STOCK PRICE PREDICTION USING MACHINE LEARNING

A SECOND YEAR PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF B.Sc. IN COMPUTATIONAL MATHEMATICS

BY

1. Saugat Bhattarai (Regd no: 026595-19)
2. Abhay Sharma (Regd no: 026610-19)
3. Sujit Acharya (Regd no: 026593-19)
4. Anmol Jha (Regd no: 026597-19)

SCHOOL OF SCIENCE

KATHMANDU UNIVERSITY

DHULIKHEL, NEPAL

April 2022

CERTIFICATION

This project entitled "STOCK PRICE PREDICTION USING MACHINE LEARNING" is carried out under my supervision for the specified entire period satisfactorily, and is hereby certified as a work done by following students

1. Saugat Bhattarai (Regd No : 026595-19)
2. Abhay Sharma (Regd No : 026610-19)
3. Sujit Acharya (Regd No : 026593-19)
4. Anmol Jha (Regd No : 026597-19)

in partial fulfillment of the requirements for the degree of B.Sc. in Computational Mathematics, Department of Mathematics, Kathmandu University, Dhulikhel, Nepal.

Harish Chandra Bhandari
Designation
Department of Natural Sciences (Mathematics),
School of Science, Kathmandu University,
Dhulikhel, Kavre, Nepal
Date:

APPROVED BY:

I hereby declare that the candidate qualifies to submit this report of the Mathematics Project (Math-252) to the Department of mathematics.

Head of the Department
Department of Natural Sciences
School of Science
Kathmandu University
Date:

ACKNOWLEDGMENTS

This report was carried out under the supervision of Mr. Harish Chandra Bhandari. I would like to express my sincere gratitude towards my supervisor for his excellent supervision, guidance and suggestion for accomplishing this work. And to the entire faculty of Department of mathematics for encouraging, supporting and providing this opportunity.

We would also like to thank everyone who helped us directly and indirectly during the duration of completing our project work.

ABSTRACT

Predicting stock market is one of the most difficult tasks in the field of computation. There are many factors involved in the prediction – physical factors vs. physiological, rational and irrational behavior, investor sentiment, market rumors,etc. All these aspects combine to make stock prices volatile and very difficult to predict with a high degree of accuracy.As per efficient market theory when all information related to a company and stock market events are instantly available to all stakeholders/market investors, then the effects of those events already embed themselves in the stock price. So, it is said that only the historical spot price carries the impact of all other market events and can be employed to predict its future movement. Hence, considering the past stock price as the final manifestation of all impacting factors we employ Machine Learning (ML) techniques on historical stock price data to infer future trend. ML techniques have the potential to unearth patterns and insights we didn't see before, and these can be used to make unerringly accurate predictions.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Forecasting the stock market is a crucial undertaking in the global stock market. Typically, stock market forecasting is concerned with accurately predicting either/both the trend and/or the price of a stock in order to increase trading profits. Due to the nonlinear and volatile nature of stock exchange, obtaining an accurate prediction of stock trend and/or price has proven to be a difficult challenge. Some people who believe in the efficient market hypothesis say that based on historical stock data, future stock prices are predictable. Others who believe in the random walk hypothesis believe that future stock prices are unaffected by prior stock data, and thus that no useful patterns can be detected in historical stock data.Many who believe in the random walk hypothesis believe that future stock prices are unaffected by past stock data, and that no helpful patterns can be detected in historical stock data to predict future stock sequences.

Many statistical models, such as the autoregressive integrated moving average (ARIMA) and seasional autoregressive integrated moving average (SARIMA) models, were developed as an obvious approach to estimate the stock price utilizing past and present data. Because of the nonlinear character of stock market exchange, these statistical models map linear relationships well but are not practical in stock market forecasting.

Deep learning has demonstrated some promising outcomes in stock market predictions in a number of research.The Long Short-Term Memory (LSTM) model or its hybridization and Gated Recurrent Unit (GRU) model or its hybridization appears to be the most preferred deep-learning architecture for stock market forecasting but it's difficult to draw an objective comparison between the performances of these two popular models.

This project's research intends to do two things: first, conduct a normalized comparison of the LSTM and GRU models' stock market forecasting performances under the identical conditions, and second, objectively analyze the significance of using closing prices in stock market forecasting.

To accomplish these objectives, we create a cooperative deep-learning architecture that can treat both the LSTM and GRU models equally with the same inputs, which include relevant characteristics from previous stock data and the news sentiment score for stock prediction.

The development of this cooperative deep-learning architecture was done utilizing existing algorithms or tools utilized in previously published works by other academics to ensure an objective comparison. This eliminates any potential bias induced by any untested algorithm we offer.

This study makes the following contributions: (a) It answers the essential question: under the same conditions, which deep-learning model, LSTM or GRU, would be a better choice for stock forecasting? (b)Comparison of the statistical model (SARIMAX) with the LSTM and GRU.

1.2 Related work

Many deep-learning approaches have been used in stock market prediction in various stock markets throughout the world in recent years. Chen et al. employed the LSTM model to forecast China's stock market on the Shanghai and Shenzhen Exchanges (SSE)[**2**]. A single input layer precedes many LSTM layers, a dense layer, and a single output layer with several neurons in this architecture. Six alternative strategies were used to predict stock prices using many stock variables such as high price, low price, open price, and closing price. The normalized features and SSE indices were found to improve forecasting accuracy in this investigation. In this analysis, financial news emotion was not taken into account.

Cho et al. created GRU, a deep-learning architecture that avoids the disappearance and ballooning of gradients seen in typical recurrent neural networks (RNNs) when learning long-term relationships. As a result, GRU and GRU-related models, including LSTM, have lately been employed in financial investment prediction, such as Bitcoin price prediction[**3**].

Saud and Shakya used stock market variables taken from the Nepal Stock Exchange to

examine the performance of three deep-learning models: Vanilla RNN, LSTM, and GRU in predicting stock price (NEPSE)[**4**]. Among the three models tested, GRU proved to be the most successful in stock price prediction. In this analysis, financial news emotion was not taken into account.

CHAPTER 2

METHODOLOGY/MODEL EQUATION

2.1 The ARIMA And SARIMAX Model

Time series analysis is a great tool for predicting future events such as market values changing. ARIMA and SARIMA are great tools for time series analysis. They require data in a 'long' format. This means a collection (preferably a pandas data-frame) of data points where each data point is associated with a time (in pandas this would be a series with different times forming the index). ARIMA stands for auto regressive integrated moving average. SARIMAX is similar and stands for seasonal auto regressive integrated moving average with exogenous factors.

[**6**] **Auto regressive (AR) model**

[**8**] The model predicts the next data point by looking at previous data points and using a mathematical formula similar to linear regression. There is something called an order (represented by "p") that determines how many previous data points will be used. When the p value is higher the model takes into account more data points that occurred a longer time ago. Below is the formula for an auto regressive model. The "c" is a constant and the "e" stands for error or noise. [**8**]

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

[8]

Figure 2.1: Auto Regression

Integrated part of ARIMA and SARIMAX.

For the auto regressive and moving average models to work the data must be stationary. This means the data needs to not have trends or seasonality. Integration is taking a difference of the time-series, subtracting the previous value from each value, which tends to make the data more stationary. There is a value called "*d*" which represents how many times the data is to be differenced. Like the p value in the auto regressive model, it is best to simply try out a few different values for d and see which model has the minimal AIC(Akaike information criterion) [**8**]

Moving average part of ARIMA and SARIMAX.

A moving average (MA) model performs calculations based on noise in the data along with the data's slope. The formula for an MA model can be found below. The "μ" represents the mean of X and the θ represents the parameters of the model.[**8**]

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

Figure 2.2: Moving Average

Combining AR and MA along with differencing (I) creates the ARIMA model. There is a value called "*q*" in the moving average model that is the order. The order is similar to the auto regressive order where it is the amount of past data points to put into the equation. [**8**] SARIMAX is used on data sets that have seasonal cycles. The difference between ARIMA and SARIMAX is the seasonality and exogenous factors (seasonality and regular ARIMA don't mix well).SARIMAX requires not only the p, d, and q arguments that ARIMA requires, but it also requires another set of p, d, and q arguments for the seasonality aspect as well as an argument called "s" which is the periodicity of the data's seasonal cycle. Choosing an s value requires an idea of when the seasonal data cycles.If the data points are separated by a monthly basis and the seasonal cycle is a year, then

s can be set to 12. Or if the data points are separated by a daily basis and the seasonal cycle is a week then s equal to 7. [8]

2.2 Recurrent Neural Network

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed or undirected graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs.] This makes them applicable to tasks such as unsegmented, connected handwriting recognition[4] or speech recognition. Recurrent neural networks are theoretically Turing complete and can run arbitrary programs to process arbitrary sequences of inputs. [8]

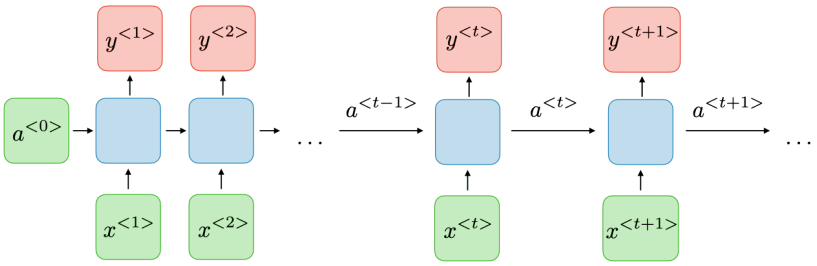


Figure 2.3: RNN Architecture

[8]

In Simple words,Recurrent Neural Network is machine learning tool that connects the previous data to recognize patterns and provide us our required data after it has predicted things from its pattern.

It is commonly used in natural language processing.

2.2.1 LSTM

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter Schmidhuber (1997), and were refined and popularized by many people in following work.1 They work tremendously well on a large variety of problems, and are now widely used. [5]

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn! [**5**] All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.
[**5**]

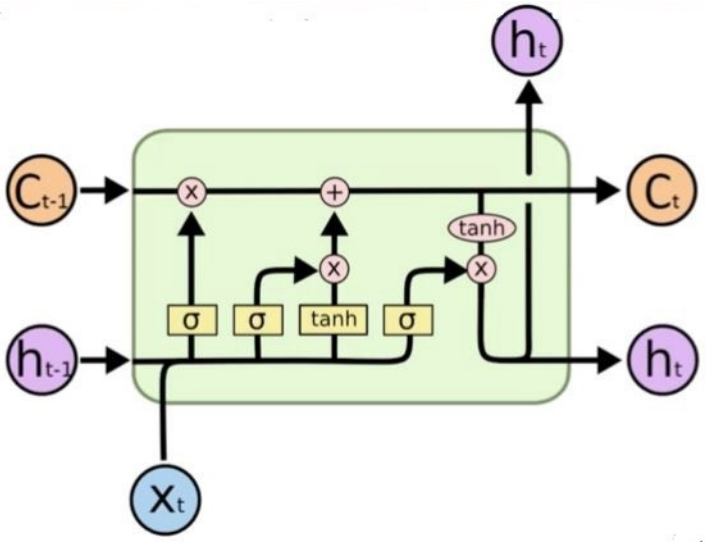


Figure 2.4: LSTM Cell
[**8**]

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.
[**5**]
In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.
[**5**]

2.2.2 Idea Behind LSTMs

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire

chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. [**5**]

The LSTM does have the ability to remove or add information to the cell state, carefully

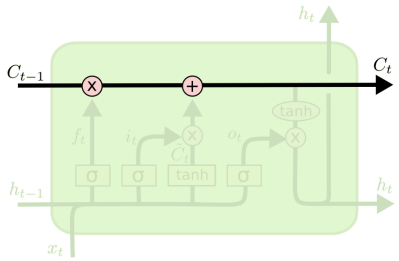


Figure 2.5: The cell state

[**8**]

regulated by structures called gates. [**5**] Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

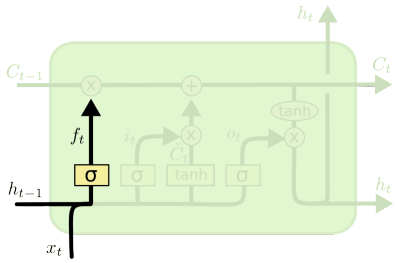
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!" [**5**] An LSTM has three of these gates, to protect and control the cell state.

2.2.3 Gates in LSTM

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

[**5**]

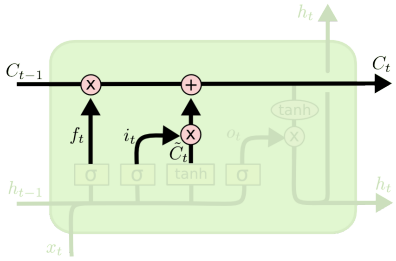
Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through *tanh* [**6**] (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to. For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case



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

Figure 2.6: Forget Gate

[**5**]

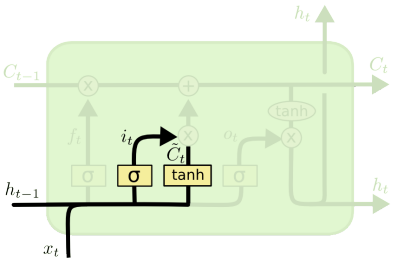


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.7: Candidate Cell State

[**6**]

that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.] The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. [**5**] Next, a tanh layer creates a vector of new candidate values,



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

[**5**]

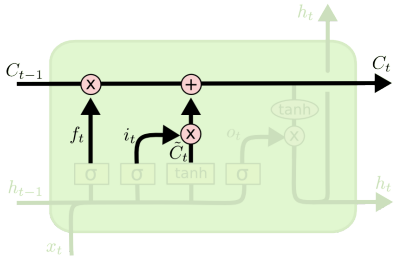
Figure 2.8: Input Gate

$\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting [**6**]

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what

parts of the cell state we're going to output. Then, we put the cell state through *tanh* [**6**] (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to. For the language model example,

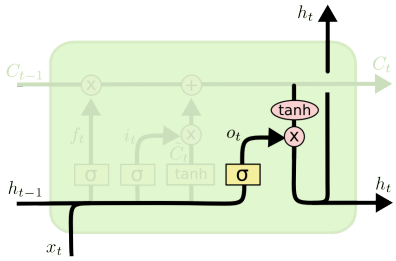


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.9: Candidate Cell State

[**6**]

since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

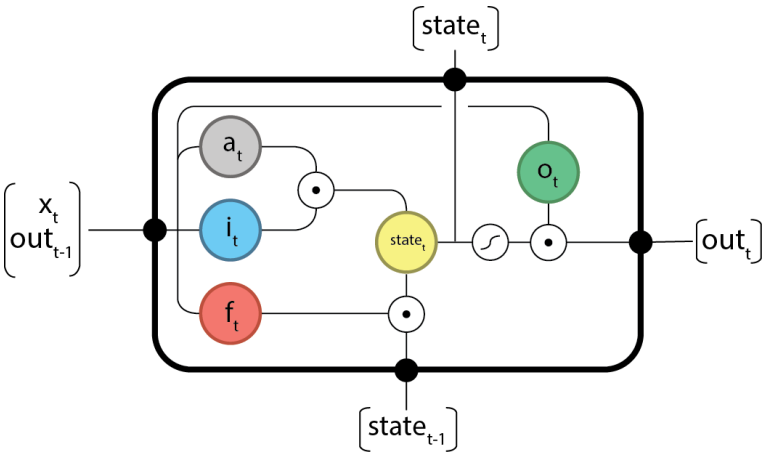


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.10: Output Gate

[**6**]

2.3 LSTM Mathematical Model



Syntactic notes

- Above $\odot$ is the element-wise product or Hadamard product.
- Inner products will be represented as .
- Outer products will be respresented as $\otimes$
- σ represents the sigmoid function

The forward components

The gates are defined as:

Input Activation:

$a_t = \tanh(w_a.x_t + u_a.out_{-1} + b_a)$

Input Gate:

$i_t = \sigma(w_i.x_t + u_i.out_{-1} + b_i)$

Forward Gate:

$f_o = \sigma(w_f.x_t + u_f.out_{-1} + b_f)$

Output Gate:

$o_o = \sigma(w_o.x_t + u_o.out_{-1} + b_o)$

Which leads to: **internal state:**

$State_t = a_t \odot i_t + f_t \odot State_{-1}$

Output:

$out_t = \tanh(State_t \odot o_t$

for simplicity we define:

$$\text{gate } s_t = \begin{bmatrix} a_t \\ i_t \\ f_t \\ o_t \end{bmatrix}, w = \begin{bmatrix} w_a \\ w_i \\ w_f \\ w_o \end{bmatrix}, U = \begin{bmatrix} u_a \\ u_i \\ u_f \\ u_o \end{bmatrix}, b = \begin{bmatrix} b_a \\ b_i \\ b_f \\ b_o \end{bmatrix}$$

The backward components

ΔT the output difference as computed by any subsequent layers (i.e. the rest of your network), and;

Δout the output difference as computed by the next time-step LSTM (the equation for t-1 is below). [**6**]

Then we Find:

$\delta out_t = \Delta_t + \Delta_{out_t}$

$\delta state_t = \delta out_t \odot o_t \odot (1 - \tanh^2(state_t)) + \delta state_1 . f_{t+1}$

$\delta a_t = \delta state_t \odot i_t \odot (1 - a_t^2)$

$\delta i_t = \delta state_t \odot a_t \odot i_t \odot (1 - o_{t+1})$

$\delta f_t = \delta state_t \odot state_{t-1} \odot f_t \odot (1 - f_t)$

$\delta o_t = \delta out_t \odot \tanh(state_t) \odot o_t \odot (1 - o_t)$

$\delta x_t = w^T . \delta gate_t$

$\Delta out_t - 1 = u^T . \delta gates_t$

The final updates to the internal parameters is computed as:

$\delta w = \sum_{n=0}^{T} \delta gates_t \otimes x_t$

$\delta u = \sum_{n=0}^{T-1} \delta gates_t + 1 \otimes out_t$

$\delta b = \sum_{n=0}^{T} \delta gates_t + 1$

EXAMPLE

Date	Close
12/12/2021	2361.27
12/13/2021	2282.01
12/14/2021	2418.85
12/15/2021	2414.62

Let us begin by defining out internal weights:

$$\mathrm{w}_a = \begin{bmatrix} 0.45 \\ 0.25 \\ 0.30 \end{bmatrix}, u_a = \begin{bmatrix} 0.15 \end{bmatrix}, b_a = \begin{bmatrix} 0.2 \end{bmatrix}$$

$$w_i = \begin{bmatrix} 0.95 \\ 0.80 \\ 0.60 \end{bmatrix}, u_i = \begin{bmatrix} 0.80 \end{bmatrix}, b_i = \begin{bmatrix} 0.65 \end{bmatrix}$$

$$w_f = \begin{bmatrix} 0.70 \\ 0.35 \\ 0.30 \end{bmatrix}, u_f = \begin{bmatrix} 0.10 \end{bmatrix}, b_f = \begin{bmatrix} 0.15 \end{bmatrix}$$

$$w_o = \begin{bmatrix} 0.60 \\ 0.40 \\ 0.20 \end{bmatrix}, u_o = \begin{bmatrix} 0.25 \end{bmatrix}, b_o = \begin{bmatrix} 0.10 \end{bmatrix}$$

And now input data:

$$\mathrm{x}_o = \begin{bmatrix} 0.1992 \\ 0.1925 \\ 0.2040 \end{bmatrix}, with label : 0.2037$$

$$x_1 = \begin{bmatrix} 0.1925 \\ 0.2040 \\ 0.2037 \end{bmatrix}, with label : 0.2005$$

Note:The input data are in normalized form

Forward Pass @ t=0

$a_o = \tanh(w_a.x_0 + u_a.out_{-1} + b_a)$

$$=\tanh(\begin{bmatrix} 0.45 & 0.25 & 30 \end{bmatrix} + \begin{bmatrix} 0.1992 \\ 0.1925 \\ 0.2040 \end{bmatrix} + \begin{bmatrix} 0.15 \end{bmatrix} . \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} 0.2 \end{bmatrix})$$

$= 0.37906$

$\mathrm{i}_o = \sigma(w_i.x_0 + u_i.out_{-1} + b_i)$

$$=\sigma(\begin{bmatrix} 0.95 & 0.8 & 0.60 \end{bmatrix} + \begin{bmatrix} 0.1992 \\ 0.1925 \\ 0.2040 \end{bmatrix} + \begin{bmatrix} 0.8 \end{bmatrix} . \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} 0.65 \end{bmatrix})$$

$= 0.75317$

$\mathrm{f}_o = \sigma(w_f.x_0 + u_f.out_{-1} + b_f)$

$$=\sigma(\begin{bmatrix} 0.7 & 0.45 & 0.30 \end{bmatrix} + \begin{bmatrix} 0.1992 \\ 0.1925 \\ 0.2040 \end{bmatrix} + \begin{bmatrix} 0.1 \end{bmatrix} . \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} 0.15 \end{bmatrix})$$

$= 0.59362$

$\mathrm{o}_o = \sigma(w_o.x_0 + u_o.out_{-1} + b_o)$

$$=\sigma(\begin{bmatrix} 0.6 & 0.4 & 0.20 \end{bmatrix} + \begin{bmatrix} 0.1992 \\ 0.1925 \\ 0.2040 \end{bmatrix} + \begin{bmatrix} 0.25 \end{bmatrix} . \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} 0.1 \end{bmatrix})$$

$= 0.58353$

$\mathrm{State}_0 = a_0 \odot i_o + f_o \odot State_{-1}$

=0.37906*0.75317+0.59362*0

=0.283496

$o_out = \tanh(State_0 \odot o_o$

=tanh(0.283496) $*$ 0.58353

= 0.162212

From here, we pass forward our state and output and begin the next time-step.

Forward Pass @ t=1

$\mathrm{a}_1 = \tanh(w_a.x_1 + u_a.out_{-1} + b_a)$

$$=\tanh(\begin{bmatrix}0.45 & 0.25 & 30\end{bmatrix} + \begin{bmatrix}0.1925\\0.2040\\0.2037\end{bmatrix} + \begin{bmatrix}0.15\end{bmatrix} \cdot \begin{bmatrix}0.162212\end{bmatrix} + \begin{bmatrix}0.2\end{bmatrix})$$

= 0.39951

$\mathrm{i}_1 = \sigma(w_i.x_1 + u_i.out_{-1} + b_i)$

$$=\sigma(\begin{bmatrix}0.95 & 0.8 & 0.60\end{bmatrix} + \begin{bmatrix}0.1925\\0.2040\\0.2037\end{bmatrix} + \begin{bmatrix}0.8\end{bmatrix} \cdot \begin{bmatrix}0.162212\end{bmatrix} + \begin{bmatrix}0.65\end{bmatrix})$$

= 0.776982

$\mathrm{f}_1 = \sigma(w_f.x_1 + u_f.out_{-1} + b_f)$

$$=\sigma(\begin{bmatrix}0.7 & 0.45 & 0.30\end{bmatrix} + \begin{bmatrix}0.1925\\0.2040\\0.2037\end{bmatrix} + \begin{bmatrix}0.1\end{bmatrix} \cdot \begin{bmatrix}0.162212\end{bmatrix} + \begin{bmatrix}0.15\end{bmatrix})$$

= 0.6115614

$\mathrm{o}_1 = \sigma(w_o.x_1 + u_o.out_{-1} + b_o)$

$$=\sigma(\begin{bmatrix}0.6 & 0.4 & 0.20\end{bmatrix} + \begin{bmatrix}0.1925\\0.2040\\0.2037\end{bmatrix} + \begin{bmatrix}0.25\end{bmatrix} \cdot \begin{bmatrix}0.162212\end{bmatrix} + \begin{bmatrix}0.1\end{bmatrix})$$

= 0.593485

$\mathrm{State}_1 = a_1 \odot i_1 + f_1 \odot State_0$

=0.39951*0.776982+0.6115614*0.285496

=0.484985

$out_1 = \tanh(State_1 \odot o_1$

$=\tanh(0.484985) * 0.593485$

$= 0.267303$

Backward Propagation @ t=1

First we had computed the difference in output from the expected (label).

$E(x, x) = \frac{(x - \hat{x})^2}{2}$

The derivate w.r.t. x is:

$\partial x E(x, x) = x - \hat{x}$

$\Delta_1 = \partial x E = 0.267203 - 0.2005$

$= 0.066703$

$\Delta_{out1} = 0$

because there is no further steps.

$\delta out1 = \Delta_1 + \Delta_{out1}$

$= 0.066703$

$\delta state_1 = \delta out1 \odot o_1 \odot (1 - \tanh^2(state_1)) + \delta state_2.f_2$

$= 0.066703 * 0.593485 * (1 - tanh(0.484985)^2) + 0$

$= 0.0315627$

$\delta a_1 = \delta state_1 \odot i_1 \odot (1 - a_1^2)$

$= 0.0206094$

$\delta i_1 = \delta state_1 \odot a_1 \odot i_1 \odot (1 - i_1)$

$= 0.002185006$

$\delta f_1 = \delta state_1 \odot state_0 \odot f_1 \odot (1 - f_1)$

$= 0.002140605$

$\delta o_1 = \delta out_1 \odot \tanh(state_1) \odot o_1 \odot (1 - o_1)$

$= 0.00724542$

$$\delta x_1 = w^T.\delta gate_i$$

$$= \begin{bmatrix} 0.45 & 0.95 & 0.70 & 0.60 \\ 0.25 & 0.80 & 0.45 & 0.40 \\ 0.30 & 0.60 & 0.30 & 0.20 \end{bmatrix} * \begin{bmatrix} 0.0206094 \\ 0.002185006 \\ 0.002140605 \\ 0.00724542 \end{bmatrix}$$

$$== \begin{bmatrix} 0.0104627 \\ 0.0107618 \\ 0.0095851 \end{bmatrix}$$

$$\Delta out_0 = u^T.\delta gates_1$$

$$= \begin{bmatrix} 0.15 & 0.80 & 0.10 & 0.25 \end{bmatrix} * \begin{bmatrix} 0.0206094 \\ 0.002185006 \\ 0.002140605 \\ 0.00724542 \end{bmatrix}$$

$$= 0.0068648$$

Now we have pass back our out and continue on computing...

Backward @ t=0

$$\Delta_0 = \partial xE = 0.162212 - 0.2037$$

$$= -0.041488$$

$$\Delta_{out0} = 0.0068648, passedback fromT = 1.$$

$$\delta out0 = \Delta_0 + \Delta_{out0}$$

$$= -0.0483528$$

$$\delta state_0 = \delta out0 \odot o_0 \odot (1 - \tanh^2(state_0)) + \delta state_1.f_1$$

$$= -0.00673245$$

$$\delta a_0 = \delta state_0 \odot i_0 \odot (1 - a_0^2)$$

$$= -0.00434209$$

$$\delta i_0 = \delta state_0 \odot a_0 \odot i_0 \odot (1 - 0_1)$$

$$= -0.00047443$$

$$\delta f_0 = \delta state_0 \odot state\_1 \odot f_0 \odot (1 - f_0)$$

$$= 0$$

$$\delta o_0 = \delta out_0 \odot \tanh(state_0) \odot o_0 \odot (1 - o_0)$$

$= 0.01095846$

$\delta x_0 = w^T.\delta gate_0$

$$= \begin{bmatrix} -0.0020725 \\ -0.0029183 \\ 0.00060440 \end{bmatrix}$$

$\Delta out\_1 = u^T.\delta gates_0$

$= 0.00170875$

Now we have updated our internal parameters according to whatever solving algorithm .We have used a simple Stochastic Gradient Descent (SGD) update with learning rate: =0.1.

We have computed how much our weights are going to change by:

$\delta w = \sum_{n=0}^{T} \delta gates_t \otimes x_t$

$$= \begin{bmatrix} -0.004342096 \\ -0.00047443 \\ -0.0 \\ 0.010958467 \end{bmatrix} \begin{bmatrix} 0.1992 & 0.1925 & 0.2040 \end{bmatrix}$$

$$+ \begin{bmatrix} 0.020609477 \\ 0.002185006 \\ 0.002140605 \\ 0.007245423 \end{bmatrix} \begin{bmatrix} 0.1925 & 0.2040 & 0.2037 \end{bmatrix}$$

$$= \begin{bmatrix} 0.003102378 & 0.003368479 & 0.003312362 \\ 0.000326107 & 0.000354413 & 0.000348302 \\ 0.000412066 & 0.000436683 & 0.000436041 \\ 0.003577670 & 0.003587571 & 0.003711420 \end{bmatrix}$$

$\delta u = \sum_{n=0}^{T-1} \delta gates_t + 1 \otimes out_t$

$$= \begin{bmatrix} 0.020609477 \\ 0.002185006 \\ 0.002140605 \\ 0.007245423 \end{bmatrix} \begin{bmatrix} 0.162212 \end{bmatrix}$$

$$= \begin{bmatrix} 0.003343104 \\ 0.000354434 \\ 0.000347231 \\ 0.001175294 \end{bmatrix}$$

$\delta b = \sum_{n=0}^{T} \delta gates_t + 1$

$$= \begin{bmatrix} 0.020609477 \\ 0.002185006 \\ 0.002140605 \\ 0.007245423 \end{bmatrix} + \begin{bmatrix} -0.004342096 \\ -0.00047443 \\ -0.0 \\ 0.010958467 \end{bmatrix}$$

$$= \begin{bmatrix} 0.016267381 \\ 0.001710576 \\ 0.002140605 \\ 0.018203890 \end{bmatrix}$$

And updating out parameters based on the SGD update function:

$w^{new} = w^{old} - \lambda * \delta w^{old}$

$$\mathrm{W}_a = \begin{bmatrix} 0.44968976 \\ 0.24996738 \\ 0.29995879 \end{bmatrix}, u_a = \begin{bmatrix} 0.1496656 \end{bmatrix}, b\_a = \begin{bmatrix} 0.19837326 \end{bmatrix}$$

$$w_i = \begin{bmatrix} 0.94968976 \\ 0.79996738 \\ 0.59995879 \end{bmatrix}, u_i = \begin{bmatrix} 0.7996656 \end{bmatrix}, b_i = \begin{bmatrix} 0.6483732 \end{bmatrix}$$

$$w_f = \begin{bmatrix} 0.69968976 \\ 0.44996738 \\ 0.29995879 \end{bmatrix}, u_f = \begin{bmatrix} 0.09966568 \end{bmatrix}, b_f = \begin{bmatrix} 0.1483732 \end{bmatrix}$$

$$w_o = \begin{bmatrix} 0.59968976 \\ 0.39996738 \\ 0.19995879 \end{bmatrix}, u_o = \begin{bmatrix} 0.24966568 \end{bmatrix}, b_o = \begin{bmatrix} 0.09837326 \end{bmatrix}$$

That completes one iteration of solving an LSTM cell.

2.4 GRU(Gated Recurrent Units)

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al. GRU is basically like a LSTM with a Forget Gate. It has fewer parameters than lstm as it lacks an output gate.GRU aims to solve the vanishing gradient problem which comes with a standard recurrent neural network. GRU can also be considered as a variation on the LSTM because both can produce equally excellent results.[**1**]

2.4.1 Architecture of GRU

As mentioned above, GRUs are improved version of standard recurrent neural network. To solve the vanishing gradient problem of a standard RNN, GRU uses, so-called, Update Gate and Reset Gate. Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction. [**1**]

2.4.2 Gated Hidden State

The key distinction between RNNs and GRUs is that the latter support gating of the hidden state. This means that we have dedicated mechanisms for when a hidden state should be updated and also when it should be reset. These mechanisms are learned and they address the concerns listed above. For instance, if the first token is of great importance we will learn not to update the hidden state after the first observation. Likewise, we will learn to skip irrelevant temporary observations. Last, we will learn to reset the latent state whenever needed. We discuss this in detail below. [**1**]

2.4.3 Reset Gate and Update Gate

The Reset Gate is responsible for the short-term memory of the network i.e the hidden state (H_t).The equation of the Reset Gate is given below. [**1**]

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)$$

Similarly, we have an Update gate for long-term memory and the equation of the gate is shown below.

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$

Mathematically, for a given time step , suppose that the input is a minibatch $X_t \in \mathbb{R}^{n*d}$ (number of examples:n , number of inputs: d) and the hidden state of the previous time step is $H_{t-1} \in \mathbb{R}^{n*d}$ (number of hidden units:h). Then, we will get the Reset Gate (R_t) and Update Gate(Z_t)

From above equation of Reset Gate and Update Gate,
W_{xr} , $W_{xz} \in \mathbb{R}^{d*h}$ and W_{hr} , $W_{hz} \in \mathbb{R}^{h*h}$ are weight parameters and $b_r, b_z \in \mathbb{R}^{1*h}$.We use sigmoid functions to transform input values to the interval (0,1)

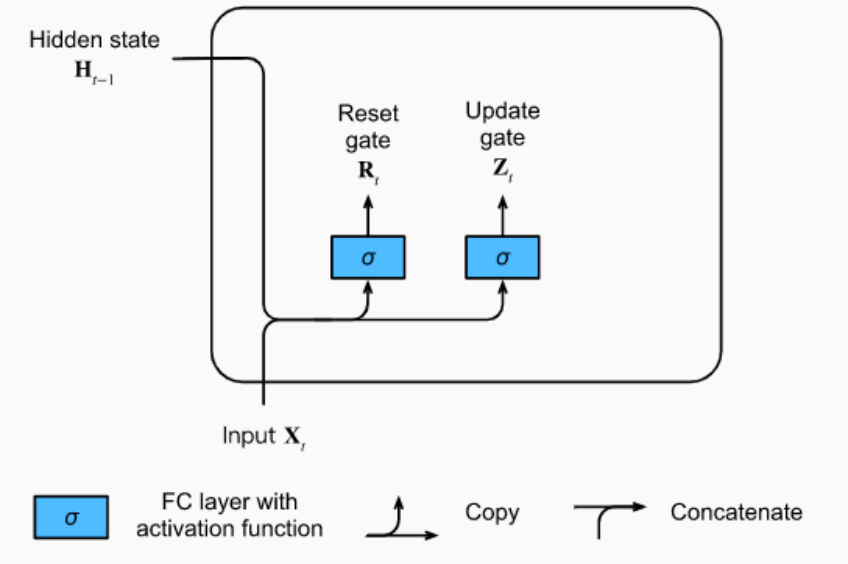


Figure 2.11: Reset Gate and Update Gate

[1]

2.4.4 Candidate Hidden State

To find the Candidate Hidden State $\tilde{H}_t$ at time step t, we have to integrate the Reset Gate with the regular latent state updating mechanism in RNN . It takes in the input and the hidden state from the previous timestamp t-1 which is multiplied by the reset gate output rt. Later passed this entire information to the tanh function, the resultant value is the candidate's hidden state.[**5**]

$\tilde{H}_t = tanh(X_t W_{xh} + (R_t \odot H_{t-1})W_{hh} + b_n)$
where $W_{xh} \in \mathbb{R}^{d*h}$ and $W_{hh} \in \mathbb{R}^{h*h}$ are weight parameters, $b_h \in \mathbb{R}^{h*h}$ is the bias, and the symbol $\odot$ is the Hadamard (elementwise) product operator. Here we use a nonlinearity in the form of tanh to ensure that the values in the candidate hidden state remain in the interval. [**1**]

The most important part of this equation is how we are using the value of the reset gate to control how much influence the previous hidden state can have on the candidate state. [**6**]

If the value of R_t is equal to 1 then it means the entire information from the previous hidden state H_{t-1} is being considered. Likewise, if the value of R_t is 0 then that means the information from the previous hidden state is completely ignored. [**1**]

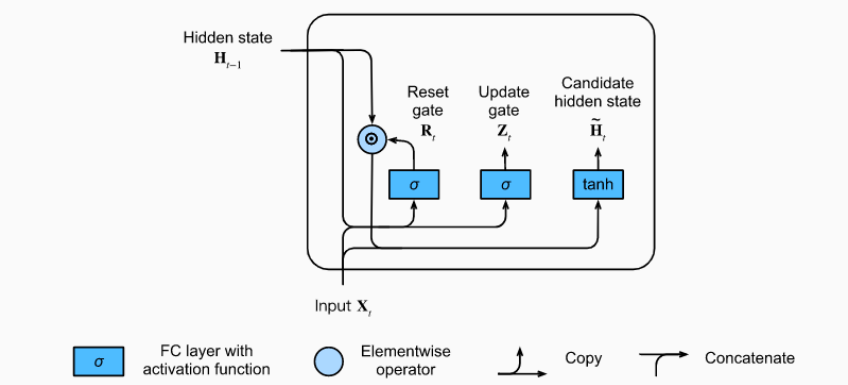


Figure 2.12: Computing the candidate hidden state in a GRU model. [**1**]

2.4.5 Hidden State

Finally, we need to incorporate the effect of the update gate Z_t This determines the extent to which the new hidden state $H_t \in \mathbb{R}^{n*h}$ is just the old state $\tilde{H_{t-1}}$ and by how much the new candidate state $\tilde{H}_t$ is used. The update gate Z_t can be used for this purpose, simply by taking elementwise convex combinations between both H_{t-1} and $\tilde{H}_t$. This leads to the final update equation for the GRU:

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t \text{ [1]}$$

Whenever the update gate Z_t is close to 1, we simply retain the old state. In this case the information from X_t is essentially ignored, effectively skipping time step t in the dependency chain. In contrast, whenever Z_t is close to 0, the new latent state H_t approaches the candidate latent state $\tilde{H}_t$. These designs can help us cope with the vanishing gradient problem in RNNs and better capture dependencies for sequences with large time step distances. For instance, if the update gate has been close to 1 for all the time steps of an entire subsequence, the old hidden state at the time step of its beginning will be easily retained and passed to its end, regardless of the length of the subsequence.[**6**]

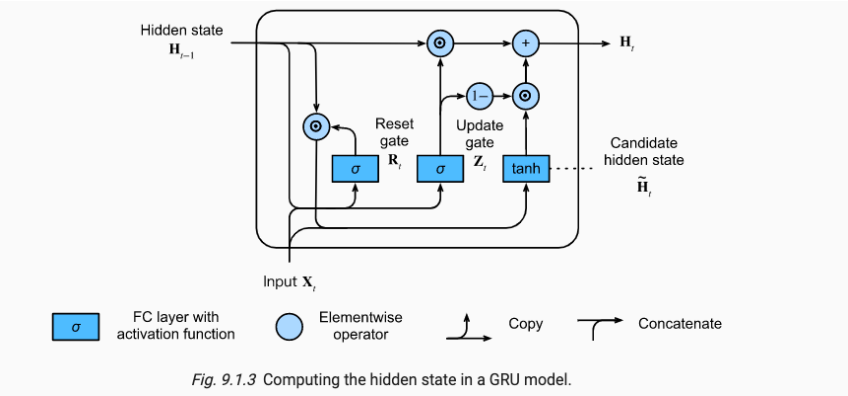


Figure 2.13: Computing the hidden state in a GRU model. [**1**]

2.5 LSTM vs GRU

2.5.1 Structural Differences

[7]

While both GRU's and LSTM's contain gates, the main difference between these two structures lies in the number of gates and their specific roles. The role of the Update gate in the GRU is very similar to the Input and Forget gates in the LSTM. However, the control of new memory content added to the network differs between these two. [7]

In the LSTM, while the Forget gate determines which part of the previous cell state to retain, the Input gate determines the amount of new memory to be added. These two gates are independent of each other. [7]

As for the GRU, the Update gate is responsible for determining which information from the previous memory to retain and is also responsible for controlling the new memory to be added. This means that the retention of previous memory and addition of new information to the memory in the GRU is NOT independent.[7]

Another key difference between the structures is the lack of the cell state in the GRU, as mentioned earlier. While the LSTM stores its longer-term dependencies in the cell state and short-term memory in the hidden state, the GRU stores both in a single hidden state. However, in terms of effectiveness in retaining long-term information, both architectures have been proven to achieve this goal effectively.[7]

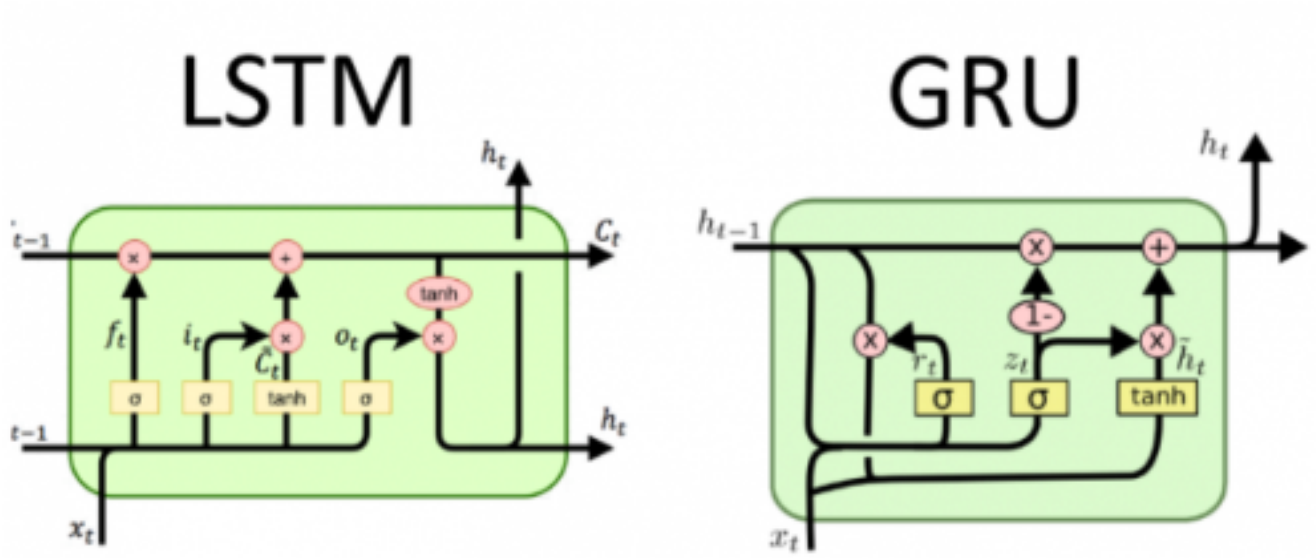


Figure 2.14: Comparison between LSTM and GRU.

[**1**]

2.5.2 Speed Differences

GRUs are faster to train as compared to LSTMs due to the fewer number of weights and parameters to update during training. This can be attributed to the fewer number of gates in the GRU cell (two gates) as compared to the LSTM's three gates. [**6**]

2.5.3 Performance Evaluation

The accuracy of a model, whether it is measured by the margin of error or proportion of correct classifications, is usually the main factor when deciding which type of model to use for a task. Both GRUs and LSTMs are variants of RNNS and can be plugged in interchangeably to achieve similar results. [**6**]

CHAPTER 3

RESULTS AND DISCUSSIONS

This chapter includes the results of the stock price prediction of NABIL Bank for the Year 2001 to 2021.For the prediction task we have used staticak model as well as machine learning model.For the comparison purpose we have used SARIMAX as statical model and RNN,LSTM and GRU as the machine learning model.

3.1 Result: SARIMAX

3.1.1 Data Ploting

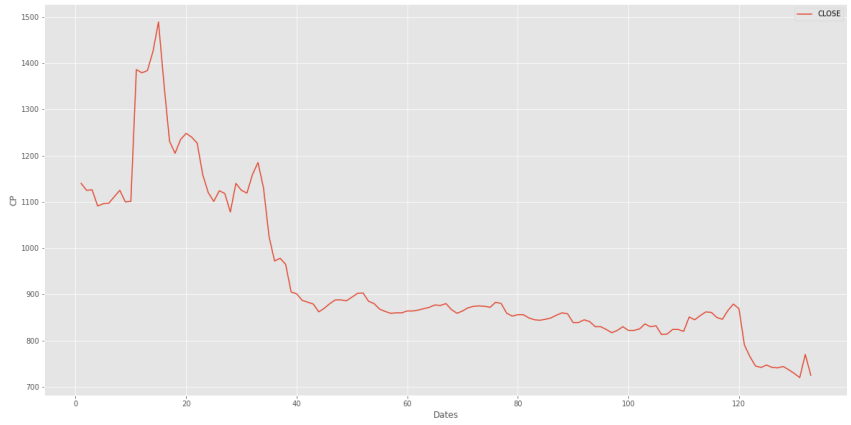


Figure 3.1: Data For NABIL Stock Prices

3.1.2 Defining Parameters and Seasonality

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

Figure 3.2: Parameter Combinations for SARIMAX

3.1.3 Prediction

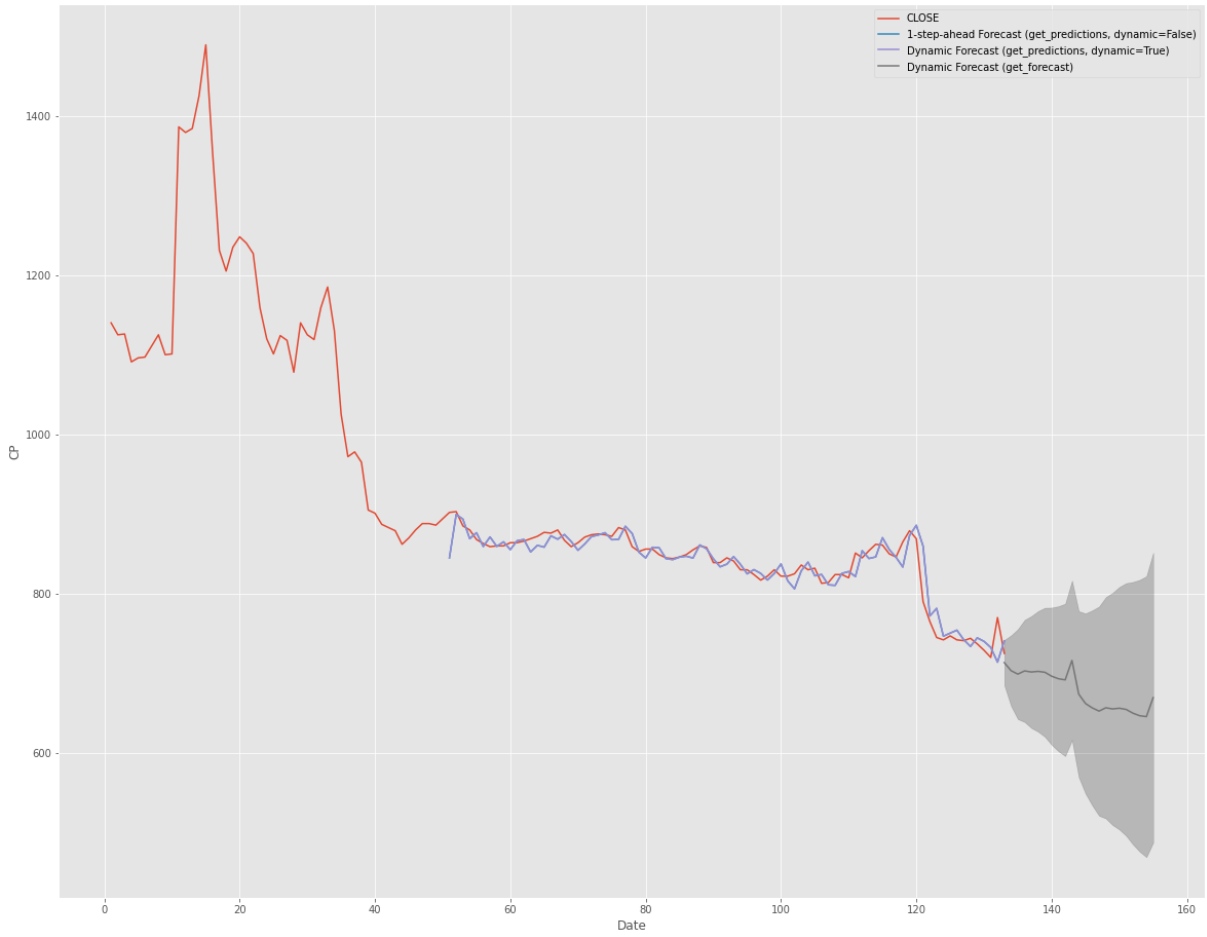


Figure 3.3: SARIMAX Prediction

3.2 LSTM AND GRU

3.2.1 DATA Exploration:

The dataset contains 6 columns associated with time series like the date and the different variables like close, high, low and volume. We will use closing values for our experimentation of time series with LSTM.

	Symbol	Date	Open	High	Low	Close
0	NEPSE	2021-12-29	2520.55	2538.78	2512.41	2524.50
1	NEPSE	2021-12-28	2523.87	2533.63	2486.05	2518.99
2	NEPSE	2021-12-27	2600.21	2616.27	2518.16	2521.65
3	NEPSE	2021-12-26	2531.43	2596.43	2531.15	2591.42
4	NEPSE	2021-12-23	2522.51	2542.29	2485.38	2520.23

Figure 3.4: NABIL BANK Data

3.2.2 Data Preprocesing:

We have use 80 % data for training and the rest 20% for testing and assign them to separate variables.

```
The shape of train dataset (3658, 6)
The shape of val data (458, 6)
The shape of test data (458, 6)
```

Figure 3.5: Shapes of train, test data

3.3 LSTM

3.3.1 Implementation of our LSTM model:

In the next step, we create our LSTM model. Here we have used the Sequential model imported from Keras and required libraries are imported.The total trainable model parameters of LSTM with dense layers are illustrated below:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 24, 4)]           0

 LSTM (LSTM)                 (None, 128)               68096

 Dense1 (Dense)              (None, 256)               33024

 Dense2 (Dense)              (None, 64)                16448

 Dense3 (Dense)              (None, 32)                2080

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 119,681
Trainable params: 119,681
Non-trainable params: 0
```

Figure 3.6: Implementation of LSTM

3.3.2 Visualization:

After fitting the data with our model we use it for prediction. We must use inverse transformation to get back the original value with the transformed function. Now we can use this data to visualize the prediction.

For Training dataset:

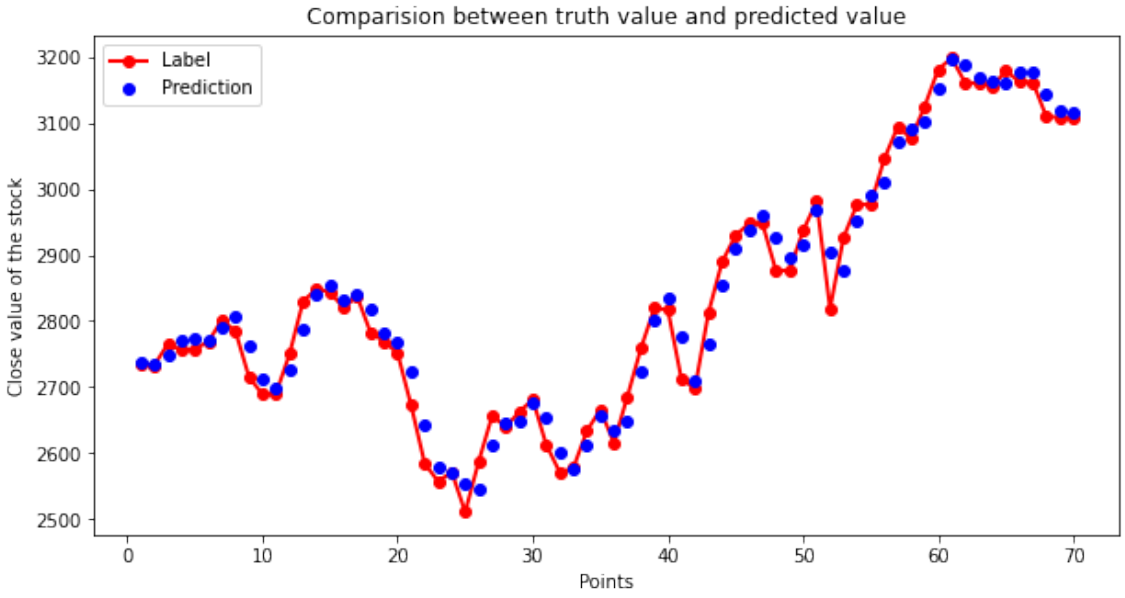


Figure 3.7: Training Dataset of LSTM

For Validation dataset:

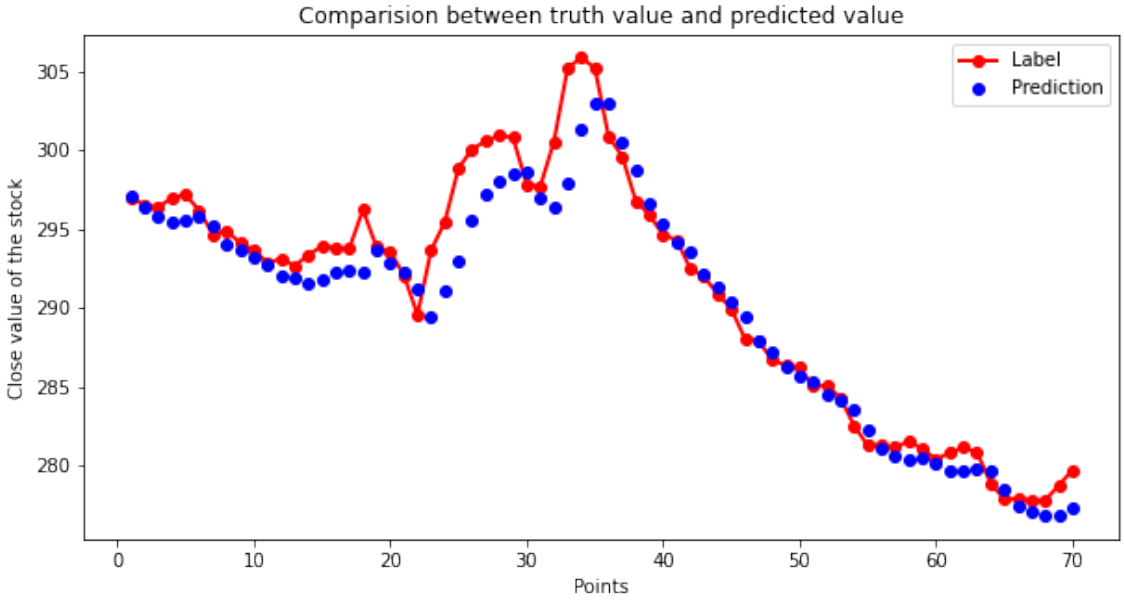


Figure 3.8: Result of LSTM Prediction

3.4 GRU Model:

3.4.1 Implementation of our GRU model:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 24, 4)]           0

 GRU (GRU)                   (None, 128)               51456

 Dense1 (Dense)              (None, 256)               33024

 Dense2 (Dense)              (None, 64)                16448

 Dense3 (Dense)              (None, 32)                2080

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 103,041
Trainable params: 103,041
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.9: Implementation of GRU model

3.4.2 Visualization:

For Training dataset:

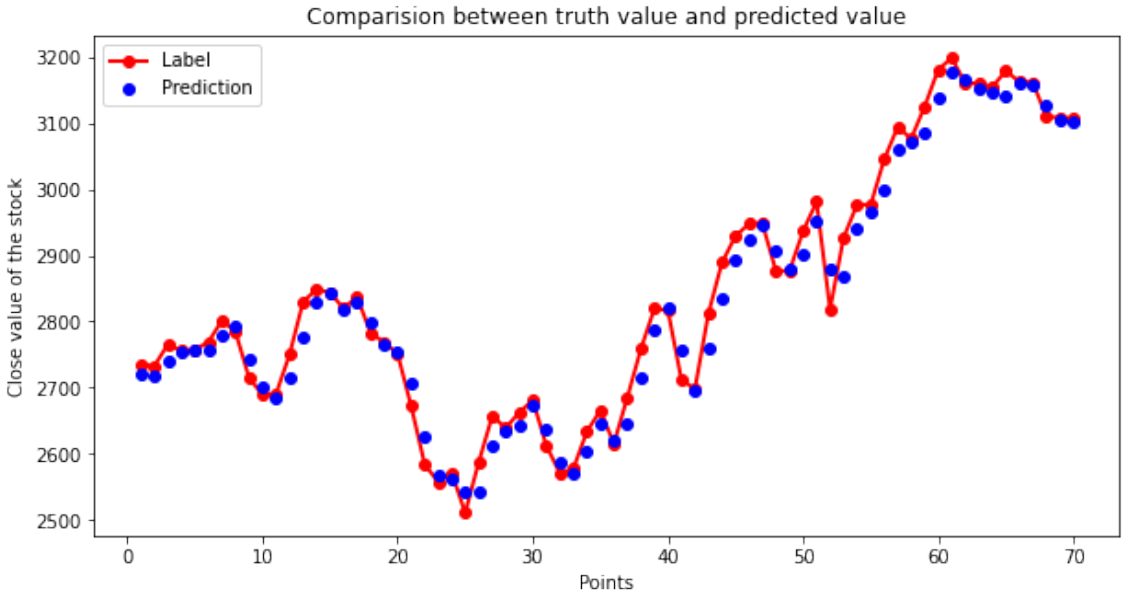


Figure 3.10: Training dataset of GRU

For Validation dataset:

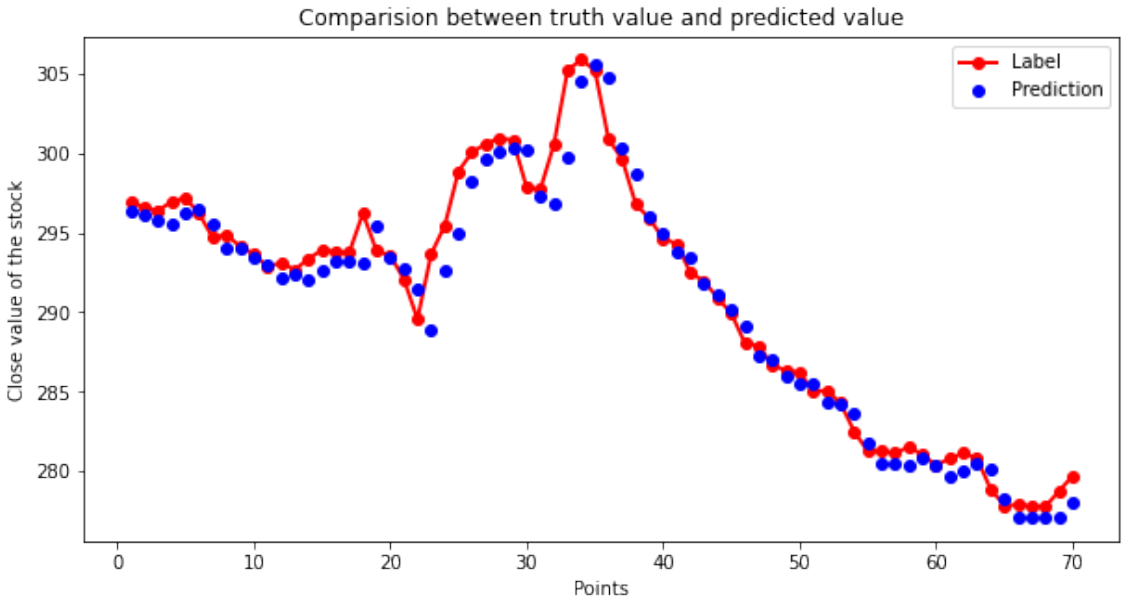


Figure 3.11: Result of GRU Prediction

CHAPTER 4

CONCLUSIONS

By using the Recurrent Network and statistical models architecture proposed in this study, we have achieved our objective of the project i.e, a normalized cpmparision on the performance of the RNN, LSTM, GRU, ARIMA And SARIMAX model for stock price with the stock features as the input in stock market forecasting. We used the data of NABIL Bank and compared the performance of out training. Our experimemtal results shows that all five models can be used to predict stock prices effectively.

Bibliography

[1] https://d2l.ai/index.html

[2] https://ieeexplore.ieee.org/abstract/document/7364089

[3] https://arxiv.org/abs/1406.1078

[4] https://www.sciencedirect.com/science/article/pii/S1877050920308851

[5] https://www.sciencedirect.com/science/article/pii/S1877050920308851

[6] Python Machine Learning Cookbook: Practical Solutions from Preprocessing to Deep Learning

[7] https://mospace.umsystem.edu/xmlui/bitstream/handle/10355/56058/research.pdf

[8] https://www.espon.eu/sites/default/files/attachments/TR$_{T}ime_{S}eries_{j}une$2012.*pdf*