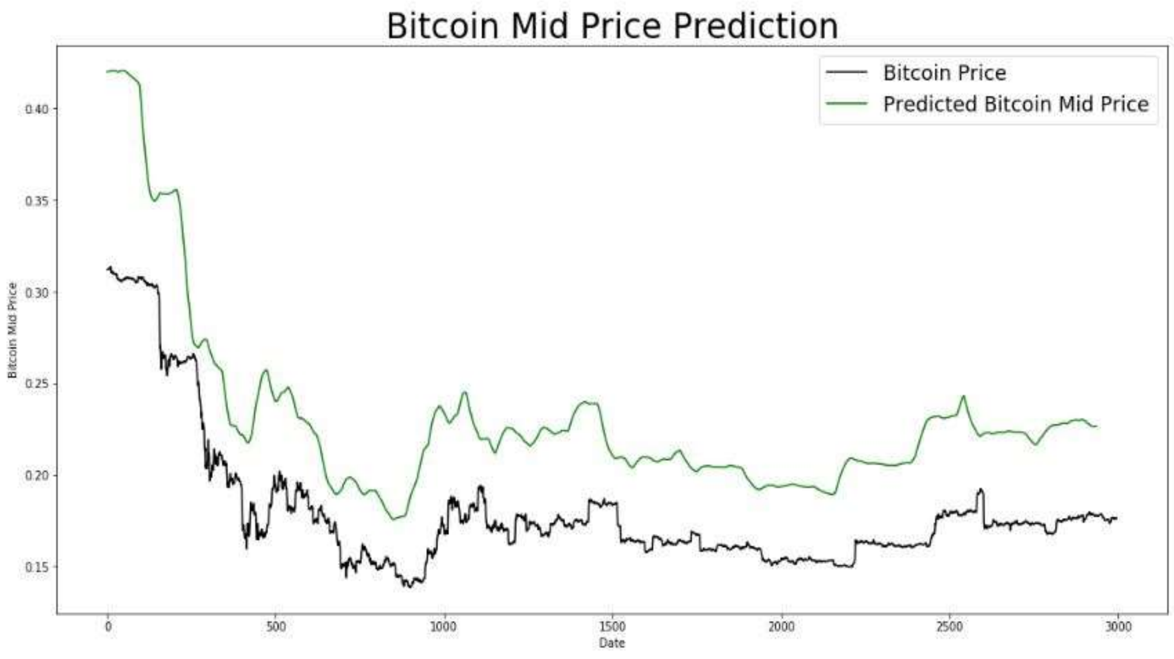


Figure 4. 2 Bitcoin Price Prediction using LSTM-with-Attention

Error Measurement Metrics

We now compared the performance of both the model in terms of ‘Mean Absolute Error’, ‘Mean Squared Error’, ‘Root Mean Squared Error’ and ‘Maximum Error’. All the metrics can be defined as follows:

Mean Absolute Error (MAE)

MAE is used to calculate the average value of errors in a dataset without any consideration of direction. The average is taken over the test set of the absolute difference between actual values and predicted values and each difference has equal weight. MAE can be mathematically defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_{actual} - y_{pred}| \tag{3.12}$$

where, ‘n’ is the total number of rows in dataset, ‘i’ is a particular value at instance, y_{actual} is actual value and y_{pred} is predicted value.

Mean Squared Error (MSE)

Also known as Mean Squared Deviation (MSD). MSE is used to calculate the average of the squares of the errors of the dataset obtained from the difference of actual and predicted observations. Value of MSE is always positive. The reason behind it is randomness. It is the quality measurement for an estimator. Values are always non-negative. The more close the value to zero, the better the results are. Formula for MSE can be written mathematically as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{actual} - y_{pred})^2 \tag{3.13}$$

where, abbreviations are same as in mean absolute error.

Root Mean Squared Error (RMSE)

As can be seen from equation, it is the square-root of the MSE. RMSE is also calculated to obtain average of error magnitude. It is a rule for quadratic scoring. Basically it is the square-root of the difference of squared values of actual and predicted values. Lower values show better results. Equation for RMSE can be shown below:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{actual} - y_{pred})^2} \tag{3.14}$$

Maximum Error (ME)

It is the maximum difference obtained by subtracting actual value and predicted values. Depending upon the difference, it can be either negative or positive in nature. Max error shows how far our predicted value is from actual value.

The table for both the models i.e. LSTM-with-attention and GRU-with-attention having above explained metrics is shown below:

Model	MAE	MSE	RMSE	ME
LSTM	0.0527239	0.003102204	0.05569743	0.146352951
GRU	0.011985815	0.011985815	0.020012971	0.096071129

Table 4. 2 Comparison of both models on the basis of different error values

Observing the above table, we can see that for every metric our model GRU-with-attention outperforms the model LSTM-with-attention. Each metric for GRU model is better than LSTM model. A visual graphical representation for the same can be shown in figure 4.3.

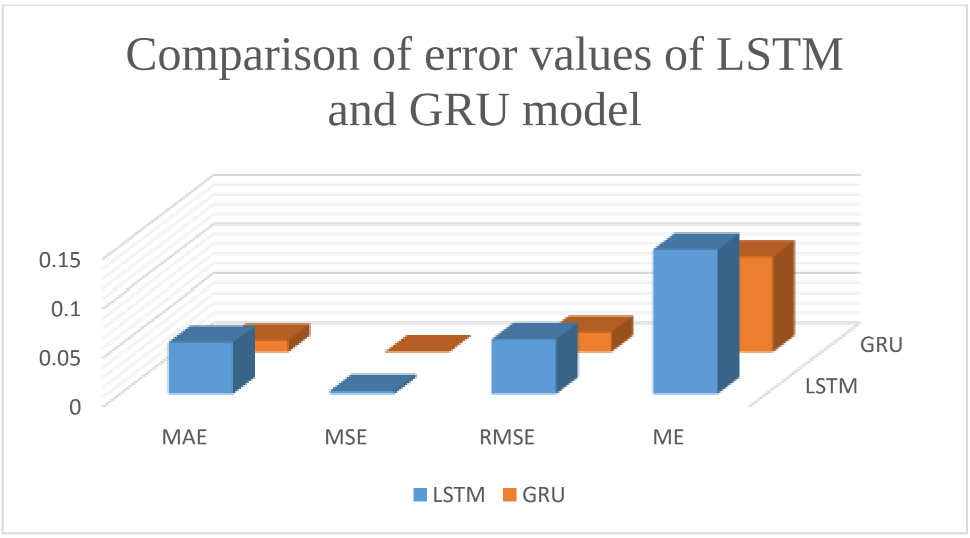


Figure 4. 3 Graphical form for comparison of error measurements

From the above graphical representation, it can be seen that the bar of LSTM model is higher than the bar of GRU model. It is required that error measurements should be close to zero for better predictions. GRU model fulfils the requirements effectively as compared to LSTM model.

CHAPTER 5 CONCLUSION AND FUTURE WORK

Blockchain technology ends up with various factors such as security, privacy, transparency, trustworthiness, etc. which a network communication demands the most for information exchange. Cryptocurrency, like bitcoin, is its one of the component which proved a leading role in decentralization. Every moment of time the price of bitcoin fluctuates and for coin holders or the customers willing to have it, it is an interesting concern for them. Hence, it is important to know the features these fluctuations depends upon. In this study, we build transformer model for predicting mid-price of bitcoin. Our work employed two types of deep learning architecture, transformer model, with Long-Short Term with attention and Gated Recurrent Unit with attention. Mid-price for bitcoin is predicted using attributes mainly ‘Open’, ‘Close’, ‘High’ and ‘Low’. Applying both these, we compared error measurements MAE, MSE, ME and RMSE for both the models. We observed that model with GRU predicted bitcoin price more precisely as compared to LSTM. GRU gives closer values of mid-price to actual values.

The proposed work can further be utilised for making predictions in healthcare, finance, business and many more. The techniques implemented in this study can further be extended by more work and research on advanced upcoming methods. The proposed model can be implemented with other upcoming technique so that it can operate in lesser time. The main aim of the work is to predict the prices with as much as possible precision so as the investors can have idea whether to invest the funds analysing the future prices.

References

[1] Acharjamayum, R. Patgiri and D. Devi, "Blockchain: A Tale of Peer to Peer Security," 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Bangalore, India, 2018.

[2] N. Chaudhry and M. M. Yousaf, "Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities," 2018 12th International Conference on Open Source Systems and Technologies (ICOSST), Lahore, Pakistan, 2018.

[3] L. S. Sankar, M. Sindhu and M. Sethumadhavan, "Survey of consensus protocols on blockchain applications," 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, 2017, pp. 1-5.

[4] U. Mukhopadhyay, A. Skjellum, O. Hambolu, J. Oakley, L. Yu and R. Brooks, "A brief survey of Cryptocurrency systems," *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, Auckland, 2016, pp. 745-752.

[5] F. Dai, Y. Shi, N. Meng, L. Wei and Z. Ye, "From Bitcoin to cybersecurity: A comparative study of blockchain application and security issues," *2017 4th International Conference on Systems and Informatics (ICSAI)*, Hangzhou, 2017, pp. 975-979.

[6] E. Garcia Ribera, "Design and implementation of a proof-of-stake consensus algorithm for blockchain," B.S. thesis, Universitat Polite`cnica de Catalunya, 2018.

[7] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof- of-stake," self- published paper, August, vol. 19, 2012.

[8] A. Porat, A. Pratap, P. Shah, and V. Adkar, "Blockchain consensus: An analysis of proof-of-work and its applications."

[9] L. Chen, L. Xu, Z. Gao, Y. Lu and W. Shi, "Protecting Early Stage Proof-of-Work Based Public Blockchain," *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Luxembourg City, 2018, pp. 122-127.

[10] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng and V. C. M. Leung, "Decentralized Applications: The Blockchain-Empowered Software System," in *IEEE Access*, vol. 6, pp. 53019-53033, 2018.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

[12] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. CoRR, abs/1406.1078, 2014.

[13] Alex Graves. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850, 2013.

[14] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems, pages 3104–3112, 2014.

[15] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.

[17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. “Attention Is All You Need”. *NIPS*, 2017.

[18] T. Phaladisailoed and T. Numnonda, "Machine Learning Models Comparison for Bitcoin Price Prediction," *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Kuta, 2018, pp. 506-511.

[19] X. Dang, H. Peng, X. Wang and H. Zhang "Theil-Sen Estimators in a Multiple Linear Regression Model," Olemiss Edu, 2008.

[20] S. McNally, J. Roche and S. Caton, "Predicting the Price of Bitcoin Using Machine Learning," *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, Cambridge, 2018, pp. 339-343.

[21] A. Greaves and B. Au, “Using the bitcoin transaction graph to predict the price of bitcoin,” 2015.

[22] R. Delfin Vidal, “The fractal nature of bitcoin: Evidence from wavelet power spectra,” *The Fractal Nature of Bitcoin: Evidence from Wavelet Power Spectra (December 4, 2014)*, 2014.

[23] L. Kristoufek, “What are the main drivers of the bitcoin price? Evidence from wavelet coherence analysis,” *PloS one*, vol. 10, no. 4, p. e0123923, 2015.

[24] B. Catanzaro, N. Sundaram, and K. Keutzer, “Fast support vector machine training and classification on graphics processors,” in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 104–111.

[25] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep, big, simple neural nets for handwritten digit recognition,” *Neural computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[26] P. V. Rane and S. N. Dhage, "Systematic Erudition of Bitcoin Price Prediction using Machine Learning Techniques," *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS),* Coimbatore, India, 2019, pp. 594-598.

[27] Aaron Visschedijk, “Trading Bitcoin Using Artificial Neural Networks”, Thesis, Radboud University, Netherlands, June 2018.

[28] Edwin Sin, Lipo Wang, “Bitcoin Price Prediction Using Ensembles of Neural networks”, 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNCFSKD), Nanyang Technological University, Singapore, 2017.

[29] N.I. Indera, I.M. Yassin, A. Zabidi, Z.I. Rizman, “Non-linear Autoregressive with Exogeneous Input (NARX)Bitcoin Price Prediction Model using PSO-Optimized Parameters and Moving average technical indicators”, Thesis, Malaysia, September 2017.

[30] X. Li, L. Yang, F. Xue and H. Zhou, "Time series prediction of stock price using deep belief networks with intrinsic plasticity," *2017 29th Chinese Control and Decision Conference (CCDC),* Chongqing, 2017, pp. 1237-1242.

[31] T. Le, J. Kim and H. Kim, "Classification performance using gated recurrent unit recurrent neural network on energy disaggregation," *2016 International Conference on Machine Learning and Cybernetics (ICMLC),* Jeju, 2016, pp. 105-110.

LIST OF PUBLICATIONS OF THE CANDIDATE’S WORK

[1]. Accepted and Presented a paper titled “Consensus Algorithms in Blockchain Technology: A Survey” in 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT) – 2019 organized by Indian Institute of Technology, Kanpur, India.