TIME SERIES MODELLING OF BITCOINS

A Project Report Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF SCIENCE

in

Mathematics and Computing

by

Mohit Sharma

(Roll No. 212123031)

to the

DEPARTMENT OF MATHEMATICS
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, INDIA

April 2023

CERTIFICATE

This is to certify that the work contained in this report entitled **"TIME SERIES MODELLING OF BITCOINS"** submitted by **Mohit Sharma** (**Roll No: 212123031**) to Department of Mathematics, Indian Institute of Technology Guwahati towards the requirement of the course **MA699 Project** has been carried out by him under my supervision.

Guwahati - 781 039
April 2023

(Dr. Arabin Kumar Dey)
Project Supervisor

ABSTRACT

The main aim of this project is to use statistical and machine learning techniques to analyze cryptocurrency prices, with a particular focus on Bitcoin prices and its daily returns. The project will explore and apply various machine learning algorithms for Bitcoin price prediction, including Recurrent Neural Networks (RNNs) such as LSTM, GRU, hybrid models using both GRU and LSTM, Highway networks, transformer networks, and their combined ensembled model (COBRA). Additionally, various aspects and technicalities of cryptocurrencies will be discussed in detail. Finally, the project will present the results of all the models on real-world Bitcoin price data.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 What is Bitcoin

Bitcoin is a digital currency that operates on a decentralized network, meaning there is no central authority or single point of trust controlling the system. This decentralized nature is made possible through the use of blockchain technology, a distributed ledger that records all Bitcoin transactions.

In traditional currencies, the issuance and transaction processes are overseen by a centralized authority, such as a government or central bank. However, with Bitcoin, both the creation of new coins and the processing of transactions are based on cryptographic principles, rather than the authority of a single entity. This means that Bitcoin transactions are verified and recorded by a network of nodes or computers, rather than a central authority.

Overall, the decentralized nature of Bitcoin and its use of cryptographic principles for transactions and coin generation provide increased security and transparency, making it a popular choice for those seeking a more secure and transparent form of currency.

1.2 Applications of Bitcoin

1. Transfer funds while maintaining anonymity.
2. To send funds abroad.
3. Function as a valuable resource.

1.3 Volatility and Bitcoins

Bitcoin's value is known to fluctuate significantly, and this volatility is often attributed to the hype surrounding it. This characteristic of bitcoin is considered one of its major drawbacks, as it goes against the primary objective of a currency, which is to provide stability to the value of goods.

Bitcoin's volatility makes it difficult for people to determine the purchasing power of their money, and it is often viewed as a speculative asset rather than a reliable currency. As a result, many people perceive bitcoin as a risky investment, and its value is often considered to be a matter of speculation rather than a reflection of its true worth.

Chapter 2

Description of Dataset

SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
1	Bitcoin	BTC	2013-12-27 23:59:59	777.5100098	713.5999756	763.2800293	735.0700073	46862700	8955394564
2	Bitcoin	BTC	2013-12-28 23:59:59	747.0599976	705.3499756	737.9799805	727.8300171	32505800	8869918644
3	Bitcoin	BTC	2013-12-29 23:59:59	748.6099854	714.4400024	728.0499878	745.0499878	19011300	9082103621
4	Bitcoin	BTC	2013-12-30 23:59:59	766.5999756	740.2399902	741.3499756	756.1300049	20707700	9217167990
5	Bitcoin	BTC	2013-12-31 23:59:59	760.5800171	738.1699829	760.3200073	754.0100098	20897300	9191325349
6	Bitcoin	BTC	2014-01-01 23:59:59	775.3499756	754.9699707	754.9699707	771.4000244	22489400	9403308145
7	Bitcoin	BTC	2014-01-02 23:59:59	820.3099976	767.210022	773.4400024	802.3900146	38489500	9781073921
8	Bitcoin	BTC	2014-01-03 23:59:59	834.1500244	789.1199951	802.8499756	818.7199707	37810100	9980135396
9	Bitcoin	BTC	2014-01-04 23:59:59	859.5100098	801.6699829	823.2700195	859.5100098	38005000	10477362437
10	Bitcoin	BTC	2014-01-05 23:59:59	952.4000244	854.5200195	858.5499878	933.5300293	72898496	11379660685
11	Bitcoin	BTC	2014-01-06 23:59:59	1017.119995	905.710022	936.0499878	953.289978	85565696	11620533603
12	Bitcoin	BTC	2014-01-07 23:59:59	965.7399902	802	946.4899902	802	81311696	9808299600

Figure 2.1: Dataset Features

- Name: This feature represents the name of the cryptocurrency, which in this case is Bitcoin.
- Symbol: This feature represents the symbol used to represent Bitcoin in trading platforms, which is usually BTC.
- Date: This feature represents the date for which the pricing data is recorded. This could be daily, hourly, or even minute-by-minute data, depending on the dataset.
- High: This feature represents the highest price that Bitcoin reached on the date in question. For example, if the dataset contains daily data, this would be the highest price that Bitcoin reached during that particular day.
- Low: This feature represents the lowest price that Bitcoin reached on the date in question. For example, if the dataset contains daily data, this would be the lowest price that Bitcoin reached during that particular day.
- Open: This feature represents the opening price for Bitcoin on the date in question. For example, if the dataset contains daily data, this would be the price at which Bitcoin started trading at the beginning of the day.
- Close: This feature represents the closing price for Bitcoin on the date in question. For example, if the dataset contains daily data, this would be the price at which Bitcoin stopped trading at the end of the day.

- Volume: This feature represents the volume of Bitcoin traded on the date in question. Volume is the number of Bitcoins that were traded on a particular day.
- Marketcap: This feature represents the market capitalization of Bitcoin on the date in question. Market capitalization is the total value of all Bitcoins in circulation on that day.

Chapter 3

Data set Preprocessing

3.1 Benchmark

We have compared different machine learning models in terms of Signal Accuracy because the bitcoin price data is highly volatile and mostly increasing. The criteria taken for comparision of different models is **mean absolute error** . The results are shown in the result section below.

3.2 Dataset separation

We have used 3 year old data of bitcoins to perform training, validation and testing of different machine learning models. 80% of the total data is used for training, 10% is validation data and the remaining 10% is testing data in each model.

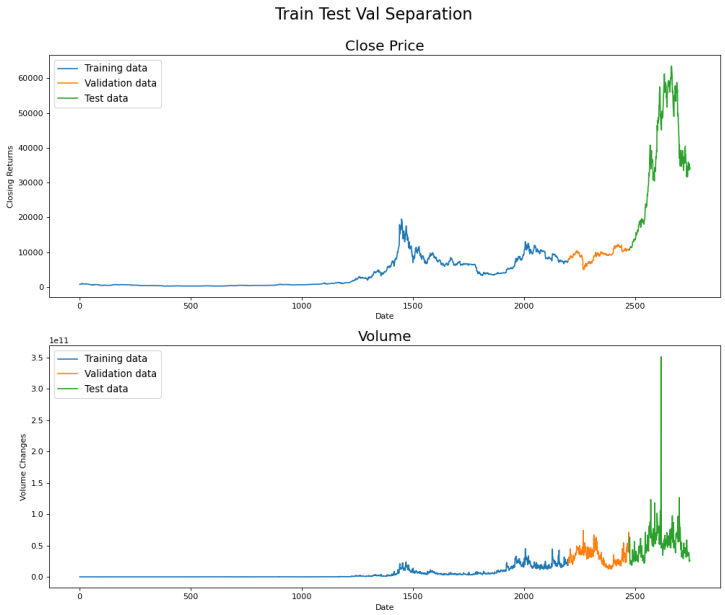


Figure 3.1: Data Separation for closing price and volume

3.3 Parameter Tuning

3.3.1 Optimization Algorithm

Optimization is an iterative process in which the model is trained iteratively and results in a maximum and minimum function evaluation. There are three different optimization algorithms are explored while learning. The algorithms are as follows:

1. Stocastic Gradient Descent
2. Adaptive Gradient Descent
3. Adam

Stocastic Gradient Descent: In Stocastic Gradient Descent, a few n samples are choosen randomly instead of the whole dataset for the iteration. It is an iterative algorithm used to minimize the cost function.

Adaptive Gradient Descent : In Adaptive Gradient Descent, the learning rate is adaptively scaled for each parameter on the basis of previous gradient information. These type of optimization algorithm avoids oscillations.

Adam: In Adam, the learning rate is adaptively scaled for each parameter on the basis of moving average of the gradient and the squared gradient. Like Adaptive Gradient Descent, Adam also avoids the oscillations.

Adaptive Gradient Descent and Adam provides dynamic weights, whereas Stocastic Gradient provides static weights.

Among all the three algorithms, we get the best fitting in case of mean absolute error in the presence of ADAM optimizer. All the following results have been generated by using the ADAM while learning the model.

3.3.2 Lookback Number

Lookback number is the number of the dataset point that our model is going to consider at a time to predict the future. We have observed that MAE decreases with the increase in lookback number till some certain point and after it MAE(mean absolute error) starts increasing with the increase in the lookback number.
The optimal lookback number at which MAE is the least as compared to others is around 20. So, here we have observed the global minimum in MAE.

3.3.3 Dropout

Dropout is a regularization method in the neural network. It prevents the over fitting of the model. The parameter dropout is the fraction of total units which will be randomly selected and turned off.

3.4 Window Feature Scaling

Feature scaling is an essential element in machine learning for model training, and one of the most frequently used techniques is data normalization. This process scales the data feature-wise to fit values within the range of 0 to 1, with min-max scaling being a widely preferred approach. Although various normalization techniques exist, this report uses a specific normalization formula as follows:

$$norm = \frac{X - X_{min}}{X_{max} - X_{min}}$$

The aim of this study is to examine how feature scaling affects the performance of machine learning models. When using a model with a fixed lookback, such as an LSTM, normalizing the data within each window of the lookback length is more advantageous. This involves scaling each training sequence to values between 0 and 1, and then converting the model's predictions back to their original scale. Scaling data in windows improves model training by allowing it to capture the relative variance in the data within a small neighborhood. This approach accelerates the learning process by enabling the model to focus on capturing the relative order of inputs instead of a universal absolute order. As a result, the model can better capture complex or highly volatile time series data.

Chapter 4

Description of weak learners

4.1 SIMPLE RNN:

4.1.1 Introduction to SIMPLE RNN:

Simple RNN (Recurrent Neural Network) is a type of neural network that is capable of processing sequential data by maintaining a hidden state that captures the context of the sequence processed so far.

In a simple RNN, the output at each time step is determined by combining the current input and the previous hidden state using a set of weights. The weights are learned during training using backpropagation through time, which allows the network to learn representations of sequential data that can be used for tasks such as sequence prediction, classification, and generation.

4.1.2 Architechture of SIMPLE RNN:

The architecture of the above model consists of a single layer of Simple Recurrent Neural Network (RNN) and a Dense output layer with 2 units. The input data is expected to have a shape of (Xtrain.shape[1], Xtrain.shape[2]), which means that the input data is a 3D tensor with dimensions (batch size, time steps, input dim).

The Simple RNN layer has 100 recurrent units and uses the ReLU activation function. It does not return the full sequence, only the last output. The Dense layer serves as the output layer and has 2 units. Overall, it is suitable for sequence prediction tasks where the input data is a time series, and the output is a single value or a small set of values.

4.1.3 Performance of SIMPLE RNN:

Here, we have made prediction on the training data, validation data and testing data, firstly with the original closing Series and secondly with the original volume series in each of the three dataset.
Validation loss: 0.08139
MAE(Mean Absolute Error): 0.1808

Closing Price

Predictions on the training data :

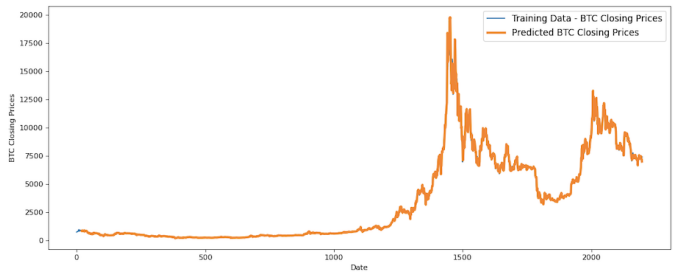


Figure 4.1: RNN prediction closing price training data

Predictions on the validation data:

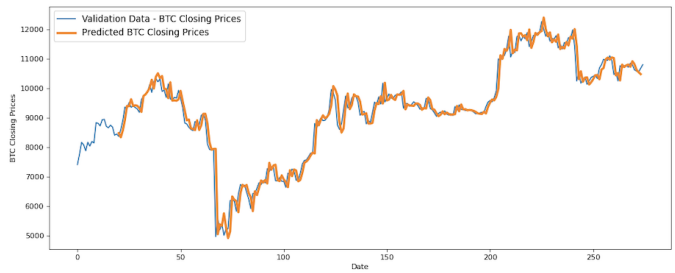


Figure 4.2: RNN prediction closing price validation data

Predictions on the testing data:

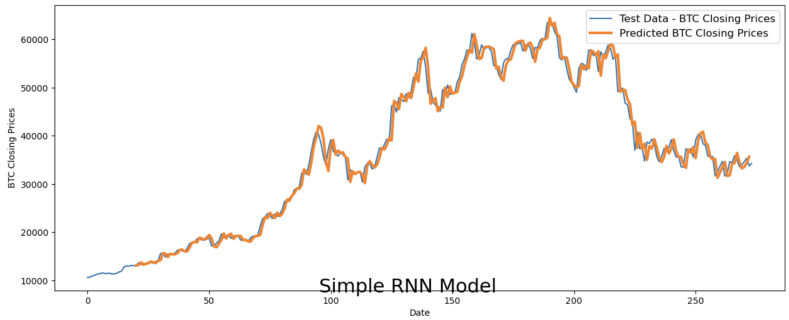


Figure 4.3: RNN prediction closing price testing data

Volumes

Predictions on the training data :

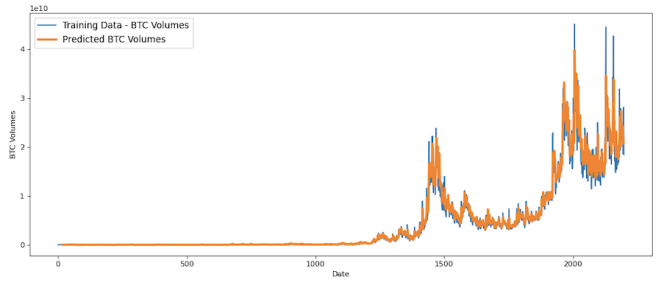


Figure 4.4: RNN prediction volume training data

Predictions on the validation data:

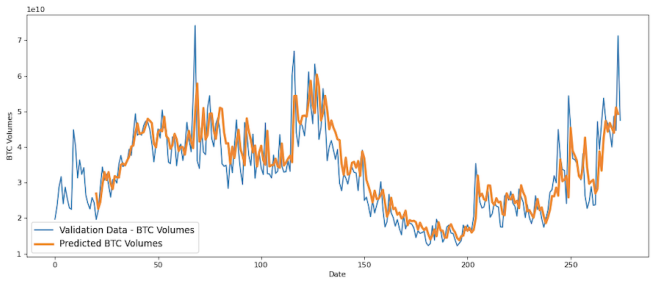


Figure 4.5: RNN prediction volume validation data

Predictions on the testing data:

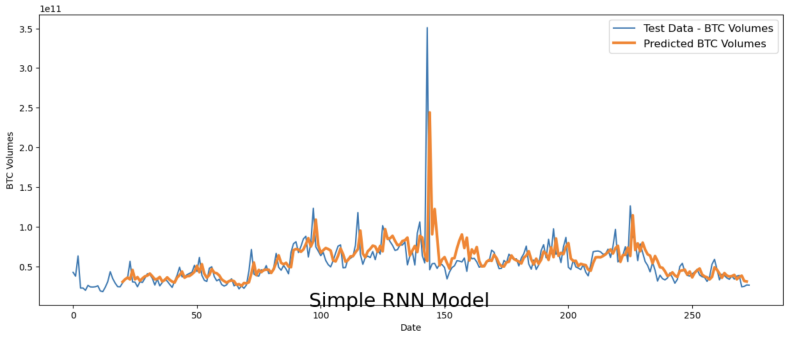


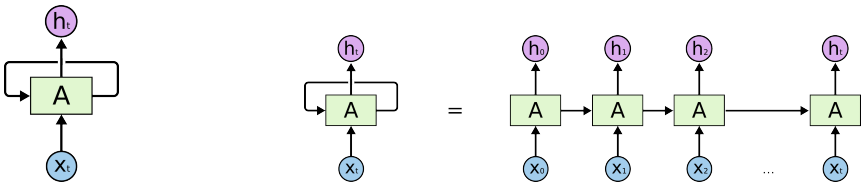
Figure 4.6: RNN prediction volume testing data

Simple RNNs suffer from the vanishing gradient problem, which limits their ability to capture long-term dependencies in sequences. Hence, we will study the following models.

4.2 LSTM:

4.2.1 Introduction to LSTM

Long Short-Term memory network generally referred as LSTM comes under Recurrent Neural Network which are capable of learning long-term dependencies. LSTM are designed in such a way that they solve the long-term dependency problem occurred in RNN. LSTM networks remember the information for long period of time.



(a) Single unit (b) Unrolled unit

LSTM have a chain of repeating modules but instead of a single neural network layer in each repeating module, there are four interacting layers. It consists of three gates namely:

1. Forget gate
2. Input gate
3. Output gate

It works on two states:

1. Hidden state
2. Cell state

In the diagram shown below, each line represents that an entire vector is carried from the output of one node to the input of the other node. The boxes represent the learned neural network layers. The circle represents the pointwise vector operations such as vector addition. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.

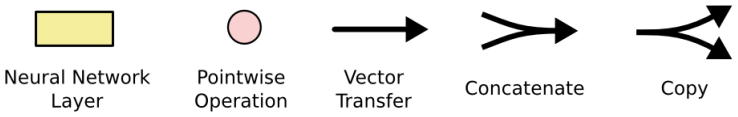


Figure 4.8: Some useful notations

Core idea about LSTM:

The important point about LSTM's is the cell state. Information can be added or removed from the cell state and is regulated by gates. Gates passes the information optionally through them. They consist of a sigmoid net layer and a pointwise

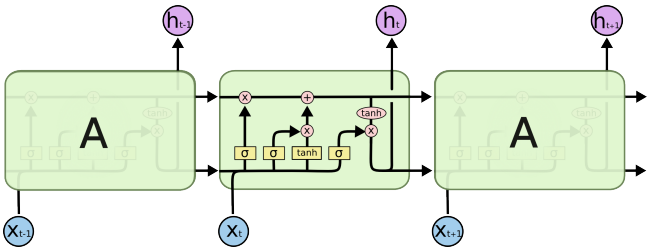


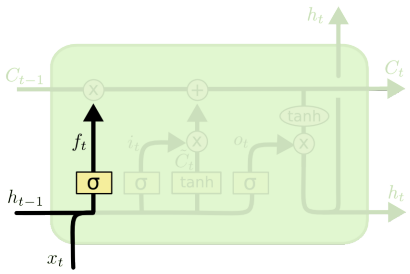
Figure 4.9: LSTM network

multiplication operation.

The sigmoid layer gives the output as 0 or 1, where 0 signifies that no information is passed through the gate whereas 1 means that whole of the information is passed through the gate. As described above, there are three gates of LSTM which control the flow of information.

FIRST STEP:

Information which is not required is removed from the cell state which is decided by a sigmoid layer known as 'forget gate layer'. Here consider:
h_{t-1}: Information carried from the last layer from time t-1.
x_t: Information passed at the time t.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 4.10

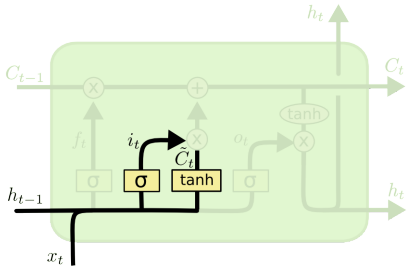
h_{t-1} and x_t vectors are examined by the forget gate layer and consequently, it produces a number between 0 and 1 for each number in the cell state. 1 represents that the information is stored completely whereas 0 implies that information will be removed from the cell state.The theory explained is governed by the equation:
$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$

So, we have multiplied the old state with f_t, which results in the removal of information which is not required.

SECOND STEP:
New Information which is going to be stored in the cell state. It consists of two parts.

1. Sigmoid layer: INPUT GATE LAYER, which is responsible for the updation in the values

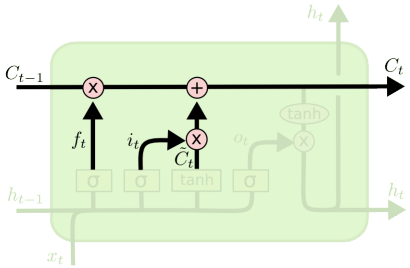
2. Tanh layer, which is responsible for the generation of new candidate values $\tilde{C}_t$ that can be added to the cell state at time t.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 4.11

Hence, as h_{t-1} and x_t vectors passes through sigmoid layer and Tanh layer, so i_t and $\tilde{C}_t$ are produces respectively, where
$i_t \;\; = \;\; \sigma(W_i.[h_{t-1}, x_t] + b_i)$
$\tilde{C}_t \;\; = \;\; \tanh(W_c.[h_{t-1}, x_t] + b_c)$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

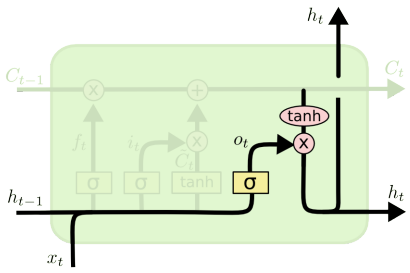
Figure 4.12

Here we will do the pointwise multiplication of the two vectors viz. i_t and $\tilde{C}_t$. As the information which is not required has been already removed from the cell state, so, we will add $i_t * \tilde{C}_t$ to cell state C_{t-1}, which gets updated to the new cell state C_t .

Hence, mathematically we can represent it as :
$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

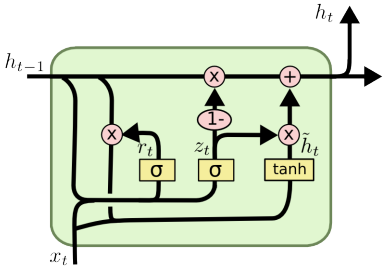
THIRD STEP:
Output is produced at this step which is based on the information present in the cell state. Firstly, cell state is passed through the sigmoid layer which acts as filter and produces a vector o_t. Secondly, cell state is passed through tanh to generate values between -1 and 1. The output vector h_t for time t is produced as the pointwise product of o_t and $\tanh(\tilde{C}_t)$, is given as:
$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$
$h_t = 0_t * \tanh(C_t)$

Hence, we get the final LSTM as follows:



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

Figure 4.13



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

Figure 4.14

4.2.2 Architechture of LSTM:

Sequential neural network model with an LSTM layer and a dense layer, compiles the model, and trains it on the training data with validation data using the Adamax optimizer, mean squared error loss function, and mean absolute error and mean absolute percentage error metrics.

The LSTM layer has 100 units, uses a bias term, applies the ReLU activation function, and takes input sequences with shape (Xtrain.shape[1], Xtrain.shape[2]). The input shape depends on the shape of the input training data, where Xtrain is a 3-dimensional array of shape (samples, time steps, features).

The dense layer has 2 units and has no activation function specified. The model is also checkpointed to save the best performing model based on the validation loss during training. The model is trained for 50 epochs, with a batch size of 32 and verbose set to 1 to display progress during training. The callbacks argument is used to pass the ModelCheckpoint object for saving the best model.

Overall, it is training a baseline LSTM neural network model for some type of prediction problem. The performance of the model can be evaluated using the history object returned by the fit method.

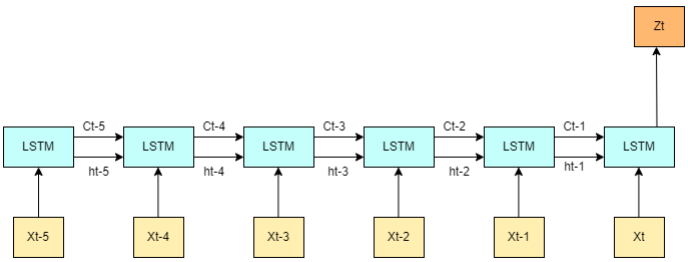


Figure 4.15: **Architecture of LSTM**.

4.2.3 Performance of LSTM

We have implemented LSTM in two different following ways:
Case1: LSTM performance without cummulative sum
Validation loss: 0.08561
MAE: 0.1917

Closing Price

Predictions on the training data :

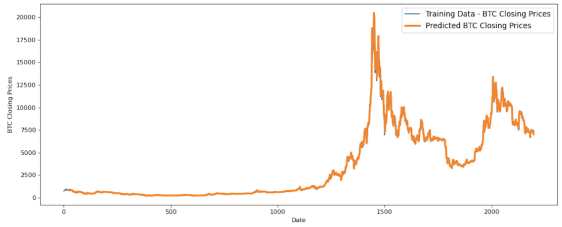


Figure 4.16: LSTM prediction closing price training data

Predictions on the validation data:

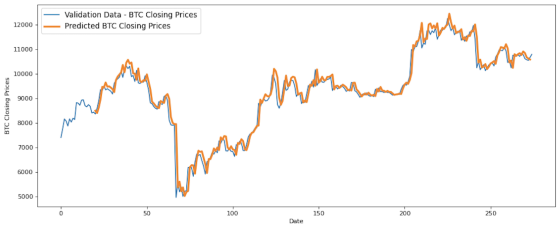


Figure 4.17: LSTM prediction closing price validation data

Predictions on the testing data:

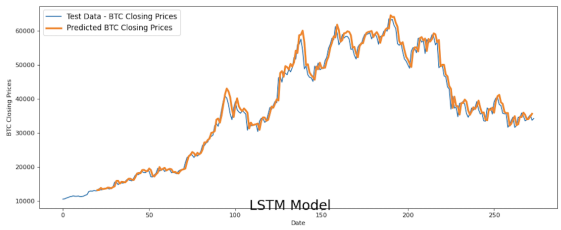


Figure 4.18: LSTM prediction closing price testing data

Volumes

Predictions on the training data :

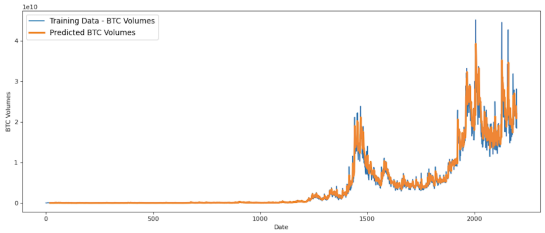


Figure 4.19: LSTM prediction volume training data

Predictions on the validation data:

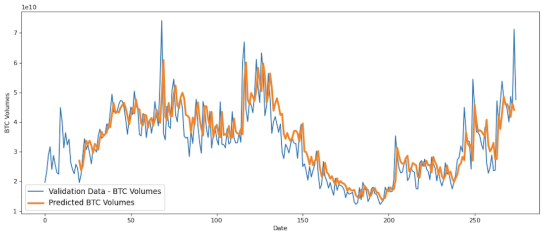


Figure 4.20: LSTM prediction volume validation data

Predictions on the testing data:

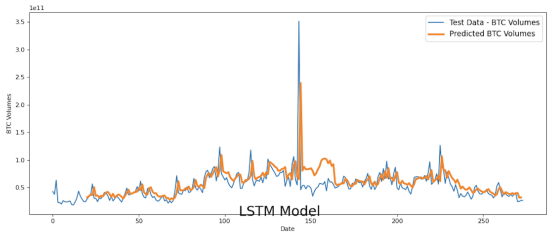


Figure 4.21: LSTM prediction volume testing data

Case2: LSTM performance with cummulative sum
Validation loss: 0.00022
MAE: 0.0565

Closing Price

Predictions on the training data :

Figure 4.22: LSTM prediction closing price training data

Predictions on the validation data:

Figure 4.23: LSTM prediction closing price validation data

Predictions on the testing data:

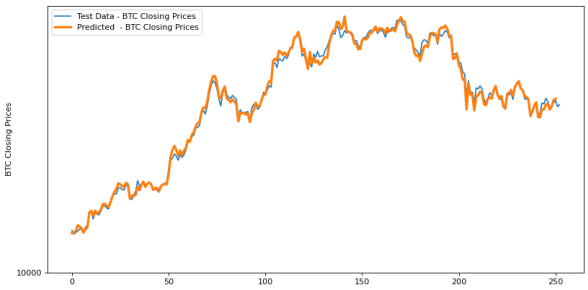


Figure 4.24: LSTM prediction closing price testing data

Volumes

Predictions on the training data :

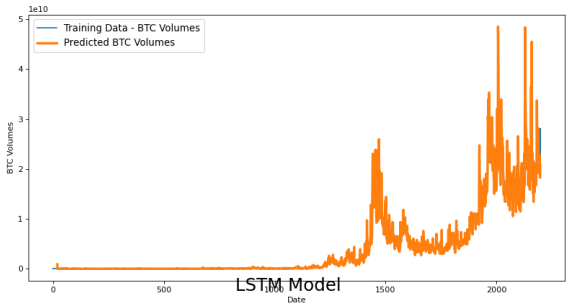


Figure 4.25: LSTM prediction volume training data

Predictions on the validation data:

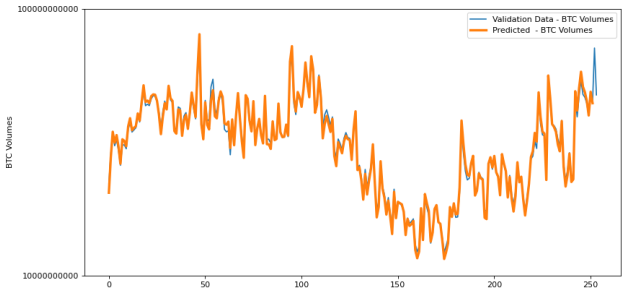


Figure 4.26: LSTM prediction volume validation data

Predictions on the testing data:

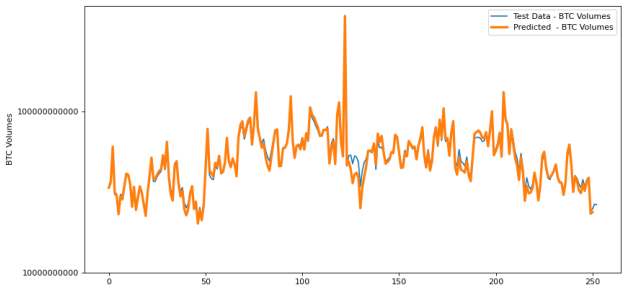


Figure 4.27: LSTM prediction volume testing data

Observations: We got better results in the second case. Hence, performance of all the models will be calculated by taking the cummulative sum of the feature set.

4.3 GRU:

4.3.1 Introduction to GRU:

Gated Recurrent units generally referred as GRU also comes under Recurrent Neural Network. It is used to tackle the problem of vanishing/exploding gradient which occurs in RNN. It only works on hidden state and there is no separate cell state. Like LSTM, GRU also introduce gates to control the amount of information to remember before updating the hidden state.GRU has fewer gates than LSTM. GRU consists of two gates namely:

1. Reset gate
2. Update gate

Core idea about GRU: Similar to LSTM here also we have the following terminologies:
h_{t-1} = hidden state at previous timestep t-1
x_t = input vector at current timestep t
h_t = hidden state at current timestep t

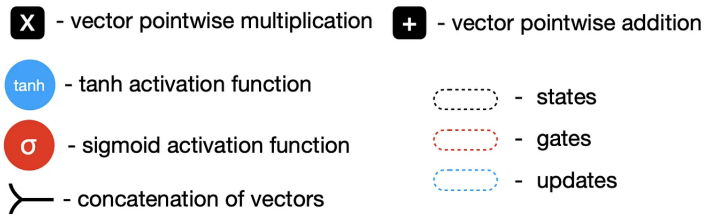


Figure 4.28

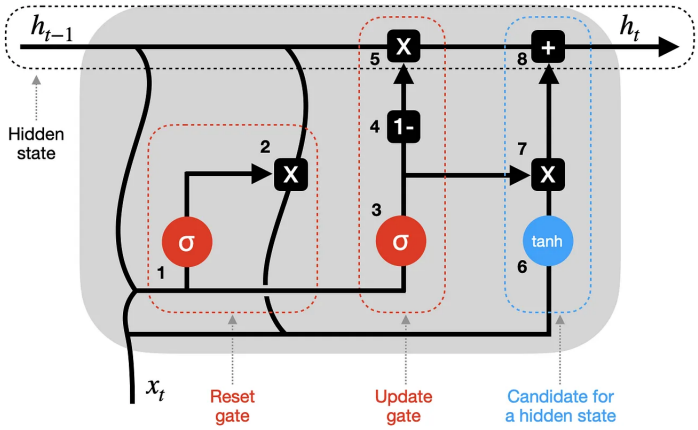


Figure 4.29

FIRST STEP:
The vector h_{t-1} from the previous hidden state and x_t, the current input state, after multiplied by respective weights and bais is added and is passed through a reset gate. So, similarly as sigmoid function produces an output between 0 and

1,where 0 represents the information which is needed to be discarded, 1 represents the information needed to be remembered, otherwise some information is partially retained if it's values lies between 0 and 1.The previous hidden state is reset by multiplying it with output vector we got.

SECOND STEP:

The vector h_{t-1} from the previous hidden state and x_t, the current input state, after multiplied by respective weights and bais is added and is passed through an update gate which uses different weights and baises used to scale these vectors. The output vector then subtracted from a vector containing all 1s. The final output vector is then multiplied by the previous hidden state which eventually updates the hidden state with the new information.

THIRD STEP:

As the previous hidden state is being reset earlier, so the outputs are combined with x_t, i.e the new input at timestep t. Before passing through a tanh activation function, the above two are multiplied by their respective weights and added biases and the whole process leads to the generation of a potential hidden state candidate, which is then multiplied to the output of an update gate and then is added to previously modified hidden state h_{t-1} to generate new hidden state h_t.

Hence, the same process repeats for next iteration for timestep $t + 1$, til it processes the entire sequence.Here, We have the GRU recurrent unit.

4.3.2 Architechture of GRU:

This code defines a sequential neural network model with a GRU layer and a dense layer, compiles the model, and trains it on the training data with validation data using the Adamax optimizer, mean squared error loss function, and mean absolute error and mean absolute percentage error metrics.

The GRU layer has 100 units, uses a bias term, applies the ReLU activation function, and takes input sequences with shape (Xtrain.shape[1], Xtrain.shape[2]). The input shape depends on the shape of the input training data, where Xtrain is a 3-dimensional array of shape (samples, time steps, features).

A dropout layer with a dropout rate of 0.2 is added after the GRU layer to help prevent overfitting. The dense layer has 2 units and has no activation function specified.

The model is also checkpointed to save the best performing model based on the validation loss during training.

The model is trained for 50 epochs, with a batch size of 32 and verbose set to 1 to display progress during training. The callbacks argument is used to pass the ModelCheckpoint object for saving the best model.

Overall, it is training a baseline GRU neural network model for some type of prediction problem. The performance of the model can be evaluated using the history object returned by the fit method.

. ’

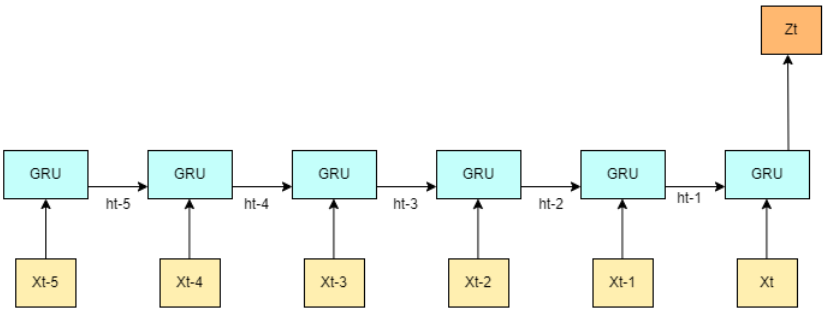


Figure 4.30: **Architecture of GRU**.

4.3.3 Performance of GRU

Here, we have made prediction on the training data, validation data and testing data, firstly with the original closing Series and secondly with the original volume series in each of the three dataset.
Validation loss: 0.00013
MAE: 0.0600

Closing Price

Predictions on the training data :

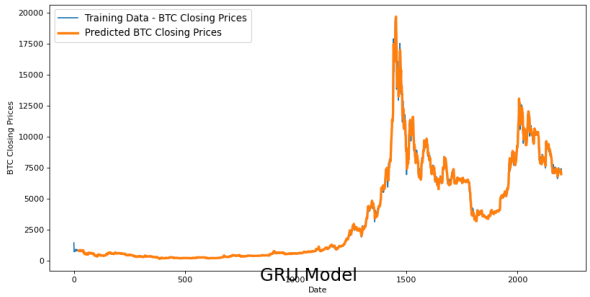


Figure 4.31: GRU prediction closing price training data

Predictions on the validation data:

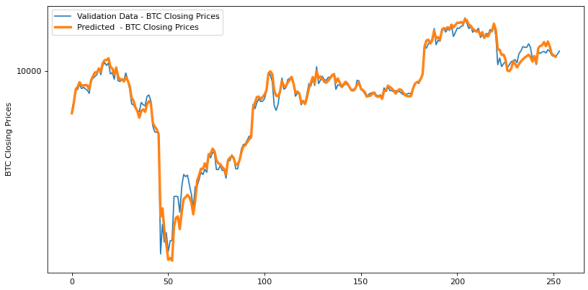


Figure 4.32: GRU prediction closing price validation data

Predictions on the testing data:

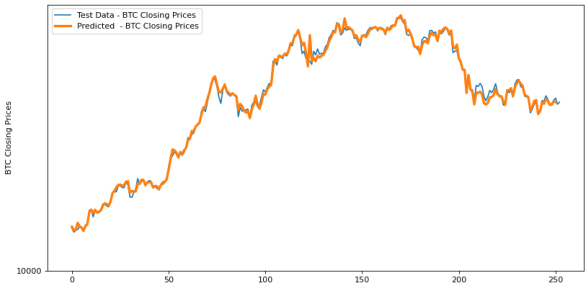


Figure 4.33: GRU prediction closing price testing data

Volumes

Predictions on the training data :

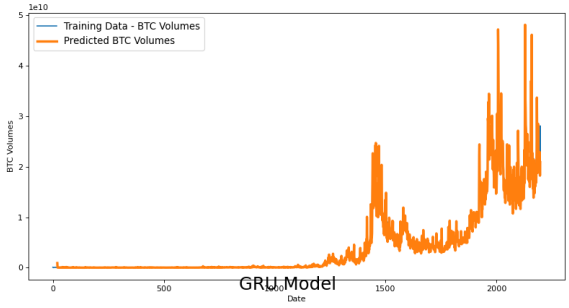


Figure 4.34: GRU prediction volume training data

Predictions on the validation data:

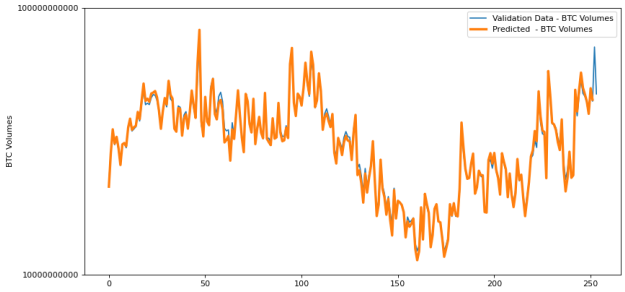


Figure 4.35: GRU prediction volume validation data

Predictions on the testing data:

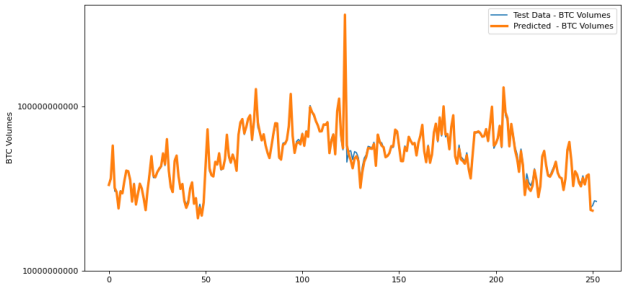


Figure 4.36: GRU prediction volume testing data

4.4 HYBRID:

4.4.1 Introduction to HYBRID

A hybrid model is a type of neural network that combines multiple network architectures to improve performance on a specific task. In this case, the hybrid model is made up of three different network architectures: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Simple Recurrent Neural Network (RNN).
By combining the strengths of these architectures, the hybrid model can potentially achieve better performance than any of the individual network architectures alone. The model is trained using mean squared error loss and other metrics, and is checkpointed to save the best model based on validation loss.

4.4.2 Architechture of HYBRID:

Sequential neural network model with a hybrid architecture that combines LSTM, SimpleRNN, and GRU layers, followed by dense layers, compiles the model, and trains it on the training data with validation data using the Adamax optimizer, mean squared error loss function, and mean absolute error and mean absolute percentage error metrics.

The first layer is an LSTM layer with 100 units, uses a bias term, applies the ReLU activation function, and takes input sequences with shape (Xtrain.shape[1], Xtrain.shape[2]). The return sequences argument is set to True to allow the output sequence to be passed to the next layer.

The second layer is a SimpleRNN layer with 100 units, uses a bias term, applies the ReLU activation function, and takes input sequences with shape (Xtrain.shape[1], Xtrain.shape[2]). The return sequences argument is also set to True to allow the output sequence to be passed to the next layer. A dropout layer with a dropout rate of 0.2 is added after the SimpleRNN layer to help prevent overfitting.

The third layer is a GRU layer with 50 units, uses a bias term, applies the ReLU activation function, and takes input sequences with shape (Xtrain.shape[1], Xtrain.shape[2]). The return sequences argument is set to True to allow the output sequence to be passed to the next layer. Another dropout layer with a dropout rate of 0.2 is added after the GRU layer to help prevent overfitting.

The fourth layer is an LSTM layer with 10 units, uses a bias term, applies the default activation function (tanh), and takes input sequences with shape (None, 50). The output sequence is not returned because return sequences is False by default. Another dropout layer with a dropout rate of 0.5 is added after the LSTM layer to help prevent overfitting.

The final layer is a dense layer with 2 units and has no activation function specified. The model is also checkpointed to save the best performing model based on the validation loss during training.

The model is trained for 50 epochs, with a batch size of 32 and verbose set to 1 to display progress during training. The callbacks argument is used to pass the ModelCheckpoint object for saving the best model.

Overall, It is training a neural network model with a hybrid architecture that combines multiple recurrent neural network layers with dense layers for some type of prediction problem. The performance of the model can be evaluated using the history object returned by the fit method.

4.4.3 Performance of HYBRID

Here, we have made prediction on the training data, validation data and testing data, firstly with the original closing Series and secondly with the original volume series in each of the three dataset.
Validation loss: 0.00036
MAE: 0.0356

Closing Price

Predictions on the training data :

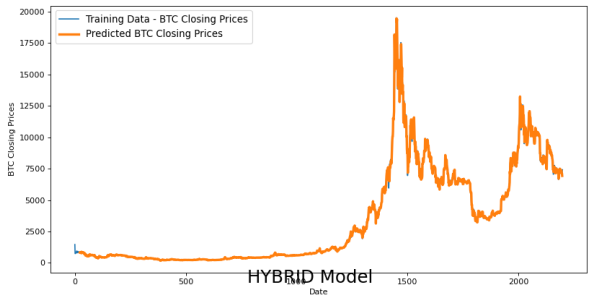


Figure 4.37: Hybrid prediction closing price training data

Predictions on the validation data:

Figure 4.38: Hybrid prediction closing price validation data

Predictions on the testing data:

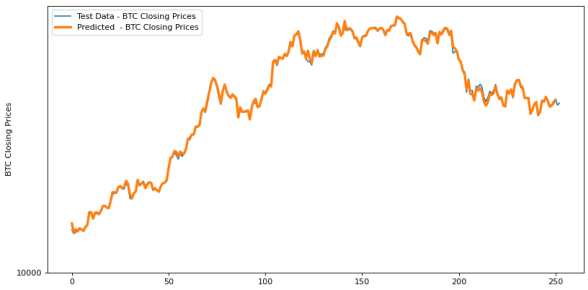


Figure 4.39: Hybrid prediction closing price testing data

Volumes

Predictions on the training data :

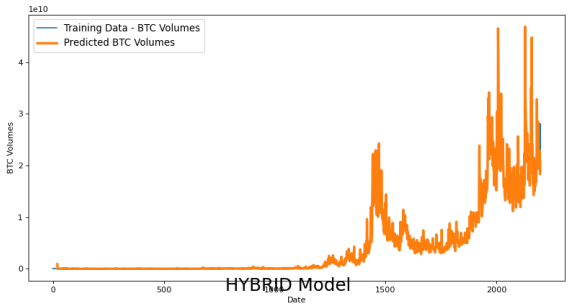


Figure 4.40: GRU prediction volume training data

Predictions on the validation data:

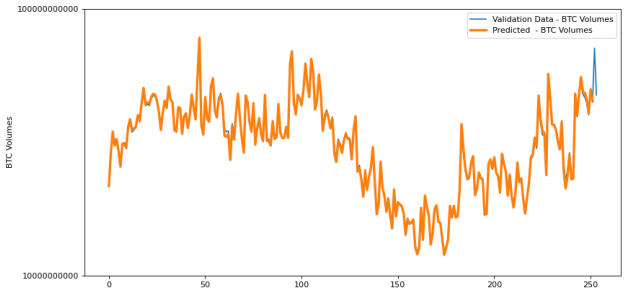


Figure 4.41: GRU prediction volume validation data

Predictions on the testing data:

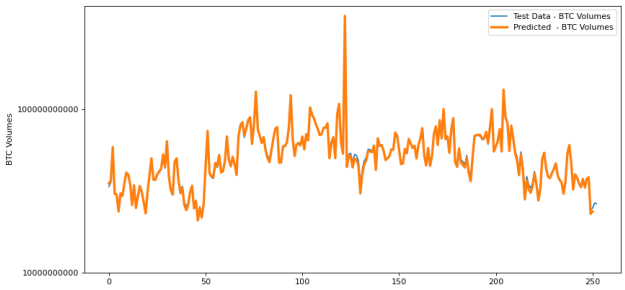


Figure 4.42: GRU prediction volume testing data

4.5 HIGHWAY LSTM:

4.5.1 Introduction to HIGHWAY LSTM

It is evident that deeper neural networks increases the accuracy, however training the deeper networks is not that straighforward. It is considerably difficult to optimize such neural networks. This can be accomplished through the use of a learning gating mechanism to regulate the flow of information as we have already seen in LSTM. As a consequence of this gating mechanism, a neural network has different paths through which information can flow. These paths are called 'Information highways' and such networks are called 'Highway Networks'.

Core idea of Highway LSTM

Let us consider a plain feedforward neural network which consists of L layers where the $l-th$ layer applies a non-linear transform H on the input, say x and generates the output, say y. It is governed by the mathematical equation if we omit biases for the sake of clarity:
$y = H(x, W_H)$
Hence, for the $i-th$ unit:
$y_i = H_i(x)$
After computing y_i it is then passes to the next layer.

In Highway network, there are two non-linear tranforms namely T and C where T is the **Transform Gate** and C is the **Carry Gate**, such that it alters the output as follows:
$y = H(x, W_H).T(x, W_T) + x.C(x, W_C)$
It reflects that how much of the output is produced by transforming and carrying the input, respectively.

As we have already seen that LSTM updates the hidden layer at each timestep t from the hidden layer h_{t-1}, where the hidden layer h_{t-1} is not saved anywhere. In order to improve LSTM, Highway Network adds a forget gate in feedward network to save the previous hidden h_{t-1}.

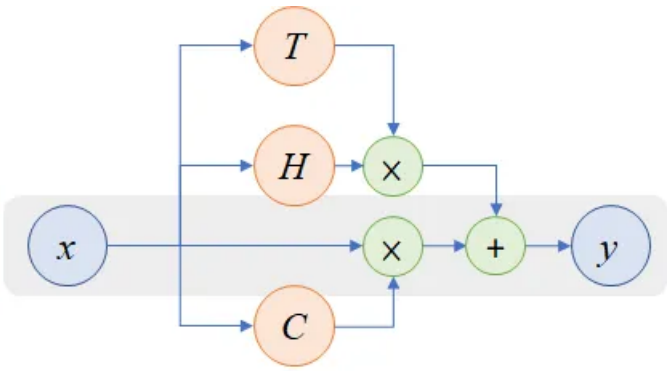


Figure 4.43

It is defined as:
$g_T = \sigma(W_T x + b_T)$
$g_C = \sigma(W_C x + b_C)$

$y = x \odot g_C + tanh(Wx + b) \odot g_T$

Here $\tanh(Wx + b)$ is a feedward network, x is the input.

4.5.2 Architechture of HIGHWAY LSTM:

It is the neural network model that combines a Long Short-Term Memory (LSTM) layer with a Highway Block layer. The LSTM layer has 100 units, uses the ReLU activation function, and returns only the last output of the sequence (return sequences=False). A dropout layer with a rate of 0.5 is added after the LSTM layer to reduce overfitting.

The Highway Block layer has 100 units and uses the ReLU activation function for the "h" (or activation) component and the sigmoid activation function for the "t" (or transformation) component. It is designed to allow the model to "learn when to carry forward information and when to transform it" by adding gating mechanisms to the network.

After the Highway Block layer, another dropout layer with a rate of 0.2 is added, and a dense layer with 2 units is added to output the predictions. The model is trained using the Adamax optimizer with mean squared error loss and mean absolute error and mean absolute percentage error metrics. It is trained for 50 epochs with a batch size of 32 and is checkpointed to save the best model based on validation loss.

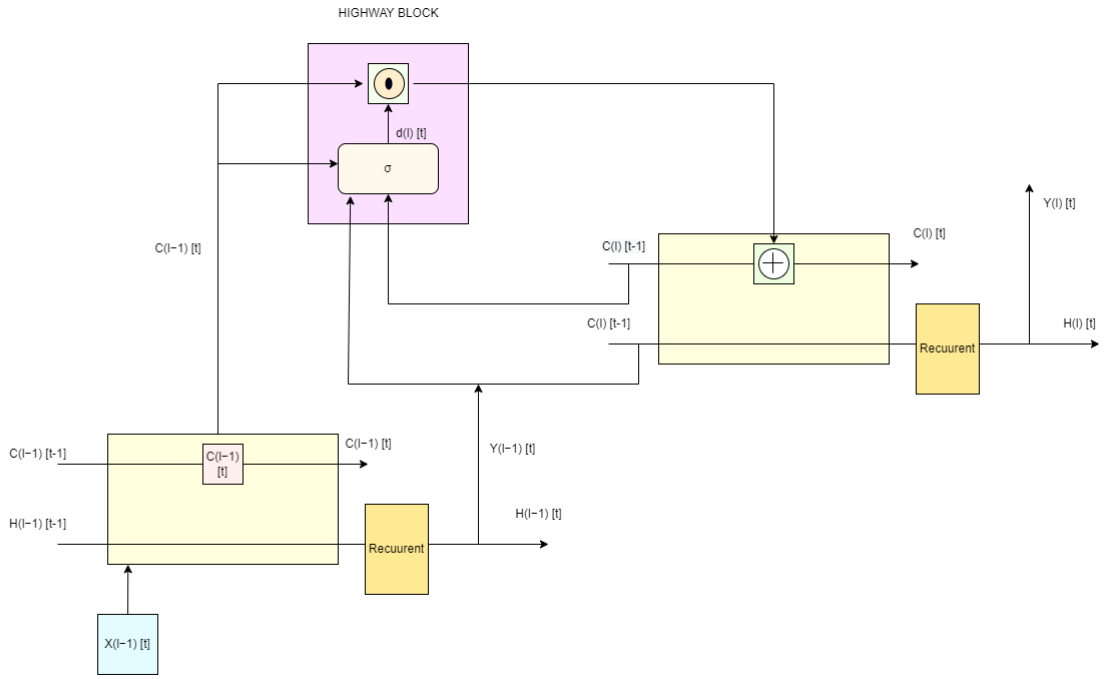


Figure 4.44: **Architecture of HIGHWAY LSTM**

4.5.3 Performance of HIGHWAY LSTM

Here, we have made prediction on the training data, validation data and testing data, firstly with the original closing Series and secondly with the original volume series in each of the three dataset.
Validation loss: 0.00010
MAE: 0.0451

Closing Price

Predictions on the training data :

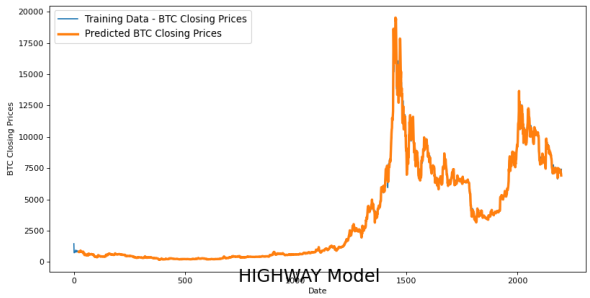


Figure 4.45: Highway LSTM prediction closing price training data

Predictions on the validation data:

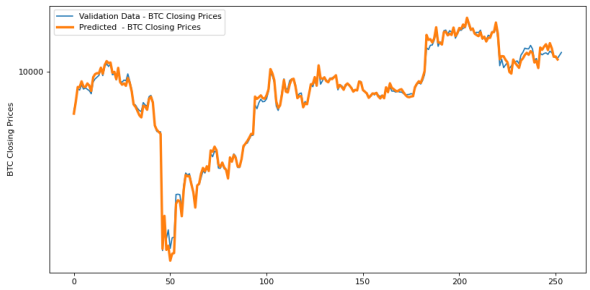


Figure 4.46: Highway LSTM prediction closing price validation data

Predictions on the testing data:

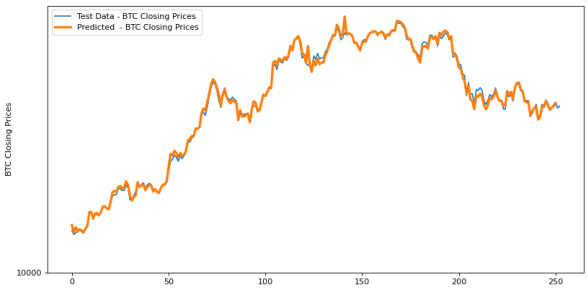


Figure 4.47: Highway LSTM prediction closing price testing data

Volumes

Predictions on the training data :

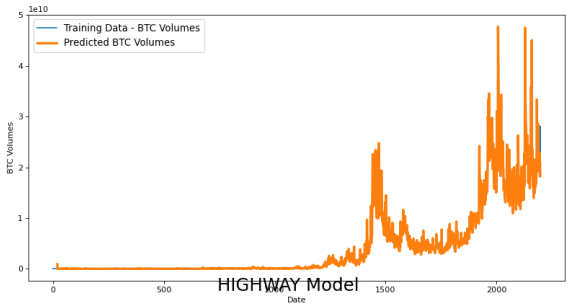


Figure 4.48: Highway LSTM prediction volume training data

Predictions on the validation data:

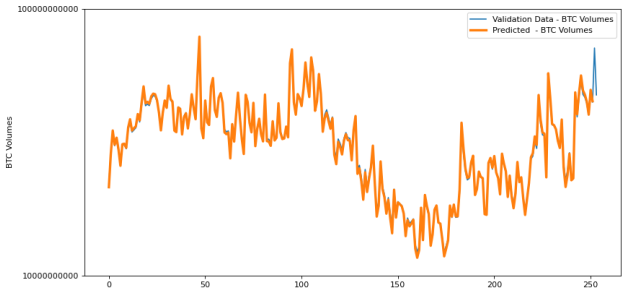


Figure 4.49: Highway LSTM prediction volume validation data

Predictions on the testing data:

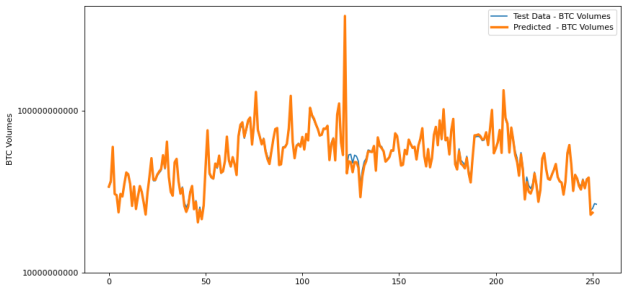


Figure 4.50: Highway LSTM prediction volume testing data

4.6 TRANSFORMER:

4.6.1 Introduction to TRANSFORMER

Transformers are the current state type model used to deal with sequences. There are no recurrent connections and hence there is no real memory of the previous states. And therefore, the model doesn't have to process element by element to develop a hidden cell state. Model overcome this lack by perceiving entire sequences simultaneously.
Transformers require more memory during training.

Core Idea of Transformer:

1) **Attention Mechanism in Transformer Architecture**:

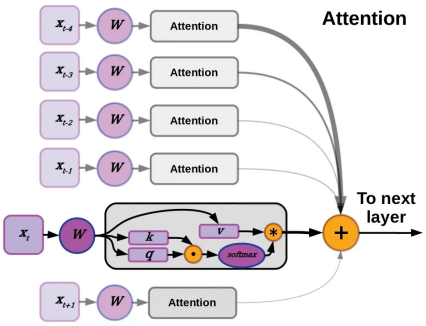


Figure 4.51

Attention basically represents the mean of selectively weighting different elemnets in the input data. This type of selection creates an impact on the hidden layers in the downstream layers. It implements attention by parsing the input into key, query and value vectors, where we get the **attention weights** from the dot product of **key** and **query** for a given position. The attention weights then passed through the softmax function so that all the attention weights sum to 1. The value vectors corresponding to each element are summed according to their attention weights calculated as above and then it passed to the next layers.

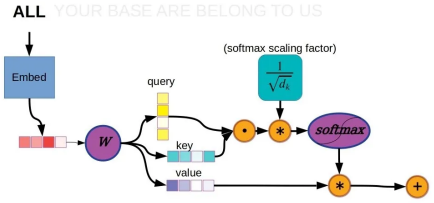


Figure 4.52

As shown the diagram, It can be easily seen that the attention layer W produces three output vectors based on the input, which are termed as above i.e. key, query.

As the whole sequence is passed to the model simultaneously, so attention mechanism is applied parallely at every element of the sequence of vectors. Hence, every element has their attention scores at the same time. As described above, these scores are scaled by the square root of the number of units in the key vector and then passed through the softmax function to scale down the attention weights between 0 and 1, such that sum of all attention weights sum to 1 and then multiplied by their corresponding value vectors, explaining the weightage of each vector in the original sequence. Then all these are summed up to get the resulting vector. Then this resulting vector is passed through the feed forward fully connected layer.

Encoder layer of transformer:
The layer described above is called the self attention layer. When it is combined with a dense feed-forward layer, we get one encoder layer. The feed-forward layer consists of the two linear layers with a rectified linear unit (ReLU) in between . firstly, the input is passed through the first linear layer and transformed, the resulting values are then kept either be 0 or always greater than 0, then finally the resultant values are then passed through the second linear layer to generate the final output for feed-forward layer. Some transformers use several encoder layers each consists of one self attention layer and other feed-forward layer.

Encoder layer of transformer:
Decoder layer consists of three layers, viz. Self-attention layer, Encoder-Decoder attention layer and feed-forward layer.

4.6.2 Architechture of TRANSFORMER:

Here, we define a function createmodel() that returns a neural network model for time series forecasting.

The model is based on the Transformer architecture and takes as input a sequence of two-dimensional data points with a specified length seqlen.

The input sequence is first passed through a Time2Vector layer that encodes the time information as a vector. The encoded time vector is then concatenated with the original input sequence and fed into three TransformerEncoder layers, each consisting of multi-head attention and feedforward neural networks.

The resulting output is then averaged along the time dimension using GlobalAveragePooling1D and passed through two dense layers with ReLU activation and dropout. The final output layer is a dense layer with linear activation that predicts the target values for the time series.

The model is compiled with mean squared error (MSE) loss, Adam optimizer, and mean absolute error (MAE) and mean absolute percentage error (MAPE) metrics.

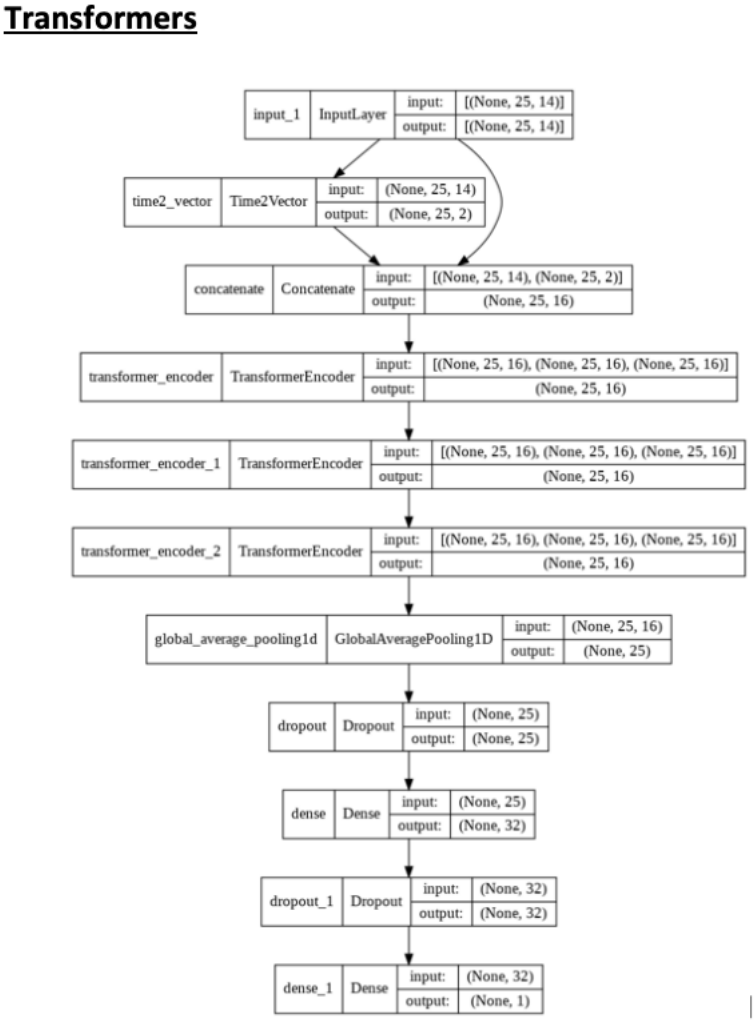


Figure 4.53

4.6.3 Performance of TRANSFORMER

Here, we have made prediction on the training data, validation data and testing data, firstly with the original closing Series and secondly with the original volume series in each of the three dataset.
Validation loss: 0.00007
MAE: 0.0542

Closing Price

Predictions on the training data :

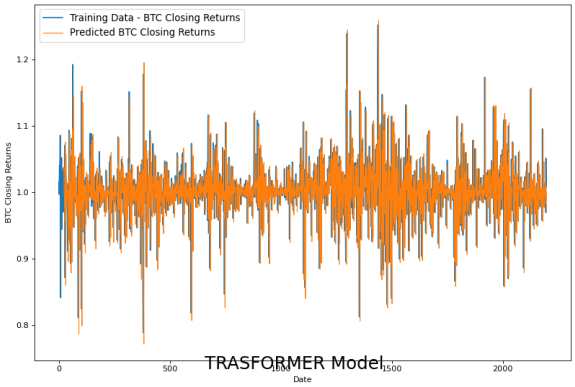


Figure 4.54: Transformer prediction closing returns training data

Predictions on the validation data:

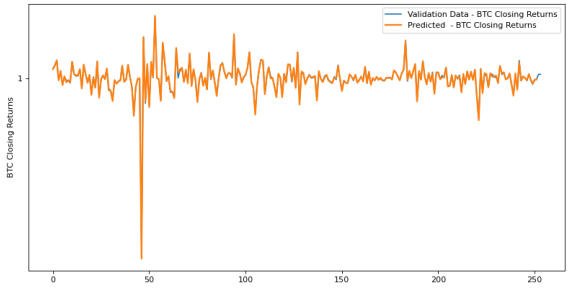


Figure 4.55: Transformer prediction closing returns validation data

Predictions on the testing data:

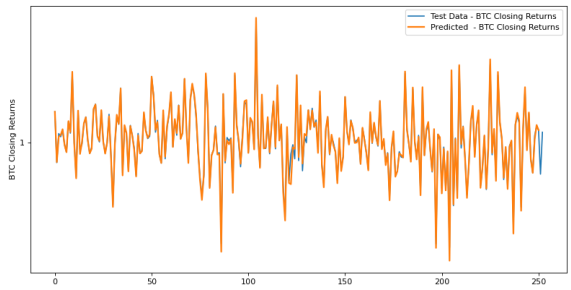


Figure 4.56: Transformer prediction closing returns testing data

Volumes

Predictions on the training data :

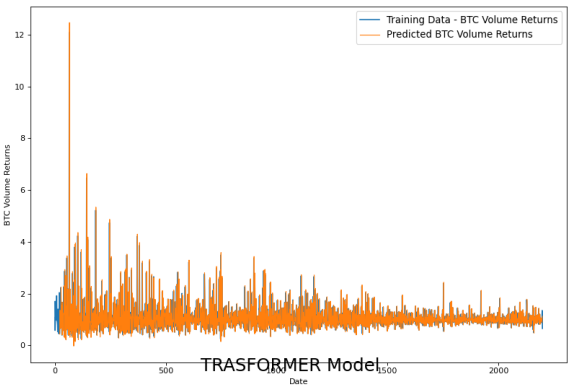


Figure 4.57: Transformer prediction volume returns training data

Predictions on the validation data:

Figure 4.58: Transformer prediction volume returns validation data

Predictions on the testing data:

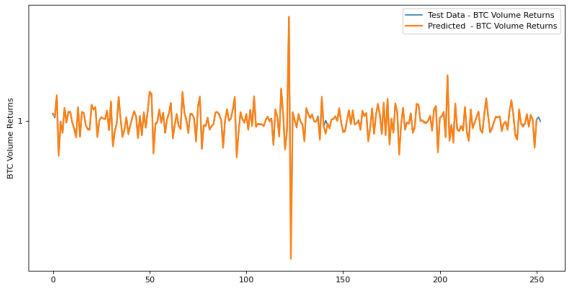


Figure 4.59: Transformer prediction volume returns testing data

Chapter 5

Description of COBRA technique

We will be starting with an original data set D_n. It will be divided into two data sequences, D_k (starting k points) and D_l (remaining $n-k$ points).

We have a set of M models which takes in a certain input, and gives an estimate r*. These basic estimators – basic machines – are assumed to be generated using only the first subsample (D_k). Hence, each model predicts provides an estimation of r*(x) on the basis ofD_k alone. Cobra provides a method to combine all the results of the model, and hence obtain a better estimate using the given models.

The goal of COBRA is to predict response for a new input point x. Using the threshold level as ϵ, we would create a ϵ-ball around the point (x_i,y_i), such that $|r_m(x_i) - r_m(y_i)| \leq \epsilon$. For the corresponding y_i's, we would be taking the average (or some other function) for the prediction of x.

Let us consider the weighted average of the corresponding values of y_i. We define the collective estimator T_n to be:

$$T_n(r_k(x)) = \sum_{i=1}^{l} W_{n,i}(x) Y_i$$

where the random weights $W_{n,i}(x)$ take the form

$$W_{n,i} = \frac{1_{\cap_{m=1}^{M} |r_{k,m}(X_i) - r_{k,m}(X_i)| \leq \epsilon_l}}{\sum_{j=1}^{l} 1_{\cap_{m=1}^{M} |r_{k,m}(X_i) - r_{k,m}(X_i)| \leq \epsilon_l}}$$

where ϵ_1 has been chosen appropriately using a certain algorithm. The restriction of the above ϵ_1-ball can be relaxed by imposing a fixed fraction $\alpha \in \frac{1}{M}, \frac{2}{M}, ..., 1$ such that:

$$W_{n,i} = \frac{1_{\cap_{m=1}^{M} |r_{k,m}(X_i) - r_{k,m}(X_i)| \leq \epsilon_l}}{\sum_{j=1}^{l} 1_{\cap_{m=1}^{M} |r_{k,m}(X_i) - r_{k,m}(X_i)| \leq \epsilon_l}}$$

Asymptomatically speaking ($\alpha \to 1$), the behaviour remains the same.
The Weak learners defined above will be used in the COBRA.

Architecture of COBRA:

We are using the object to perform an ensemble of pre-trained deep learning models. We are calling the method get models predictions() on our ensemble object to obtain the predictions of each individual model on some data Xtrain. We are then computing the difference between consecutive predictions of each model using the Extracount method of our ensemble object.

We are then performing a grid search over a range of values for epsilon and alpha to find the best hyperparameters for our ensemble. We are using the MyCobra class to define and train the ensemble.

During each iteration of the grid search, we are fitting the ensemble to the training data Xtrain and ytrain, and using the trained ensemble to make predictions on a validation set Xval. We are then computing the mean squared error (MAE) between the predicted and true labels for the validation set. We are updating the values of best mae, best epsilon, and best alpha if the current MAE is better than the previous best.

Overall, we are trying to optimize the hyperparameters of our ensemble to minimize the MAE on a validation set.

Performance of COBRA:
Here, We have made prediction on the training data, validation data and testing data, firstly with the Original closing Series and secondly with the original volume series in each of the three dataset.
Validation loss: 0.00005
MAE: 0.0453

Closing Price
Predictions on the training data :

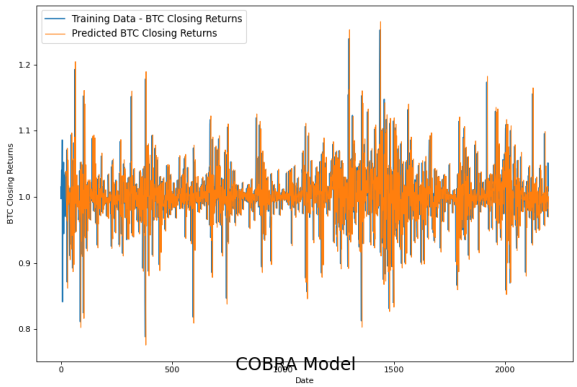


Figure 5.1: COBRA prediction closing returns training data

Predictions on the validation data:

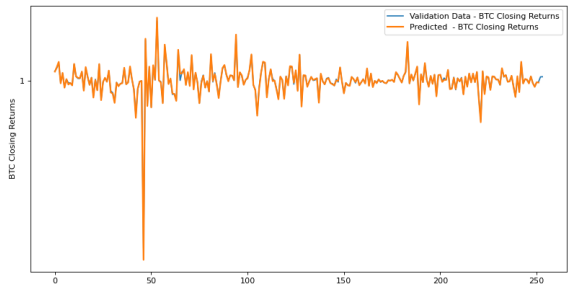


Figure 5.2: COBRA prediction closing returns validation data

Predictions on the testing data:

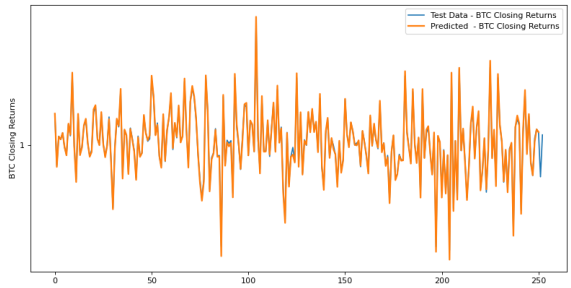


Figure 5.3: COBRA prediction closing returns testing data

Volumes

Predictions on the training data :

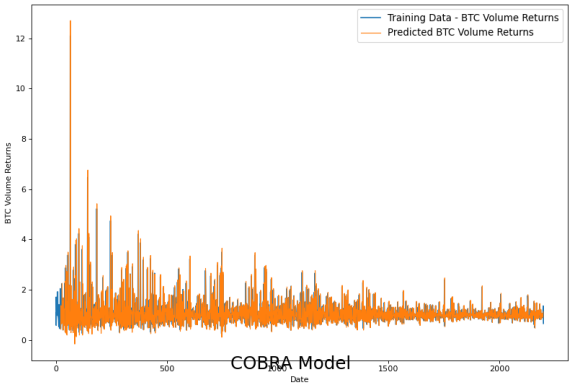


Figure 5.4: COBRA prediction volume returns training data

Predictions on the validation data:

Figure 5.5: COBRA prediction volume returns validation data

Predictions on the testing data:

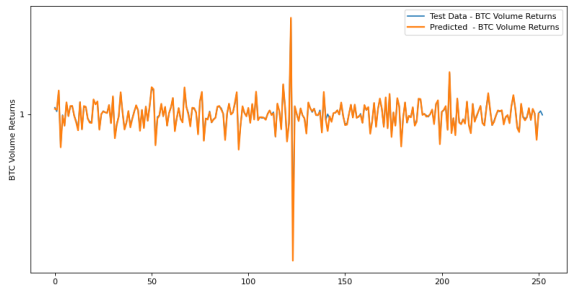


Figure 5.6: COBRA prediction volume returns testing data

Chapter 6

Results for Closing Price (Modified)

We have presented the outcomes of our implementation of various models including RNN, LSTM, GRU, Transformer, Hybrid RNN, LSTM, GRU, Highway LSTM, and COBRA. The training, validation, and testing data predictions for each model were obtained and are illustrated in previous sections. Our evaluation metrics consist of various performance measures.

Mean Squared Error (MSE) and **Validation Loss**.

Models	Test		Validation		Train	
	Loss	MAE	Loss	MAE	Loss	MAE
Simple RNN	0.1099	0.1423	0.0812	0.1606	0.0713	0.1371
LSTM (Without CumSum)	0.1113	0.1475	0.0858	0.1605	0.0809	0.1424
LSTM	0.0011	0.0322	0.0008	0.0291	0.0011	0.0288
GRU	0.0010	0.0311	0.0008	0.0287	0.0011	0.0289
Hybrid	0.0009	0.0279	0.0007	0.0259	0.0010	0.0251
Highway LSTM	0.0005	0.0178	0.0003	0.0153	0.0005	0.0148
Transformer	0.0003	0.0091	0.0002	0.0093	0.0004	0.0092
COBRA	0.000006	0.001811	0.000005	0.001247	0.000004	0.001347

Table 6.1: Evaluation of Weak learners and COBRA

- The results obtained in Cobra have not undergone hyper-parameter tuning, which means that we have used arbitrary values for Epsilon and Alpha.

Chapter 7

Results for Volume (Modified)

We have presented the outcomes of our implementation of various models including RNN, LSTM, GRU, Transformer, Hybrid RNN, LSTM, GRU, Highway LSTM, and COBRA. The training, validation, and testing data predictions for each model were obtained and are illustrated in previous sections. Our evaluation metrics consist of various performance measures.

Mean Squared Error (MSE) and **Validation Loss**.

Models	Test		Validation		Train	
	Loss	MAE	Loss	MAE	Loss	MAE
Simple RNN	0.1739	0.2056	0.1872	0.2138	0.1716	0.2061
LSTM (Without CumSum)	0.1896	0.2316	0.1942	0.2279	0.1853	0.2282
LSTM	0.0281	0.0239	0.0253	0.0215	0.0268	0.0247
GRU	0.0280	0.0248	0.0265	0.0243	0.0279	0.0269
Hybrid	0.0264	0.0250	0.0254	0.0248	0.0257	0.0264
Highway LSTM	0.0168	0.0157	0.0140	0.0126	0.0163	0.0178
Transformer	0.0110	0.0129	0.0102	0.0112	0.0125	0.0159
COBRA	0.000472	0.010759	0.000125	0.008182	0.000638	0.014739

Table 7.1: Evaluation of Weak learners and COBRA

- The results obtained in Cobra have not undergone hyper-parameter tuning, which means that we have used arbitrary values for Epsilon and Alpha.

Chapter 8

Conclusion

- Based on the results you provided, it seems that the Transformer model performed the best, with the lowest validation loss of 0.00007. This is followed closely by the LSTM and Highway LSTM models, which both achieved validation losses of 0.00009 and 0.00010, respectively.
- The GRU model achieved a slightly higher validation loss of 0.00013, but still performed well overall. The RNN and Hybrid models achieved higher validation losses of 0.00010 and 0.00047, respectively.
- Overall, these results suggest that the Transformer, LSTM, and Highway LSTM models may be particularly well-suited for predicting bitcoin prices. However, it's important to note that the specific characteristics of your dataset, such as its size, structure, and underlying patterns, can impact the performance of different models. Therefore, it's always a good idea to experiment with different models and evaluate their performance on your specific dataset.

Chapter 9

Future Plans

Our approach entails the meticulous tuning of hyperparameters, which is crucial for achieving fine-tuned and dynamically responsive predictions.

Furthermore, we aim to implement a combined regression methodology, incorporating an ensemble model of LSTM, GRU, highway LSTM and transformer with dynamically modulated epsilon values and varying alpha. This approach would aid us in enhancing the accuracy of our predictions. However, the scarcity of time has posed a challenge in executing these plans.

Bibliography

1. Biau, Gérard & Fischer, Aurélie & Guedj, Benjamin & Malley, James. (2015). COBRA: A combined regression strategy. Journal of Multivariate Analysis. 146. 10.1016/j.jmva.2015.04.007.

2. Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Shortterm Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

3. Alfarraj, Motaz & Alregib, Ghassan. (2018). Petrophysicalproperty estimation from seismic data using recurrent neural networks. 2141-2146. 10.1190/segam2018-2995752.1.

4. Sherstinsky, Alex. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. Physica D: Nonlinear Phenomena. 404. 132306. 10.1016/j.physd.2019.132306.

5. Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia. (2017). Attention Is All You Need.