New Jersey Institute of Technology
Digital Commons @ NJIT

Theses | Electronic Theses and Dissertations

5-31-2021

Time series forecasting with applications to finance

Viswapriya Misra
New Jersey Institute of Technology

Follow this and additional works at: https://digitalcommons.njit.edu/theses

 Part of the Data Science Commons

Recommended Citation

Misra, Viswapriya, "Time series forecasting with applications to finance" (2021). *Theses*. 1836.
https://digitalcommons.njit.edu/theses/1836

This Thesis is brought to you for free and open access by the Electronic Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Theses by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

NJIT New Jersey's Science & Technology University

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

TIME SERIES FORECASTING WITH APPLICATIONS TO FINANCE

by
Viswapriya Misra

In finance, many phenomena are modeled as time series. This thesis investigates time series forecasting problems in finance, precisely the stock price prediction problem. We employ and compare traditional statistical algorithms like MA, ARIMA, and ARMA-GARCH with newly developed deep learning-based algorithms such RNNs, LSTMs, GRUs, TCNs, and bidirectional LSTMs and GRUs for predicting stock prices. We perform a comprehensive study and present all the experimental results on different datasets. We find that ARIMA and GRU perform better for single-step stock price prediction than other deep learning architectures. Adding market and economic indicators do not improve the performance of the deep learning models. In the case of multistep forecasting, ARIMA outperforms multistep GRU/TCN and Seq2Seq GRU/TCN. Also, transfer learning helps to improve the performance of the deep learning models.

TIME SERIES FORECASTING WITH APPLICATIONS TO FINANCE

by
Viswapriya Misra

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master's of Sciences in Data Science

Department of Computer Science
New Jersey Institute of Technology

May 2021

APPROVAL PAGE

TIME SERIES FORECASTING WITH APPLICATIONS TO FINANCE

Viswapriya Misra

Dr. Jason T.L. Wang, Dissertation Advisor — Date
Professor, New Jersey Institute of Technology

Dr. Grace Wang , Committee Member — Date
Professor, New Jersey Institute of Technology

Dr. Katherine Hebert , Committee Member — Date
Associate Professor, MontClair State University

BIOGRAPHICAL SKETCH

Author: Viswapriya Misra

Degree: Master of Science

Date: May 2021

Undergraduate and Graduate Education:

- Master of Science in Data Science
  New Jersey Institute of Technology, 2021
- Bachelor of Technology in BioTechnology
  UIET, Kurukshetra University, 2017

Major: Data Science

It is far better to foresee even without certainty than not to foresee at all.

Henri Poincare

ACKNOWLEDGMENT

I want to thank Dr. Jason T.L. Wang for guiding me through the Master's thesis process. I would also like to extend my gratitude towards Dr. Katherine Herbert and Dr. Grace Wang for accepting my invitation to join as my thesis committee members.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

Stocks are one of the most important asset classes for any investor. There are several advantages while investing in stocks. Historically seen, stocks provide liquidity and a good return over time. The 5 year Nasdaq Composite return is 171.64 percent[1]. Also, publicly traded companies are by regulation mandated to disclose their financial status; this provides a lot of information for investing. Share markets are also regulated, which prevents different kinds of risks in financial transactions.

Since stocks are such an important asset class due to their advantages, it is essential to know when to buy and sell the stock to make a profit. To do so, one needs to predict the price of the stock accurately. The methods to predict the stock price are mainly divided into two categories, fundamental analysis and technical analysis[2].

There is a widely held belief in finance that the fundamental value of the share is nothing but the present value of future benefits. All demand and supply of shares are supposed to originate from the expected future benefits from the stocks. If the prevailing price of the share is below the fundamental value, the shareholder considers it a good buy. If the current price is greater than the fundamental value, the shareholder would like to sell the stock. The process of dealing with the estimation of the fundamental value is known as fundamental analysis[3].

Another type of analysis is called technical analysis. Technical analysts believe that the current stock price contains all the information. The technical analysis avoids human subjectivity and emotion and relies upon various graphs, charts, and mathematical methods like linear regression, ARIMA, GARCH, etc. Many factors influence stock price like historical prices, market trends, global economy, industry trends, etc. These factors can be used to predict the price of the stock. If only the

one-time price is used to predict the future price, then it is a univariate time series prediction problem. If more than one factor is used to indicate the price, then it is a multivariate time series problem.

This study provides an analysis of different time series prediction methods and compares the performances. With the recent advancement in deep learning, it has become very promising to use deep learning methods for time series prediction. Therefore, a comparison of traditional statistical methods with the current deep learning methods is studied. The performance of the deep learning models has been compared with the performance of Simple Moving Average, Exponential Moving Average, ARIMA, and ARMA GARCH for stock price prediction. The deep learning models considered for the study include Recurrent Neural Networks(RNN) and its variants like LSTM, GRU, Bi-GRU, and Bi-LSTM. Convolution Neural Networks(CNN) has also been used to model the data as suggested by some previous research work [4], and the performance has been compared. Also, a variant of the CNN model explicitly used for time series modeling called Temporal Convolutional Networks or (TCN) has been employed for predictions. Moreover, various combinations above models have also been used, and the performances of these models are discussed.

The study includes univariate single-step forecasting, multivariate single-step forecasting, univariate multi-step forecasting analysis, and analyzing the effect of transfer learning for stock price prediction. Under each section of the study, there are a few more minor research questions that we answer. We are studying the stock price data of three companies, namely, Apple, Tesla, and Facebook, over six years from 2015 until 2020. The stock price is time-series data, and it follows some patterns. Through our modeling techniques, we try to learn these patterns and see whether or models can capture more signal than noise. We split the data in a 70:30 ratio for training and testing. We aim to predict the stock's closing price based on the

historical closing price for the univariate time series model; for multivariate time series problems, open, high, close, volume, and market indices are considered.

CHAPTER 2

LITERATURE REVIEW

Traditionally statistical time series models have been used frequently for stock price prediction. The most popular time series forecasting method in finance is the ARIMA model, which is generalized by Box and Jenkins[5]. ARIMA model stands for Auto-Regressive Integrated Moving Average model. It includes both an Auto-Regressive component and a Moving Average component with differencing. The main benefit of using the ARIMA model is to transform a non-stationary series into a series without seasonality or trend by applying finite differencing of data points.

The authors of [6] have looked into the critical limitations of MA models like ARIMA, SARIMA, and ARIMAX. The major hurdles for these models are that they are regression-based approaches, and hence they are seldom able to model data with non-linear relationships between parameters. In addition to this, they require certain statistical assumptions about the data to be held in order to have a meaningful insight, for instance, the constant standard deviation in error terms.

An ARIMA model, when integrated with a Generalized Auto-regressive Conditional Heteroskedasticity (GARCH) model, provides the benefit of relaxing the assumption of a constant standard deviation in errors. While this limitation is overcome, the optimization of a GARCH model and its parameters might be puzzling and tricky [7]. Another type of model that can be used for stock price prediction is the ARMA GRACH model. Garcia et al. [8] had used the ARMA GARCH model to effectively predict the day ahead electric prices in Spain and California. In this study, both the ARIMA model and ARMA GRACH model are used to predict the stock price.

New techniques in deep learning have come up to solve problems related to sequential data. LSTM (Long Short-Term Memory), which is a particular case of the

Recurrent Neural Network (RNN) method introduced by Hochreiter and Schmidhuber [9], solves the vanishing gradient problem[10] of RNNs. LSTM architecture has proven to be stable and robust for long-range modeling dependencies in various previous studies [11,12].

Selvin et al. [4] applied three different neural networks like LSTM, RNN, and CNN to predict NSE-India-listed conglomerates. The authors evaluated the proposed methods using the sliding window approach. The results showed that CNN was able to capture the dynamical change of data when compared to other models.

Siami-Niamini et al. [13] conducted a comparative study between LSTM and ARIMA. The authors used historical monthly financial time series from Jan 1985 to Aug 2018 from the Yahoo finance Web. The experiment results showed that LSTM, compared to the ARIMA model, facilitated the best overall performance, confirmed through RMSE values. Peter Yamak et al.[14] compared ARIMA, LSTM, and GRU networks for bitcoin price prediction. They found that ARIMA outperforms the deep learning models, and GRU performs better than LSTM.

In the above papers of Siami-Niamini et al. and Selvin et al., a small collection of models are tested on different datasets. In our study, we aim to test a wide array of models and compare their performance on three given datasets. It will help us to know which model performs well and are best suited for the problem of stock price prediction. We also compare hybrid models like CNN GRU and LSTM GRU. Each of these hybrid models has advantages of its own due to the unique individual characteristics.

Siami-Niamini et al. had implemented LSTM with a window size of one. Whereas Chen et al. [15] had implemented LSTM on Chinese stocks with a window size of 30. Window size is the length of past data the network uses as a feature to forecast the next day's price. We aim to compare the performance of our model with different window sizes and try to see what impact window size has on the results.

Sirignano et al. [16] reported a universal price formation mechanism in stock markets. We use this knowledge to test whether a single model trained on multiple datasets can outperform models trained for specific stocks. The aim is to pre-train a single deep learning model on different datasets and fine-tune its parameters by training on a specific stock before predicting the prices for that stock. This process is called as transfer learning, and we try to implement transfer learning both in the case of univariate price prediction and multivariate stock price prediction.

Jaydip Sen et al. [17] had implemented a multistep one-week forecast for the Nifty 50 Index using CNN and LSTM. We do a similar study but on stock prices and for 3 days, 5 days, and 7 days forecast horizon. The models considered for multistep forecast comparison are ARIMA, GRU and Sequence to Sequence GRU model.

Shaojie Bai et al. [18] conclude that the usual association between recurrent networks and sequence modelling should be revised and Convolutional Networks should be considered as an obvious starting point for sequence modeling tasks.

Shumin Deng et al.[19] observed that TCNs significantly outperformed baseline models like ARIMA, LSTM, and CNN on the stock trend prediction tasks. It achieves much better performance than either traditional ML models, or deep neural networks (such as LSTM and CNN), suggesting that TCN has a more obvious edge in sequence modeling and classification problems. Hence, they employed TCN as their basic prediction model in this paper.

CHAPTER 3

METHODS

3.1 Background

The study tries to compare different models for stock price prediction and attempts to find which model can best learn the data structure and make the most accurate predictions. The best overall model is then used further for different use cases like multi-step prediction and multivariate prediction on the same dataset. We also perform transfer learning to test whether a model can learn from other stock price movements and improve prediction accuracy when further fine-tuned on a given stock. There are two traditional statistical methods used, namely - ARIMA and ARMA GARCH model. We also use moving averages as a baseline for checking the performance of a model. The deep learning models included in the study are RNN, LSTM, GRU, Bi-LSTM, Bi-GRU, LSTM GRU, CNN, and CNN GRU.

The advantages of ARIMA are that it is simple to implement, does not require any parameter tuning, and is quick to run. The benefits of RNN are that it requires no pre-requisites like stationarity, and it can model non-linear functions. Also, statistical models are probabilistic, whereas deep learning models try to capture the structure in the data.

In deep learning models, RNNs are better suited for stock price prediction than fully connected deep learning networks because, firstly, RNNs can take variable-length input. Secondly, it can model long-range dependencies. Fully secured neural networks have independent parameters, whereas, in RNNs, parameters are shared across time steps that help in learning across time.

We also compare the usefulness of applying Temporal convolutional networks (TCN). This framework employs casual convolutions and dilations.

3.1.1 Moving Average

Moving averages are very commonly used technical indicators in financial trading. There are many types of moving averages; the most common ones are the simple moving average(SMA) and exponential moving average(EMA)[18].

A simple moving average of order n is the average of the past n values. Since it is an average, it averages out noise in the data and captures the movement of the time series without the fluctuations.The formula for SMA is as follows:

$$SMA = \frac{A_1 + A_2 + ...A_n}{n} \tag{3.1}$$

where $A_1, A_2, ...A_n$ are the asset prices for n period.

Short-term averages are more sensitive to price changes as compared to long-term moving averages. Hence long-term SMA curves are smoother than short-term SMA curves. The most common use case of SMA is to identify if an asset is in an uptrend or downtrend.

The other type of moving average is the Exponential Moving Average(EMA). In EMA, the maximum weight is given to the current asset price, and the weights decrease exponentially as we move back in the time series. An exponential moving average is more sensitive to recent price changes as it gives more weight to current prices. Due to this, EMA follows the actual price trend more closely than SMA. The formula for EMA is as follows:

$$EMA_{Today} = (Value_{Today} * (\frac{Smoothing}{1 + Days})) + (EMA_{Yesterday} * (1 - \frac{Smoothing}{1 + Days})) \tag{3.2}$$

According to Appel (2005)[19], the exponential moving average is better than the simple moving average for identifying trends in a price series. For further details on the simple moving average, see Vandewalle et al. (1999)[20].

The study uses SMA and EMA as the baseline while comparing the performance of various time series models. In the survey, SMA and EMA of order 1 and order 60 are used to compare. An order of 60 is used because specific deep learning models use the past 60 day stock price as a feature to predict the next day's stock price.

3.1.2 ARIMA Model

ARIMA stands for AutoRegressive Integrated Moving Average. The ARIMA model is a generalization of the ARMA model that is suited to elucidate non-stationary time-series. The main advantage of using the ARIMA model is that it transforms a non-stationary series into a series without seasonality or trend by applying finite differencing of data points[5].

A stationary time series has its statistical properties constant over time. If it has no trend, its variance is around its mean and has a constant amplitude. Also, its autocorrelations remain constant over time. Based on these assumptions, a stationary time series could be considered a combination of signal and noise. The ARIMA model after separating the signal from noise, outputs a single step-ahead to produce forecasts.

An ARIMA model has three components, the Autoregression (AR) component, the Moving Average (MA) component, and the Integrated(I) differencing component [21].

Autoregression is a type of regression in which past lagged values determine the current value. The order of the lag can be determined by looking into the plots for autocorrelation or partial autocorrelation. The differencing component makes the time series stationary, and the differenced values replace the actual values. The Moving average (MA) part incorporates the dependency between an observation and a residual error that is obtained from a moving average model applied to lagged observations. The order of each of the three components is denoted by the terms p,

d, and q. Here p is the number of lag observations required in the AR model, d is the number of times the raw data needs to be differenced to make it stationary, and q is the size of the moving average window.

The full model can be written as:

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t. \tag{3.3}$$

where y' is the differenced series (it may have been differenced more than once). The right side of the equation include the lagged values of y' and the lagged errors.

In the study, the ARIMA model is implemented in Python with the help of the stat model library and the auto ARIMA library. ARIMA is used for both single-step forecast and multiple step forecast using the historical price as the only variable.

3.1.3 ARMA GARCH Model

The generalized autoregressive conditional heteroscedasticity (GARCH) was designed to model the volatility of financial assets. The equation of the GARCH model has two parts the conditional mean part and the conditional variance part. By representing the conditional mean equation as an ARMA(Autoregressive Moving Average) process, we can combine ARMA and GARCH to obtain an ARMA-GARCH model that can be used to predict the stock price.

To clearly understand the ARMA-GARCH model, one must understand the difference between unconditional mean and variance and conditional mean and variance. The unconditional mean and variance are the mean and variance of the time series distribution, which is constant over the time period considered.

In an ARMA-GARCH model, the prediction made is not the same as the current estimates. Instead, they are higher or lower than the mean over a short period of time; as the prediction horizon increases, the forecasts converge to long-term unconditional values.

The conditional mean equation of a GARCH model can be formulated in various ways. In our study, we expect that the series follows an autoregressive moving average, ARMA, model developed by Box and Jenkins. The GARCH model was introduced by Bollerslev[22], which is a generalization of the ARCH model, developed initially by Engle[23]. The ARCH model allows for many lags in the conditional variance, and the GARCH model extends it by also allowing for lags in the error terms.

The GARCH error parameter measures the reaction of the conditional volatility to market shocks. When the parameter is significant, the volatility is very sensitive to market changes. The GARCH lag parameter measures the persistence in conditional volatility, irrespective of the market. The sum of the two parameters determines the rate of convergence of the conditional volatility to the long-term average level.

The ARMA-GARCH model used ARMA for the linear part and GARCH for the residual part.

The GARCH error parameter, measures the reaction of the conditional volatility to market shocks. When the parameter is large, the volatility is very sensitive to market changes. The GARCH lag parameter, measures the persistence in conditional volatility, irrespective of the market. The sum of the two parameters determines the rate of convergence of the conditional volatility to the long-term average level

The ARMA-GARCH model used ARMA for the linear part and GARCH for the residual part.

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \epsilon_{i-1}^2 + + \sum_{i=1}^{p} \beta_i \sigma_{i-1}^2 \tag{3.4}$$

$$\epsilon_t = \sigma_t e_t \tag{3.5}$$

$$X_i^t = c + \epsilon_i + \sum_{i=1}^{p} \psi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \tag{3.6}$$

where parameters in equation 3.6 are given by ARMA and 3.4 are given by GARCH.

3.1.4 Recurrent Neural Network (RNN)

Recurrent Neural Networks is a class of deep neural networks designed to work on sequential data[24]. The usual fully connected deep neural networks are not designed for sequential data since the input size is fixed. The weights for each node at each layer are different, so they don't capture time-dependent features; hence deep neural networks can't remember information from the past. Recurrent Neural Networks have an internal state for every time step. At each time step in a recurrent neural network has two inputs, the previous internal state, and the new information. Both the internal state and the new data are multiplied by a weight matrix and stacked. Actual world tasks like stock market prediction or Natural Language Processing make use of sequential information. These real-world tasks cannot be handled by traditional neural networks since simple feed-forward neural networks or ANN assume that all inputs (and outputs) are independent. Hence, they would perform poorly with regards to predicting the next time-steps of the sequential information. Therefore to overcome such problems, researchers came up with the idea of Recurrent Neural Networks or RNNs. They called these neural architectures as RNNs because they perform the same task for every element of the sequence, with the output being dependent on the previous computations. In theory, these RNNs have the capability of making use of information in arbitrarily long sequences, however, in practice, they suffer from exploding or vanishing gradient problem that limits them to looking back only a few steps.

Unlike a traditional deep neural network, that uses different parameters at each layer, an RNN shares the same parameters ($W_h h, W_x h$ above) across all steps, as shown in Figure 3.1. This fact indicates that the RNNs perform the same task at

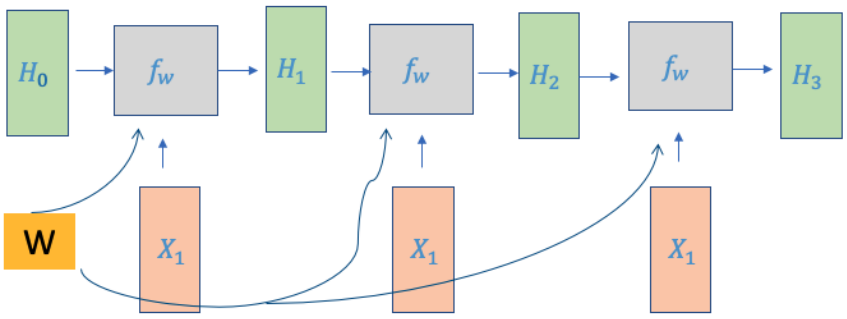


Figure 3.1 The image shows that the same weight is used for every time step.

each stage, just with different inputs. It dramatically reduces the total number of parameters the RNN needs to learn.

Training an RNN is similar to preparing a traditional Neural Network but with a bit of surprise. As the parameters are shared across all time steps in the network, the gradient at each output relies not only on the calculations of the current time step but also on the previous time steps. Vanilla RNNs trained with Back Propagation Through Time have difficulties learning long-term dependencies viz. dependencies between actions that are far apart) due to what is known as the vanishing/exploding gradient problem.

The Vanishing Gradient Problem RNNs have difficulties learning long-range dependencies - the interactions between inputs several time steps apart[9]. This is because the 2-norm of the Jacobian matrix has an upper bound of 1. We also see that the activation functions like tanh and sigmoid have derivatives of 0 at both ends of the real number line. Thus they have zero gradients and drive other gradients in previous layers to zero. Hence with small values in the matrix and the multiple matrix multiplications, the gradient values shrink fast and eventually, after a few time steps tend to zero. And the state at those steps that are far behind do not contribute to what we are learning.

To overcome the difficulties of the vanishing gradient problem, we need to initialize the weight matrices W of the RNN unit properly. Proper regularization techniques can also come in handy in solving the vanishing gradient problem. A much more desirable solution is to use ReLU in place of tanh or sigmoid activation functions. The derivative of ReLU is a constant of either zero or one; hence it is not probable to suffer from vanishing gradients. An even more popular solution is to deploy either Long Short - Term Memory (LSTM) or Gated Recurrent Unit (GRU) architectures. LSTMs came into existence in 1997 and are most widely used in NLP applications today. GRUs, first proposed in 2014, are simplified versions of LSTMs. Both of these RNN architectures were explicitly designed to deal with vanishing gradients and efficiently learn long-range dependencies.

3.1.5 Long Short Term Memory (LSTM)

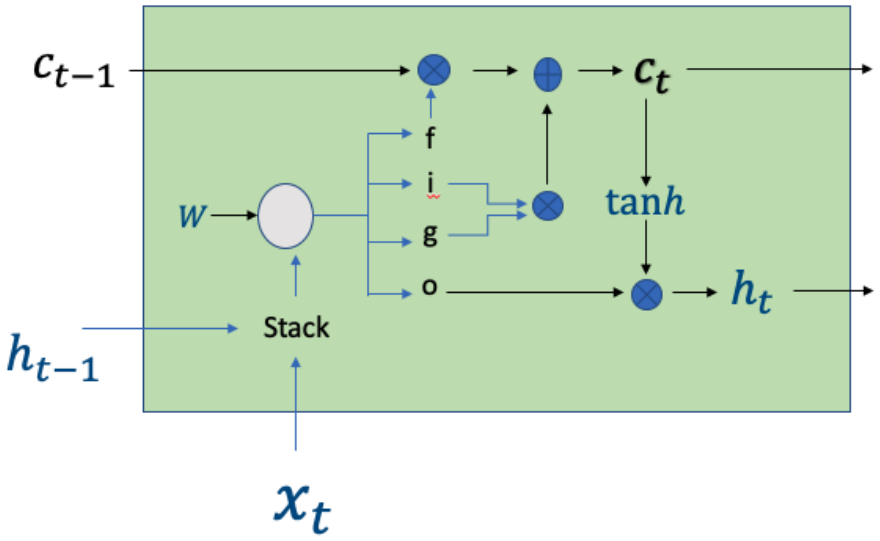


Figure 3.2 The LSTM Unit showing various gates.

The challenge to preserve long-term information and short-term input skipping existed for a long time. The earliest approaches to address this challenge were laid forward by Hochreiter Schmidhuber in the year 1997[9]. LSTM's design is motivated by the logic gates of a computer. The design lends control to manage the memory cell through several gates. As we look into Figure 3.2, we try to understand the implications of the gates and their mechanism. A gate is required to capture the entries from the cell. We refer to this gate as the output gate represented by the letter "o" in figure 3.2. A second gate is required to decide when to read data into the cell. This gate, referred to as the input gate, is represented by the letter "g" in figure 3.2. Finally, a mechanism to reset the cell's content is managed by a forget gate of the "f" gate, as shown in figure 3.2. The input at the current time step and data from the hidden state of the previous time step passes into the LSTM gates. Three fully connected layers process them with a sigmoid activation function to compute the values of the input, forget and output gates. The values of the three gates lie in the range (0,1).

The first step is to decide what information is to be discarded away from the cell state and this is done by the sigmoid layer which is called the forget gate. This gate looks at the vectors from previous hidden state and the current input, and outputs a value between zero and one for each number in the cell state from previous time step. A "one" represents retaining information from previous time steps in the cell state while a "zero" accounts for the information to be forgotten entirely.

The next step for the LSTM architecture lies in deciding what new information we are going to store in the cell state, and this is achieved in two stages. Firstly, a sigmoid layer called the "input gate layer" allows which values to update. Secondly, a tanh layer creates a vector of new candidate values, that gets added to the state. And the next step lies in combining the computations as mentioned earlier to update the state.

To finally update the old cell state into a new cell state, we multiply the old state by the output of the forget layer , omitting the things we decided to omit earlier. At the same time, we add the output of the input layer at time step (t) with the vector containing new candidate values.

The value of the hidden state depends upon the three gates, namely - the forget gate, the input gate, and the output gate. The forget gate is element-wise multiplied by the previous cell state and added to the output of element-wise multiplication between the input at time step (t) and the vector for new candidate values.

A slight variation of the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014)[25]. The model design combines the forget and input gates into a single "update gate." Furthermore, it even incorporates the cell state and hidden state and makes some other changes. The final model is more straightforward than standard LSTM models.

3.1.6 Gated Recurrent Unit (GRU)

In RNNs, a long product of matrices can lead to vanishing/exploding gradients. In practice, such gradient anomalies would be unfit to capture a logical breakdown between a bear and a bull market for securities. Then in such cases, it would be more meaningful to have a method of resetting our internal state representation. In GRUs [25], as seen in Figure 3.3, the hidden states have gates that permit for update and reset of the internal state portrayal.

The Reset Gate allows us to control how much of the previous state we might want to retain. This gate captures short-term dependencies.

The Update Gate allows us to control how much of the new state is just a copy of the old state. Unlike the reset gate, this gate captures long-term dependencies.

The reset gate and the update gate are both set to be vectors in the range between zero and one to perform convex combinations.

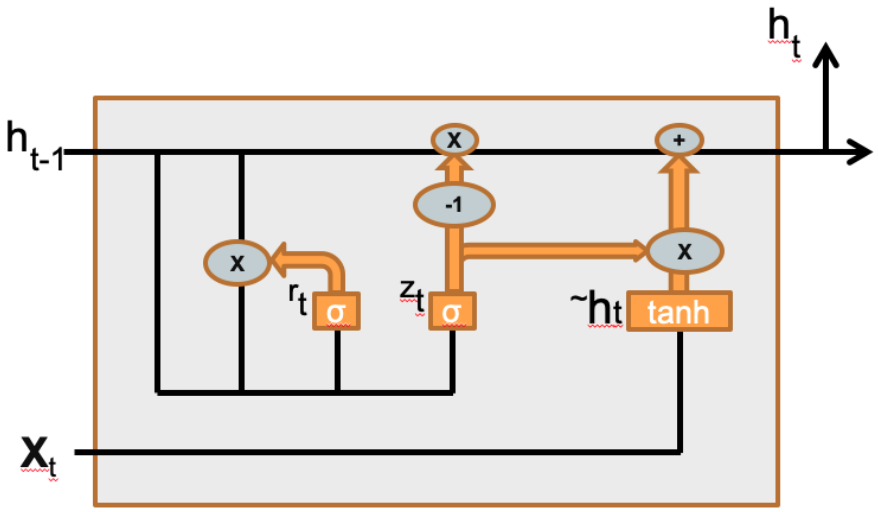


Figure 3.3 GRU

The reset gate and the update gate are both sigmoid layers. The sigmoid function transforms inputs between zero and one.

The following equation gives the candidate hidden state, and the elementwise Hadamard operator operates between the reset gate and the hidden state from the previous time step. We use the tanh nonlinearity to keep the values between -1 and 1. The influence of the previous state can be reduced with the elementwise multiplication of the reset gate and the hidden state of the previous time step. Whenever the entries in the reset gate are close to 1, we redeem a vanilla RNN. For entries of the reset gate that are close to 0, the candidate hidden state results from an MLP. Any pre-existing hidden state is thus reset to defaults.

Finally, we need to introduce the effect of the update gate Z_t. This gate helps us determine how the new hidden state is similar to the old hidden state and the degree to which the new candidate state would be utilized. The aforesaid purpose can be solved by using convex combinations between $\tilde{}H_t$ and H_t. When Z_t is close to one, we retain the old state and skip the current input X_t for that time step in the dependency chain. In contrast, when Z_t is close to zero, the new latent state approaches the candidate

latent state. These designs can help us alleviate the vanishing gradient problem in RNNs and better capture dependencies for sequences with large time step distances. Suppose for the entire time step of a subsequence; the update gate has been close to 1, then the old hidden state at the time step of its beginning will be easily captured and passed to its end, notwithstanding the length of the subsequence.

3.1.7 Bidirectional-LSTM/Bidirectional-GRU

In sequence learning, so far, our goal was to model the subsequent output given a set of historical data in the paradigm of a time series. While it is a typical scenario, it is not the only one we encounter.

Bidirectional RNNs(LSTMs) as shown in Figure 3.4 were introduced by [Schuster Paliwal, 1997][28].

In a bidirectional RNN, information from both ends of the sequence estimates the output. In training the model, we use both the past and the future observations to predict the current one. In practice, however, we do not have the privilege of looking into the future time steps of the sequences. It creates a limitation in the BRNN models, and hence the model will underperform during testing In addition to this, BRNNs are very slow to train. The prominent reasons for this are that the forward propagation requires both forward and backward recursions in bidirectional layers. The backpropagation depends on the outcomes of the forward propagation. Hence, gradients would have a very long dependency chain.

Bidirectional layers are seldom in use due to computation expense from an application's point of view. Its applications lie, namely filling in missing words, annotating tokens (e.g., for named entity recognition), and encoding sequences wholesale as a step in a sequence processing pipeline (e.g., machine translation).

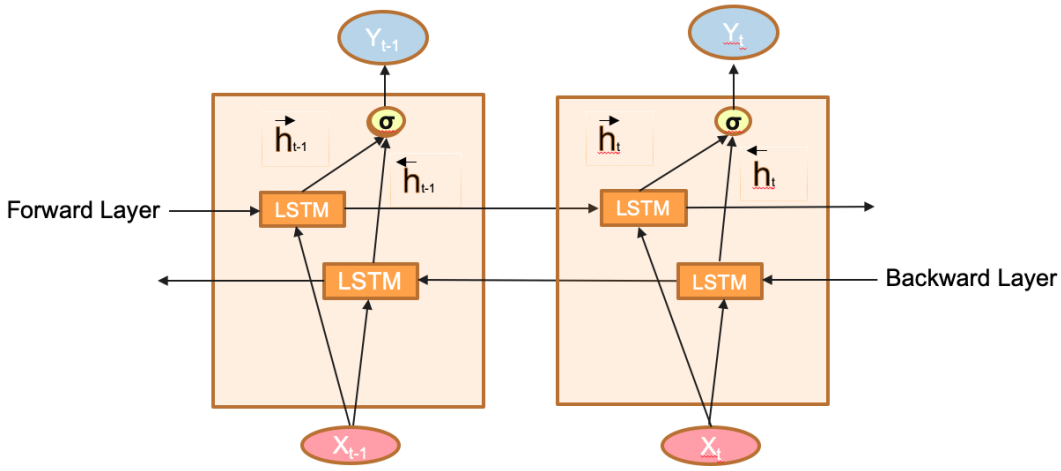


Figure 3.4 BI-LSTM

3.1.8 Temporal Convolutional Network

Temporal Convolutional Networks (TCNs) are a class of time-series models that overcame the earlier hurdles by capturing long-range patterns using the order of temporal convolutional filters. There are essentially two types of TCNs: The EncoderDecoder TCN(ED-TCN) that makes use of an order of temporal convolutions, pooling, and upsampling but can competently capture long-range temporal patterns. A Dilated TCN employs dilated convolutions in place of pooling and upsampling and includes skip connections between layers. TCNs are an adaptation of the recent WaveNet [29] model. The Dilated TCN has more layers, but each uses dilated filters that only operate on a small number of time steps.

We define TCNs which have the following properties: (1) computations are layer-wise, meaning every time-step is updated simultaneously, instead of updating sequentially per-frame (2) convolutions compute across time, and (3)predictions at each frame are a function of a fixed length of time and referred to as the receptive field.

The TCN relies upon two principles: Firstly, the network's output is of the same length as that of the input. Secondly, there is no information leakage from the future

time steps to the past. The TCN employs a 1D fully-convolutional network (FCN) architecture, and each hidden layer is of the same length as that of the input layer. Zero paddings of size (kernel size-1) maintain succeeding layers of the same length as previous ones. Also, TCNs, through causal convolutions, convolve an output at time t only with elements from time t and earlier in the previous layer. Simply put: TCN = 1D FCN + causal convolutions.

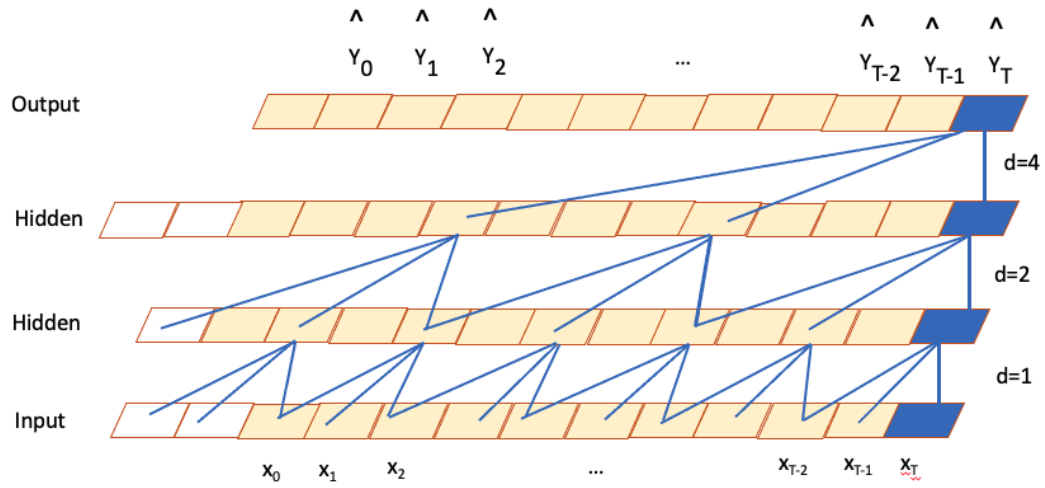


Figure 3.5 Dilated Temporal Convolutional Network (TCN)

Dilated Convolutions A significant drawback of a simple causal convolution is that it can only access a history of time steps linearly in size with the depth of the network. It makes it demanding to apply the aforementioned causal convolution on sequence tasks, especially those requiring more extended history. To overcome this drawback, we employ dilated convolutions that let an exponentially larger receptive field. In other words, for a 1-D sequence input $x \epsilon R^n$ and a filter $f : \{0, ..., (k-1)\} \Rightarrow R$, the dilated convolution operation F on element s of the sequence is defined as

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i).x_{(s-d).i} \tag{3.7}$$

here d is the dilation factor, k is the filter size, and s d ·i holds for the direction of past time steps. Dilation is hence parallel to introducing a fixed step between every two adjacent filter taps. When d = 1, a dilated convolution reduces to a regular convolution. Using larger dilation facilitates an output at the top level to represent a broader range of inputs, therefore effectively expanding the receptive field of a ConvNet. It thus provides us with a pair of ways to increase the receptive field of the TCN: choosing larger filter sizes k and increasing the dilation factor d, where the adequate history of one such layer is (k-1)d. As is usual, while using dilated convolutions, we increase d exponentially with the depth of the network (i.e., d = $O(2^i)$ at level i of the network). This makes it possible that there is at least some filter that maps each input within the effective history while also permitting for an extremely large effective history using deep networks. We provide an illustration in Figure 1(a).

There are notable advantages to using TCNs, and they are as follows :

- Parallelism: Convolutions come with an inherent advantage, i.e., it can be done in parallel as the same filter is used in each layer. Hence, in both training and evaluation, along input sequence can be operated as a whole in TCN, instead of sequentially as in RNN.

- Flexible receptive field size: TCNs confer the unique ability to control the model's memory size better and are simple to adapt to different domains. We can manipulate TCNs receptive field size through stacking more dilated (causal) convolutional layers, using more significant dilation factors increasing the filter size are viable options.

- Stable gradients: TCN has a backpropagation path different from the temporal direction of the sequence. Hence, it can overcome the problem of exploding/vanishing gradients, a unique challenge in RNNs.

- Low memory requirement for training: Owing to the model architectures of LSTMs and GRUs, they quickly use a lot of memory to store the partial results of their multiple cell gates. While, in a TCN, the filters are shared across a layer, with the backpropagation depending only on network depth. So in practice, a gated RNN architecture is likely to use multiplicative factor more memory than TCNs.

- Variable-length inputs: Just like RNNs, which model inputs with arbitrary lengths in a recurrent way, TCNs can also take in inputs of variable sizes by sliding the 1D convolutional kernels. As a result, TCNs could be replacements for RNNs for sequential variable-length data.

While there are also two significant disadvantages to using TCNs and they are as follows:

• Data storage during evaluation: RNNs in order to generate a prediction only need to maintain a hidden state and take in a current input x_t. A synopsis of the entire history is provided by the fixed-length set of vectors h_t, while the actual observed sequence can be eliminated. While TCNs works to take in the raw sequence up to the effective history length, therefore possibly requiring more memory during evaluation/testing.

• Potential parameter change for a transfer of domain: Different domains have their own requirement on the amount of past time-steps the model gets trained on in order to predict accurately. So while transferring a model from one domain where little memory is required (i.e., small k and d) to a domain where much longer memory (i.e., much larger k and d) is required, TCN may not perform as expected for not having large enough receptive field.

3.1.9 Transfer Learning

Transfer learning improves a learner from one domain by transferring information from a related field [29]. Many machine learning applications use transfer learning, including text sentiment classification, image classification, human activity classification, and multi-language text classification.

Here we use data of other technology companies to learn the price movements, and then we use the pre-trained model to predict the price for our test data. The transfer of knowledge occurs by learning the price movement of other technology companies and using it to predict the prices of given technology stock. Transfer learning relaxes the hypothesis that the training data must be independent and identically distributed (i.i.d.) with the test data, which motivates us to use transfer learning to solve the problem of insufficient training data[30]. There are only 253 data points (trading days) per year; hence it is a small dataset, and we can use Transfer Learning to learn about price movements from similar stocks.

According to Sirignano et al. [16], it is better to fit a single model to multiple stock price data and use that single model to make predictions. The universal model trained on data from all stocks outperforms, in terms of out-of-sample prediction accuracy[16], asset-specific linear and nonlinear models that get trained on time series of any given stock, show that the universal nature of price formation weighs in favor of pooling together financial data from various stocks, instead of designing asset or sector-specific models as usually done. It is assumed that the stock price of different companies have similar underlying dynamics. The model tries to learn these underlying dynamics to predict the price for a given stock better.

Our study took stock of ten of the largest technology companies and trained our models on the historical prices of these ten stocks. The trained model is then used to predict the prices for Apple, Tesla, and Facebook.

3.2 Data

In the study mainly the data of three technology companies are used namely, Apple, Facebook and Tesla. The data ranges from 1st January 2015 to 22nd December 2020 and it includes the daily prices of each of the stocks. Below are the plots of the stock price for each company over the six year time period.

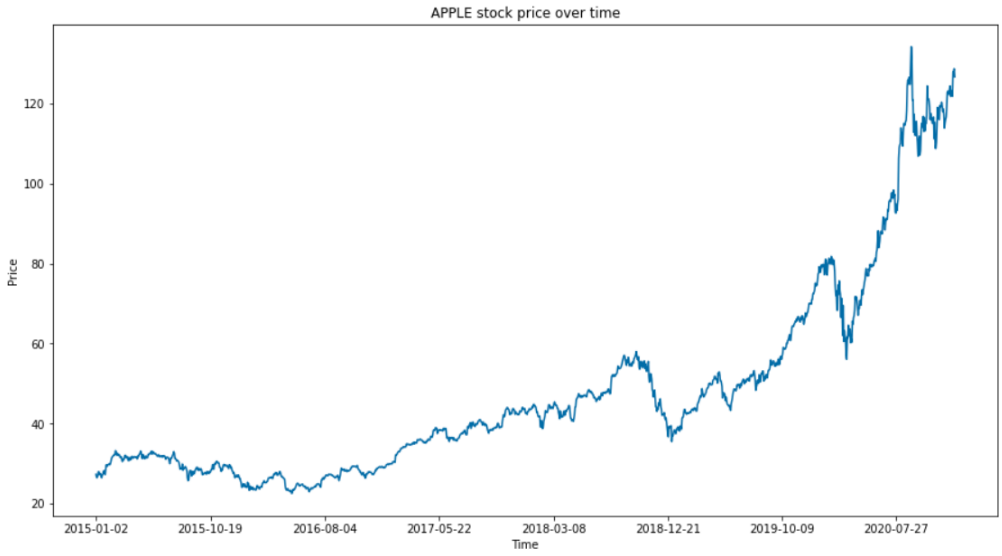


Figure 3.6 Apple Stock Price from beginning of 2015 till end of 2020

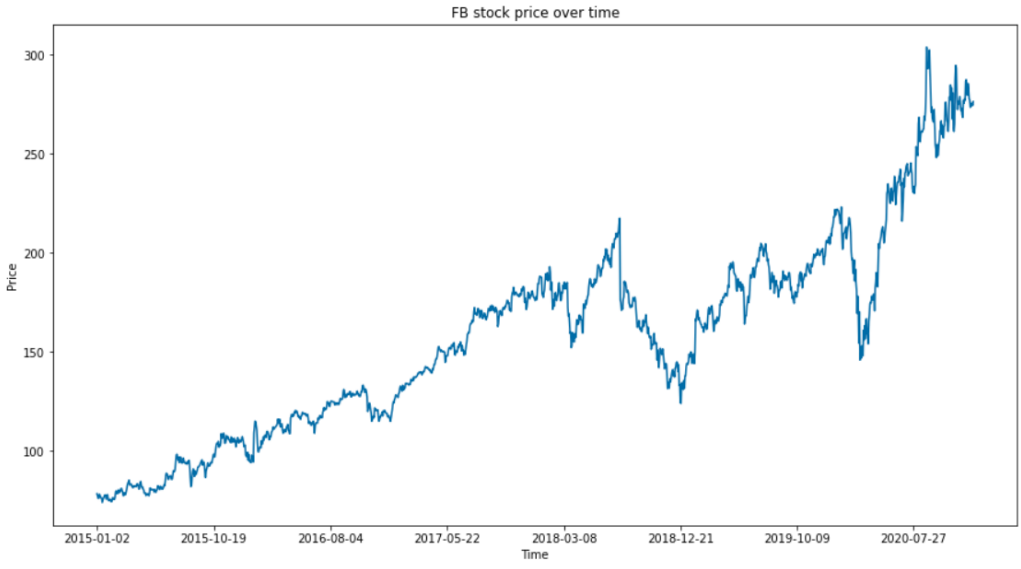


Figure 3.7 Facebook Stock Price from beginning of 2015 till end of 2020

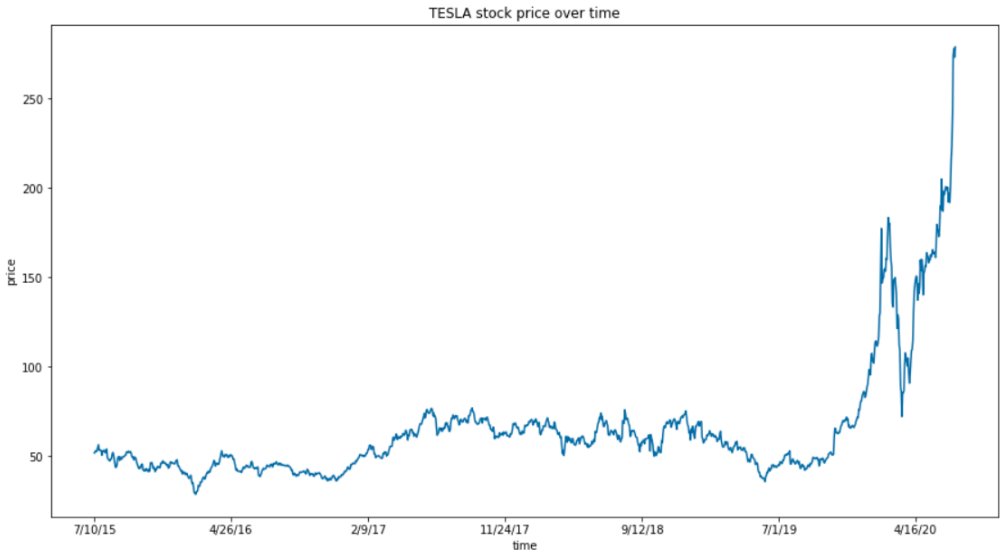


Figure 3.8 Tesla Stock Price from beginning of 2015 till end of 2020

Since it is time series data we use various time series modelling techniques to predict the stock price. For the analysis the data is split into 70:30 ratio for training and testing.

3.2.1 Data for Univariate Models

For modelling by sequential deep learning models the data is converted into fragments of window of size 60. So to predict the price of the 61st day we look into the values of the past 60 days. For example, if we consider Apple, then there are 1503 rows of data in total. For training we consider 70 percent of it, that is 1052 rows. For univariate time series modelling the only feature considered is the historical closing price. Thus the shape of the input data is [No. of trainig rows - 60, 60, 1]. Therefore, in the case of Apple Inc.,the input vector for training our deep learning model build in Keras is a 3D vector of the shape [992,60,1]. The performance of the best performing model is compared for different window sizes to look for trends in window size versus performance of the model. The result of it is discussed in the results section.

We deploy two other deep learning models, namely, CNN and CNN-GRU. For these two models the input shape is different as compared to all of the other deep learning models as described in the previous paragraph. In CNN-GRU, the CNN layer comes first, followed by the GRU layer. A 1D convolution operation acts in both the architectures on each slice of the input vector. Meaning, for every time step, a convolution operation is performed. The shape of the input vector for the one-time step is [60,1] as we take a window size of 60.

For statistical modelling like ARIMA and ARMA GARCH, the data is differenced to make it stationary for modelling. In ARIMA, there is little difference in performance if we use just the past 60 days or the entire history to predict the stock price.

3.2.2 Data for Multivariate Models

For multivariate modeling, three different datasets are picked to improve the accuracy of our prediction. The datasets included are the Nasdaq Composite Index, the Nasdaq Technology 100 index, and the ten-year yield of a treasury bill. The Nasdaq Technology 100 index comprises 100 technology companies that represent the internet, software, electronics, and other fields. The Nasdaq Technology 100 index dataset is incorporated as there is usually a correlation between the stock price movement of a technology company and the change in the price of the related market index. The Nasdaq 100 Technology index is an index specific to the Technology sector, whereas the Nasdaq composite index represents every sector in the market.

The Nasdaq composite can be considered a representation of how well a country's economy is doing and whether the market is in a bull run or bear run. The final dataset chosen is the ten-year yield data; this dataset shows the people's confidence in the United States economy. If the people are confident that the United States economy will do well, they will not invest in treasury bills resulting in a fall

in the price of the bills and an increase in yield. Treasury bills are considered a safe investment, and hence when the economy does well, the investors like to take risks in stock markets rather than invest in the safe Treasury bills, which have a low return. All these datasets are chosen to keep in mind that market and economic factors affect the price of a stock. The results of the analysis are presented in the results section.

3.3 Analysis

3.3.1 Single Step Forecasting

Several different architectures are considered for univariate single-step forecasting. All the different models are presented one after the other. First, the statistical models are presented, followed by the deep learning models.

ARIMA Model We find out the parameters of the ARIMA model using the auto arima library in Python. The parameters are chosen based on AIC values. The model which has the least AIC is selected.

The model is fitted on the historical data, and the stock price is predicted for the next day. As we predict the stock price for each day, we add the stock's actual price on that day to the historical data and fit the model again to make the following prediction. The advantage of the ARIMA model is that it fits the data very quickly; hence fitting the model at each step is not time taking.

ARMA GARCH Model For fitting the ARMA GARCH model. The time-series data are differenced to make it stationary. An ARMA model is fit for the mean equation, and a GARCH model is fit for the variance part. The stats model library and the arch library in Python are used to model the data. The different parameters of the model are derived from this two-equation. The model is used to predict the log return rate, which is then inverse transformed to predict the stock price.

RNN/LSTM/GRU/Bi-LSTM/Bi-GRU Model The models were implemented in Keras. A single layer of LSTM was used to model the given time series from three companies namely - Apple, Tesla, and Facebook. The LSTM layer had seven hundred units, and the input shape had a 60-day window for each time step. A dropout layer with a dropout parameter of 0.1 was added to the architecture. Dropout is proposed to inject noise into the network layer before calculating the subsequent hidden units during training. This idea enforces regularization. The method is called dropout because we drop out ten percent of neurons during training.

```
np.random.seed(1)
model = Sequential()
model.add(LSTM(units = 700, input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.1))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train, y_train, epochs = 30, batch_size = 32)
```

Figure 3.9 LSTM Architecture and it's parameters.

The model was compiled using the Adam optimizer, and loss was reported as a mean squared error loss. Adam realizes the gains of both SGD with momentum and RMSProp.Precisely, the Adam optimizer calculates an exponential moving average of the gradient and the squared gradient while the parameters β_1 and β_2 control the decay rates of these moving averages.

LSTM GRU Model We combine both the LSTM and the GRU models into a single model to take advantage of both the LSTM layer and the GRU layer. The idea behind this is that it may be possible that the LSTM may underperform, whereas the GRU over-predicts the price. To compensate for the shortcomings of each layer, we combine the two layers.

```
np.random.seed(1)
model = Sequential()
model.add(LSTM(units = 700, return_sequences=True, input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.1))
model.add(GRU(units = 700,  input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.1))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train, y_train, epochs = 30, batch_size = 32)
```

Figure 3.10 LSTM Architecture and it's parameters.

In this architecture we employ the LSTM layer first and then the GRU layer. We also determined that the alternate arrangement, i.e., the GRU layer first and then the LSTM layer, and observed that it underperformed the prediction task.

CNN Model The Convolutional Neural Network Model is implemented to check whether there is any temporal dependence in the data. IF the data does not have temporal dependence, then the CNN will be able to model the data and predict the next day's price.

```
np.random.seed(1)
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=128, kernel_size=11, activation='relu')
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train, y_train, epochs = 30, batch_size = 32)
```

Figure 3.11 CNN Architecture and its parameters.

In the model architecture, a time-distributed 1D convolution is applied. The time distributed layer helps to apply convolution to each time step of the data. The 1D convolution layer is followed by a max pool layer and a dense layer to predict the next day's price. The 1D convolution layer applies 128 filters with a kernel size of 11. These hyperparameters were selected after tuning the deep learning model to achieve better performance.

CNN GRU Model A CNN-GRU model is implemented to extract features from the time series through the CNN layer and then pass the output features to a GRU layer. We prefer the GRU layer over an LSTM layer as the GRU layer performs better as compared to the LSTM layer in the univariate single-step forecasting problem.

```
np.random.seed(1)
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu')
model.add(TimeDistributed(MaxPooling1D(pool_size=1)))
model.add(TimeDistributed(Flatten()))
model.add(GRU(units = 700))
model.add(Dropout(0.1))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train, y_train, epochs = 30, batch_size = 32)
```

Figure 3.12 CNN GRU Architecture and it's parameters.

Similar to the previous CNN architecture, the CNN-GRU architecture has a time distributed 1D convolution layer followed by a GRU layer. The convolution layer extracts features from the time series and passes it on to the GRU layer. The GRU layer is followed by a dense layer that predicts the next day's price.

TCN Model The model uses only one TCN stack as more stacks are not needed. A kernel of size four is used to cover the window length, and the maximum dilation is 16. The number of filters is kept as 64 after testing with a different number of filters. There are no skip connections in the TCN model as the skip connections reduce the model's performance.

```
model = Sequential()
model.add(TCN(nb_filters=64, kernel_size=4, nb_stacks=1, dilations=[1, 2, 4, 8, 16 ]
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train, y_train, epochs = 100, batch_size = 32)
```

Figure 3.13 TCN Architecture and it's parameters.

GRU + TCN Model The model has two branches, namely, a GRU branch and a TCN branch. The same input is given to both the branches and the embedding of each of these branches is then concatenated to form a single output which is then passed through a single dense unit to give the output of the model. This model tries to take advantage of both the sequential GRU model and the hierarchical convolution model.

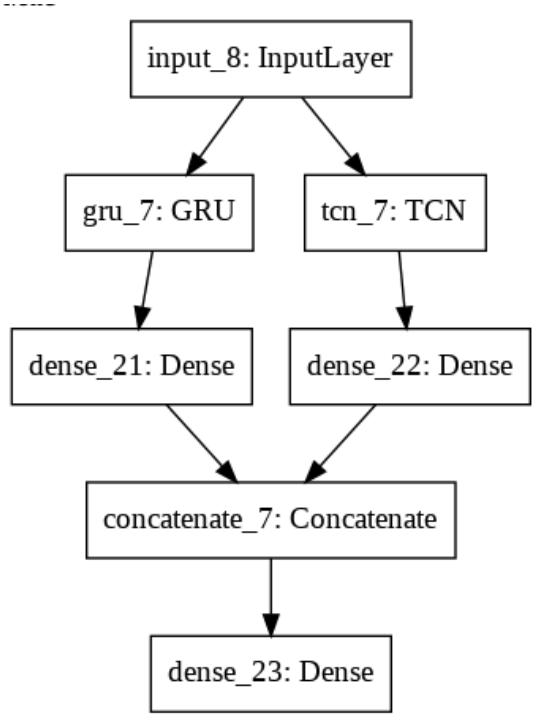


Figure 3.14 GRU + TCN Architecture and it's parameters.

3.3.2 Multivariate Forecasting

As mentioned in the Data section, we have considered three more datasets (Nasdaq Composite, Nasdaq 100 Technology Index, ten-year Treasury Bill yield) to help us determine the price of the stocks. Together with the three different datasets, the open, high, low, volume, and adjusted close columns for each stock are also considered features. In total, there are nine features to predict the stock price. The best performing univariate model is chosen for the multivariate analysis. Below is a

snapshot of the network model and input shape used for multivariate stock price prediction. The result of this analysis, along with a comparison with the univariate model performance, is provided in the results section.

```
model = Sequential()
model.add(GRU(units = 300 , input_shape = (X_train90.shape[1], 9)))
model.add(Dropout(0.1))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train90, y_train90, epochs = 30, batch_size = 32)
```

Figure 3.15 Multivariate Model Architecture with nine input features.

3.3.3 Multi Step Forecasting

From the results of single-step forecasting, we come to know the best performing networks for most of the cases. We compare the performance of these networks with different modifications suited for making a multistep prediction.

In the study, we predict the stock price for the next 3 days, 5 days, and 7 days. The error is measured by the mean absolute error and mean squared error metrics.

Multistep ARIMA We use the stats model library to build the ARIMA model. The parameters of the model are chosen with the help of AIC. We fit the model into the complete historical data and predict the value for the next seven days. As we walk forward, we add the actual price to the historic price, fit the model again, and predict the next horizon.

Multistep GRU For multistep prediction using GRU, we change the number of units in the final dense layer to be equal to the number of days we want to predict.

Seq - Seq GRU A sequence to sequence network consists of an encoder and decoder. The encoder encodes the information and reduces the dimension of the

input data. The decoder takes the reduced information and tries to recreate the original data. Here we use a similar network to input historical prices and output a sequence of futures prices.

```
model = Sequential()
model.add(GRU(units = 700, input_shape = (X_train90.shape[1], 1)))
model.add(Dropout(0.1))
model.add(RepeatVector(7))
model.add(GRU(700, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(100, activation='relu')))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train90, y_train90, epochs = 30, batch_size = 32)
```

Figure 3.16 Using Seq - Seq GRU for multistep forecasting.

The network architecture consists of a GRU encoder, followed by several repeat vectors once for each time step in the output sequence. This sequence of vectors will be presented to the GRU decoder. The decoder layer has 700 units, and time-distributed dense layers follow it. In general, we expect the sequence to sequence model to perform better than the multistep GRU model as it is a more complex architecture.

Multistep TCN For multistep prediction using TCN, we change the number of units in the final dense layer to be equal to the number of days we want to predict.

3.3.4 Transfer Learning

In transfer learning, the model is first trained on other stock datasets to learn the parameters and then make predictions for that specific stock. In our study, we try to find which model out of TCN and GRU is better able to learn the price dynamics and predict the stock price of a given company. We train the GRU and the TCN models on ten different technology stocks and then use the trained model to make the predictions. The models used are the same as the GRU and TCN models previously

used. The GRU model has a single GRU layer with 700 units. The TCN model has a single stack with 64 filters and maximum dilation of 16.

```
model = Sequential()
model.add(GRU(units = 700, input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train, y_train, epochs = 30, batch_size = 32)
```

Figure 3.17 GRU Transfer Learning Model Architecture.

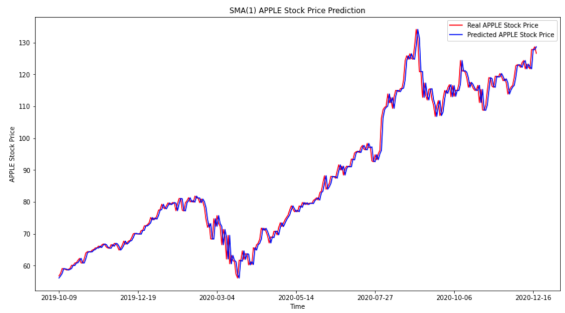
CHAPTER 4

RESULTS

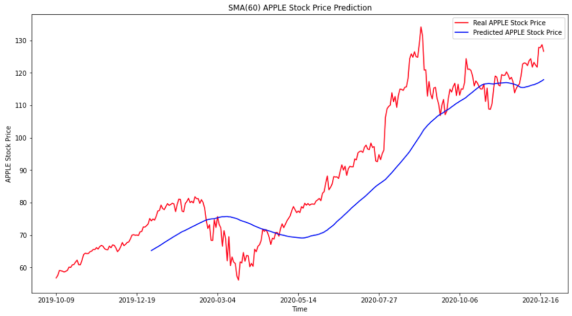
4.0.1 Univariate Single Step Forecasting

We would look into the results of Apple Inc., followed by Facebook Inc., and finally into the results for Tesla Inc. The Red-colored Line represents the actual price in each subplot, while the Blue-colored line shows the predicted price. Each subplot has a caption that informs about the model it represents, the mean absolute error, and the root mean squared error.

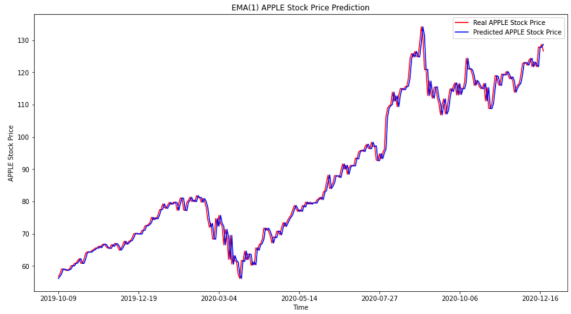
Apple Stock Price Prediction As we see in Figure 4.1, the best models are ARIMA (e), GRU(i), LSTM GRU (j), Bi-GRU(l), and CNN GRU (n). The mean absolute error for each of these models is less than the MA(1) and EMA(1) baseline models. From the deep learning model performances, we can note that the GRU layer can learn the dynamics of price movement better than any other layer. This may be attributed to the structure of the GRU cell.



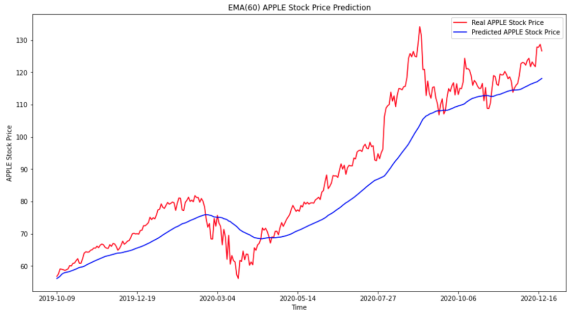
(a) MA(1) (1.61,2.37)



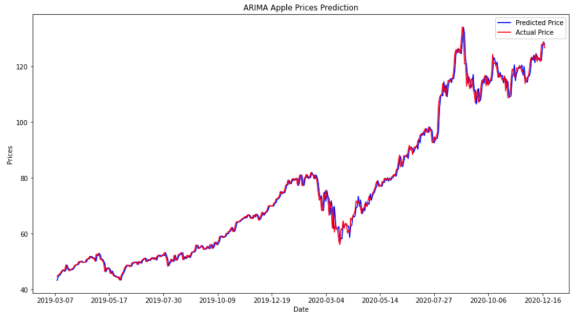
(b) MA(60) (9.34,11.51)



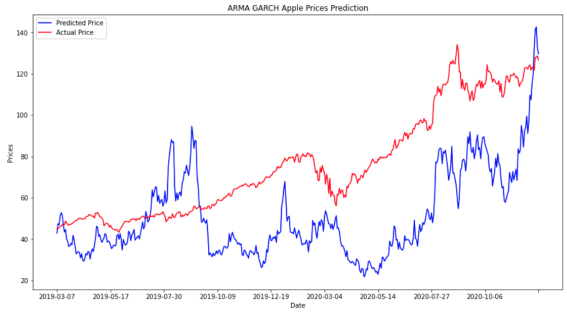
(c) EMA(1) (1.61,2.37)



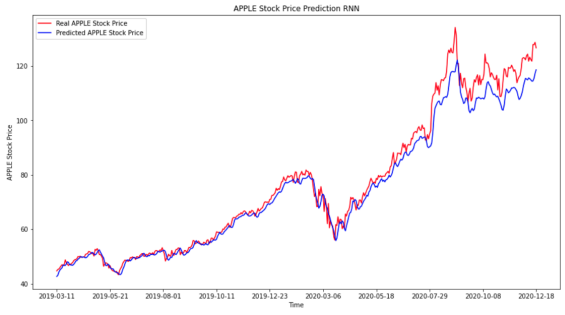
(d) EMA(60) (8.29,10.11)



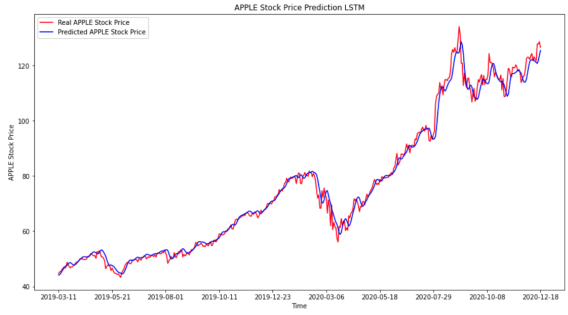
(e) ARIMA (1.30,1.99)



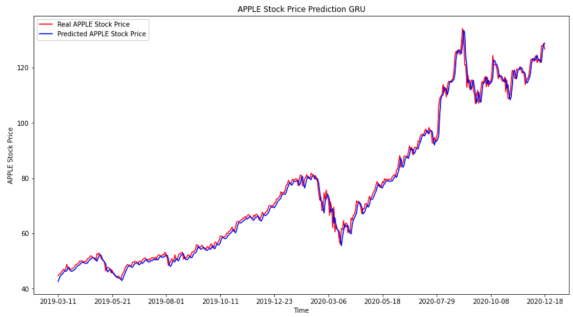
(f) ARMA GARCH (28.45,32.75)



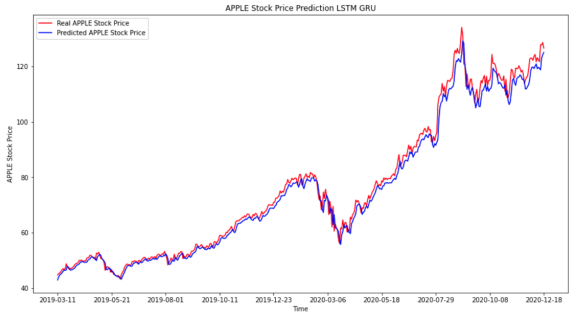
(g) RNN(2.39,4.29)



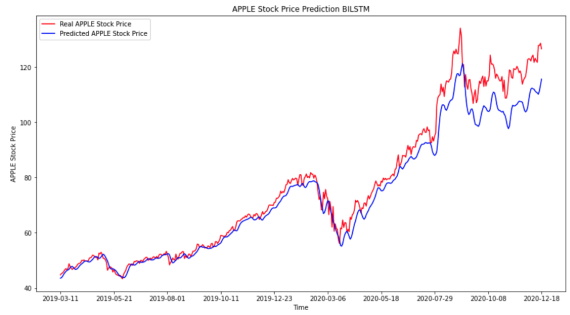
(h) LSTM(1.65,2.54)



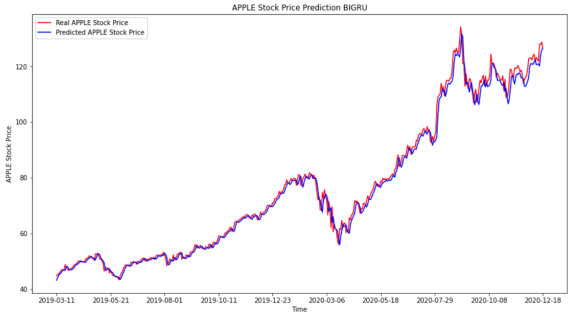
(i) GRU(1.49,2.11)



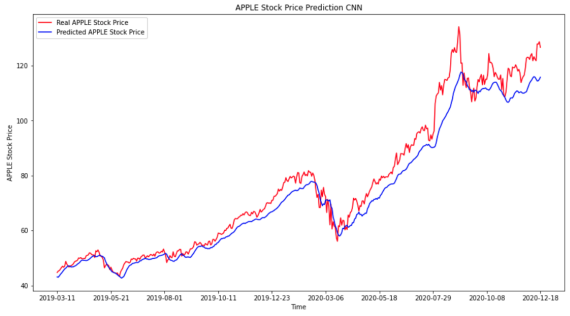
(j) LSTM-GRU(1.29,1.99)



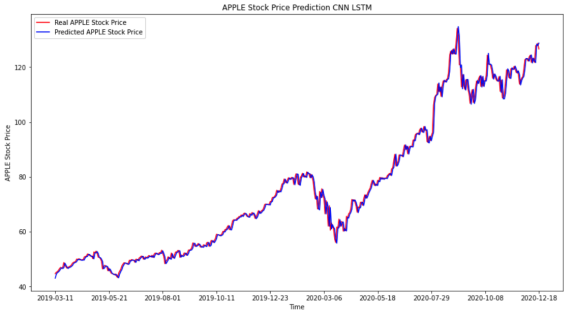
(k) Bi-LSTM(2.80,3.94)



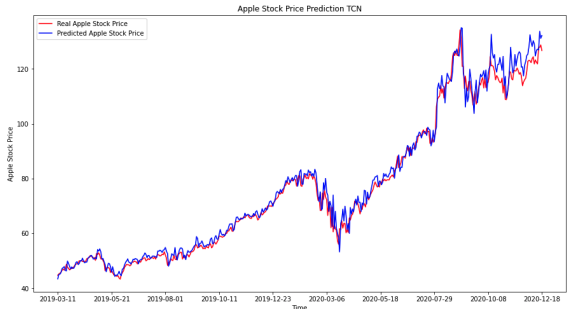
(l) Bi-GRU(1.52,2.24)



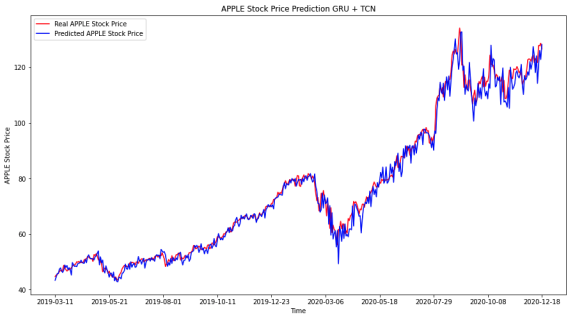
(m) CNN (2.50,3.62)



(n) CNN-GRU (1.32,2.09)



(o) TCN (2.11,3.10)



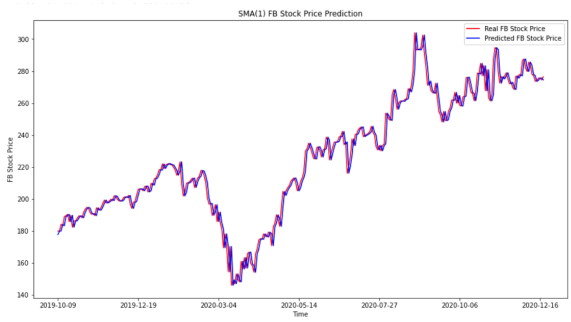
(p) GRU + TCN (2.04,3.03)

Figure 4.1 Apple Stock Price prediction. (a) to (d) showcase the performance of the baseline models, (e) represents the ARIMA model. (g) to (p) represent the deep learning architectures.

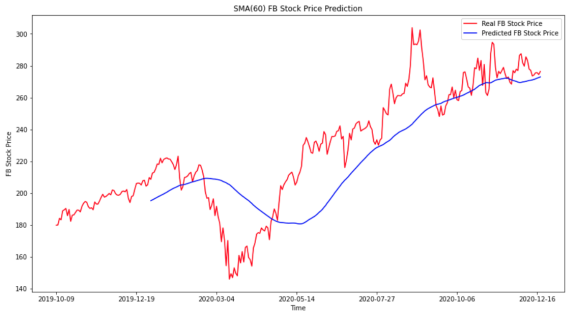
Table 4.1 Comparative Table for Apple Stock Price Prediction

Algorithm	MAE	RMSE
ARIMA	1.30	1.99
ARMA-GARCH	28.45	32.75
SMA(1)	1.61	2.37
EMA(1)	1.61	2.37
SMA(60)	9.34	11.51
EMA(60)	8.29	10.11
RNN	2.39	4.29
LSTM	1.65	2.54
GRU	1.49	2.11
Bi-LSTM	2.80	3.94
Bi-GRU	1.52	2.24
LSTM-GRU	1.29	1.99
CNN	2.50	3.62
CNN-GRU	1.32	2.09
TCN	2.11	3.10
GRU + TCN	2.04	3.03

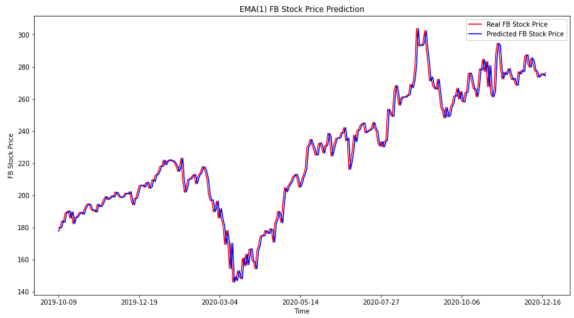
Facebook Stock Price Prediction As we see in Figure 4.2, the best models are ARIMA (e), GRU(i), Bi-GRU(l), TCN(o) and GRU+TCN (p). The mean absolute error of only the ARIMA models is equal to the MA(1) and EMA(1) baseline models. Here too, the GRU architecture outperforms the other deep learning architectures. This is a common trend across all three stocks. The TCN models also have a low error, and the performance is comparable to that of GRU models



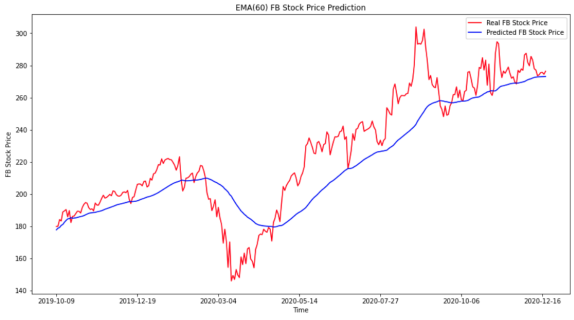
(a) SMA(1) (4.04,5.81)



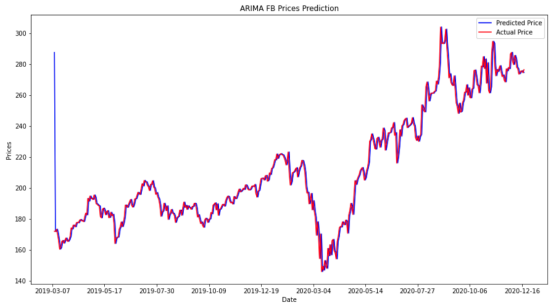
(b) SMA(60) (19.86,24.92)



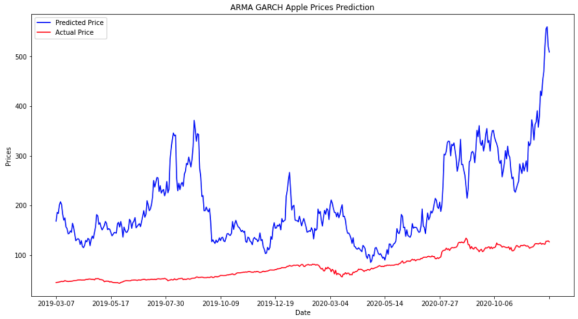
(c) EMA(1) (4.04,5.81)



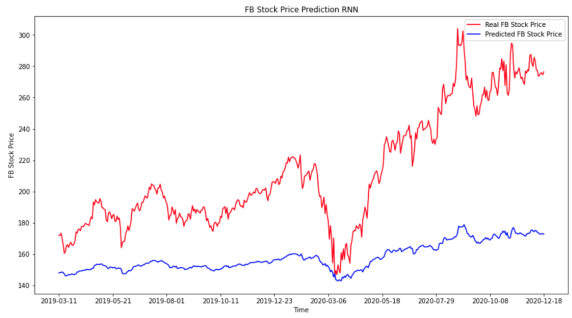
(d) EMA (60) (17.08,21.10)



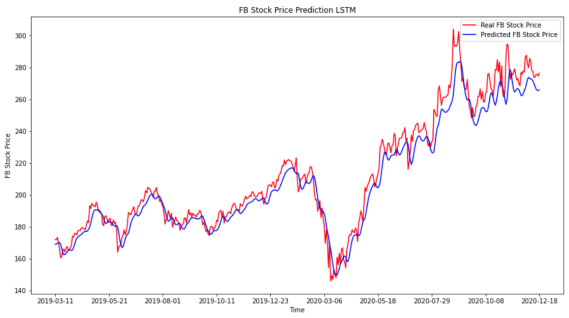
(e) ARIMA(3.73,7.43)



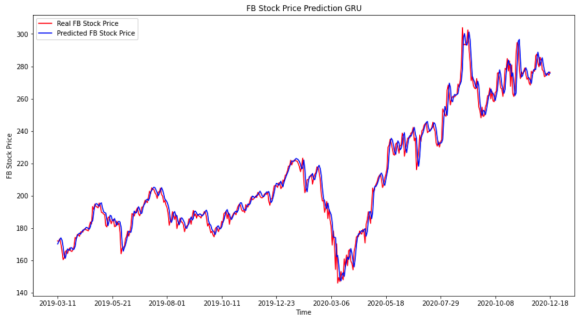
(f) ARMA GARCH(58.15,70.85)



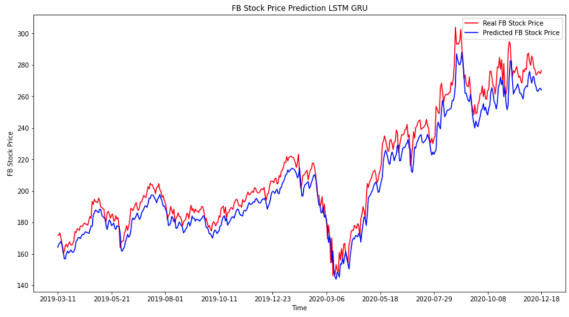
(g) RNN (54.36,61.39)



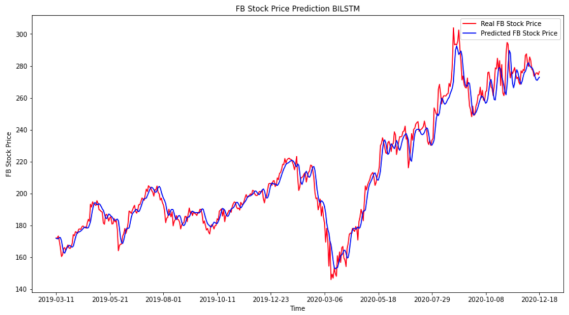
(h) LSTM(6.51,8.53)



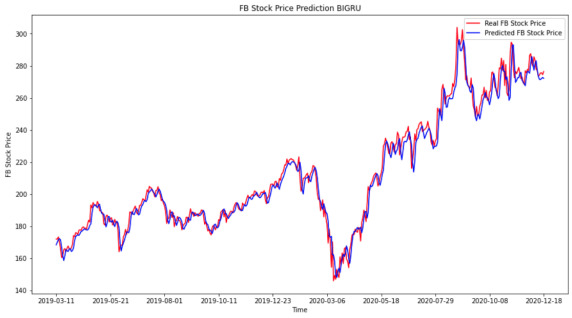
(i) GRU(3.67,5.29)



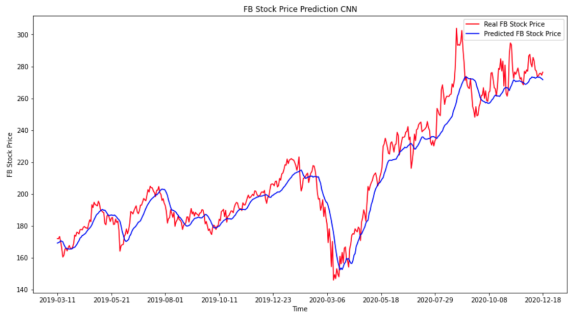
(j) LSTM-GRU(7.88,9.19)



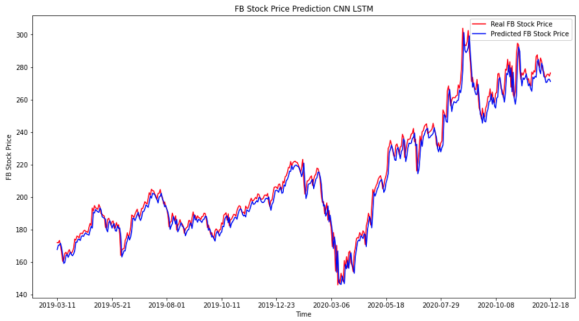
(k) Bi-LSTM(4.55,6.37)



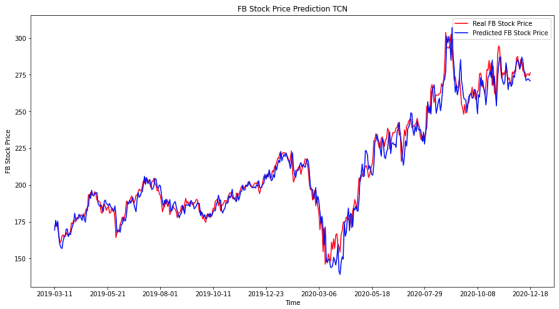
(l) Bi-GRU(4.033,5.60)



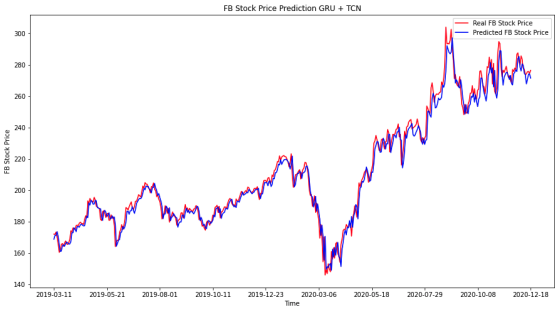
(m) CNN(7.59,9.99)



(n) CNN-GRU(4.16,5.59)



(o) TCN(4.13,5.69)



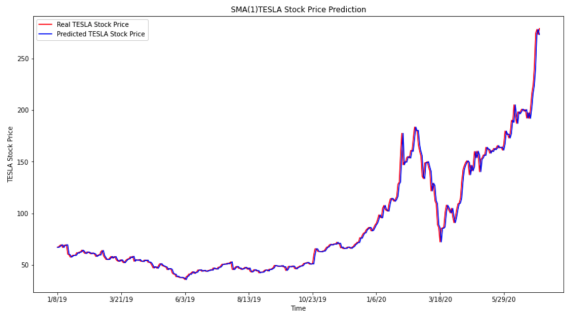
(p) GRU + TCN(3.90,5.52)

Figure 4.2 Facebook Stock Price prediction. (a) to (d) showcase the performance of the baseline models, (e) represents the ARIMA model. (f) represents the ARMA GARCH model and it underforms as compared to the ARIMA model. (g) to (p) represent the deep learning architectures.

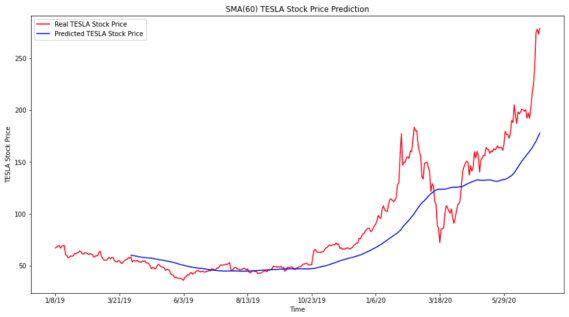
Table 4.2 Comparative Table for Facebook Stock Price Prediction

Algorithm	MAE	RMSE
ARIMA	3.73	7.43
ARMA-GARCH	58.15	70.85
SMA(1)	4.04	5.81
EMA(1)	4.04	5.81
SMA(60)	19.86	24.92
EMA(60)	17.08	21.10
RNN	54.36	61.39
LSTM	6.51	8.53
GRU	3.67	5.29
Bi-LSTM	4.55	6.37
Bi-GRU	4.033	5.60
LSTM-GRU	7.88	9.19
CNN	7.59	9.99
CNN-GRU	4.16	5.59
TCN	4.13	5.69
GRU + TCN	3.90	5.52

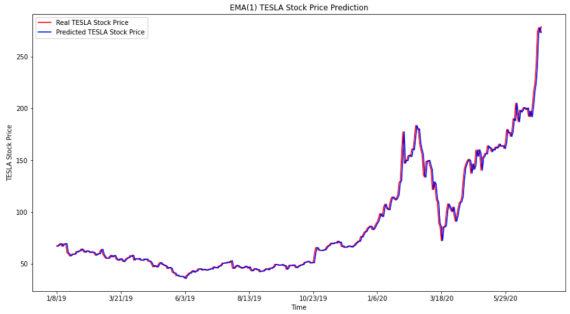
Tesla Stock Price Prediction As we see in Figure 4.3, the best models are ARIMA (e), GRU(i), Bi-GRU(l), TCN(o) and GRU+TCN (p). The mean absolute error of only the ARIMA models is equal to the MA(1) and EMA(1) baseline models. Here as well the GRU arhitecture outperforms the other deep learning architectures. It is a common trend across all the three stocks. The TCN models also have low error and the performance is comparable to that of GRU models.



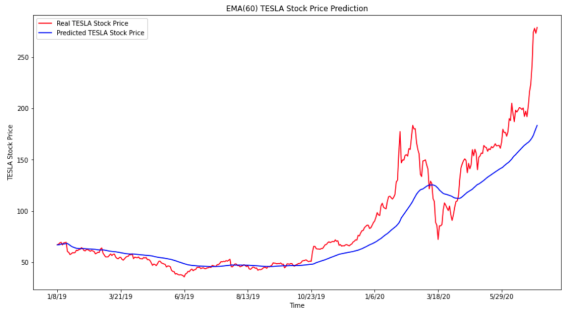
(a) MA(1) (2.87,5.35)



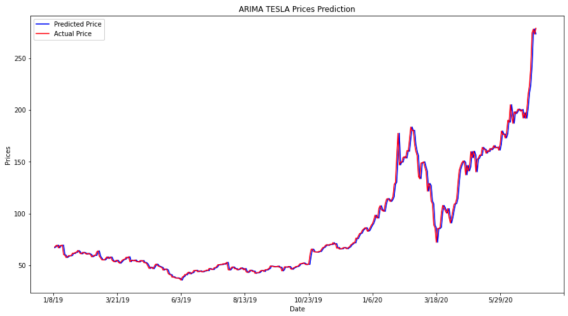
(b) MA(60) (2.87,5.35)



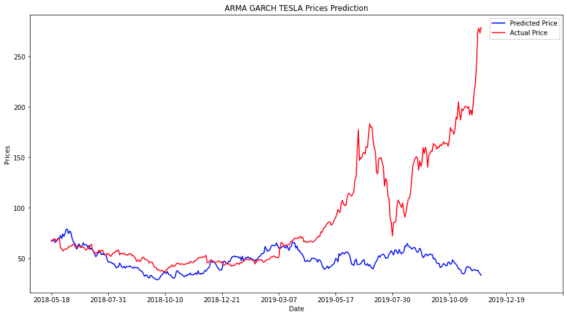
(c) EMA(1) (2.87,5.35)



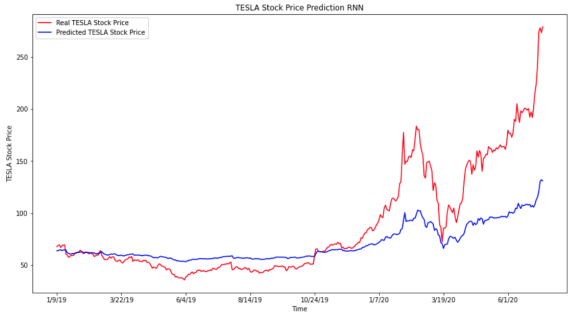
(d) EMA(60) (2.87,5.35)



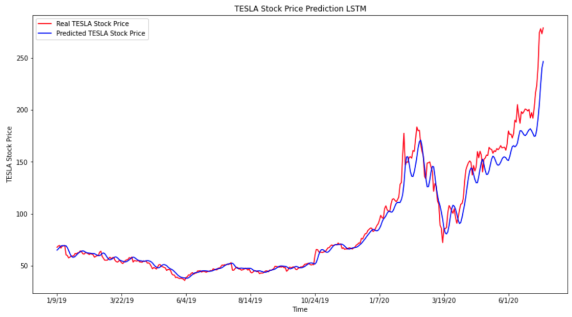
(e) ARIMA (2.87,5.35)



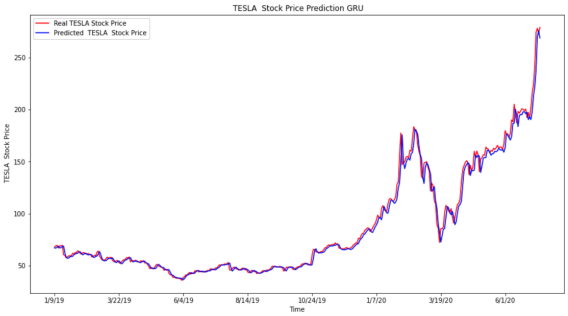
(f) ARIMA-GARCH (40.94,66.09)



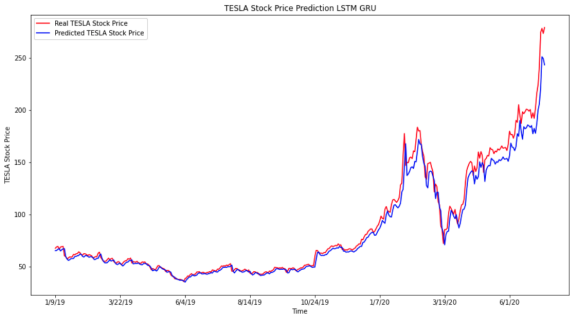
(g) RNN(25.02,38.88)



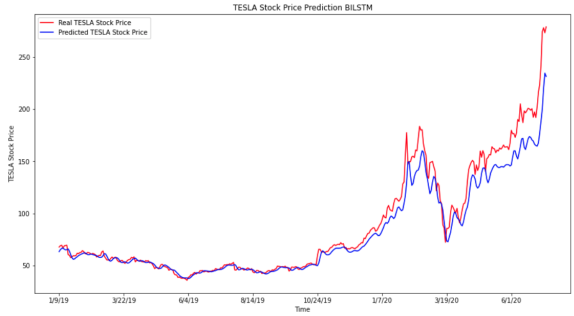
(h) LSTM(6.28,11.22)



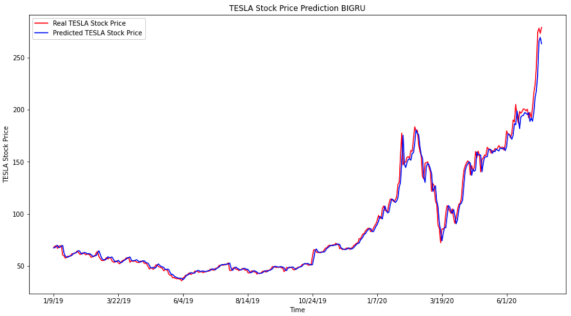
(i) GRU (3.29,5.87)



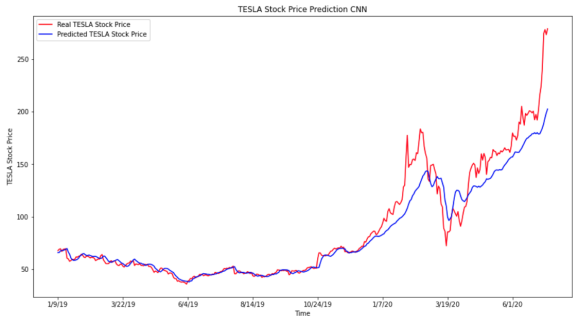
(j) LSTM-GRU(5.39,8.83)



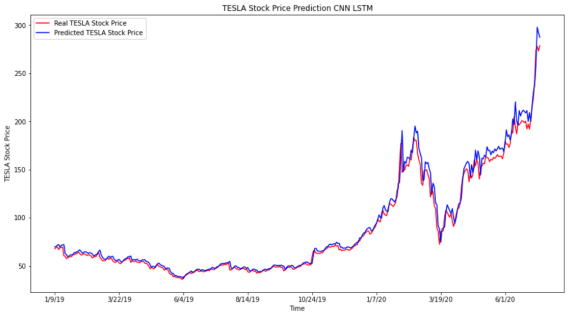
(k) Bi-LSTM(7.94,13.62)



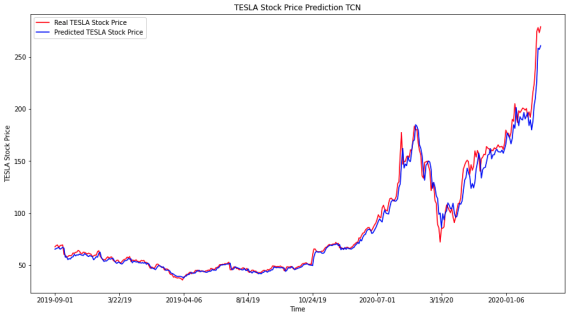
(l) Bi-GRU(3.16,5.86)



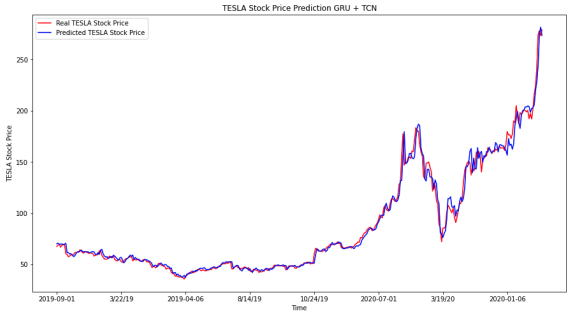
(m) CNN(8.94,16.01)



(n) CNN-GRU(4.19,6.69)



(o) TCN(3.86,6.65)



(p) GRU + TCN(3.40,5.91)

Figure 4.3 Tesla Stock Price prediction. (a) to (d) showcase the performance of the baseline models, (e) represents the ARIMA model. (f) represents the ARMA GARCH model and it underforms as compared to the ARIMA model. (g) to (p) represent the deep learning architectures.

Table 4.3 Comparative Table for Tesla Stock Price Prediction

Algorithm	MAE	RMSE
ARIMA	2.87	5.35
ARMA-GARCH	40.94	66.09
SMA(1)	2.87	5.35
EMA(1)	2.87	5.35
SMA(60)	19.59	28.39
EMA(60)	17.40	25.57
RNN	25.02	38.8
LSTM	6.28	11.22
GRU	3.29	5.87
Bi-LSTM	7.94	13.62
Bi-GRU	3.16	5.86
LSTM-GRU	5.39	8.83
CNN	8.94	16.01
CNN-GRU	4.19	6.69
TCN	3.86	6.65
GRU + TCN	3.40	5.91

From the error metrics, one can determine that for Apple Inc., the best model is LSTM GRU. The other GRU-related models like CNN-GRU, Bi-GRU, and simple GRU perform well too. The ARIMA model also performs equally well. Concerning Facebook Inc., the GRU model performs the best, followed by the ARIMA model, the GRU+TCN model, and the TCN model, respectively. Regarding Tesla Inc, the ARIMA model performs slightly better than the GRU and the BI-GRU models. The GRU+TCN model also performs well. Overall the ARIMA models and the GRU

models are best suited for the univariate single-step stock price prediction problem followed by the TCN models. The GRU models are the best model for two out of the three stocks under consideration. Our results align with that of [14], where the ARIMA and GRU models are the best fit for price predictions.

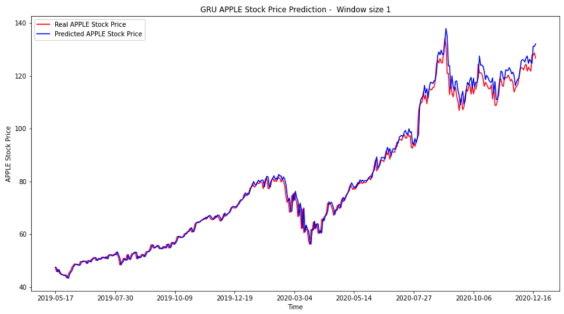
The advantage of ARIMA is that it is quick to run, and the model can be easily updated with new data while making forecasts. On the other hand, the Deep Learning models are hard to update while performing predictions. Hence in the case of ARIMA, we kept on updating the model while performing predictions. Whereas in Neural Networks, the models were trained only once and then used for prediction.

While training the model, different hyperparameters were chosen to improve the performance. In the GRU model, the best performance was obtained only with a single GRU layer having seven hundred units and a dropout of 0.1. In the TCN model, a single stack of TCN layer was used, with 64 filters, kernel size of 4, and maximum dilation of 16.

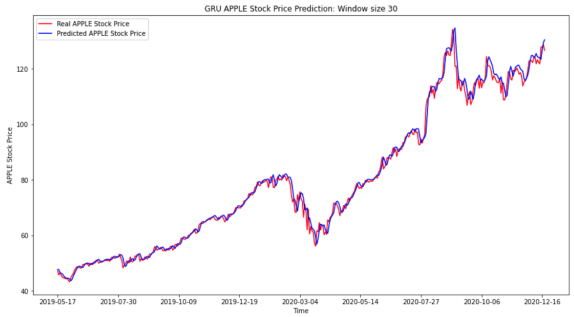
We use GRU and TCN models for all of our further analyses as these models have performed considerably better than other deep learning models.

4.0.2 Comparing Different Window Sizes

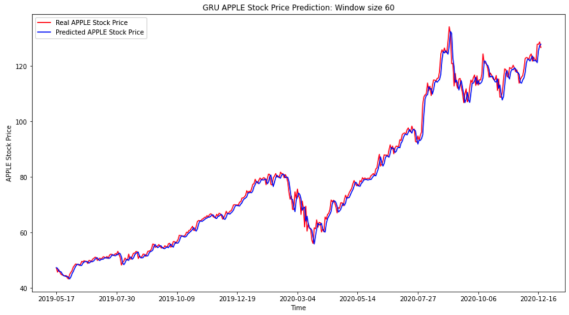
The authors of [15] presented a long short-term memory (LSTM) model for predicting the return of Chinese shares. The authors considered 30-day sequences of historical data for predicting the returns. Whereas the authors of [13] have used a single day window to predict the value for next day. In this part the results of the performance for different window sizes is provided for each dataset. The model chosen to do the analysis are the GRU model and the TCN model as these two models performs the best across all the three datasets.



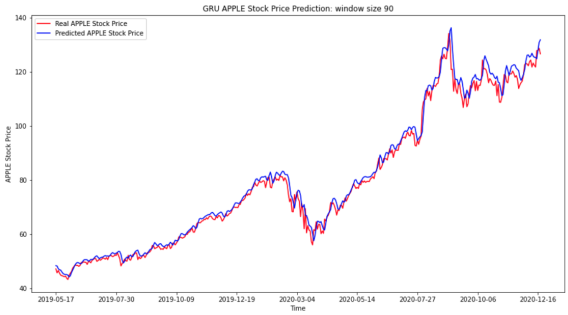
(a) Apple Inc. Window Size (1) : (1.62,2.47)



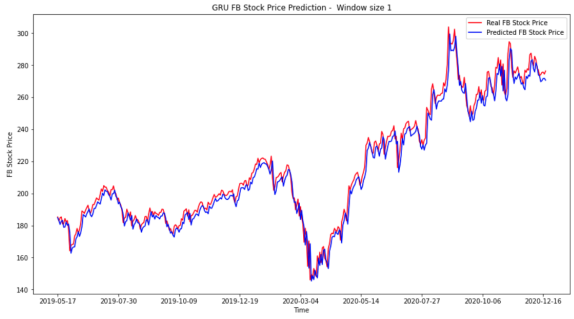
(b) Apple Inc. Window Size (30) : (1.48,2.2)



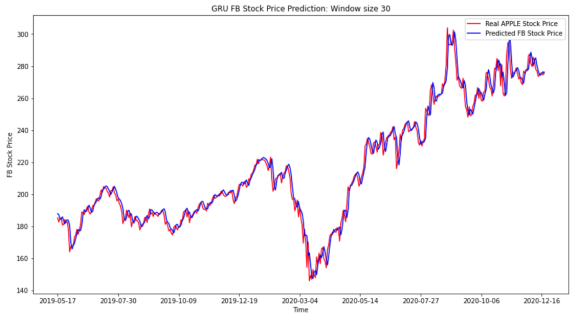
(c) Apple Inc. Window Size (60) : (1.46,2.17)



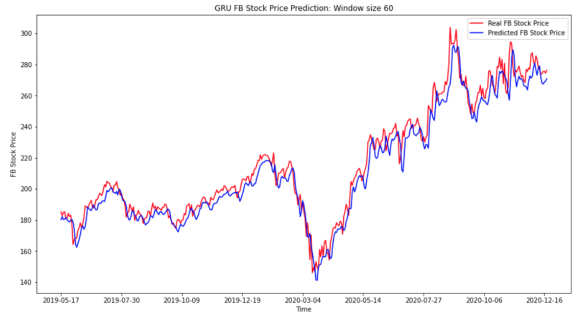
(d) Apple Inc. Window Size (30) : (1.89,2.65)



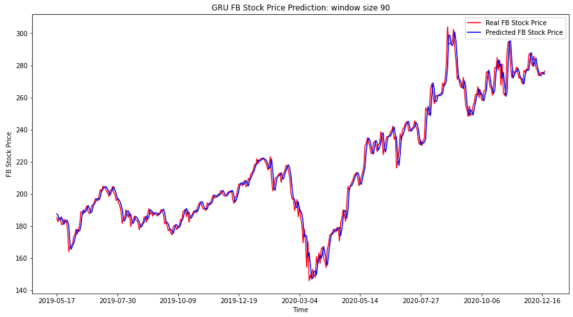
(e) Facebook Inc. window size(1):(4.61,6.06)



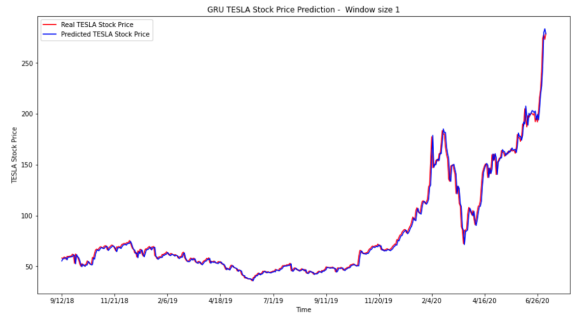
(f) Facebook Inc. window size(30):(3.79,5.43)



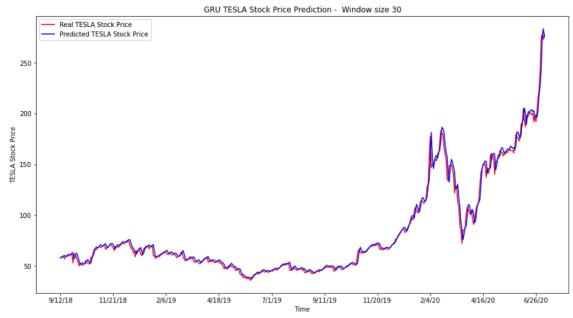
(g) Facebook Inc. window size (60) : (4.83,6.5)



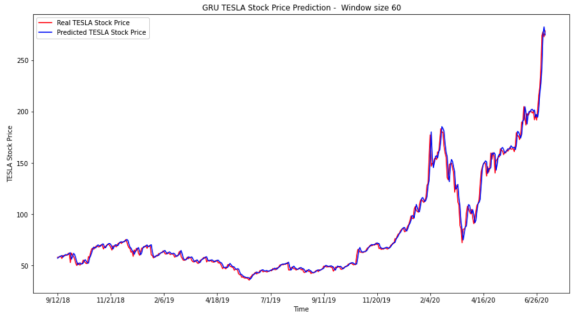
(h) Facebook Inc. Window Size(90) : (3.76,5.37)



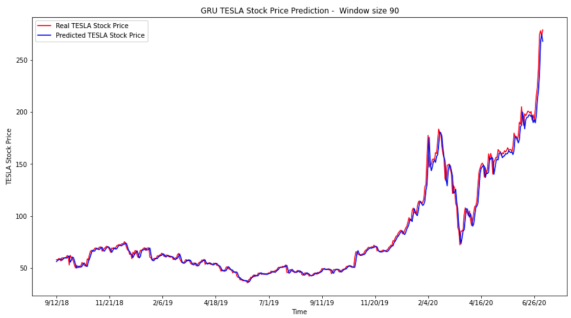
(i) Tesla Inc. Window Size(1) : (2.82,4.98)



(j) Tesla Inc. Window Size(30) : (3.08,5.18)



(k) Tesla Inc. Window Size(60) : (2.84,5.05)



(l) Tesla Inc. Window Size(90) : (3.07,5.45)

Figure 4.4 Stock Price Prediction for Different Window Sizes - GRU Architecture

Table 4.4 Comparative Table of Different Window-Size Used With GRU for Stock-Price Prediction

Window / Stocks	1	30	60	90
	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)
Apple	(1.62, 2.47)	(1.48,2.2)	(1.49,2.11)	(1.89, 2.65)
Facebook	(4.61, 6.06)	(3.79, 5.43)	(3.67, 5.29)	(3.76, 5.37)
Tesla	(2.82, 4.98)	(3.08, 5.18)	(3.29, 5.87)	(3.07, 5.45)

Table 4.5 Comparative Table of Different Window-Size Used With TCN for Stock-Price Prediction

Window / Stocks	1	30	60	90
	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)
Apple	(1.99, 2.71)	(2.06,2.76)	(2.11,3.10)	(1.61, 2.35)
Facebook	(4.60, 6.10)	(6.49, 8.09)	(4.13, 5.69)	(4.31,5.95)
Tesla	(2.90, 5.37)	(3.09,5.41)	(3.86, 6.65)	(4.27, 7.29)

We compared different window sizes for the GRU model on each of the three datasets. Looking at the results given in the table there is no clear trend that the error decreases or increases with the window size. In this study mostly a window size of 60 is used to predict the 61st day stock price as for two out of the three stocks the 60 day window size has the least error. In the case of TCN model, we do not see any clear trend.

4.0.3 Univariate Multi Step Forecasting

In the case of Apple, the RMSE(Root Mean Squared Error) of Seq2Seq GRU, Multi-Step GRU and TCN are less than that of ARIMA whereas ARIMA has the least MAE(Mean Absolute Error). Suggesting ARIMA predicts more extreme values as compared to Multistep GRU or Seq2Seq GRU. In the case of Facebook ARIMA performs the best for most of the metrics, similary for Tesla too ARIMA performs better as compared to Multistep GRU, Seq2Seq GRU and TCN.

Days / Algorithms	3	5	7
	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)
ARIMA	(1.87, 4.52)	(2.23,5.01)	(2.54,5.12)
Multi-step GRU	(2.79, 4.06)	(3.40, 4.80)	(3.27, 4.87)
Seq2Seq GRU	(2.88, 3.85)	(2.69, 3.87)	(3.70, 5.09)
TCN	(3.04, 4.47)	(3.28, 4.45)	(3.04, 4.47)

Table 4.6 Error Metric of Apple Stock Prices in Multi-Step Forecasting

Days / Algorithms	3	5	7
	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)
ARIMA	(5.01, 8.37)	(6.09, 9.61)	(6.68,10.26)
Multi-step GRU	(5.29, 7.50)	(7.26, 9.78)	(8.94, 11.72)
Seq2Seq GRU	(5.49, 7.79)	(6.58, 9.17)	(9.58, 12.71)
TCN	(5.89, 8.09)	(7.102, 9.74)	(8.69, 11.369)

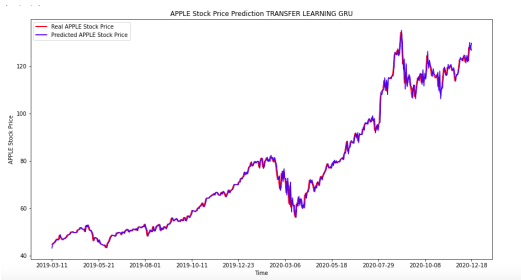
Table 4.7 Error Metric of Facebook Stock Prices in Multi-Step Forecasting

Table 4.8 Error Metric of Tesla Stock Prices in Multi-Step Forecasting

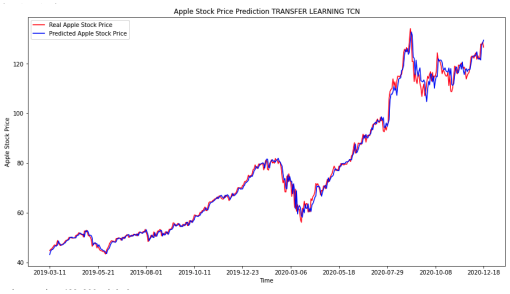
Days / Algorithms	3	5	7
	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)
ARIMA	(4.29, 8.04)	(5.80,14.58)	(6.47,13.78)
Multi-step GRU	(6.08, 10.44)	(7.69, 13.25)	(9.05, 15.78)
Seq2Seq GRU	(6.04, 11.47)	(5.6, 15.65)	(11.46, 20.64)
TCN	(6.21, 10.00)	(6.13, 10.68)	(7.71, 14.01)

4.0.4 Transfer Learning for Single Step Forecasting

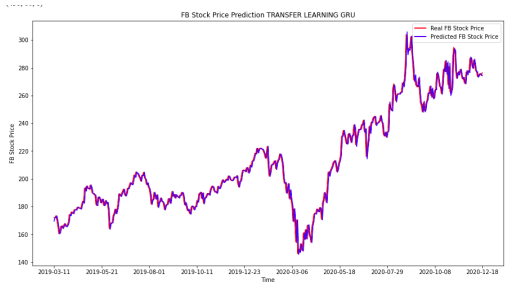
The advantage of a deep learning model over a statistical model is that a deep learning model can learn from similar datasets and perform predictions on a given data. This is called as transfer learning. In transfer learning we first train the GRU model and the TCN model on other stock prices data. Once the parameters are trained we use the model to predict the stock price of a given stock. Below we have included the plots of the predicted and real stock price values for each of the three datasets. The plots are followed by a table showcasing the error metrics in each case.



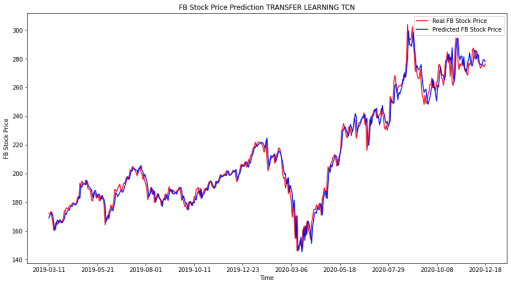
(a) Prediction of Stock Prices for Apple Using Transfer Learning - GRU



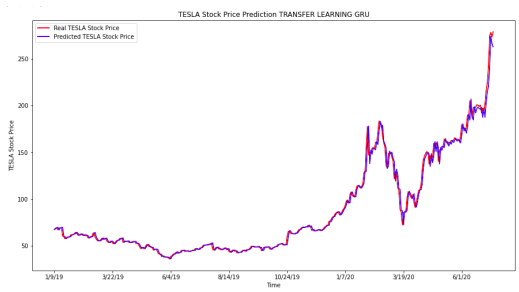
(b) Prediction of Stock Prices for Apple Using Transfer Learning - TCN



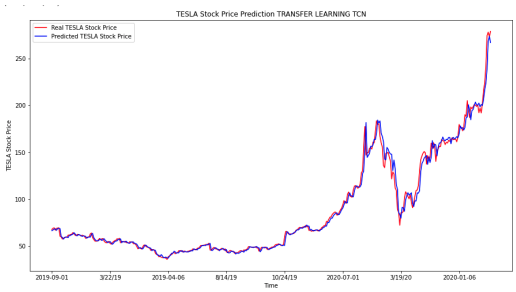
(c) Prediction of Stock Prices for FB Using Transfer Learning - GRU



(d) Prediction of Stock Prices for FB Using Transfer Learning - TCN



(e) Prediction of Stock Prices for Tesla Using Transfer Learning - GRU



(f) Prediction of Stock Prices for Tesla Using Transfer Learning - TCN

Figure 4.5 Single step stock price Prediction via transfer learning.

Table 4.9 Reported Errors of Transfer Learning on Stock Price Prediction - GRU Architecture

Stocks / Learning Method	Apple	Facebook	Tesla
	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)
Without Transfer Learning-GRU	(1.49, 2.11)	(3.67, 5.29)	(3.29, 5.87)
With Transfer Learning-GRU	(1.35, 2.06)	(2.93, 5.40)	(3.56, 5.15)

Table 4.10 Reported Errors of Transfer Learning on Stock Price Prediction - TCN Architecture

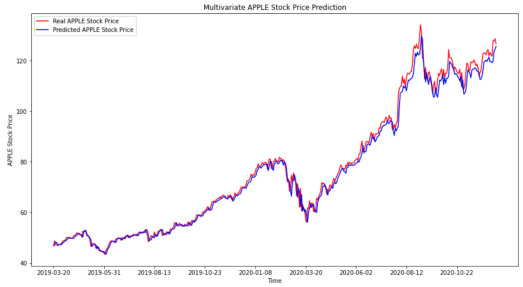
Stocks / Learning Method	Apple	Facebook	Tesla
	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)
Without Transfer Learning-TCN	(2.11, 3.10)	(4.13, 5.69)	(3.86, 6.65)
With Transfer Learning-TCN	(1.39, 2.19)	(3.83, 5.47)	(3.17, 5.89)

From our experimental study we find that Transfer Learning improves the performance of both the GRU model and the TCN model for all the three stocks. If

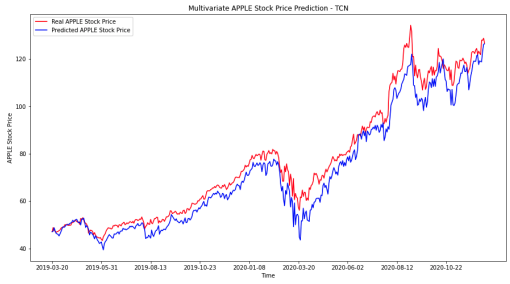
we add more datasets to train our model we may achieve an even higher accuracy and a lesser error rate.

4.0.5 Multivariate Single Step Forecasting

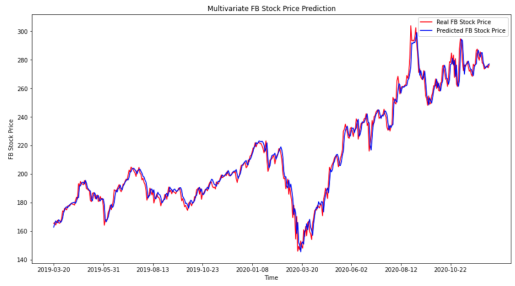
We conduct the Multivariate analysis in two parts. First we perform general multivariate analysis with variables like open, close, high, volume, market indices, interest rate, etc. Then we perform the same analysis but with the help of transfer learning. The plots and table for both the analysis are presented below.



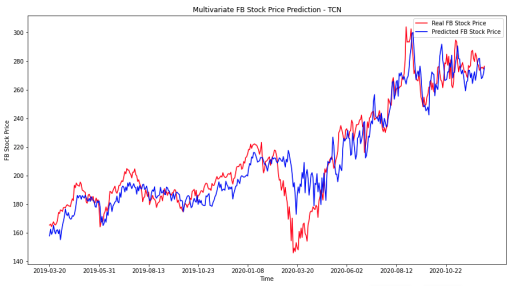
(a) Prediction of Apple Stock Prices Using GRU model



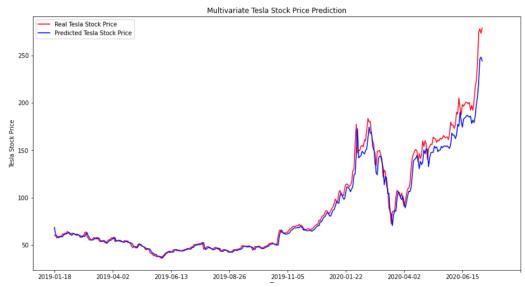
(b) Prediction of Apple Stock Prices Using TCN model



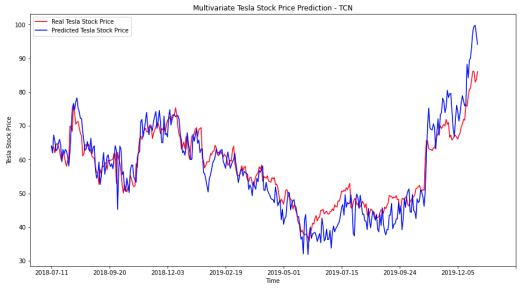
(c) Prediction of Facebook Stock Prices Using GRU model



(d) Prediction of Facebook Stock Prices Using TCN model



(e) Prediction of Tesla Stock Prices Using GRU model



(f) Prediction of Tesla Stock Prices Using TCN model

Figure 4.6 Multivariate Stock Price Prediction - a comparison between GRU and TCN Architecture

Table 4.11 Univariate and Multivariate GRU error metrics for Stock Price Prediction

Stocks / Learning Method	Apple	Facebook	Tesla
	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)
Univariate-GRU	(1.49, 2.11)	(3.67, 5.29)	(3.29, 5.87)
Multivariate-GRU	(1.71, 2.50)	(3.48, 5.09)	(4.09, 8.58)

Table 4.12 Univariate and Multivariate TCN error metrics for Stock Price Prediction

Stocks / Learning Method	Apple	Facebook	Tesla
	(MAE, RMSE)	(MAE, RMSE)	(MAE, RMSE)
Univariate-TCN	(2.11, 3.10)	(4.13, 5.69)	(3.86, 6.65)
Multivariate-TCN	(4.48, 5.92)	(9.72, 13.28)	(3.80, 4.80)

In the case of Facebook multivariate GRU is performing better as compared to univariate GRU to predit the next day's stock price but in the case of Apple and Tesla the multivariate GRU model does not improve the performance.

CHAPTER 5

DISCUSSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

In our study, we not only compared ARIMA with simple LSTM, GRU, and RNN but also compared it with TCNs, Bi-LSTM, and Bi-GRU and models derived by combining two different architectures. We find that ARIMA and GRU-related architectures perform well for stock price prediction from the experimental studies. The advantage of deep learning models over a statistical model is that there are no pre-requisites like stationarity, one can model non-linear data and deep learning models are flexible enough to include different datasets and it can also perform transfer learning which statistical time series methods cannot.

While training the neural networks, some crucial conclusions were that more epochs did not improve the model's accuracy. Besides that, adding more layers also reduced the accuracy of the model. Initially, we tested with four layers (Stacked RNN/LSTM) model, and as we reduced the number of layers to one, the accuracy kept on increasing. One of the reasons for this is that the more complex model has far more parameters to train, and with the small dataset we have, it is not enough to train a large number of parameters. Hence, as we reduced the number of layers, the performance went up.

The time taken to train LSTM and GRU was less than RNN due to improved gradient flow. Bidirectional GRUs and LSTMs took considerably more time to prepare than simple LSTM, RNN, and GRUs because information flows in both directions, and the models are more complex. Our findings could not proclaim that training the sequential data twice and learning from the future and the past time-steps would help achieve better forecasts of the stock prices. The results obtained by

us reveal that Bi-LSTMs and Bi-GRUs perform comparably to regular LSTMs and GRUs.

In our analysis we found that ARIMA, GRU and TCN are one of the best performing model for univariate single step price prediction. In the multi-step analysis, it was expected that the Seq2Seq model would outperform the ARIMA model and the multi-step GRU/TCN model. Yet, the ARIMA multi-step model outperformed the other models. Only in the cases of Apple Inc. and Facebook Inc., the RMSE of the Seq2Seq model is less than the RMSE of ARIMA, suggesting that it predicts lesser extreme values than the ARIMA model.

In our multivariate analysis, we do not use technical indicators (Moving Average) as features because they are derived from the stock price itself, and they can have co-founding pitfalls. The dataset used in the multivariate analysis is based on the financial theory that market price movement and economic factors can influence the stock price of the company. Still, from our research, we see that adding these datasets did not improve the prediction.

To improve the performance of the prediction algorithm, Transfer Learning is used. Transfer Learning enhances the performance of model for univariate single step price prediction. This shows that the model can learn the typical price dynamics across the datasets. We expect that the performance of Deep Learning models will improve if more datasets were included to pre-train the models. The GRU model performs better as compared to the TCN model in the task of transfer learning.

Future research work may include the following improved Multivariate and Multi-step time series prediction with Transfer Learning. Furthermore, Natural Language Processing methods can be used to understand the effect of positive and negative news on stock prices.

Also the future of the research can be extended towards Generative Adversarial Networks(GANs). The GAN network can be used to make representations of time

series. It has been observed that there is no significant difference between GAN and usually used LSTM. We can explore different GAN architectures to simulate time series, especially those that involve structures traditionally used for time series in deep learning, such as LSTM.

BIBLIOGRAPHY

[1] Nasdaq Total Returns (Annual Performance), url: https://www.nasdaq.com/market-activity/total-returns.

[2] A. V. Devadoss and T. A. A. Ligori, "Forecasting of stock prices using multi layer perceptron," Int J Comput Algorithm, vol. 2, pp. 440–449, 2013.

[3] V. Raghunathan and Prabina Rajib, Stock Exchanges, Investments and Derivatives: Straight Answers to 250 Nagging Questions, McGraw Hill Education; 3rd edition, 1 July 2017.

[4] Selvin, S.; Vinayakumar, R.; Gopalakrishnan, E.A.; Menon, V.K.; Soman, K.P. Stock price prediction using LSTM, RNN and CNN-sliding window model. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 13–16 September 2017; pp. 1643–1647.

[5] Box, G.E.P.; Jenkins, G.M. Time Series Analysis: Forecasting and Control; Holden-Day: San Francisco, CA, USA, 1970.

[6] A. Earnest, M. I. Chen, D. Ng, L. Y. Sin, "Using Autoregressive Integrated Moving Average (ARIMA) Models to Predict and Monitor the Number of Beds Occupied During a SARS Outbreak in a Tertiary Hospital in Singapore," in BMC Health Service Research, 5(36), 2005.

[7] M. J. Kane, N. Price, M. Scotch, P. Rabinowitz, "Comparison of ARIMA and Random Forest Time Series Models for Prediction of Avian Influenza H5N1 Outbreaks," BMC Bioinformatics, 15(1), 2014.

[8] Reinaldo C. Garcia, Javier Contreras, Senior Member, IEEE, Marco van Akkeren, and João Batista C. Garcia, "A GARCH Forecasting Model to Predict Day-Ahead Electricity Prices", IEEE TRANSACTIONS ON POWER SYSTEMS, VOL. 20, NO. 2, MAY 2005.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.

[10] Razvan Pascanu and Tomas Mikolov and Yoshua Bengio, "On the difficulty of training Recurrent Neural Networks", arXiv 1211.5063v, 2013.

[11] Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to sequence learning with neural networks. arXiv 2014, arXiv:1409.3215v3.

[12] Yue-Hei Ng, J.; Hausknecht, M.; Vijayanarasimhan, S.; Vinyals, O.; Monga, R.; Toderici, G. Beyond short snippets: Deep networks for video classification.

In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 4694–4702.

[13] Siami-Namini, S.; Tavakoli, N.; Namin, A.S. A Comparison of ARIMA and LSTM in Forecasting Time Series. In Proceedings of the 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, USA, 17–20 December 2018; pp. 1394–1401.

[14] Yamak, Peter T. and Yujian, Li and Gadosey, Pius K.,"A Comparison between ARIMA, LSTM, and GRU for Time Series Forecasting", Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence, pp. 49–55. DOI: 10.1145/3377713.3377722

[15] K. Chen, Y. Zhou and F. Dai, "A LSTM-based method for stock returns prediction: A case study of China stock market," 2015 IEEE International Conference on Big Data (Big Data), Santa Clara, CA, 2015, pp. 2823-2824, doi: 10.1109/BigData.2015.7364089.

[16] Sirignano, Justin and Cont, Rama, Universal Features of Price Formation in Financial Markets: Perspectives From Deep Learning (March 16, 2018). Available at SSRN: https://ssrn.com/abstract=3141294 or http://dx.doi.org/10.2139/ssrn.3141294

[17] Jaydip Sen and Sidra Mehtab, Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models,2020, arXiv, 2010.13891.

[18] Shaojie Bai, J. Zico Kolter, Vladlen Koltun, An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling, arXiv,1803.01271v2

[19] Shumin Denf et al., Knowledge-Driven Stock Trend Prediction and Explanation via Temporal Convolutional Network,https://doi.org/10.1145/3308560.3317701

[20] de Souza, M.J.S., Ramos, D.G.F., Pena, M.G. et al. Examination of the profitability of technical analysis based on moving average strategies in BRICS. Financ Innov 4, 3 (2018). https://doi.org/10.1186/s40854-018-0087-z

[21] Appel G (2005) Technical Analysis: Power Tools for Active Investors. Prentice Hall Publishing, NJ

[22] Vandewalle N, Ausloos M, Boveroux P (1999) The Moving Averages Demystified. Physica A: Statistical Mechanics and Its Applications 269(1):170–176Return

[23] Ratnadip Adhikari and R. K. Agrawal, "An Introductory Study on Time Series Modeling and Forecasting" , 2013, arXiv 1302.6613.

[24] Tim BOLLERSLEV, GENERALIZED AUTOREGRESSIVE CONDITIONAL HETEROSKEDASTICITY , Journal of Econometrics 31 (1986) 307-327. North-Holland

[25] Engle, Robert F. "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation." Econometrica, vol. 50, no. 4, 1982, pp. 987–1007. JSTOR, www.jstor.org/stable/1912773. Accessed 23 Jan. 2021.

[26] Jeffrey L. Elman. Finding structure in time. COGNITIVE SCIENCE, 14(2):179–211, 1990.

[27] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 1724–1734,

[28] Z. Cui, R. Ke, Y. Wang, "Deep Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction," arXiv:1801.02143, 2018.

[29] J. Kim, N. Moon, "BiLSTM model based on multivariate time series data in multiple field for forecasting trading area." Journal of Ambient Intelligence and Humanized Computing, pp. 1-10.

[30] M. Schuster and K.K. Paliwal. 1997. Bidirectional recurrent neural networks. Trans. Sig. Proc. 45, 11 (November 1997), 2673–2681. DOI:https://doi.org/10.1109/78.650093

[31] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. CoRR, abs/1609.03499, 2016.

[32] Weiss, K., Khoshgoftaar, T.M. Wang, D. A survey of transfer learning. J Big Data 3, 9 (2016). https://doi.org/10.1186/s40537-016-0043-6

[33] Chuanqi Tan and Fuchun Sun and Tao Kong and Wenchang Zhang and Chao Yang and Chunfang Liu, "A Survey on Deep Transfer Learning", 2018, arXiv 1808.01974.