Dhaka Stock Exchange Price Prediction Using Artificial Neural Networks

Mashfiqur Rahman

Department of Computer Science and Engineering

United International University

A thesis submitted for the degree of

MSc in Computer Science & Engineering

April 2023

Declaration

I, Mashfiqur Rahman, declare that this thesis titled, Dhaka Stock Exchange Price Prediction Using Artificial Neural Networks and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a MSc degree at United International University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at United International University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Mashfiqur Rahman

Certificate

I do hereby declare that the research works embodied in this thesis entitled "Dhaka Stock Exchange Price Prediction Using Artificial Neural Networks" is the outcome of an original work carried out by Student Mashfiqur Rahman under my supervision.

I further certify that the dissertation meets the requirements and the standard for the degree of MSc in Computer Science and Engineering.

Signed: _____

Date: _____

Prof. Dr. Mohammad Nurul Huda

Professor and Director - MSCSE

Department of Computer Science and Engineering,

United International University,

Dhaka-1209, Bangladesh.

Abstract

Stock trading is one of the most important economic activities in the financial world. However, stock markets are susceptible to changes due to their complex and non-linear nature. Stock trend prediction maximizes profit while minimizing risk by developing approaches to foresee the stocks. In this study, we will focus on predicting the price of a Dhaka Stock Exchange-listed company using historical transaction data. The study compares five forecasting models, i.e., Autoregressive Integrated Moving Average (ARIMA), Recurrent neural network (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Convolutional Neural Network (CNN). Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Root Mean Squared Error (RMSE), and R-square are used to evaluate the performance of the observed and predicted values for selected models. Finally, the thesis provides a comparative analysis of the models' performance based on the results.

This thesis is dedicated to Almighty Allah who has blessed us with good health, patience, and dedication.

I am grateful to my supervisor, Prof. Dr. Mohammad Nurul Huda, for his continuous support, advice, and motivation throughout my M.Sc. I am grateful to him for the guidance and inspiration he provided during my research and writing. As a student, I could not have imagined a better supervisor and mentor for my Master of Science degree.

My sincere thanks go out to the rest of the exam committee for their encouragement and insightful comments.

I am forever grateful to my beloved parents, who are the inspiration behind every success of mine, who have taught me to read and learn, and who have encouraged me to question things I do not comprehend. At the same time, they taught me to look for answers with patience and dedication. Thank you also to my beautiful wife, Afsana Ahmed Munia, for her expert advice on this thesis and in life, for her love and support, and for every precious moment we have shared together. It would not have been possible to earn this degree without her unconditional love and support.

Published Papers

In conjunction with the research presented in this thesis, the author has published in the following peer-reviewed journals and conferences:

1. Mashfiqur Rahman, Mohammad Nurul Huda, A Review: Machine Learning for Stock Market, SCRS CONFERENCE PROCEEDINGS ON INTELLIGENT SYSTEMS, April 2022, pp. 305–312.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

A financial market is a place where securities are traded. The most commonly known form of financial market is the stock market. A society cannot function without financial markets. By creating a trading platform, investors with funds can invest in companies listed on the stock market and make the funds available for business or [2]. IPOs are ways companies raise capital. Shares can be traded among various buyers and sellers on a secondary market. A stock represents ownership of a fraction of a company [3]. The unit of a stock is called a share. Stocks are traded in stock markets. The stock market refers to a market or exchange where activities of buying, selling, and issuance of shares of listed companies take place [4]. It is also frequently referred to as Capital Market, Equity Market, Stock Exchange, etc. Stock exchanges are subsets of the stock market. One stock market can comprise one or more stock exchanges. For instance, Bangladesh has two stock exchanges, the Dhaka Stock Exchange and the Chittagong Stock Exchange. As powerful as the New York Stock Exchange (NYSE) is, it was not the first exchange to influence the markets. In the 1300s, Venetian moneylenders began selling debt issues. It is believed, however, that modern stock trading was developed by the East India Company in London. Stock market trends have been predicted through different approaches over time. Successfully predicting the movement would result in a significant profit yield. Stock market predictions can be based on trend predictions or price predictions. There are two major approaches to making stock market predictions: 1) Fundamental Analysis, and 2) Technical Analysis.

As part of Fundamental Analysis, companies are analyzed for the factors that will influence their stock price in the future, including profit/loss, management changes, industry changes, etc. By analyzing a company's balance sheet, income statement, cash flow statement, etc., it determines its value. According to technical analysts, stock charts provide hints about price movements. The objectives of Fundamental and Technical analysts are, therefore, different. Another difference is that fundamental research focuses on long-term investing than technical analysis's short-term method. However, 90% of investors use technical analysis for stock market trend prediction. The field of artificial intelligence (AI) aims to make machines (computer programs) behave intelligently, mimicking human behaviour. As a result of Alan Turing's paper "Computing Machinery and Intelligence" [5] and the Turing Test, artificial intelligence has evolved into many branches. Machine Learning (ML) is one of these branches. In general, machine learning uses computer algorithms to make predictions or decisions based on data without explicitly programming [6]. The complexity of machine learning varies based on the task and the algorithm used to accomplish it. Modern machine learning technologies have evolved rapidly due to the massive amount of data produced every day due to digitalization. Computational power has also increased rapidly. In the past two decades, investors and researchers have been exploring different ways to use machine learning for stock market prediction. Algorithms such as Rectified Linear Unit (ReLU) have also reduced computing time. The Stock Market's price is influenced by various factors, such as national and international politics, environmental disasters, religious occasions, human sentiments, company policies, etc. Because the stock market is nonlinear and complex, it is very hard to predict its price. The stock market has been rigorously analyzed by academics and industry experts for these reasons. Time series data, heterogeneous market data, and microeconomic variables were used by researchers. This technology is showing promising results for the future as stock market data becomes more accessible, machines become more capable of processing data, and machine learning advances. In light of this, we have focused our research on predicting the price movements of the Dhaka Stock Exchange.

During our background research of the domain, we have studied numerous recent and past research papers and additional materials. We have summarized our findings in a paper titled "A Review: Machine Learning for Stock Market" which has been published in ICSISCET 2021.

1.2 Objectives of the Thesis

As part of this thesis, a significant data set containing carefully selected features is used to examine the performance of the statistical algorithm ARIMA and four deep neural networks based on carefully selected features. By comparing the neural network models to ARIMA, we will be able to determine which configuration of neural networks is most accurate for predicting the future price of the company.

1.3 Research Challenges

Research Challenges

- Where can we get the data?
- Which factors influence the prices in DSE?
- How to find relevant works from the plethora of research?
- Will the dataset fit the model?
- How to find the best model configuration for Neural Networks?
- How can we improve our model accuracy?
- Can we reuse the model for a similar dataset?

1.4 Organization of the Thesis

The thesis is organized as follows:

Chapter 2 Background and literature review are presented in this chapter. The intention is to explain the terms and concepts later discussed in the research. So that the users feel comfortable when a term is encountered.

Chapter 3 Here we outline the methodology of this thesis work, such as data collection, feature selection, feature extraction, data preprocessing, dataset description, steps to run the tests

Chapter 4 The results found from the models are presented and discussed in this chapter

Chapter 5 Finally, the Conclusion and future work chapter will discuss our overall thesis work, outcome, and future work scope.

Chapter 2

Background and Literature Review

2.1 Background

2.1.1 Data

Any fact, value, text, sound, or image that can be examined and analyzed to gather information might be considered data. The most crucial component of machine learning, and artificial intelligence is data. We can't train any model without data. Machine learning methods learn from examples. These examples are known as data and the corresponding data collection. These datasets are then divided into various subsets for training, validating, and testing the models.

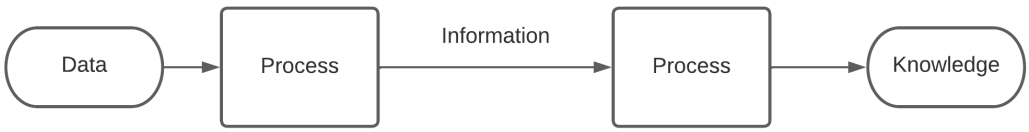

Figure 2.1: Data to knowledge flow -

The data can be mainly of two types:

1. Structured Data: Also known as quantitative data, it is well-organized and easy to understand for machine learning algorithms.

2. Unstructured Data: Often known as qualitative data. It cannot be handled or evaluated using standard data tools and procedures.

Data can be primarily divided into two categories:

1. Qualitative (Categorical) Data:
   The data that falls into categories is known as qualitative data. Rather than being specified in terms of numbers, categorical measures are defined in terms of plain language specifications. Categorical variables describe characteristics like a person's gender or hometown zip code etc. Quantitative data can be further divided into two types:

   (a) Nominal:
   Nominal data facilitates in variable labelling without revealing the numerical value. Usually they cannot be measured or ordered. However, the data can be qualitative and quantitative. Examples include letters, symbols, words, gender, etc. Generally nominal data are analyzed by grouping them and calculating their frequencies or percentages.

   (b) Ordinal:
   Natural order is observed in ordinal data. Example can be T-shirt sizes Large, medium and small.

2. Quantitative (Numerical) Data:
   Quantitative data represents the numerical value. Some examples of numerical data are height, length, size, weight, and so on. The quantitative data can be classified into two different types:

   (a) Discrete:
   Discrete data can take only discrete values. Discrete information contains only a finite number of possible values which can not be subdivided meaningfully. For example, number of boy and girls in the class.

   (b) Categorical:
   Data that can be calculated is referred to as continuous data. Here, within a predetermined range, there are an endless number of possible values that can be chosen. For instance, the current temperature.

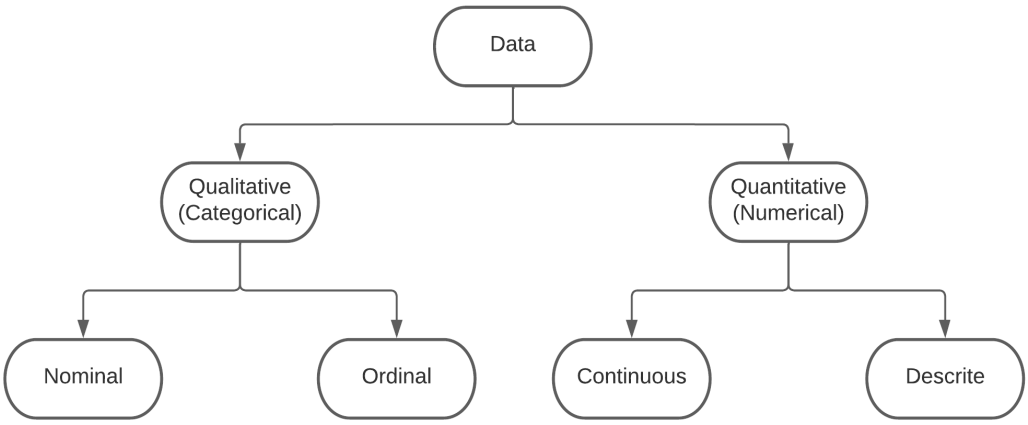

Figure 2.2: Types of Data -

The data has different properties. These are commonly referred to as 5Vs of data. They are as follows.

1. Volume:
   A volume is a measure of how much data is available. It can be considered big data if the volume of data is large enough. To determine the value of data, size of data plays a very crucial role.

2. Velocity:
   It describes the rate at which data is created and transferred. Rapidly analyzing and processing this information is often required.

3. Value:
   The value is the benefit that data may offer. Depending on the insights that may be drawn from the data, its worth rises dramatically.

4. Variety:
   Variety refers to the diversity of data types. The value of data derived from various sources may differ. The standardisation and distribution of all the data being gathered pose a problem in terms of variety. It also refers to nature of data that is structured, semi-structured and unstructured data.

5. Veracity:

   Veracity, in general, refers to the degree of confidence in the data that has been gathered. It refers to the quality and accuracy of data. The gathered information can be incomplete, erroneous, or unable to offer any useful, insightful information.

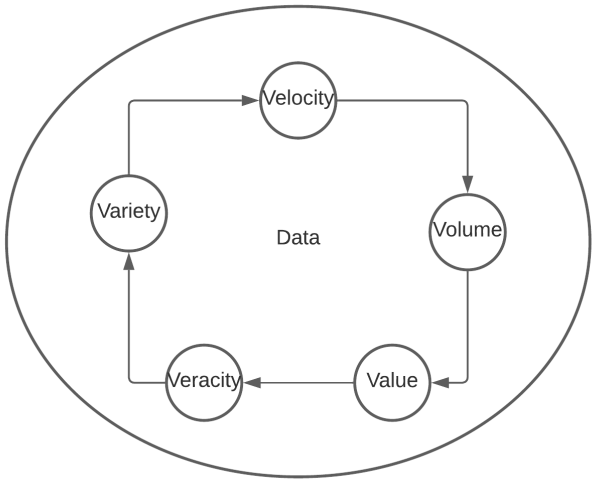

Figure 2.3: Properties of Data -

2.1.2 Overfitting

Supervised machine learning tries to model a function (f) that maps input variables (X) to output variables (Y).

$$Y = f(X) \tag{2.1}$$

A critical consideration when learning a function from training data is how well the model generalizes to new data since the collected data may not be representative of all possible domain data and may be incomplete and noisy. We can determine if a machine learning model is overfitting or underfitting based on how effectively it picks up new information and generalizes it to new data. Over-fitting occurs when a model learns the training data too well but fails to generalize. In this case, the training error is relatively low, but the test error is high. Most data studied contain some level of mistake or random noise. Attempting to make the model adhere to slightly erroneous

data too closely will infect the model with significant flaws and diminish its prediction potential. Overfitting can be caused by the following conditions:

- The models are trained using too many parameters
- The hypothesis space searched by the learning algorithm is high
- The training dataset is not a proper representation of the domain data

Following are a few popular techniques to avoid over-fitting:-

- Early stopping:
  With this approach, training is halted before the model begins to learn about the noise in training data. This strategy runs the danger of prematurely stopping the training process, which would result in the opposite issue of underfitting.
- Training with more data:
  By providing the model additional chances to identify the main relationship between the input and output variables, expanding the training set can enhance model performance when clean, appropriate data is included. If not, overfitting might result from the model just becoming more complex.
- Feature selection:
  The act of selecting the most crucial features from the training data and removing the redundant or unnecessary ones is known as feature selection. This also helps with dimensionality reduction.
- Regularization:
  Regularization is the technique used by statisticians to help them understand which features to leave out of a model. Through regularisation, the larger coefficient input parameters are given a "penalty," which ultimately lowers the variance in the model. Dropout, Lasso, and L1 regularisation are all excellent choices for regularisation.
- Ensemble methods:
  Ensemble learning methods are made up of a set of classifiers, i.e. decision trees, and their predictions are aggregated to identify which ones are most likely to be true. The most well-known ensemble methods are bagging and boosting.

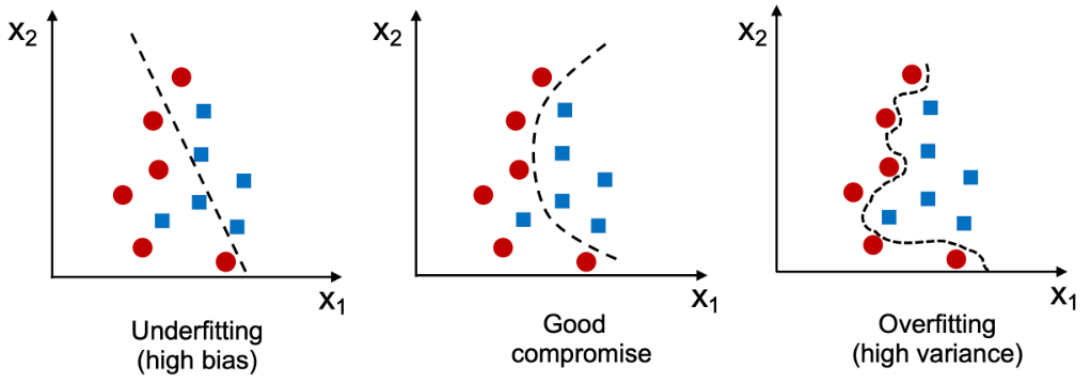

Figure 2.4: Overfitting vs underfitting [1]

2.1.3 Underfitting

The underfitting occurs when a model fails to extract enough patterns from the data. This is caused by weak training models, inappropriate models, a lack of data points, or simply a lack of data. In such cases, the model performs poorly on all data since it does not have enough parameters to capture the underlying system's trends. It is possible to try alternate machine learning algorithms, generate more features, etc, to remedy this problem.

The gold standard of applied machine learning is cross-validation, which determines model accuracy on unknown data. The most commonly used resampling method is K-fold cross-validation. By training and testing your model on various subsets of training data k times, you can estimate how well a machine learning model will perform on untried data. Figure 2.4 shows a pictorial representation of underfitting, balanced, and overfitting models.

In order to understand a machine learning algorithm over time, we can plot both its performance on a training dataset and its performance on a test dataset as it is learning training data. The model training error should decrease as the algorithm learns, and the testing error should decrease as well. When the model is overfitted and learns irrelevant details and noise from the training dataset, the performance on the training dataset may continue to decrease if it is trained for too long. The model's generalization ability decreases at the same time, leading to an increase in error for the test set. In a sweet spot, the model has good skill on both the training dataset and

the unseen test dataset just before the error on the test dataset increases. Figure 2.5 shows the validation loss for training and test data.

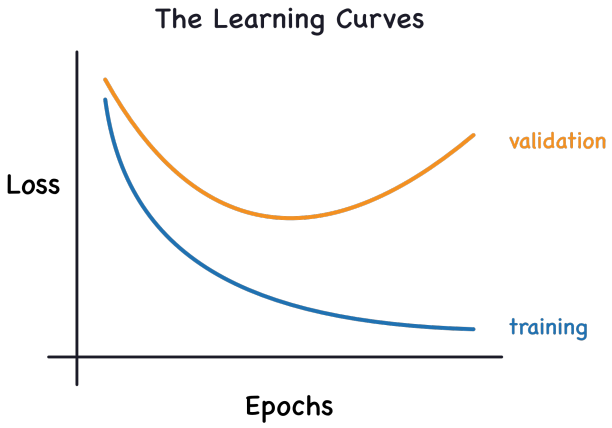

Figure 2.5: Model learning curve - [7]

Recent studies [8] show that despite being trained to "perfectly fit or interpolate," complicated models like deep learning models and neural networks function with great accuracy. Figure 2.6 shows a model double descent curve. The performance of the model increases as it learns past the interpolation threshold. The methods that avoid overfitting, such as early stopping and regularization, can actually prevent interpolation.

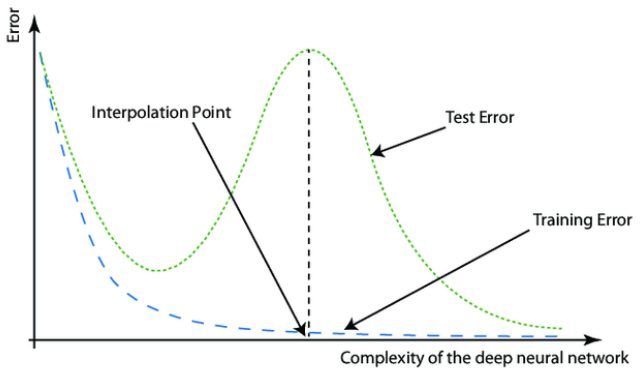

Figure 2.6: Double descent curve - [9]

Follow is a few reasons [10] for underfitting:

- High bias and low variance
- Smaller training data
- Weak Training model
- Noisy data
- Early Stopping

A few techniques to reduce [10] underfitting is listed below:

- Exploring new models
- Increasing model complexity
- Adding new features
- Removing noise from data by preprocessing
- Increasing number of epochs to train

2.1.4 ARIMA (AutoRegressive Integrated Moving Average)

One of the most popular linear time series forecasting models is ARIMA (AutoRegressive Integrated Moving Average). There are three main components: autoregressive (AR), differenciation (I), and moving average (MA). By analyzing historical data, it predicts the future values of univariate time series. ARIMA is expressed as ARIMA(p, d, q), where p, d, and q are the orders of AR, I, and MA components, respectively. The components are as follows:

- The autoregressive component (AR) represents the dependency between an observation and a specific number of lagged observations (previous steps). The AR term is defined by the parameter 'p', which represents the order of the autoregressive components. By increasing 'p', the model takes into account more historical data.
- By subtracting the previous observation from the current observation, differencing (I) makes the time series stationary. 'D' represents the order of differencing

in the differencing term. The higher the 'd', the more times the differencing operation will be applied. Stationarity is important because many time series models assume stationary data, including ARIMA. The mean, variance, and autocorrelation structure of stationary time series are constant over time.

- A moving average is a statistical representation of time series error that combines previous errors. This approach addresses the impact of past errors on current observations. The parameter 'q' determines the order of the moving average component. The higher the 'q,' the more past error terms the model considers.

Figure 2.7 shows an ARIMA model forecast.

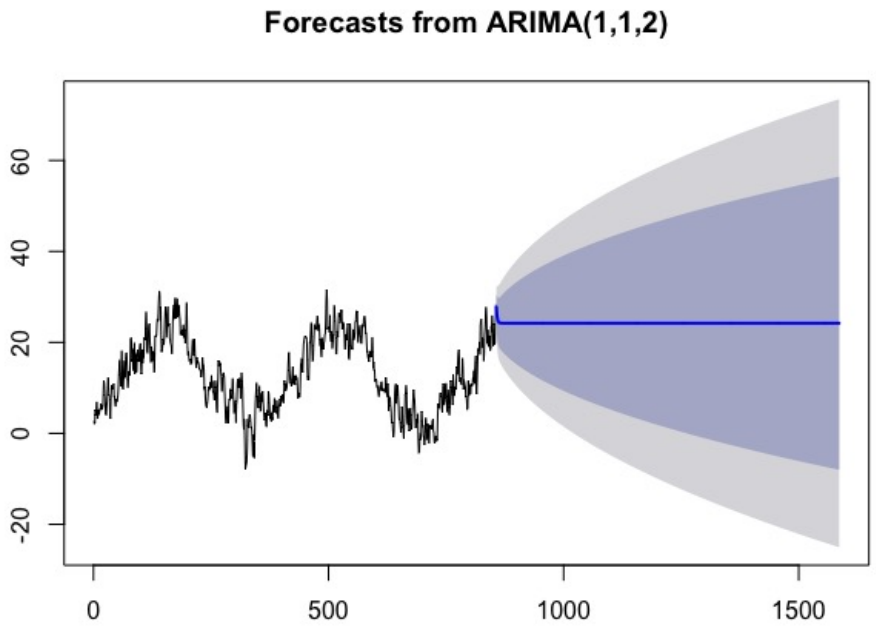

Figure 2.7: Forecasting using ARIMA - [11]

2.1.5 Artificial Neural Network-ANN

With collaboration from the US Navy, Cornell University's Frank Rosenblatt created the Perceptron system in 1958 [12]. This was perhaps the first neural network prototype. A human brain has billions of neurones. Neurones joined by dendrites act as an

information relay process and transform electrical and chemical impulses that make up human ideas. The figure [13] displays the biological brain structures, which are made up of several neurones (vertices) and dendrites (branches) connected by axons (edges).

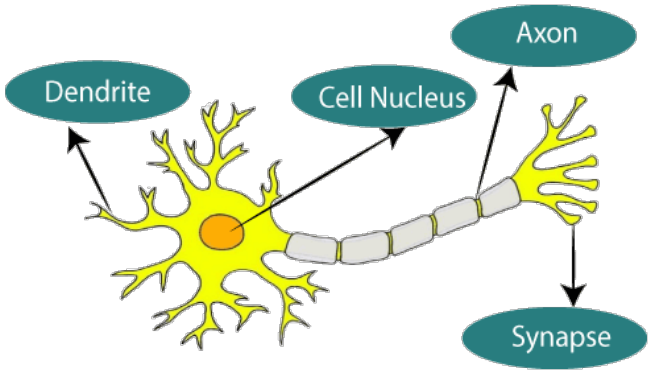

Figure 2.8: Biological Neural Network - [13]

An artificial neural network, also referred to as ANN or simply a neural network is a computational algorithm based on the principles of brain activity. An artificial nervous system is made up of linked units or nodes, which are loosely resembling biological brain neurons. An artificial neuron is the unit of an artificial neural network. It receives signals, processes them, and signals neurons connected to it. These connections are known as edges. Neurons are arranged into layers. Figure 2.9 shows the architecture of a simple ANN that adapts to changing information and finds hidden patterns. It is famous for its ability to find hidden patterns in the data and adapt to change.

It is the hidden units that connect the layers of input and output. Below is a brief description of the layers.

- Input Layer: Input layer of an ANN is responsible for receiving input data from the external environment. Data is entered into an ANN through the input layer, which serves as a link between the input data and the network's internal structure. Each neuron in the input layer corresponds to an attribute or feature in the input dataset. There is no transformation or activation function applied to the input layers in order to pass the input values directly to the neurons in the subsequent hidden layers. A few key points about the input layer are as follows:

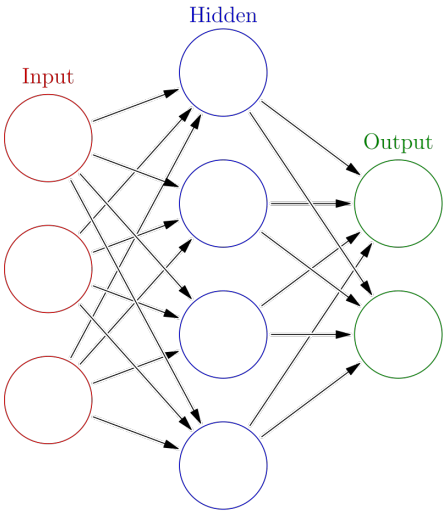

Figure 2.9: An artificial neural network - [14]

  - Number of neurons: This is determined by the number of input features or attributes in the dataset. For example, if you have a dataset with 10 input features, there will be 10 neurons in the input layer.
  - No activation function: There are no activation functions applied to input data by neurons in the input layer. The input values are simply passed on to the next layer (i.e., the first hidden layer).

- Hidden Layer: In ANNs, hidden layers consist of multiple neurons that are connected through weighted connections to neurons in adjacent layers to learn specific features or patterns. The number of hidden layers and neurons in an ANN affects the complexity and performance of the ANN. This weight is adjusted during training to minimize the difference between predicted and actual outputs. Adding more hidden layers and neurons to an ANN allows it to learn more complex patterns and representations, but may also increase the risk of overfitting.

- Output Layer: The output layer of an artificial neural network is responsible for generating the final predictions and plays a crucial role in the training process. It is dependent on the type of problem being solved, such as classification or regression, whether the output layer uses specific architectures and activation functions. The output layer has the following characteristics:

- Number of Neurons: In classification tasks, it corresponds to the number of classes, while in regression tasks, it is usually a single neuron that predicts a continuous value.
- Activation function: Neurons in the output layer use activation functions to transform their input into final output. The choice of an activation function depends on the problem being solved.
  * A sigmoid function is often used for binary classification tasks to convert input into a probability range between 0 and 1.
  * Multi-class classification is commonly done using the softmax function, which converts inputs into probability distributions.
  * Regression tasks typically use linear activation functions, which leave the input unchanged, providing continuous output.
- Loss function: The output layer is associated with a loss function (also called a cost or objective function) that measures the difference between the predicted values and the actual target values. Training reduces this loss function. Regression problems commonly use Mean Squared Error (MSE), binary classification problems use Binary Cross-Entropy, and multiclass classification problems use Categorical Cross-Entropy.
- Backpropagation: The output layer plays a critical role during neural network training. From the output layer to the hidden layers, backpropagation computes the loss function gradients with respect to weights and biases within the network. Weights and biases are updated by gradient descent, Adam, RMSprop, etc.

In the absence of an activation function, the output of every layer is a linear function of the input of the preceding layer regardless of the layer count. When the activation function is employed, it contributes nonlinear components to the neurone, allowing the neural network to approach any nonlinear function arbitrarily and be used with several nonlinear models. An ANNś architecture refers to the number of layers between the input and output layers. There are two types of ANN architecture:

1. Single Layer Perceptron
2. Multi-Layer Perceptron

2.1.6 Single Layer Perceptron-SLP

One of the simplest types of artificial neural networks is the Single Layer Perceptron (SLP), which consists of a single layer of neurons (also called nodes or units) that are directly connected to inputs. This neural network is designed to solve binary classification problems and serves as the basis for more complex models. Using the Perceptron Learning Algorithm, the Single Layer Perceptron is trained through an iterative process that adjusts the weighting of the connections according to the difference between the predicted output and the actual output. Only linearly separable data can guarantee convergence for the Perceptron Learning Algorithm. It converges when the maximum number of iterations has been reached or the model correctly classifies all training samples. Single Layer Perceptron is also known as the Feed Forward Neural Network. Figure 2.10 shows the architecture of a single-layer perceptron.

The following is the perceptron model:

$$f = \sum_{j=0}^{k} x_j w_i + b \tag{2.2}$$

$$y^{'} = f(y) \tag{2.3}$$

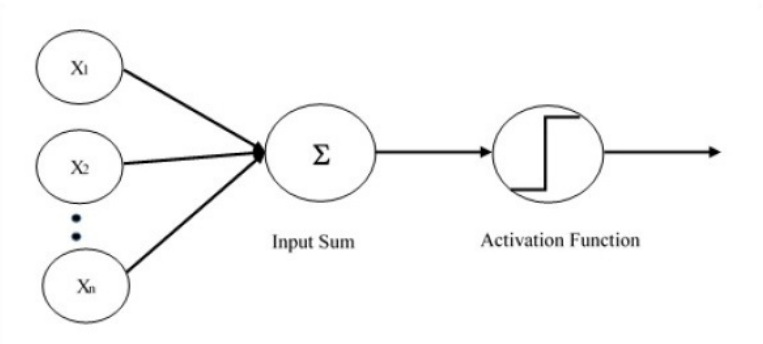

Figure 2.10: Architecture of Single Layer Perceptron - [15]

2.1.7 Multi-Layer Perceptron-MLP

In the context of neural networks, the Multi-Layer Perceptron (MLP) typically consists of at least three layers:

1. Input Layer,

2. Hidden Layer and

3. Output Layer.

Accept the input nodes; Each node in the input layer is connected to a nonlinear activation function. A backpropagation learning algorithm is used to train neurons for the MLP algorithm. Data is then fed back into the network in a forward feeding manner. Among the main advantages of using an MLP is the ability to distinguish between non-linear and linear data. Figure 2.11 shows a simple MLP architecture with two hidden layers.

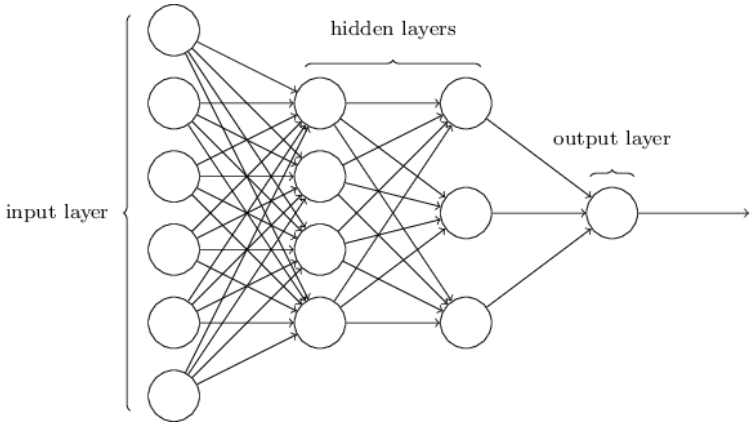

Figure 2.11: Architecture of Multilayer Perceptron - [16]

2.1.8 Recurrent Neural Network-RNN

As humans do not continuously begin to think, their ideas persist and this seems to be a significant limitation of conventional neural networks. However, recurrent neural networks address this limitation. As a result of the following problems with feed-forward neural networks, recurrent neural networks were developed. They are networks containing loops that allow information to be persistent.:

- can't deal with consecutive data
- unable to remember earlier inputs

A recurrent Neural Network, commonly known as RNN, is a particular type of ANN which is generally used for sequential data modeling. Unlike deep neural networks that

assume inputs and outputs are mutually exclusive, recurrent neural networks' outputs depend on the previous items in sequence. It is capable of handling sequential data, such as natural language, speech, time-series, video, music, and stock market data. Unlike feedforward networks, which have distinct weights across each node, recurrent networks have a "memory" cell that stores all the information about the calculations. Each layer of the network shares the same weight parameter. In contrast to feedforward networks, which have distinct weights for each node, they all share the same weight parameter. The weights are still modified by the back-propagation and gradient descent techniques [17] to support learning algorithms. The internal architecture of an RNN is shown in Figure 2.12.

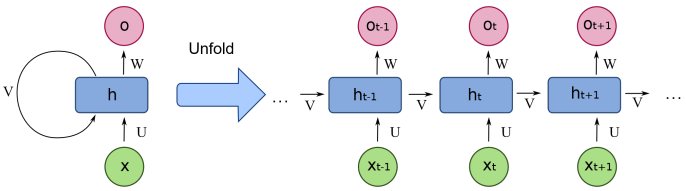

Figure 2.12: Archiecture of RNN - [18]

'X' is the input layer of the neural network that receives and analyzes input before passing it on to the middle layer. 'H' is the middle layer where there may be one or more hidden layers, each with its own activation functions, weights, and biases. You can use a recurrent neural network if the different parameters of different hidden layers are not affected by the preceding layer, i.e., the neural network doesn't have memory. Recurrent Neural Networks will standardize the varied activation functions, weights, and biases, resulting in identical values for each hidden layer. The biggest problem with an RNN is the vanishing gradient problem (VGP) or exploding gradient problem. Instead of producing numerous hidden layers, it will generate only one and loop over it as many times as needed. The problem is primarily caused by poorly customized network parameters and hyperparameters. The gradient shrinks too much and consistently drops to a minimum value due to the vanishing gradient problem (sometimes negative). Because of this, the model performs poorly and is inaccurate. It is possible that the model is unable to categorize or predict what it is supposed to. An exploding gradient occurs when the slope of a neural network increases exponentially

rather than decreasing during training. During training, large error gradients build up, leading to very large changes in neural network weights, which cause this issue. Overly long training times, poor performance, and poor accuracy are the main concerns regarding gradient problems. At timestep (t), here is the equation:

$$h_t = \sigma(U.X_t + W.h_t - 1) \tag{2.4}$$

$$y_t = softmax(V.h_t) \tag{2.5}$$

where:

σ = Tan H activation function
U = The hidden layer weight
V = The output layer weight
W = Same weight vector for different timesteps
X = Input vector with sequence
y = Output vector with sequence

In feedforward networks one input is mapped to one output. Recurrent neural networks (RNNs) are used for a variety of purposes, including sentiment categorisation, machine translation, and music synthesis where the inputs and outputs can have varying lengths. RNN has the following models:

- One to one
  Traditional neural networks employ a one to one architecture which is shown in Figure 2.13

- One to many
  A single input in a one-to-many network can result in many outputs. One application of one to many networks is in the production of music. Figure 2.14 shows One to many RNN architecture.

- Many to one
  In this instance, a number of inputs from several time steps result in a single output. These networks are used in sentiment analysis or emotion identification when a word sequence determines the class label. Figure 2.15 shows Many to one RNN architecture

Figure 2.13: One to one RNN architecture - [17]

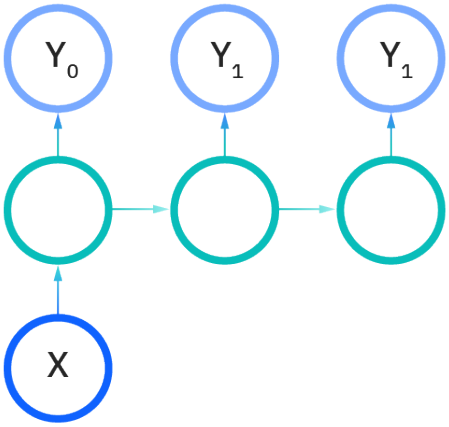

Figure 2.14: One to many RNN architecture - [17]

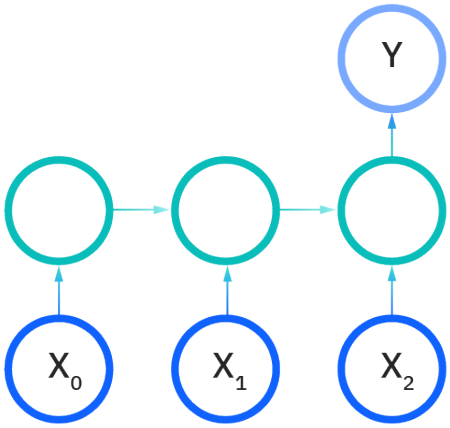

Figure 2.15: Many to one RNN architecture - [17]

- Many to many
  Machine translation employs many-to-many networks. Figure 2.16 Many to many RNN architecture

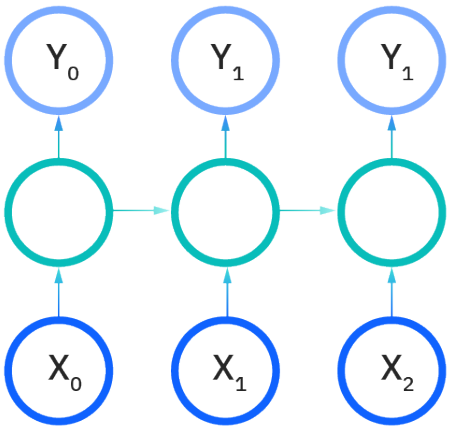

Figure 2.16: Many to many RNN architecture - [17]

2.1.9 Long Short-Term Memory-LSTM

In Hochreiter et al. [19], LSTM was developed as a variation of the basic RNN algorithm. An RNN has the problem of only holding information about past states, causing the vanishing gradient problem. This has been solved by LSTM, which has an identical chaining structure instead of just one neural network layer. Machine translation, speech recognition, and other complex problem domains require this behavior. As shown in figure 2.17, the LSTM has four gates: forget gates, candidate cate gates, input gates, and output gates. Except for the candidate gate, which uses the tanH activation function, all gates are single-layer neural networks with sigmoid activation functions. Similar to the nodes of a neural network, these gates act on the signals they receive and block or pass on information based on its intensity and significance, which they filter using their own sets of weights. As a recurrent neural network learns, it alters these weights, along with those controlling inputs and hidden states. Essentially, cells make educated predictions, back-propagate error, and modify weights by gradient descent to determine when to allow data to enter, exit, or be erased.

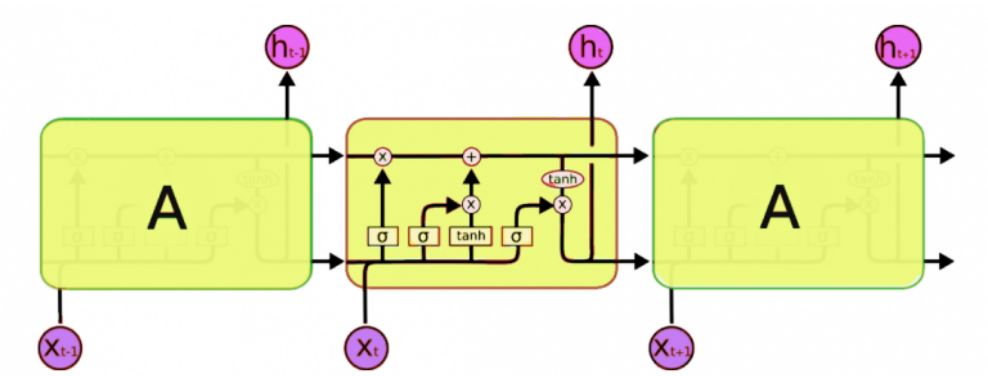

Figure 2.17: Architecture of LSTM - [20]

Figure 2.18 shows LSTM at the T time step.

2.1.10 Gated Recurrent Unit-GRU

GRUs were introduced by Kyunghyun Cho et al. /citecho2014learning in 2014. They employ update and reset gates in order to solve the vanishing gradient problem associated with recurrent neural networks. Output data is determined by these two vectors. GRUs are LSTMs because they are designed similarly and produce similar results in

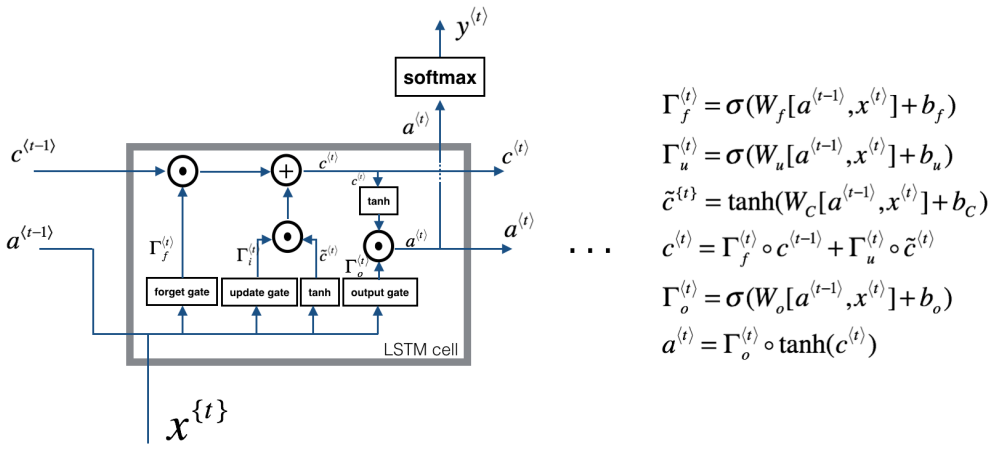

Figure 2.18: LSTM with formula - [21]

some cases. These two vectors are responsible for determining output data. A model can be trained to maintain old data without washing it or removing irrelevant data. The update gate helps determine how much historical data should be transmitted to the future. This is especially powerful since the model can select to duplicate all historical data and remove the possibility of the vanishing gradient problem. As a result of this gate, the model determines how much past data is to be forgotten.

The GRU network has fewer parameters than an LSTM. It only has two gates, while the LSTM network has three gates. A LSTM consists of an input gate, a forget gate, and an output gate. GRU consists of a Reset gate and an Update gate. LSTMs have two states. The hidden or "cell" state and the long term or "cell" state are both types of memory. The only possible state for GRU is "Hidden" (Ht).

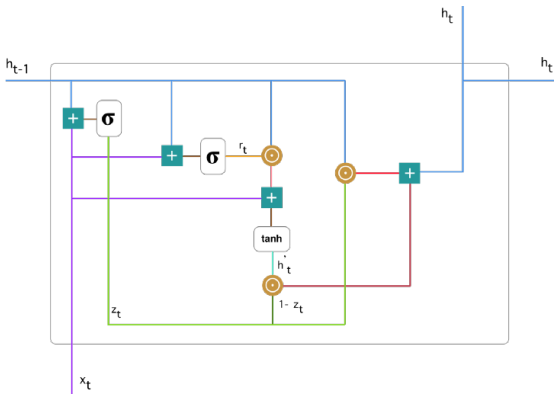

Figure 2.19: Detailed version of that single GRU - [22]

2.1.11 Convolutional Neural Network-CNN

A CNN was developed and used for the first time in the 1980s. It is mainly used for image recognition, image classification, object detection, and face recognition. The convolutional neural network structure is based on the neurobiological observation that neurons in the visual cortex can recognize changes in direction and location without problems. It can learn unique characteristics regardless of its orientation or position, unlike a multilayer perceptron. Figure 2.20 shows the architecture of CNN.

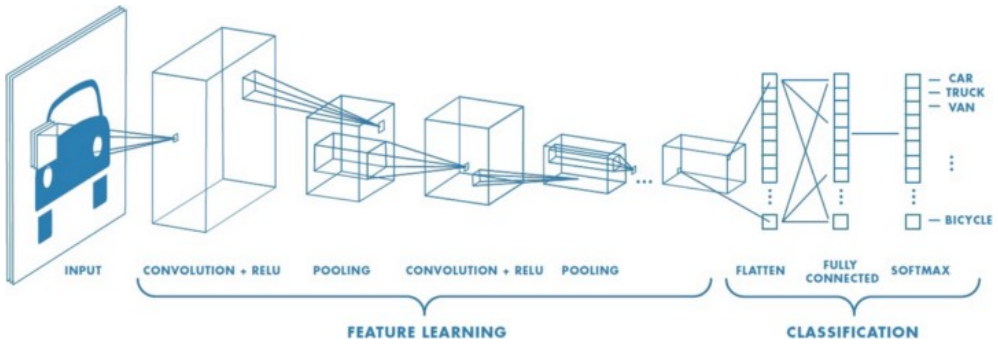

Figure 2.20: Architecture of a CNN - [23]

2.1.12 Support Vector Machine-SVM

Among the most widely used supervised machine learning algorithms is the Support Vector Machine. A hyperplane maximizing the margin between the support vectors is the goal of the SVM in high dimensional feature spaces, significantly when the number of features exceeds the number of training examples. It is possible to train the SVM to handle nonlinear input spaces by using different kernel functions. Figure 2.21 shows a simplified explanation of the SVM.

2.1.13 Rough Set Theory

RS is a method for discovering structural relationships in noisy data based on establishing equivalence classes within the given training data. It was first introduced by Pawlak [25].

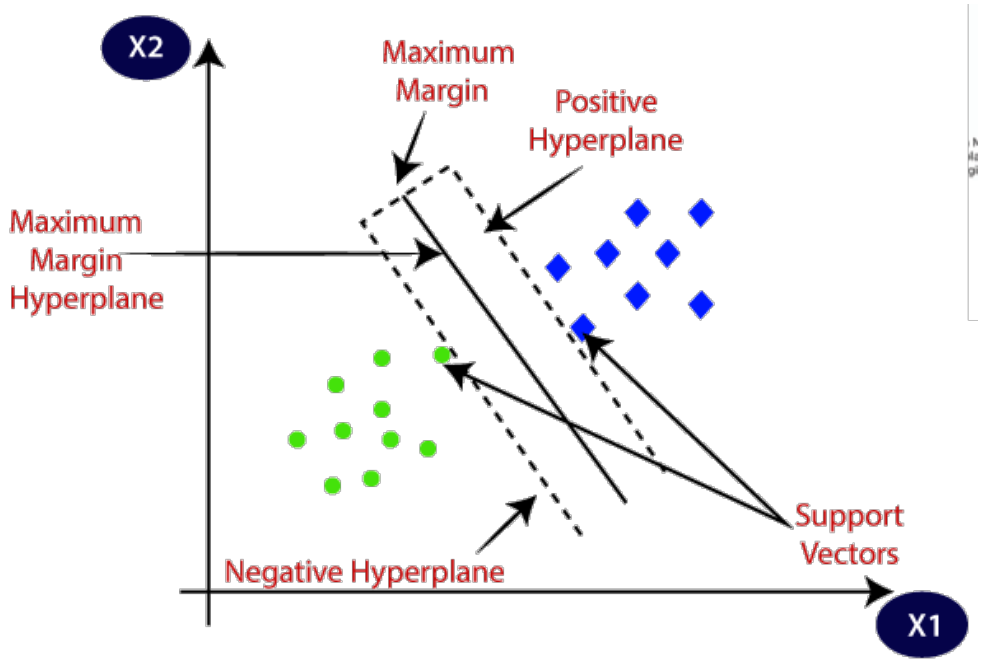

Figure 2.21: Support Vector Machine - [24]

2.1.14 Moving Average Convergence/Divergence-MACD

The MACD indicator is a trend-following momentum indicator that represents the relationship between two moving averages. The Formul is:

$$MACD = EMA_{12} - EMA_{26} \tag{2.6}$$

Figure 2.22 shows an example MACD Graph.

2.1.15 Simple Moving Average-SMA

The simple moving average, or SMA for short, represents the average price over an extended period of time. MA5, MA12, and MA26 are some examples of SMAs that are commonly used. An example of SMA is shown in the below Figure 2.23

2.1.16 Rate of Change-ROC

A rate-of-change (ROC) is a measure of the difference between the price at the moment and the price "n" periods ago. This oscillates around the zero line as it fluctuates in

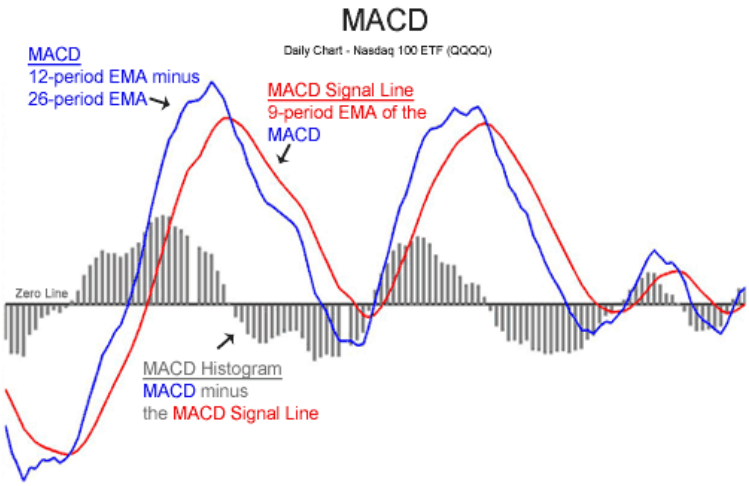

Figure 2.22: MACD Graph - [26]

Figure 2.23: Simple Moving Average - [27]

a positive or negative direction based on the rate of change. The formula of ROC is given below

$$ROC = \frac{(Closet – Closet_n)}{Closet_n} * 100 \tag{2.7}$$

2.1.17 Relative Strength Index-RSI

The Relative Strength Index (RSI) is a popular technical indicator used in financial markets to assess the strength and momentum of price movements. It was developed by J. Welles Wilder and introduced in his 1978 book, "New Concepts in Technical Trading Systems" [28]. The RSI is primarily used in the analysis of stocks, but it can also be applied to other financial instruments such as commodities, currencies, and indices. The formula of Relative Strength is given below:

$$RSI = 100 – \frac{100}{(1 + RS)} \tag{2.8}$$

$$RS(\text{Relative Strength}) = \frac{\text{Average of X days up closes}}{\text{Average of X days down closes}} \tag{2.9}$$

Figure 2.24 shows the RSI graph. RSI is usually calculated over a 14-day period, but the time frame can be adjusted according to the analyst's preferences. The RSI is useful for identifying overbought and oversold conditions in a market, as well as reversed or diverging trends. In general, RSI values above 70 indicate overbought conditions and could signal a price correction or reversal. The RSI value below 30 indicates that an instrument is undervalued, which could result in a price increase or trend reversal. Additionally, some traders use the 50 level as a signal to indicate a trend change, with values above 50 indicating bullish momentum and values below 50 indicating bearish momentum. In order to make more informed trading decisions, the RSI should be used in conjunction with other technical analysis tools and indicators.

2.1.18 Chaikin Money Flow

Chaikin Money Flow (CMF) is a technical analysis indicator developed by Marc Chaikin [30] to measure the flow of money into and out of a financial instrument over a specified period. Based on the accumulation/distribution concept, the indicator combines price and volume information to provide insight into the buying and selling pressure within a market. It is used to track the Money Flow Volume over a specific period, generally 20

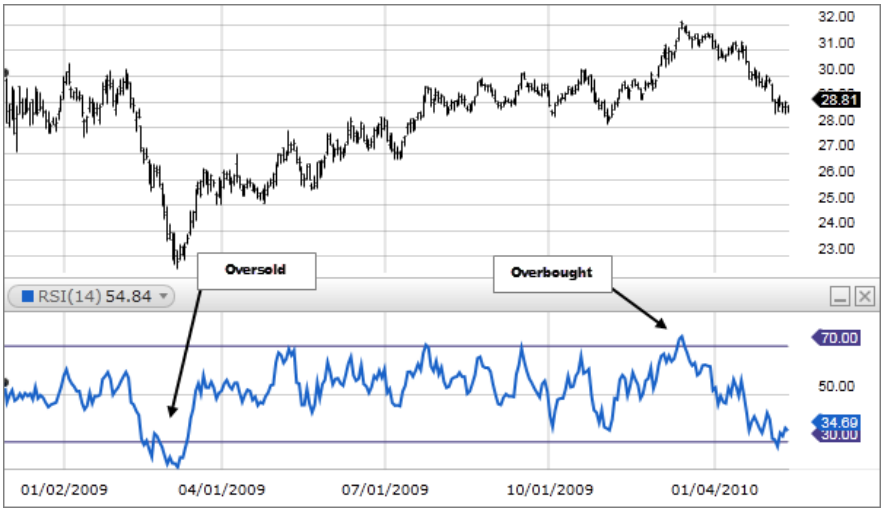

Figure 2.24: RSI Graph - [29]

to 21 days. The indicator oscillates around the zero-base line called the accumulation distribution line. Figure 2.25 Chaikin Money Flow shows a graph of Chaikin Money Flow. The calculation of Chaikin Money Flow involves three main steps:

1. **Calculate the Money Flow Multiplier (MFM) for each period:**

$$MFM = \frac{(Close - Low) - (High - Close)}{(High - Low)}$$

Where:

Close = Closing price for the period

Low = Lowest price for the period

High = Highest price for the period

2. **Calculate the Money Flow Volume (MFV) for each period:**

$$MFV = MFM \times Volume$$

Where:

Volume = Trading volume for the period

3. **Calculate the Chaikin Money Flow (CMF) over a specified number of periods (typically 20 or 21):**

$$CMF = \frac{(\sum of MFV for the specified number of periods)}{(\sum of Volume for the specified number of periods)}$$

Figure 2.25: Chaikin Money Flow - [31]

The resulting CMF value ranges between -1 and 1. Positive values indicate that the financial instrument is being accumulated, which may lead to a rise in price. Traders and analysts use the CMF to identify trend changes, divergences, and confirmations of price movements based on the fact that there is more selling pressure. A negative CMF value indicates that the instrument is being traded and that the price may fall. Alternatively, a divergence between the CMF and the price of a financial instrument may signal a potential trend reversal. Additionally, a cross above or below zero may be interpreted as a buy or sell signal, respectively. In order to make more informed trading decisions, Chaikin Money Flow should be used in conjunction with other technical indicators.

2.1.19 Mean Squared Error (MSE)

The Mean Square Error (MSE) is a popular metric in machine learning and statistics for evaluating regression models which predict continuous values. One of the advantages of MSE is its simplicity and interpretability. Essentially, it measures the average squared difference between the actual value (ground truth) and the predicted value (estimation). As part of the training process, the algorithm aims to minimize the MSE by adjusting its parameters in order to produce more accurate predictions. As a result of squaring the differences, the metric emphasizes the importance of accurate predictions by giving more weight to error rates that are larger. However, the squared terms may cause the metric to be sensitive to outliers, since large errors can have a disproportionately significant impact on the overall score. Whenever this sensitivity is undesirable, alternative metrics can be used, such as Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE). MSE is defined as follows:

$$MSE = \frac{1}{n} \sum (actual - forecast)^2 \tag{2.10}$$

Where: n = number of items, = summation notation, Actual = original or observed y-value, Forecast = y-value from regression.

2.1.20 Root Mean Squared Error (RMSE)

The Root Mean Square Error (RMSE) is a widely used metric in machine learning and statistics to evaluate regression models that predict continuous values. This is

the square root of Mean Squared Error (MSE). RMSE is calculated by taking the square root of MSE in order to restore the error to its original scale, which makes it easier to interpret. It is used in a variety of machine learning algorithms, such as linear regression, decision trees, and neural networks. During the training process, the algorithm tries to minimize RMSE by adjusting its parameters to produce more accurate predictions. The key advantage of RMSE over MSE is that it is expressed in the same units as the predicted and true values. This makes it easier to understand the errors' magnitude. Like MSE, RMSE is sensitive to outliers, as large errors have a substantial impact on the overall score. In cases where this sensitivity is undesirable, alternative metrics, such as Mean Absolute Error (MAE), can be used. RMSE is defined as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_i - Y_i)^2} \tag{2.11}$$

Where:

- n: the number of observations
- P_i: the predicted value for observation i
- Y_i: the actual value for observation i

2.1.21 Multicollinearity

When multiple independent variables in a multiple regression model are highly correlated with one another, multicollinearity occurs [32]. A high correlation between independent variables can make it difficult to estimate model parameters accurately. It is difficult to determine the contribution of each variable to the outcome when multicollinearity exists [33]. It can lead to incorrect inferences about relationships between the independent variables and the dependent variables. Calculating Variance Inflation Factors (VIF) and analyzing condition indices are typically used to detect multicollinearity [34]. Multicollinearity can be addressed by removing one or more of the correlated independent variables, combining them, or penalizing high coefficient values through regularization techniques [35, 36].

2.1.22 Mean Absolute Percentage Error- MAPE

The Mean Absolute Percentage Error (MAPE) is a widely used metric for assessing forecasting model accuracy, particularly in regression and time series analysis [37]. In machine learning applications, where the aim is to evaluate predictive models' performance and identify opportunities for improvement, it is especially useful. [37] A MAPE is calculated by dividing the actual value by the predicted value and then averaging these percentages over all data points. In addition, it is expressed as a percentage, making it easier to compare and understand different models' relative performance. In order to compute MAPE, percentage errors are summarised without regard to sign. Because absolute percentage errors are used, positive and negative errors don't cancel one another out. The smaller the MAPE, the better the forecast. The Mean Absolute Percentage Error (MAPE) equation is as follows:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|Actual_i - Predicted_i|}{|Actual_i|}\right) \times 100 \tag{2.12}$$

where n is the number of data points, and $Actual_i$ and $Predicted_i$ represent the actual and predicted values, respectively.

MAPE has several advantages in machine learning:

- Interpretability: As a percentage, MAPE is straightforward to interpret and communicate. It provides an intuitive measure of the average error as a proportion of the actual values [37].
- Scale-independent: Since it is calculated as a percentage, MAPE is independent of the scale of the data, making it suitable for comparing models with different units or scales [37].
- Applicable to various models: MAPE can be used to assess the performance of a wide range of machine learning models, including regression, time series forecasting, and even classification problems with ordinal outcomes [37].

However, MAPE has some limitations:

- Undefined for zero actual values: MAPE is undefined when actual values are zero, as it involves dividing by the actual value. This can be problematic when dealing with data containing zeros or very small values [37].

- Asymmetric: MAPE can be biased towards underpredictions, as the error percentage will be higher for underpredictions than overpredictions when dealing with the same absolute error [37].

Despite these limitations, MAPE remains a popular metric for evaluating model performance in machine learning and is particularly useful for comparing the accuracy of different models in a way that is easy to understand and interpret.

2.1.23 R2 Score

The R-squared (R^2) score measures the goodness of fit of a regression model, indicating how much of the variation in the dependent variable is explained by the independent variables [38]. R^2 values range from 0 to 1, with a higher value indicating a better fit. Here's a more detailed explanation:

1. R^2 compares the model's performance to a baseline model. A baseline model is a simple model that only considers the mean of the dependent variable for prediction, ignoring the independent variables. An R^2 of 0 means that the model is no better than the baseline model, while an R^2 of 1 indicates that the model perfectly explains the variation in the dependent variable.

2. R^2 can be calculated using the following formula:

$$R = 1 - \frac{SumofSquaredResiduals}{TotalSumofSquares} \tag{2.13}$$

   where Sum of Squared Residuals represents the sum of the squared differences between the predicted values and the actual values, and Total Sum of Squares is the sum of the squared differences between the actual values and the mean of the dependent variable.

3. In machine learning, R^2 is commonly used as an evaluation metric for regression problems. It provides a quantitative measure of how well the model's predictions match the actual data. However, R^2 alone may not be sufficient to evaluate the model's performance fully, as it doesn't account for overfitting or the complexity of the model. In such cases, adjusted R^2, which penalizes the model for adding unnecessary variables, can be a more appropriate metric.

4. It is important to note that a high R^2 does not necessarily imply causality. A high R^2 can be observed even in cases where there is no causal relationship between the independent and dependent variables.

2.1.24 Mean Absolute Error- MAE

Mean Absolute Error (MAE) is a popular evaluation metric used in Machine Learning, especially for regression problems. It is a measure of the average magnitude of the errors between the predicted values and the actual values, without considering their direction. The MAE is calculated as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.14}$$

where:

- n is the number of data points,
- y_i is the actual value of the ith data point,
- $\hat{y}_i$ is the predicted value of the ith data point,
- $\sum$ denotes the summation over all data points.

The primary usage of MAE in Machine Learning is to evaluate the performance of regression models. A lower MAE indicates better predictive accuracy of the model, as it represents the average error made by the model when predicting the target variable. MAE is particularly useful when comparing different models, as it provides a simple and interpretable measure of their predictive performance.

2.2 Literature Review

Sachdeva et al. [39] developed a deep learning model using LSTM to forecast the National Stock Exchange (NSE) of the Indian equity market. The dataset consisted of data from the NIFTY 50 index and INFOSYS Ltd obtained from Yahoo finance between 11 Dec 2007 to 12 Dec 2018. The features included Date, Open, High, Low, and Close prices. The data was divided into train and test sets to prevent snooping bias, where the train set contained data from 11 Dec 2007 to 11 Dec 2017, and the test set

contained the remaining data. MinMaxScaler from the sklearn library was used to scale the data. Two-time steps of 20 and 60 were used in the experiment. The RNN model was trained with 'n' stock prices before time 't' to capture correlation and trend in price movement, and predict the price of 't+1' day. Dropout was used for regularization, and 'Adam' 'RMSProp' were optimizers. After experimenting with different combinations of the optimizer, time steps, and LSTM, the best output was achieved for RMSProp with 60-time steps. The model achieved 97% accuracy in predicting Infosys and Nifty 50 stock market results. The test set Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) were 0.5625 and 0.74, respectively, leading to the conclusion that the deep learning model can predict the NSE stock market.

Banik et al. [40] proposed a hybrid model that combines Rough Set (RS) and Artificial Neural Network (ANN) for predicting movement in the Dhaka Stock Exchange (DSE) with an overall accuracy of 97.89%. The authors used a dataset spanning eight years of DSE data, from January 2004 to June 2012, which was split into a 70-30 train-test set. The dataset included MACD, MA5, MA12, PROC, and RSI technical features, commonly used by financial analysts to predict time series data. The authors first built an RS model using the Rosetta Rough Set Toolkit, applying Johnhon's reducer algorithm and the equal binning discretized method, which generated 12 reducts. The RS model achieved 87% overall prediction accuracy, with 72.31% accuracy in predicting falling stock prices and 80.80% in predicting rising stock prices. For the ANN model, the authors used a 9:15:18:1 architecture, with a learning rate of 0.01 and a momentum factor of 0.90, selecting the sigmoid transfer function for the input and output layers. They used the Levenberg Marquardt's backpropagation algorithm for backpropagation. On average, the ANN model achieved 72.86% accuracy in predicting falling stock prices, 80.74% accuracy in predicting rising stock prices, and an overall accuracy of 78.37%. The authors used the RS model to perform reducts and generate decision rules based on the predicted index for the hybrid RS-ANN model, which achieved higher prediction accuracy than the RS and ANN models, with an accuracy of 97.89%, 95.93%, and 96.87% for overall, falling, and rising stock price prediction, respectively. Based on their experiment results, the authors recommended the hybrid model as a useful tool to guide investors in the DSE.

Usmani et al. [41] utilized Single Layer Perceptron (SLP), Multi-Layer Perceptron (MLP), Radial Basis Function (RBF), and Support Vector Machine (SVM) to predict

the Karachi Stock Exchange (KSE). The models were trained using nine parameters, including Oil rate, gold rate, silver rate, Interest rate, Foreign Exchange Rate (FEX), News, Social Media feed, Simple Moving Average (SMA), and Autoregressive Integrated Moving Average (ARIMA). The dataset covered the period from September 2015 to January 2016, and the attribute values were normalized between -1 to 1 due to their negative/positive nature. Because of dependencies on News and Social Media Feed (Twitter), which were only available in limited quantities, only 100 records were collected and split into 70-30 ratios for the train-test sets. OpinionFinder library was used for text mining, which resulted in many non-English words in the corpus, potentially compromising the models' performance. Among the four developed models, MLP achieved the highest test set accuracy of 77%, while SVM showed surprising 100% accuracy on the training set, but its test set performance was the lowest at 60%. Despite limited resources and data, Usmani et al. [41] concluded that machine learning techniques could be used to predict the KSE.

In their study, Bhowmick et al. compared the prediction efficiencies of classical statistical models for financial forecasting with advanced neural networks such as RNN, LSTM, LSTM peephole, and Gated recurrent unit (GRU). They found that NN models outperformed classical models like ARIMA, indicating that NN models could capture the nonlinearity of the market better. The performance of the four NN models was comparable, and the dataset comprised five companies from DSE. They dropped the NAN values before applying MinMaxScaler from the sklearn library and used four prices (Average, Close, Low High) as input to the models. The dataset was split into 80-10-10% for the train-test-validation set. For the NN models, they used a 4-20-50-1 architecture with a learning rate of 0.00001 and the Adam optimizer, which is suited for non-stationary and sparse data. The experiment was run for 100 epochs, and TensorFlow and Keras were used for NN models. On the other hand, the Statsmodels library trained classical models such as autocorrelation (ACF), partial-autocorrelation (PCF), the first differential autocorrelation, and ARIMA. The Moving Average Convergence/Divergence (MACD) indicator was also used to validate the experiment, but it was later deemed uninformative for prediction.

Selvin and colleagues developed a short-term prediction model for NSE-listed companies using RNN, LSTM, and CNN models with a sliding window approach. They used minute-wise stock data of 1721 NSE-listed companies between July'14 and June'15

and focused on three companies, two from the IT sector and one from the pharmaceutical industry. They used a sliding window size of 100 minutes, which was chosen based on the error rate of various window sizes. The training data was of Infosys from 1 July 2014 to 14 October 2014, and the test data consisted of all three companies chosen earlier and lied between 16 October 2014 and 28 November 2014. They trained all models for 1000 epochs and found that CNN outperformed RNN and LSTM models. CNN's superior accuracy was due to its ability to predict values based on the current window and capture dynamic changes and patterns. They compared NN models' results with ARIMA output and found that NN models performed better than ARIMA.

Khare and colleagues [42] developed models to predict short-term prices of 10 individual stocks on the New York Stock Exchange using two types of artificial neural networks: feed-forward neural networks and recurrent neural networks. The dataset contained minute-by-minute stock price data recorded over one year. They normalized the data using MinMaxScaler and split it into a 70-30 ratio for training and testing. Although the study did not mention which features were provided to the models, the mentioned categories (trend, oscillator, and momentum) belong to the parts. The experiment results showed that both MLP and LSTM models could predict the up-down price trend, but MLP model exhibited higher accuracy in predicting the stock price compared to the LSTM model.

In their study, Li et al. [43] examined the China Stock Market using six forecasting models and nine feature combinations chosen from 23 commonly used technical indicators. To analyze the impact of time periods, they applied each of the forecasting methods 400 times over 12 different periods using a moving-window approach. Furthermore, they compared the relationship between different feature combinations and model performance. Their research aimed to compare the accuracy of shallow machine learning methods such as SVM, Naive Bayes, and Decision Tree with deep understanding methods like MLP, RNN, and LSTM. The experimental results revealed that the deep learning methods outperformed the other models in terms of accuracy.

In their study, Porshnev et al. [44] aimed to investigate whether Twitter sentiment analysis could enhance the accuracy of stock market prediction. However, they found that using SVM, the best accuracy they could achieve was only 64.10%, leading them to reject their hypothesis. They further noted that the low accuracy could be due to the limited number of data points available.

Most of the authors use MinMaxScaler for normalization and RNN for model training. Data was split by 70/80-30/20 ratio for training and testing. Table 1 Comparison of Literature Review shows a tabular form of different aspects of the literature review. Most of the authors use MinMaxScaler for normalization and RNN for model training. Data was split by 70/80-30/20 ratio for training and testing

Table 2.1: Comparison of Literature Review

Ref	Source	Features	Scaler	Train Test split	Models	Conclusion	Research Gaps
Sachdeva et al. [39]	NSE	Date, Open, High, Low, and Close prices	Min Max Scaler	-	RNN	the deep learning model could predict the NSE stock market.	Based on Indian equity markets; Optimizer, time steps, and LSTM combined
Banik et al. [40]	DSE	MACD, MA5, MA12, PROC, RSI	-	70-30	RS, ANN, Hybrid (RS-ANN)	the hybrid model as a tool to guide investors of DSE	Focused on Dhaka Stock Exchange; Hybrid model

Usmani et al. [41]	KSE	Oil rate, gold rate, silver rate, Interest rate, Foreign Exchange Rate (FEX), News, Social Media feed, Simple Moving Average (SMA), Autoregressive Integrated Moving Average (ARIMA)	-	70-30	SLP, MLP, RBF, SVM	the models proved that the KSE could be predicted using machine learning techniques	Limited dataset; Non-English words in corpus; Dependencies on News and Social Media Feed
Bhowmik et al. [45]	DSE	Four prices, Average, Close, Low & High	Min Max Scaler	80-10-10	RNN, LSTM, LSTM peephole, GRU, ARIMA	-	Limited to Dhaka Stock Exchange; Performance comparison between classical models and advanced neural networks

Selvin et al. [46]	NSE	-	-	-	RNN, LSTM, CNN, ARIMA	NN models outperformed ARIMA	Focused on NSE-listed companies; Short-short-term prediction model; Comparison with ARIMA
Khare et al. [42]	York Stock Exchange	Trend, Oscillator, and Momentum	Min Max Scaler	70-30	Feed Forward Neural Networks, RNN, MLP, LSTM	both LSTM and MLP models could predict the up-down price trend.	Limited to New York Stock Exchange; Short-term price prediction
Li et al. [43]	China Stock Market	23 technical indicators	-	-	SVM, Naive Bayes, Decision Tree, MLP, RNN, LSTM	The performance of deep learning modes is better than other models in terms of accuracy	Comparing shallow and deep learning models; Focused on China Stock Market
Porshnev et al. [44]	-	-	-	-	SVM	They achieved the best accuracy of 64.10% using SVM, therefore rejecting their earlier hypothesis	Low data points; Limited accuracy attributed to Twitter sentiment analysis

Chapter 3

Methodology

Time series are groups of data points stored based on when collected. Time-series modelling is used to identify past patterns and predict future events. Data science says "Garbage in, garbage out" is very true. This chapter describes the procedures used to obtain the results.

3.1 Data normalization

As a rule of thumb, neural networks perform best when their input values are small, typically between zero and one. The model is more likely to learn from small numbers because the learning algorithms are based on gradient updates being made to its weight parameters. Increasing values will cause the update to occur more rapidly, resulting in a faster learning process, while decreasing values will result in a slower learning process. Typically, a dataset contains large values, so in order to incorporate them into our model, we need to rescale and normalize them to a range of 0 to 1.

3.2 Dataset Spliting

The dataset is divided into three subsets for evaluating a model: a training set, a validation set, and a test set. In the training phase, the model is trained using the training dataset, while in the validation phase, it is evaluated using the validation dataset. Having a separate test dataset is extremely important to avoid information leaks and to evaluate the model after it has been completed. It is important to note, however, that some information about the validation data is filtered into the model so

that the model can be tuned based on the model's performance. Because the model is trained for the validation data, it is possible to end up with a model that works artificially well with the validation data. We must use a separate and never-before-seen dataset to evaluate our model since actual performance is due to the use of previously unseen data, as opposed to validation data. This is referred to as the test dataset. It is very important that the model does not have any access to any information about the test dataset, not even indirectly. In order to avoid information leaks, it is imperative to have a separate test dataset.

3.3 Time steps

Based on our literature review and background research, we have used four deep neural networks for predicting the stock price as these models work well with sequential data. These models are RNN, LSTM, GRU, and CNN. We have used the five most current share alues as input to the model to predict the price of the sixth day. This allows us to search for temporal patterns. This can be written as:

equation

where xt is the share value at time t.

3.4 Raw data details

A prerequisite to performing machine learning is to have available data to learn from. Because learned models are derived essential, the data must be suitable. Data were collected from [47]. The data is from a DSE-listed entity, 'BEXIMCO.'. Details can be found in [48]. The data lay between 1999-01-27 and 2021-07-29, spanning over 22 years, comprising 5380 rows in total. The Table 3.1 presents the first five entries in a dataset, which contains information on daily stock prices and trading volume. Transaction date (txn_date), opening price (open), highest price (high), lowest price (low), closing price (close), and trading volume (vol) are included within the dataset. An entry dated January 27, 1999, indicated an opening price of 59, a high of 60.4, a low of 59, a closing price of 59.4, and a trading volume of 16,000. The following entries represent daily stock data for the following days: 1/28/1999, 1/30/1999, 1/31/1999, and 2/01/1999.

Table 3.1: Head of Stock Data

txn_date	open	high	low	close	vol
1/27/1999	59	60.4	59	59.4	16000
1/28/1999	58.5	59	58.5	58.6	4100
1/30/1999	58.6	58.7	58.5	58.5	2400
1/31/1999	58	58.8	58	58.4	3200
2/01/1999	58.5	58.5	58	58.2	3700

Table 3.2: Statistical description of the raw data

	mean	std	min	25%	50%	75%	max
open	6.19	1.68	3.90	4.80	6.10	7.00	13.30
high	6.31	1.72	4.00	4.90	6.20	7.20	13.90
low	6.08	1.62	3.90	4.70	6.00	6.90	13.00
close	6.17	1.65	3.90	4.80	6.00	7.00	13.00
vol	621787.05	1152663.00	1.00	109067.50	249273.00	622014.25	16106172.00

They provide an overview of the stock's price movement and trading activity during these dates.

A statistical description of the raw data in Table 3.2 provides insight into the stock's price components (open, high, low, and close) and trading volume. An analysis of the price movement and trading volume of the stock provides a basis for further technical analysis. The average opening price is BDT 6.19, with a standard deviation of BDT 1.68, ranging from a minimum of BDT 3.90 to a maximum of BDT 13.30. The high prices exhibit a mean value of BDT 6.31, a standard deviation of BDT 1.72, and range from BDT 4.00 to BDT 13.90. The low prices have an average of BDT 6.08, a standard deviation of BDT 1.62, and span between BDT 3.90 and BDT 13.00. The closing prices show a mean value of BDT 6.17, a standard deviation of BDT 1.65, with a minimum of BDT 3.90 and a maximum of BDT 13.00. The trading volume has a considerably larger standard deviation of 1,152,663 compared to its mean value of 621,787.05, indicating significant fluctuations in trading activity. The high prices exhibit a minimum of 1 to a maximum of 16,106,172, with quartile values at 109,067.50 (25%), 249,273 (50%), and 622,014.25 (75%).

Figure 3.1 Close price shows the distribution of the close price for the duration.

The red horizontal line shows the mean value during the duration. Between 2008 and 2013 the close price shoot above the mean.

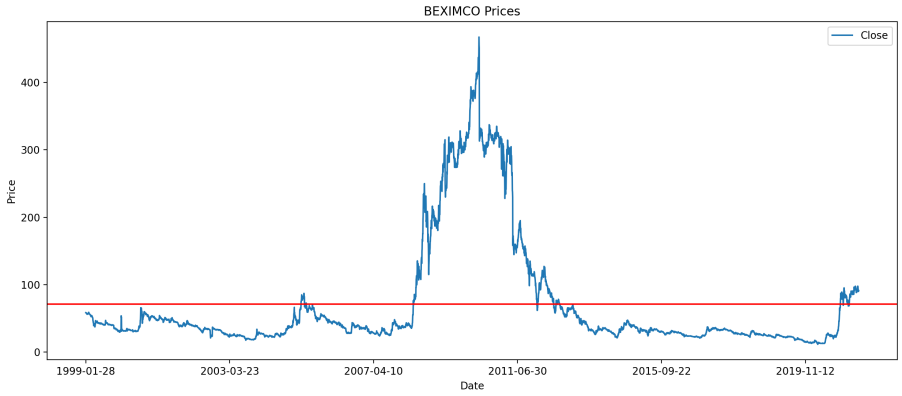

Figure 3.1: Beximco Close Price -

The Figure 3.2 displays a line plot depicting the stock's trading volume over time, with the x-axis representing the timeline and the y-axis representing the trading volume in numerical values. Based on the plot, it is evident that trading volume fluctuates significantly throughout the observed period, with some data points showing noticeable spikes at specific times. As a result of increased market interest, news, or events affecting the stock, these spikes may occur. The volume fluctuated sharply from 2015 onward. It appears from the plot that there have been periods of high activity, as indicated by large values, as well as periods of relatively low activity, as indicated by smaller values. In order to understand the price movements and market dynamics of the stock, it is essential to consider trading volume as a factor.

3.5 Feature Engineering

In addition to the OHLC prices (Opening, High, Low, Close), volume data was also included in the dataset. Our algorithms calculated the percentage rate of change (PROC) of the close price using the pandas [49] method and NumPy [50] method, respectively, as input features, and the log PROC of the close price as input features. Our next step was to calculate the RSI, MACD, and Chaikin Money Flow utilizing OHLC and volume data using Python [51] and then apply the models we developed.

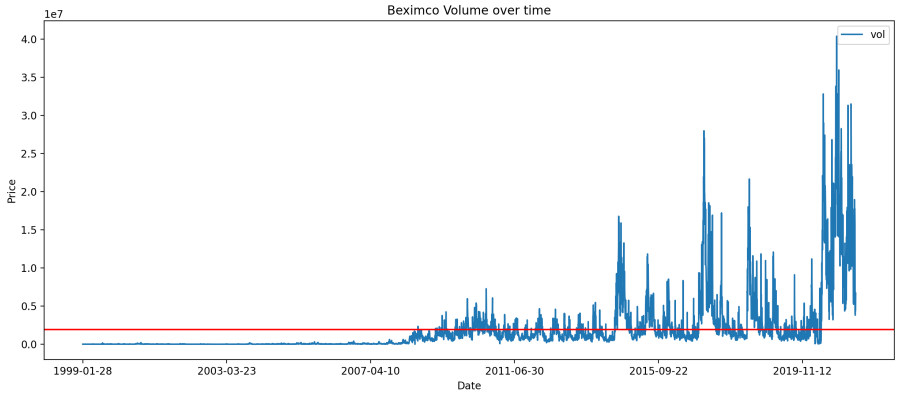

Figure 3.2: Beximco Volume over time -

Table 3.3: Dataset information

	close	vol	close_roc	close_log_roc	rsi	macd_signal	cmf
count	5380	5.38E+03	5380	5380	5380	5380	5380
mean	71.34498	1.90E+06	0.000621	0.000083	47.43512	0.043675	-0.233244
std	85.60475	3.85E+06	0.033361	0.032677	13.35644	3.989931	0.201273
min	11.6	5.00E+00	-0.39341	-0.499904	8.469354	-31.998695	-0.81075
25%	27.5	1.73E+04	-0.0137	-0.013793	38.64222	-0.746857	-0.371534
50%	36.1	7.23E+05	-0.00302	-0.003026	46.16808	-0.238272	-0.253508
75%	59.25	1.85E+06	0.01087	0.010811	55.45625	0.551263	-0.10276
max	467.3	4.04E+07	0.733119	0.549923	94.54839	26.060103	0.549461

Table 3.3 shows the statistical values of the features. Here's a brief technical analysis based on the provided summary statistics:

- Closing Price (close): The average closing price is BDT 71.34, with a standard deviation of BDT 85.60, indicating considerable variability in the closing price over the observed period. The stock price ranged from a minimum of BDT 11.60 to a maximum of BDT 467.30.
- Trading Volume (vol): The average trading volume is 1.90 million shares, with a high standard deviation of 3.85 million shares. This suggests that trading volume is highly variable.

- Close_ROC: The average rate of change for the closing price is 0.0621%, indicating small daily price changes on average. However, the standard deviation of 3.3361% suggests that there have been some days with more significant price moves.

- Close_Log_ROC: The average logarithmic rate of change for the closing price is 0.0083%, which is very close to the average close_ROC value. This further suggests small daily price changes on average.

- RSI: The average RSI value is 47.43, which is close to the midpoint of the RSI range (50). This suggests that, on average, the stock is neither overbought nor oversold. The RSI ranges from a minimum of 8.47 to a maximum of 94.55, indicating that there have been periods of both extreme oversold and overbought conditions.

- MACD_Signal: The average MACD signal line value is 0.0437, which is close to zero. This suggests that, on average, the stock has not shown strong trends in either direction. However, the MACD signal line has ranged from -31.9987 to 26.0601, indicating periods of both bearish and bullish trends.

- Chaikin Money Flow (CMF): The average Chaikin Money Flow value is -0.2332, which is negative, suggesting that, on average, there has been more distribution than accumulation of the stock. The CMF ranges from -0.8108 to 0.5495, indicating periods of both strong distribution and accumulation of money flow.

With an average closing price of BDT 71.34 and a variable trading volume, the technical analysis indicates the stock has experienced periods of both bullish and bearish trends. The stock has seen periods of both overbought and oversold conditions, as well as periods of accumulation.

Table 3.4: Feature Correlation

	open	high	low	close	vol	roc	log_roc	rsi	macd	cmf
open	1.00	1.00	1.00	0.99	.009	-0.002	-0.003	.178	0.141	0.149
high	1.00	1.00	1.00	1.00	.011	0.008	0.007	.184	0.144	0.154
low	1.00	1.00	1.00	1.00	.008	0.008	0.007	0.18	0.139	0.147
close	0.99	1.00	1.00	1.00	0.01	0.017	0.016	.185	0.142	0.153
vol	.009	.011	.008	.011	1.00	0.122	0.120	0.38	0.153	0.379

roc	-.002	.008	.008	.017	.122	1.000	0.994	.354	0.015	0.184
log_roc	-.003	.007	.007	.016	0.12	0.994	1.000	.352	0.012	0.179
rsi	.178	0.18	0.18	.185	.388	0.354	0.352	1.00	0.456	0.642
macd	0.14	0.14	0.13	0.14	0.15	0.015	0.012	0.45	1.000	0.282
cmf	0.14	0.15	.14	0.15	0.37	0.184	0.179	0.64	0.282	1.000

According to Table Table 3.4, the price components of the stock (open, high, low, and close) are highly correlated, suggesting that they move together. Though trading volume has a weak correlation with price components, its impact on momentum indicators, trend indicators, and money flow indicators (RSI (0.38), respectively, is considerably greater. The stock's momentum, measured by RSI, is closely associated with its trend (MACD, correlation of 0.456) and accumulation/distribution of money flow (CMF, correlation of 0.642). By contrast, ROC and Log_ROC have a moderate correlation with RSI (0.354 and 0.352, respectively) but weaker correlations with other indicators and price components. Based on this analysis, it is important to understand the interplay between various technical indicators, in particular momentum (RSI), trend (MACD), and money flow (CMF), in determining the stock's price movement.

In the heatmap in Figure 3.3, the correlation matrix between price components and technical indicators can be seen, with a color scheme ranging from dark blue (strong negative correlation) to dark red (strong positive correlation). Since the price components (open, high, low, and close) are strongly correlated, they tend to move in tandem, whereas trading volume is weakly correlated with these price components. Indicators based on momentum, such as the ROC, RSI, and MACD, and those based on money flow, such as the CMF, have a moderate to strong positive correlation with trend-based indicators. Therefore, the article emphasizes the importance of understanding the stock's momentum in relation to how money flows into and out of the market affect the stock's price direction. To determine the direction of a stock's price, momentum, trend, and money flow indicators are all interrelated.

3.6 Preprocessing

In our study, we focus on time-series data pertaining to the Dhaka Stock Exchange. To process this data effectively, we utilized the transaction date (txn date) as the index for our data frame. Upon loading the dataset from the CSV file, we assigned the trading

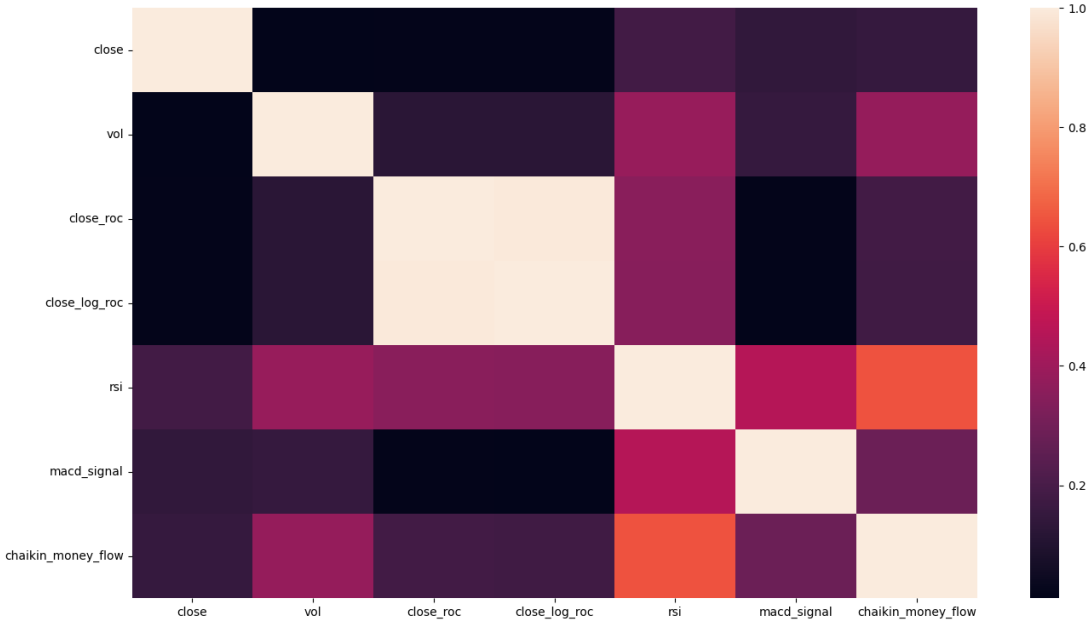

Figure 3.3: Feature Correlation Heatmap -

day (txn date) as the index, which proved essential for organizing and processing the data.

Handling missing data is a crucial aspect of preprocessing time-series datasets, particularly since many machine learning algorithms cannot accommodate missing values. Various strategies exist for addressing missing data, such as eliminating rows with missing values, imputing missing values from existing rows, or employing algorithms that can handle missing values. In our case, we opted to remove the rows containing missing values, as there were only a few instances. Consequently, we eliminated the NaN values from the dataset and resolved seven rows with zero volume values.

Feature scaling is a commonly employed technique to standardize data features within a similar range. This process is particularly important as it prevents algorithms from disproportionately weighting larger values. For example, an individual's age may range from 0 to 150, while their salary could be in the millions. To mitigate such disparities in magnitude, we performed feature scaling before inputting the data into machine learning algorithms. Notably, some algorithms demonstrate improved performance when data is appropriately scaled.

In the preprocessing stage, we used two MinMaxScalers (x scale, y scale) to scale

the input features and output values. We selected the closing price as the output value. Given that the newly computed values included both positive and negative values, we scaled the features and outputs between -1 and 1. To partition the dataset, we divided it into training, validation, and test sets with a 70-15-15 ratio based on the txn date index value. This division resulted in sequential time blocks for our dataset. The training dataset comprised the initial 70% of the data, spanning from 1999-01-27 to 2014-09-16. The test and validation datasets each contained 15% of consecutive rows, ranging between 2014-09-17 and 2021-07-29.

Our objective was to utilize the sequential values of the stock to predict future values. To achieve this, we used five days of data as input for the model to predict the price on the 6th day. We reshaped the input features into five (n) rows, each containing seven features, which served to predict the output (closing price) of the 6th day. Table 3.5 shows the input to the model and expected result for a single instance

Table 3.5: Input and output of a single instance

Input	[[-0.79667687 -0.79535684 -0.78253932 -0.79021286 -0.99920755 -0.27115265 -0.01530007 0.54697151 0.10631538 -0.43805979] [-0.79886314 -0.80137575 -0.78481878 -0.79372394 -0.99979712 -0.32546333 -0.07347706 0.05333146 0.10684838 -0.48947535] [-0.79842589 -0.80266552 -0.78481878 -0.79416283 -0.99988134 -0.30458234 -0.05089887 0.00744055 0.10711887 -0.57361293] [-0.80104941 -0.8022356 -0.78709824 -0.79460171 -0.99984171 -0.30458752 -0.05090444 -0.03835573 0.10714467 -0.47827196] [-0.79886314 -0.80352537 -0.78709824 -0.79547948 -0.99981694 -0.30763272 -0.05418054 -0.1260589 0.10689476 -0.43091519]]
Output	[[-0.79767391]]

3.7 Platform details

The implementation of our study's methodology was carried out using the Python programming language [51], a popular choice for its versatility and extensive library support in the machine learning domain. We conducted our experiments and developed the code within Jupyter Notebook [52], an interactive computing environment that fosters seamless integration of code, visualizations, and narrative text. The complete code for our project is publicly accessible and can be found at the following reference [53].

To train the deep learning models, we leveraged two widely-used and powerful libraries: TensorFlow [54] and Keras [55]. TensorFlow, an open-source library developed by Google, provides a comprehensive ecosystem of tools, libraries, and resources for machine learning and artificial intelligence research. Keras, a high-level neural network API, is built on top of TensorFlow, enabling the rapid and efficient design and training of deep learning models. The combination of these libraries allowed us to streamline the model development process and ensure robust, efficient training of our deep learning models for the analysis of Dhaka Stock Exchange time-series data. The Table 3.6 presents a list of the main libraries utilized in our study, along with their respective versions. These libraries are essential components of the Python ecosystem for machine learning and data analysis tasks:

1. **NumPy (v1.19.5)**: A fundamental library for numerical computing in Python, NumPy provides support for multi-dimensional arrays, mathematical functions, and linear algebra operations [50].

2. **pandas (v1.2.4)**: A powerful data manipulation library, pandas offers data structures like DataFrame and Series, which enable efficient handling of large datasets, time-series data, and data cleaning tasks [49].

3. **matplotlib (v3.3.4)**: A widely-used library for creating static, interactive, and animated visualizations in Python, matplotlib facilitates the plotting of graphs, histograms, scatter plots, and more [56].

4. **scikit-learn (v0.24.2)**: A comprehensive library for machine learning and data mining tasks, scikit-learn provides tools for data preprocessing, model training and evaluation, as well as various machine learning algorithms [57].

5. **TensorFlow (v2.3.0)**: An open-source machine learning library developed by Google, TensorFlow offers a versatile ecosystem for implementing, training, and deploying machine learning models, including deep learning networks [54].

6. **Keras (v2.4.3)**: A high-level neural network API built on top of TensorFlow, Keras simplifies the design and training of deep learning models, promoting rapid experimentation and development [55].

Table 3.6: Used libraries and versions

Serial	Library	Version
1	NumPy	1.19.5
2	pandas	1.2.4
3	matplotlib	3.3.4
4	sklearn	0.24.2
5	TensorFlow	2.3.0
6	Keras	2.4.3

To benchmark the different models seamlessly, a standard test environment was implemented. The computational environment and hardware specifications used for the implementation of the machine learning models were as follows:

- **Operating System:** The experiments were performed on a Windows 10 operating system.
- **Processor:** The computer used for the experiments was equipped with an Intel Core i7-6500U processor.
- **RAM:** The system had 16 GB of RAM installed.
- **Disk Drive:** An SSD was used as the primary storage drive for the system.
- **Python:** The experiments were conducted using Python 3.8 programming language.
- **Anaconda:** The Anaconda distribution (version 4.10.3) was used as the Python environment management system.

The environment configuration is given in Table 3.7.

Table 3.7: Execution Environment

Serial	Description	Configuration
1	Operating System	Windows 10
2	Processor	Corei7, 6500U
3	RAM	16 GB
4	Disk Drive	SSD
5	Python	3.8
6	Anaconda	4.10.3

3.8 Algorithms

Activation functions played a crucial role in training each model in our study. We experimented with various combinations of hyperparameters, including epochs, optimizers, regularization techniques, loss functions, layers, and the number of neurons in each layer. While the list of tested configurations is extensive, we will focus on discussing only the models that demonstrated the best performance in this thesis.

The Mean Squared Error (MSE) was employed as the loss function to optimize the model weights during training. To prevent overfitting and improve generalization, we configured the models for early stopping based on the validation loss. This approach allowed us to save the best-performing models at each stage of the training process.

Subsequently, we loaded the saved models and conducted further analysis to evaluate their performance. It is important to note that, throughout the training process, we observed that the validation loss was consistently lower than the training loss for various configurations. This observation suggested that our models were learning effectively and generalizing well to unseen data.

3.9 ARIMA

We utilized the Auto ARIMA algorithm to fit a model to historical stock close prices and generate forecasts for future values. It showcases how the pmdarima library can be used to automate the process of selecting the best ARIMA model and provide valuable insights for stock price prediction.The Auto ARIMA model is fitted to the close_prices data. The 'auto_arima' function from pmdarima is used with the following parameters:

- seasonal=False: Specifies that the model does not consider seasonal patterns.
- stepwise=True: Enables the algorithm to search for the best-fit ARIMA model using a stepwise approach.
- suppress_warnings=True: Suppresses warnings that may arise during the model fitting process.
- error_action='ignore': Specifies to ignore any errors encountered during the model fitting.
- trace=False: Disables the detailed tracing of the model fitting process.

The best-fitting ARIMA model was found to be ARIMA(5,1,5)(0,0,0)

0

. The ARIMA(5,1,5)(0,0,0)[0] model represents an autoregressive integrated moving average model with non-seasonal components. The This model incorporates autoregressive terms up to lag 5, differencing of the data once, and moving average terms up to lag 5. It does not include any seasonal components. This particular model configuration is suitable for analyzing and forecasting time series data that exhibit non-seasonal patterns and require differencing to achieve stationarity. It has the following parameters:

- Autoregressive terms: $p = 5$
- Degree of differencing: $d = 1$
- Moving average terms: $q = 5$
- Seasonal autoregressive terms: $P = 0$
- Degree of seasonal differencing: $D = 0$
- Seasonal moving average terms: $Q = 0$
- Seasonal period: $s = 0$

3.9.1 LSTM

To identify an optimal deep neural network (DNN) model for the Dhaka Stock Exchange (DSE) dataset, we manually explored various architectural combinations. Through a trial and error process, we determined that a 5:40:40:40:20:1 architecture for the Long Short-Term Memory (LSTM) model offered promising results. We utilized the Keras.Layers.LSTM class for the model implementation.

To enhance the model's generalization capabilities and prevent overfitting, we employed a 10% dropout rate for regularization between the second and third hidden layers, as well as between the third and fourth hidden layers.

Our early stopping configuration led to the termination of the training process after 20 generations. Figure 3.4 displays a graph that illustrates the training and validation loss for the LSTM model across different epochs during the learning process. The x-axis represents the number of epochs, while the y-axis indicates the loss value. The blue line represents the training loss, and the orange line represents the validation loss.

Initially, both training and validation losses exhibit a rapid decline, indicating that the model is effectively learning from the data. As the number of epochs increases, the decrease in losses becomes more gradual, and the lines begin to stabilize. After six epochs, the training and validation losses fluctuate less, suggesting that the model has reached a relatively stable state in its learning process. This graph provides valuable insights into the performance of the LSTM model and aids in identifying the optimal stopping point for the training process.

The Table 3.8 displays the training performance of the LSTM model at each step, detailing the time taken, loss, and validation loss values. It provides insights into the model's progress throughout the training process and how the loss and validation loss change over time. The table comprises three columns: Time, Loss, and Validation Loss, each representing a specific metric related to the training process.

The Time column indicates the duration taken for each training step, which varies from 2 seconds to 17 seconds. The Loss column presents the model's loss values, which measure the difference between the predicted and true outputs. It is essential for the loss values to decrease as the training progresses, indicating that the model is learning and improving its performance. The Validation Loss column represents the model's

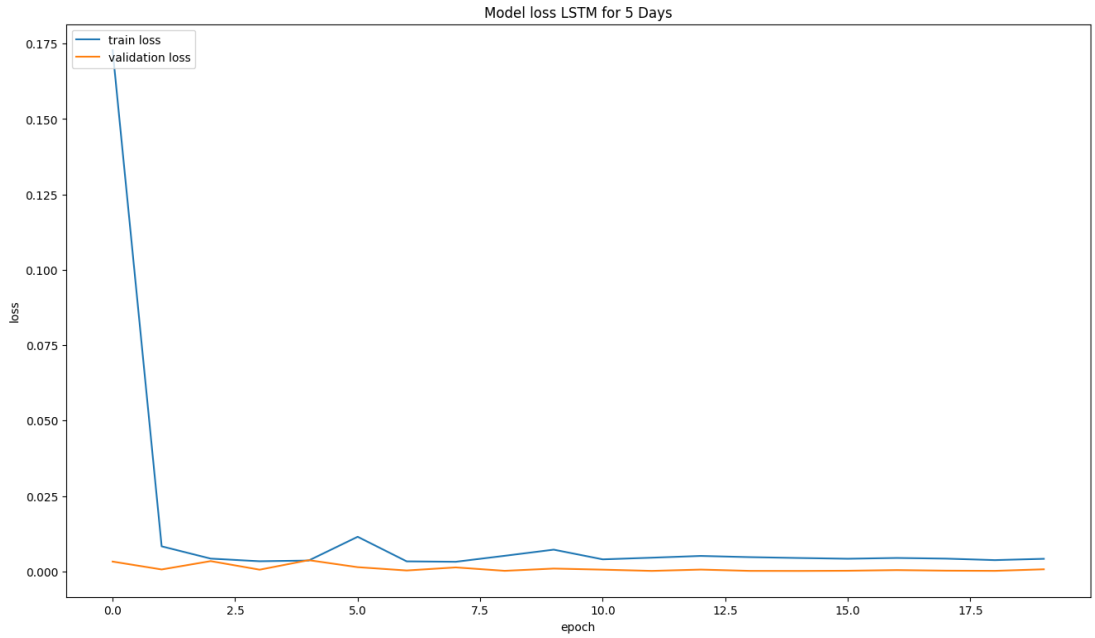

Figure 3.4: Model Loss LSTM for 5 Days -

performance on a separate validation dataset, which is not used during training. Lower validation loss values correspond to better generalization of the model to unseen data.

This table allows for the evaluation of the model's learning progress and helps identify potential overfitting or underfitting issues. It is crucial to observe both the loss and validation loss values, ensuring that they decrease over time, to confirm that the model is converging to an optimal solution.

The Table 3.9 presents the architecture of the best LSTM model's summary used in this study. A total of 39,141 parameters were trained for the model. The model consists of several layers, including four LSTM layers, two dropout layers, and a dense layer. The first LSTM layer (lstm) has an output shape of (None, 5, 40) and contains 7,680 parameters. The second LSTM layer (lstm_1) also has an output shape of (None, 5, 40) and has 12,960 parameters. A dropout layer (dropout) follows, which helps prevent overfitting by dropping out neurons with an output shape of (None, 5, 40) and no parameters. The third LSTM layer (lstm_2) shares the same output shape and parameter count as the previous LSTM layers. Another dropout layer (dropout_1) is added with the same output shape as the previous dropout layer and no parameters. The fourth LSTM layer (lstm_3) has a reduced output shape of (None, 20) and 4,880

Epoch	Time	Loss	Validation Loss
1	17s 72ms/step	0.1728	0.0033
2	5s 54ms/step	0.0083	6.6234e-04
3	4s 44ms/step	0.0043	0.0034
4	4s 42ms/step	0.0034	5.8920e-04
5	5s 50ms/step	0.0036	0.0037
6	2s 23ms/step	0.0115	0.0014
7	2s 23ms/step	0.0033	3.2754e-04
8	2s 23ms/step	0.0032	0.0013
9	3s 26ms/step	0.0052	2.1584e-04
10	3s 29ms/step	0.0072	9.5900e-04
11	2s 22ms/step	0.0040	6.0098e-04
12	2s 23ms/step	0.0046	1.8981e-04
13	2s 23ms/step	0.0051	6.2457e-04
14	3s 33ms/step	0.0047	1.9578e-04
15	5s 49ms/step	0.0045	1.7594e-04
16	4s 43ms/step	0.0042	2.4134e-04
17	4s 40ms/step	0.0045	4.5727e-04
18	3s 26ms/step	0.0043	2.6839e-04
19	2s 23ms/step	0.0038	2.0488e-04
20	2s 22ms/step	0.0042	7.1843e-04

Table 3.8: Training details for each step of LSTM model

parameters. Finally, a dense layer (dense) is used for the model's output with a shape of (None, 1) and 21 parameters. This table provides a comprehensive overview of the LSTM model's structure and the number of parameters involved in each layer.

3.9.2 RNN

An optimal architecture for the RNN model when applied to stock market data was determined to be a 5:50:15:1 configuration. This structure was implemented using the SimpleRNN class from the Keras library's Layers module. To enhance the model's generalization capabilities, a 10% dropout rate was introduced between the first and second hidden layers as well as between the last hidden layer and the output layer. This regularization strategy aimed to prevent overfitting and improve the model's performance

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 5, 40)	7680
lstm_1 (LSTM)	(None, 5, 40)	12960
dropout (Dropout)	(None, 5, 40)	0
lstm_2 (LSTM)	(None, 5, 40)	12960
dropout_1 (Dropout)	(None, 5, 40)	0
lstm_3 (LSTM)	(None, 20)	4880
dense (Dense)	(None, 1)	21

Table 3.9: Architecture of the LSTM Model

on unseen data.

The training process was configured with an early stopping mechanism based on the validation loss, ensuring that the model would cease training once its performance on the validation set ceased to improve. Consequently, the training process terminated automatically after 17 epochs.

The Table 3.10 presents the training details for the Recurrent Neural Network (RNN) model, specifically the epoch number, training time per step (in seconds and milliseconds), training loss, and validation loss. Over the course of 19 epochs, the RNN model's loss and validation loss values decrease, indicating the model's improving performance in learning the stock market data.

In the initial epochs, the training loss and validation loss decrease rapidly, suggesting that the model is learning essential patterns in the data. As the training progresses, the loss values continue to decrease, but at a slower rate, implying that the model is fine-tuning its understanding of the underlying structure of the stock market data.

It is essential to monitor the validation loss to avoid overfitting the model. Overfitting occurs when the model becomes too specialized in learning the training data, resulting in poor generalization to new, unseen data. In this case, the validation loss remains low throughout the training process, indicating that the model is not overfitting and can likely generalize well to new data.

The Table 3.11 presents the architecture of a Recurrent Neural Network (RNN) model, which consists of five layers and 3,906 parameters. The first layer, a SimpleRNN layer, has an output shape of (None, 5, 50) and 2,900 trainable parameters. This layer

Epoch	Time (ms/step)	Loss	Validation Loss
1	3s 13ms/step	0.2406	0.0079
2	1s 9ms/step	0.0805	0.0059
3	1s 12ms/step	0.0488	0.0017
4	1s 15ms/step	0.0318	0.0028
5	2s 16ms/step	0.0236	0.0015
6	1s 10ms/step	0.0192	8.8182e-04
7	1s 8ms/step	0.0163	0.0018
8	1s 8ms/step	0.0144	0.0011
9	1s 8ms/step	0.0129	8.0992e-04
10	1s 8ms/step	0.0121	0.0019
11	1s 8ms/step	0.0122	0.0011
12	1s 8ms/step	0.0138	0.0013
13	1s 7ms/step	0.0130	0.0037
14	1s 8ms/step	0.0111	5.1499e-04
15	1s 8ms/step	0.0103	8.5248e-04
16	1s 8ms/step	0.0093	0.0029
17	1s 8ms/step	0.0092	6.0445e-04
18	1s 8ms/step	0.0098	0.0011
19	1s 11ms/step	0.0097	0.0032

Table 3.10: Training details for the RNN model

serves as the initial input layer for the RNN model and processes the input data using recurrent connections.

Following the first SimpleRNN layer, a dropout layer is added to the model to prevent overfitting. This Dropout layer has an output shape of (None, 5, 50) and 0 parameters, as it does not have any trainable weights. The dropout layer randomly sets a fraction of input units to 0 during training, which helps in regularization.

The third layer in the model is another SimpleRNN layer with an output shape of (None, 5, 15) and 990 trainable parameters. This layer adds more complexity to the model, allowing it to learn more intricate patterns in the input data.

A second Dropout layer with an output shape of (None, 5, 15) and 0 parameters is added after the third layer, providing additional regularization for the model.

Finally, a Dense layer with an output shape of (None, 5, 1) and 16 trainable pa-

Table 3.11: RNN Layer Information

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 5, 50)	2900
dropout_2 (Dropout)	(None, 5, 50)	0
simple_rnn_1 (SimpleRNN)	(None, 5, 15)	990
dropout_3 (Dropout)	(None, 5, 15)	0
dense_1 (Dense)	(None, 5, 1)	16

rameters is used as the output layer of the model. This layer takes the features learned by the previous layers and produces the final output of the RNN model.

In summary, the RNN model has a total of 3,906 trainable parameters and consists of two SimpleRNN layers, two Dropout layers for regularization, and a Dense layer for generating the output.

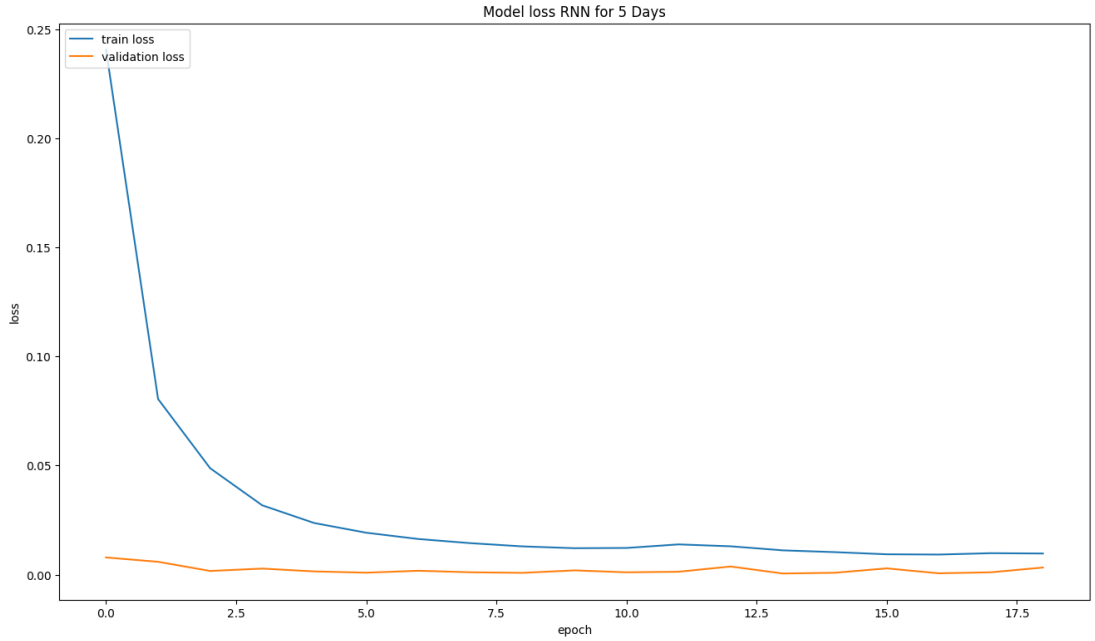

Figure 3.5: Model Loss RNN for 5 Days -

The Figure 3.5 illustrates the training and validation loss of a Recurrent Neural Network (RNN) model during the training process for stock market analysis. The x-axis represents the number of epochs, while the y-axis shows the loss values. The graph displays two lines: one for the training loss and the other for the validation loss.

Throughout the training process, the training loss consistently decreases, which indicates that the model is effectively learning from the training data. On the other hand, the validation loss also generally decreases, but there are fluctuations observed at certain points during training. This fluctuation in validation loss is a common occurrence and could be attributed to the stochastic nature of the training process or the presence of noise in the data.

The goal of the training process is to minimize both the training and validation loss, aiming to achieve a balance between fitting the training data well and generalizing to new, unseen data. The decreasing trend of both the training and validation loss in the graph suggests that the RNN model is converging towards an optimal solution for predicting stock market trends based on the given input data.

3.9.3 GRU

After exploring various architectural configurations for the Gated Recurrent Unit (GRU) model, the 5:50:1 architecture was identified as the most effective for this particular application. The GRU model was implemented using the GRU class from the Keras library, which provides a powerful and easy-to-use interface for designing and training recurrent neural networks.

A 10% dropout rate was introduced between the hidden layer and the output layer to incorporate regularization, preventing overfitting and improving the model's generalization capabilities.

To optimize the training process and avoid overfitting, early stopping was employed based on the validation loss. As a result, the training process automatically terminated after 24 epochs when the validation loss did not show significant improvement. The corresponding training and validation loss graph is illustrated in Figure 18, which provides a visual representation of the model's performance during training and its convergence towards an optimal solution. This 5:50:1 GRU architecture demonstrates the model's ability to effectively predict stock market trends based on the given input data and serves as a valuable tool for future stock market analysis.

The training details for each epoch, including loss and validation loss, are presented in Table 3.12. It presents information on the training time, loss, and validation loss for each of the 24 epochs. It shows that the GRU model underwent 24 epochs of training, effectively minimizing both the training loss and validation loss.

In the first epoch, the training took 6 seconds and 14 milliseconds per step, with a loss value of 0.1627 and a validation loss of 0.0494. As the training progressed through the epochs, the time per step and the loss values varied. For example, in the 11th epoch, the training took 1 second and 9 milliseconds per step, with a loss of 0.0043 and a validation loss of 1.9463e-04.

The Loss column represents the training loss for each epoch, which is the difference between the predicted output and the actual output for the training dataset. The objective of the training process is to minimize this loss value. As observed in the table, the loss value consistently decreased with each epoch, such as in the 24th epoch, where the training loss reached 0.0034.

The Validation Loss column displays the loss value calculated using the validation dataset, which is separate from the training dataset. This dataset is used to evaluate the model's performance on unseen data. Ideally, the validation loss should also decrease with each epoch, indicating that the model generalizes well to new data. In this case, the validation loss decreased overall, with some fluctuations, such as in the 14th epoch, where the validation loss was 2.7194e-04, suggesting that the model is not overfitting to the training data.

In summary, t For instance, in the final epoch, the model achieved a training loss of 0.0034 and a validation loss of 2.8291e-04. This indicates that the model has effectively learned from the training data and is expected to perform well on new, unseen data.

The Table 3.13 summarizes the GRU model architecture used in this study, consisting of three layers with different types, output shapes, and parameter counts. The first layer is a Gated Recurrent Unit (GRU) with an output shape of (None, 5, 50) and 8,850 trainable parameters, responsible for learning temporal dependencies in the input data. The second layer is a Dropout layer for regularization, maintaining the output shape of (None, 5, 50) but randomly setting a fraction of input units to 0 during training to prevent overfitting. This layer has no trainable parameters. The third and final layer is a Dense (fully connected) layer with an output shape of (None, 5, 1) and 51 trainable parameters, responsible for producing final predictions by mapping the previous layers' output to target values. In total, the GRU model has 8,901 trainable parameters that are adjusted during training to minimize the loss function and improve predictive performance.

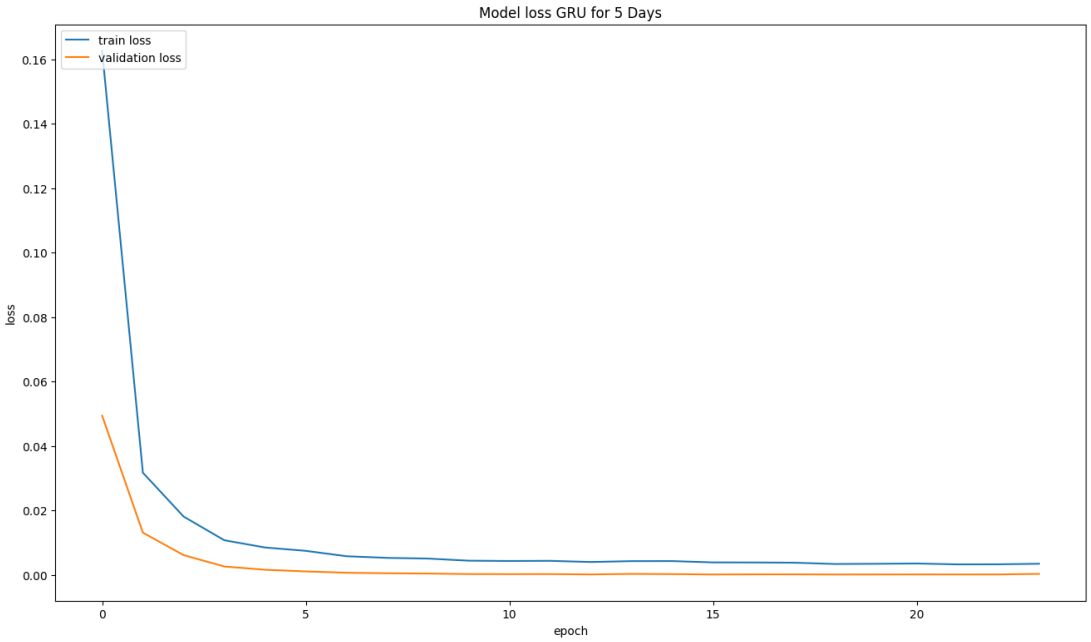

Figure 3.6: Model Loss GRU for 5 Days -

The Figure 3.5 provides a visual representation of the GRU model's training progress, displaying the reduction in training and validation losses as the number of epochs increases. The plot highlights the model's ability to learn from data and improve its predictive performance while avoiding overfitting. The x-axis represents the number of epochs or training iterations, while the y-axis shows the loss values. Two curves are plotted on the graph: the training loss (blue) and validation loss (orange).

As the number of epochs increases, both the training and validation loss curves show a decreasing trend, indicating that the model is learning from the data and improving its performance over time. The training loss curve shows a steeper decline initially and continues to decrease, while the validation loss curve exhibits a similar pattern but with a less steep slope. The minor fluctuations in the validation loss curve suggest that the model is adjusting to new data and generalizing well without overfitting.

3.9.4 CNN

Conv1D class from Keras. layers were used. 20% dropout was added between the Max pooling and flatten layers for regularization. Table 167 shows CNN training details

for each epoch. The training automatically stopped for early stopping after 16 epochs. Corresponding Training vs. Validation loss is shown in Figure 19.

The best CNN model's summary is shown in . A total of 1,473 parameters were trained for the model. The Table 3.14 presents the training progress of the model in terms of the loss and validation loss for each epoch. In the first epoch, the model starts with a relatively high loss of 0.3557 and a validation loss of 0.0325. As the training progresses, both the loss and validation loss decrease significantly. By the second epoch, the loss drops to 0.2027, and the validation loss reduces to 0.0039. Throughout the following epochs, the model continues to improve as its loss and validation loss values decrease further. For instance, in the fifth epoch, the loss is 0.0986, while the validation loss is 3.8808e-04. By the 16th epoch, the loss value reaches 0.0258, and the validation loss value reaches 7.8930e-04. As the loss and validation loss values decline, the model learns effectively from the data and improves its performance over time.

The Table 3.15 presents the architecture of a 1D convolutional neural network (CNN) model designed for time-series data analysis. It has a total of 1,473 trainable parameters and aims to capture temporal dependencies in time-series data through convolution and pooling operations. The model consists of five layers, including an input layer, a pooling layer, a dropout layer, a flatten layer, and an output layer.

1. **Input Layer (Conv1D)**: The first layer is a 1D convolutional layer (Conv1D) with 64 filters, each with a kernel size of 3. This layer is responsible for learning local patterns in the input data by performing a convolution operation. It results in an output shape of (None, 3, 64) and has 1,408 trainable parameters.

2. **Pooling Layer (MaxPooling1D)**: The second layer is a max pooling layer that downsamples the input data by taking the maximum value from non-overlapping regions of the input feature maps. In this case, it reduces the spatial dimension of the data to (None, 1, 64) without any trainable parameters.

3. **Dropout Layer (Dropout)**: The third layer is a dropout layer, which applies a regularization technique to prevent overfitting. It randomly sets a fraction of input units to 0 at each training step, helping the model generalize better. The dropout layer maintains the same output shape as its input, (None, 1, 64), and does not have any trainable parameters.

4. **Flatten Layer (Flatten)**: The fourth layer is a flatten layer that reshapes the input data into a one-dimensional array. It converts the input shape of (None, 1, 64) to (None, 64), facilitating the connection between the convolutional layers and the fully connected dense layer. This layer has no trainable parameters.

5. **Output Layer (Dense)**: The final layer is a dense (fully connected) layer with a single output neuron. It maps the 64 input features to a single output value, resulting in an output shape of (None, 1). The dense layer has 65 trainable parameters.

The Figure 3.7 illustrates the training and validation loss curves of a 1D Convolutional Neural Network (CNN) model designed for time-series stock market data analysis. Both training and validation data demonstrate the model's ability to capture temporal patterns and dependencies in time-series stock market data while generalizing to new data points well. This plot illustrates the model's performance over a series of training intervals. In the graph, the x-axis represents the number of epochs, whereas the y-axis represents the loss values. Observing the graph, it is evident that the training loss (depicted in blue) decreases consistently with the number of training epochs, suggesting the model is effectively learning from the training data. There is also a general downward trend in the validation loss (shown in orange), suggesting that the model generalizes well to unseen data. As a common characteristic of neural network training, the validation loss curve exhibits some fluctuations. The overall trend, however, is still decreasing, which suggests that the model is not overfit.

The Figure 4.6 illustrates a bar chart comparing the total number of parameters for four different deep learning models: LSTM, RNN, GRU, and CNN. These models were used in the context of stock market analysis. The chart clearly demonstrates that the LSTM model has the highest number of parameters, amounting to 38,501. On the other hand, the RNN model has the second-lowest number of parameters, with a total of 3,906. The GRU model has 8,901 parameters, and the CNN model has the lowest number of parameters among all four models, with only 1,473.

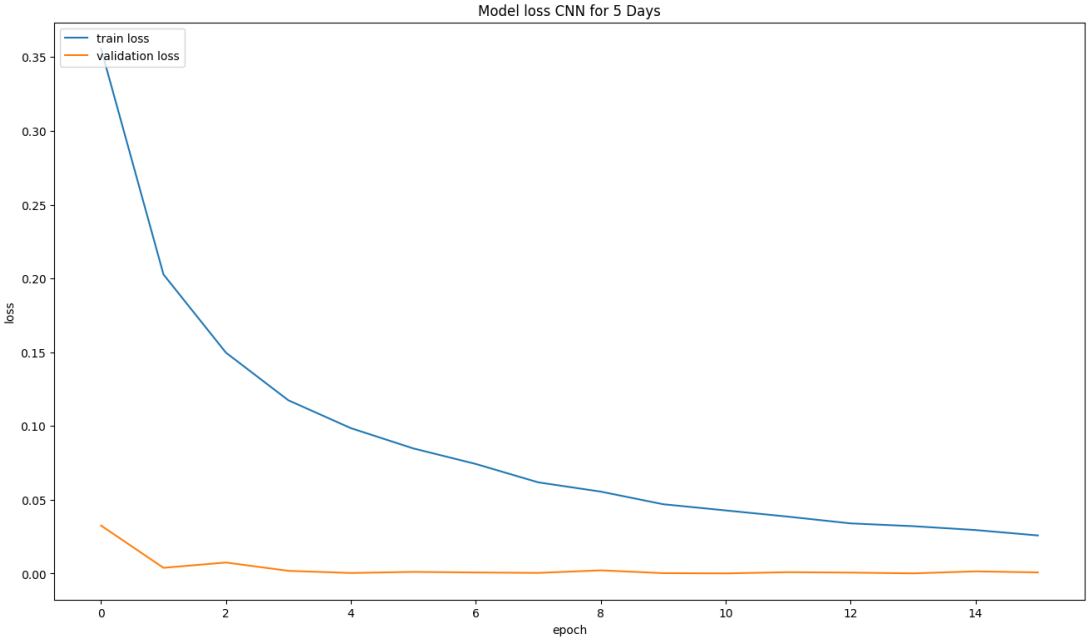

Figure 3.7: Model Loss CNN for 5 Days -

Table 3.12: GRU Training Details

Epoch	Training Time	Loss	Validation Loss
1	6s 14ms/step	0.1627	0.0494
2	1s 10ms/step	0.0317	0.0131
3	1s 11ms/step	0.0181	0.0061
4	2s 16ms/step	0.0107	0.0026
5	2s 17ms/step	0.0085	0.0016
6	2s 15ms/step	0.0074	0.0010
7	1s 10ms/step	0.0058	6.2851e-04
8	1s 9ms/step	0.0052	4.6431e-04
9	1s 9ms/step	0.0050	3.8359e-04
10	1s 9ms/step	0.0044	2.1227e-04
11	1s 9ms/step	0.0043	1.9463e-04
12	1s 9ms/step	0.0043	2.0919e-04
13	1s 9ms/step	0.0040	1.0018e-04
14	1s 9ms/step	0.0042	2.7194e-04
15	1s 9ms/step	0.0043	1.9838e-04
16	1s 9ms/step	0.0038	8.3926e-05
17	1s 11ms/step	0.0038	1.1999e-04
18	2s 15ms/step	0.0037	1.2137e-04
19	1s 12ms/step	0.0033	8.3582e-05
20	1s 9ms/step	0.0034	9.0623e-05
21	1s 10ms/step	0.0035	9.6265e-05
22	1s 9ms/step	0.0032	9.3931e-05
23	1s 9ms/step	0.0032	9.5712e-05
24	1s 9ms/step	0.0034	2.8291e-04

Table 3.13: GRU Model Summary

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 5, 50)	8850
dropout	(None, 5, 50)	0
dense	(None, 5, 1)	51

Epoch	Training Time	Loss	Validation Loss
1	1s 5ms/step	0.3557	0.0325
2	0s 4ms/step	0.2027	0.0039
3	0s 4ms/step	0.1497	0.0075
4	0s 3ms/step	0.1174	0.0018
5	0s 4ms/step	0.0986	3.8808e-04
6	0s 4ms/step	0.0848	0.0011
7	0s 3ms/step	0.0743	7.3271e-04
8	0s 3ms/step	0.0618	4.4164e-04
9	0s 4ms/step	0.0555	0.0022
10	0s 4ms/step	0.0470	2.8352e-04
11	0s 4ms/step	0.0428	1.1644e-04
12	0s 4ms/step	0.0386	9.2082e-04
13	0s 4ms/step	0.0340	6.4610e-04
14	0s 4ms/step	0.0321	1.3315e-04
15	0s 3ms/step	0.0295	0.0015
16	0s 4ms/step	0.0258	7.8930e-04

Table 3.14: CNN training details

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 3, 64)	1408
max_pooling1d (MaxPooling1D)	(None, 1, 64)	0
dropout_1 (Dropout)	(None, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Table 3.15: CNN model summary

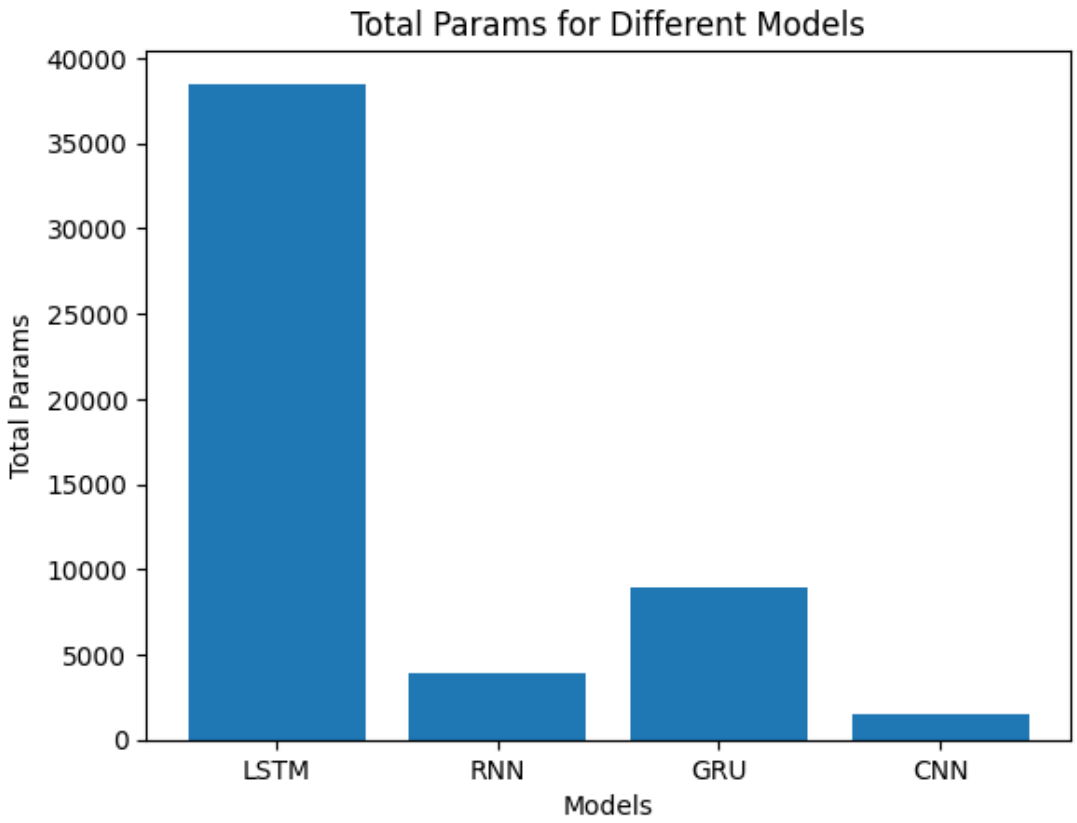

Figure 3.8: Parameters Learned by different Models -

Chapter 4

Results

In this chapter, we present the results obtained from our analysis of stock market prediction using various machine learning models, including Auto ARIMA, Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Convolutional Neural Networks (CNN). The primary objective of this research is to compare the performance of these models in predicting stock close prices and identify the most suitable model for this task.

To evaluate the performance of each model, we used four evaluation metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R2) score. These metrics provide insights into the accuracy of the models and their ability to capture the underlying trends in the stock market data. By comparing these evaluation metrics across different models, we can determine the strengths and weaknesses of each approach and make informed decisions about their applicability in predicting stock market trends.

In the following sections, we will discuss the results obtained from each model in detail, focusing on their performance in terms of the evaluation metrics. We will also provide visual representations of the predicted stock close prices for each model, allowing for a more intuitive comparison of their prediction capabilities. Finally, we will synthesize the results to draw conclusions about the effectiveness of the different models in predicting stock market trends and offer recommendations for future research in this area.

Figure 4.1 illustrates the accuracy of the Auto ARIMA model in forecasting stock prices over a period of 10 days. X represents the time period, which could be days,

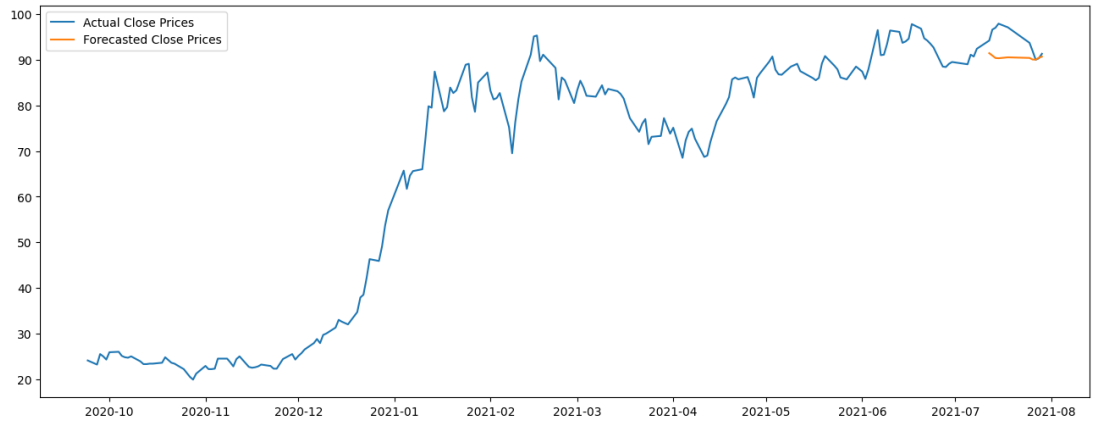

Figure 4.1: ARIMA actual and predicted values -

weeks, or any other unit based on the data. Y represents the stock price values. The graph consists of two lines: the blue line represents the actual stock prices, while the orange line represents the Auto ARIMA model's predictions. Over a specified time period, the graph shows how well the model has captured stock price movements.

The graph 4.2 presents a comparison between the actual stock close prices and the predicted values generated by the Long Short-Term Memory (LSTM) model. The x-axis represents the time steps, while the y-axis displays the stock close prices. The blue line in the graph represents the actual stock close prices, and the orange line represents the predicted values from the LSTM model.

Upon visual inspection of the graph, it is evident that the LSTM model has effectively captured the overall trends in the stock close prices, following the general patterns of the actual data. The predicted values, in most cases, are in close proximity to the actual values, indicating the LSTM model's ability to make accurate predictions. However, there are a few instances where the predicted values deviate from the actual stock close prices, resulting in some discrepancies. This could be attributed to the inherent volatility and unpredictability of stock market data, as well as limitations in the model's architecture or training process.

Despite these discrepancies, the overall performance of the LSTM model in predicting stock close prices is promising, as evidenced by its ability to capture the general trends in the data. This finding is further supported by the evaluation metrics computed for the LSTM model, which demonstrate its relatively low error rates and

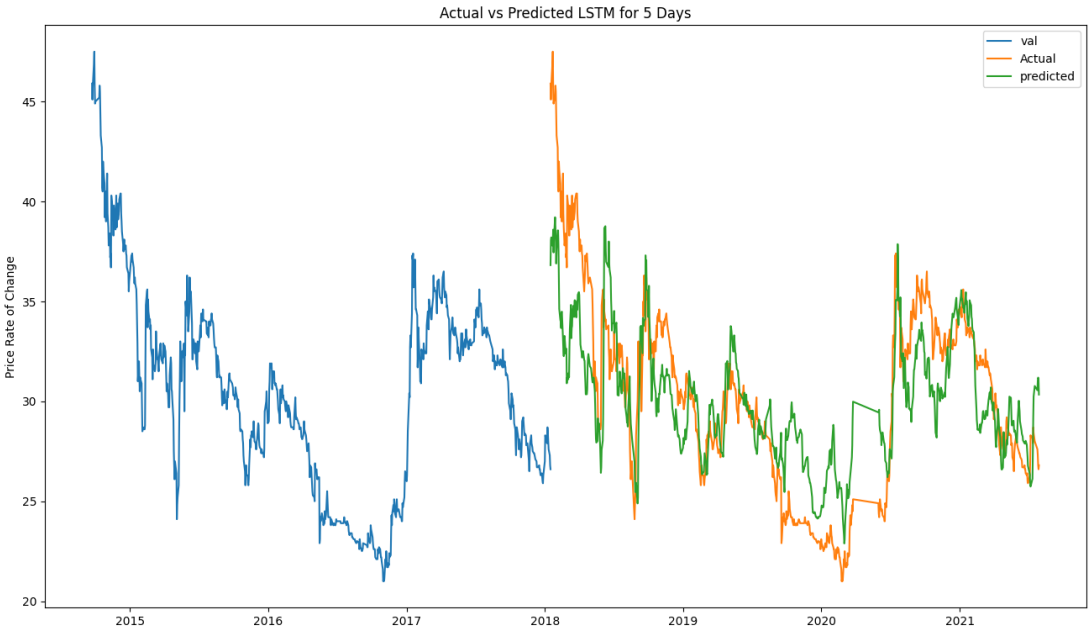

Figure 4.2: LSTM model's ability to predict stock close prices over time despite observed discrepancies between actual and predicted values (blue line and orange line). -

high R-squared (R2) score. These metrics serve as quantitative evidence of the LSTM model's effectiveness in predicting stock market trends and validate the visual observations made from the graph.

The graph 4.3 showcases a comparison between actual stock close prices (blue line) and predicted values (orange line) generated by the RNN model over time. The model appears to struggle with capturing the exact peaks and troughs of the actual stock prices. For instance, around the year 2017, the actual stock prices exhibit a sharp upward movement, while the RNN model's predictions fail to accurately capture this increase. Similarly, during 2018, the actual stock prices show a significant decline, but the RNN model's predictions underestimate the magnitude of this decrease.

On the other hand, the model seems to be successful in capturing the general trends of the stock prices, as seen between the years 2016 and 2017, and 2019 to 2020, where both the actual and predicted values follow similar patterns. However, the noticeable deviations between the actual and predicted values throughout the graph indicate that the RNN model's performance might not be optimal for accurately forecasting stock

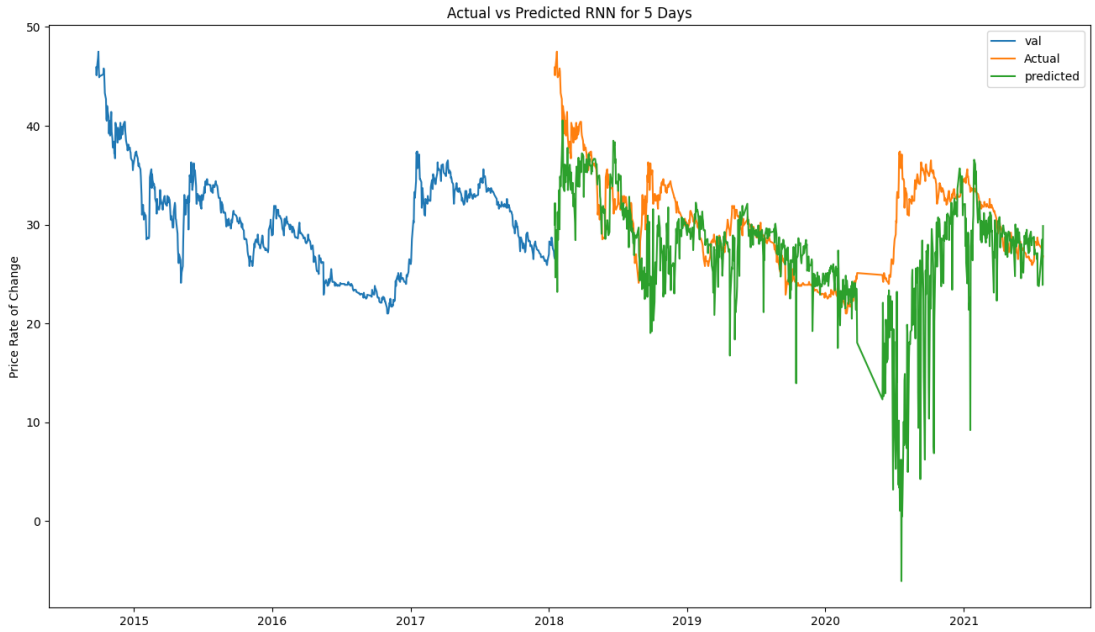

Figure 4.3: Comparison of actual stock close prices and RNN model predictions, highlighting the model's ability to capture general trends but struggling with precise peaks and troughs. -

close prices in this particular case.

The graph 4.4 showcases a comparison between actual stock close prices and predictions made using a GRU model, where the x-axis represents the years and the y-axis represents the stock prices. As an example, we can observe that around the year 2019, both the actual and predicted prices exhibit a downward trend followed by a sharp increase. Similarly, during the first half of 2020, both lines illustrate a steep decline in stock prices followed by a significant recovery.

However, the model struggles to accurately predict certain peaks and troughs. For instance, in late 2020, the actual prices show a sharp spike that the GRU model fails to capture. Despite these limitations, the GRU model appears to provide a satisfactory representation of the stock market movements, highlighting its potential usefulness in forecasting future price trends.

The graph 4.5 showcases a comparison between the actual and predicted stock close prices using a Convolutional Neural Network (CNN) model. The x-axis represents the time in years, and the y-axis displays the stock prices. The blue line in the graph

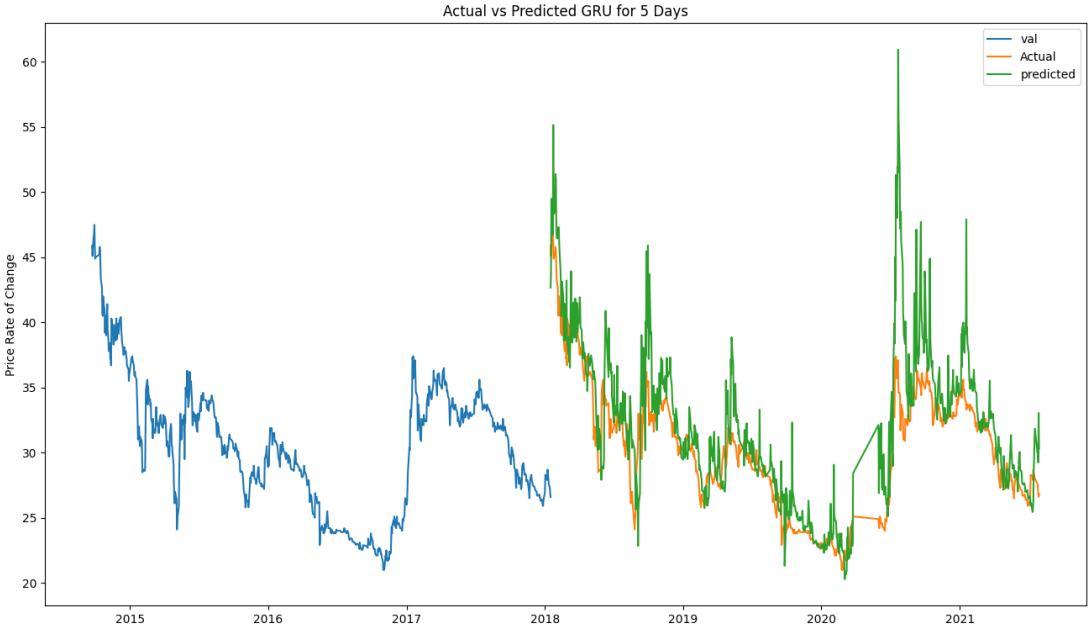

Figure 4.4: GRU Model: Actual vs. Predicted Stock Close Prices Over Time -

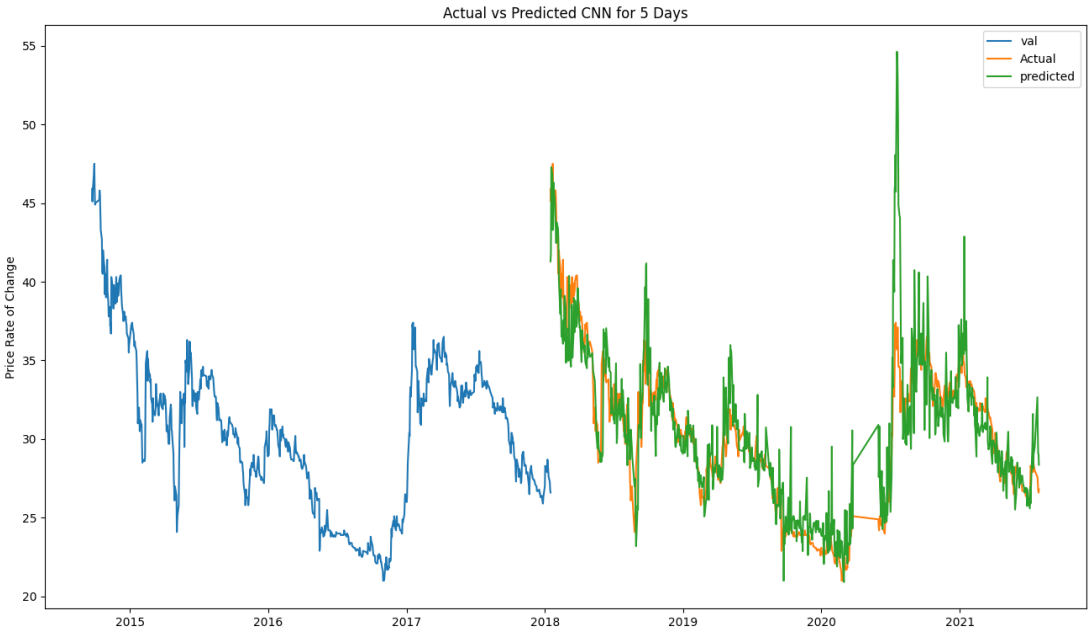

Figure 4.5: Actual vs. Predicted Stock Prices for the Next 5 Days using CNN Model -

Model	Total Params	MSE	MAE	RMSE	R2
ARIMA	0	19.9367	3.5500	4.4650	-1.428
LSTM	38,501	8.8056	2.3748	2.9674	0.6312
RNN	3,906	55.3703	4.5187	7.4411	-1.3188
GRU	8,901	12.4595	2.3088	3.5298	0.4782
CNN	1,473	6.0696	1.5758	2.4637	0.7458

Table 4.1: Evaluation metric results for LSTM, RNN, GRU, and CNN models

indicates the actual close prices, while the orange line represents the predicted close prices generated by the CNN model.

In the earlier part of the graph, the predicted prices closely follow the actual prices, suggesting that the model has a good degree of accuracy. However, as the graph progresses, some fluctuations and discrepancies between the actual and predicted prices become more apparent. In some instances, the predicted prices are slightly higher or lower than the actual prices, indicating that the model may not always be able to capture every nuance in the stock price movement.

Despite these minor discrepancies, the overall trend of the predicted prices aligns well with the actual prices. This suggests that the CNN model is relatively effective in forecasting the stock close prices, although it may not capture every detail perfectly. Further analysis and evaluation of the model's performance using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R2 score would provide a more comprehensive understanding of the model's accuracy and reliability in predicting stock close prices.

A Table 4.1 presents the evaluation metric results for different models, including ARIMA, LSTM, RNN, GRU, and CNN. These models were used in a stock market analysis, and various metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R2) were calculated to assess their performance. Let's analyze the results and determine the best model based on these metrics.

Starting with the ARIMA model, it has 0 total parameters, indicating a simpler model compared to the others. However, the evaluation metrics reveal that the ARIMA model has relatively higher values for MSE, MAE, RMSE, and a negative R2 score. These values suggest that the ARIMA model may not be capturing the underlying

patterns and trends in the stock market data effectively, resulting in poor predictive accuracy.

Moving on to the LSTM model, it has the highest number of total parameters (38,501), indicating a more complex and potentially more flexible model. The LSTM model performs relatively better than the ARIMA model, with lower values for MSE, MAE, and RMSE, and a positive R2 score (0.6312). These results indicate that the LSTM model demonstrates better predictive accuracy and captures the stock market trends more effectively than the ARIMA model.

Next, the RNN and GRU models have fewer total parameters compared to LSTM, but their evaluation metrics are generally higher. Both models exhibit higher values for MSE, MAE, RMSE, and have negative R2 scores. This suggests that these models may not perform as well in accurately predicting stock market trends compared to the LSTM model.

Lastly, the CNN model has a moderate number of total parameters (1,473) and demonstrates relatively low values for MSE, MAE, RMSE, and a high R2 score of 0.7458. These results indicate that the CNN model performs well in predicting stock market trends, with lower errors and a higher proportion of the variance explained by the model.

Figure 4.6 Evaluation metric results show the graphical representation of Table **??**. With over 38,000 parameters required for training, the LSTM model required the most parameters. Around 9,000 parameters were required for the GRU model, while 1,473 parameters were required for the CNN model. RNN models required 3,906 parameters. This illustrates that the number of parameters required by different machine learning models can vary widely, and that models with fewer parameters can still provide competitive results.

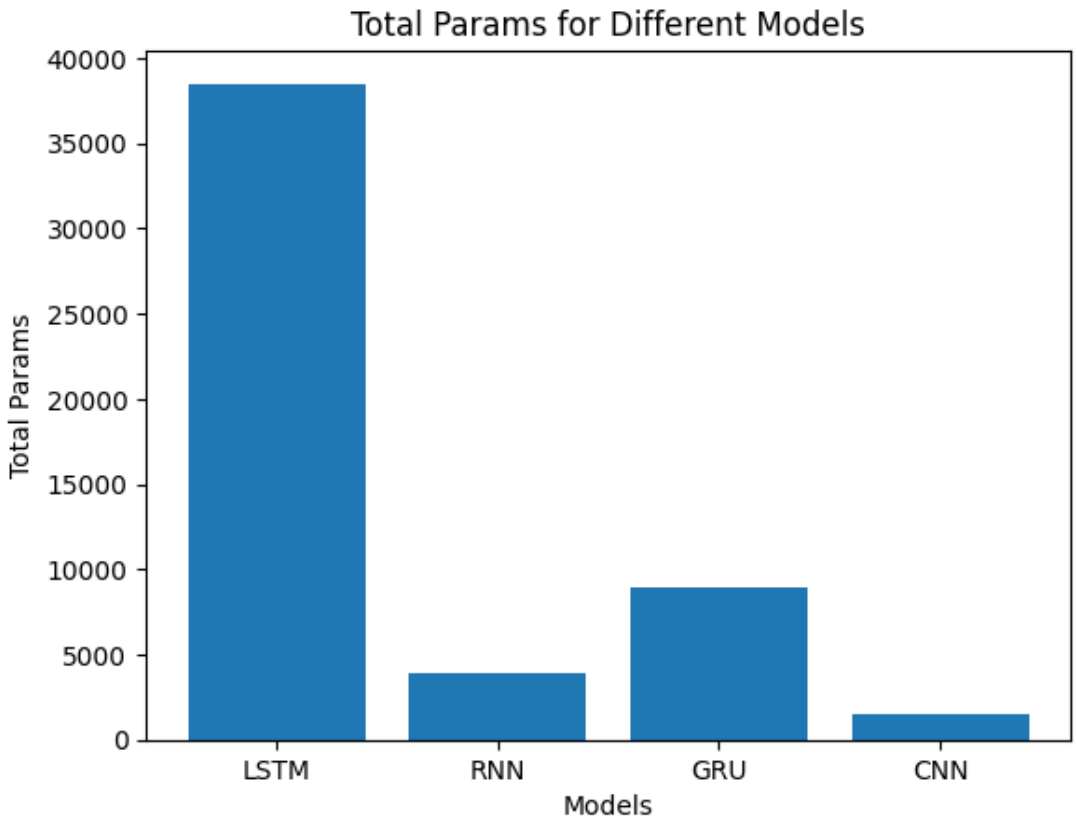

Figure 4.6: Total parameters for different models -

Chapter 5

Conclusion

In conclusion, this thesis investigated the performance of different models, including ARIMA, LSTM, RNN, GRU, and CNN, for stock market analysis. The evaluation of these models was based on various metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R2).

The findings from the table 4.1 analysis indicate that the ARIMA model, with its simplicity, exhibited poorer performance compared to the other models. It had higher values for MSE, MAE, RMSE, and a negative R2 score, suggesting limited predictive accuracy in capturing the stock market trends.

On the other hand, the LSTM model, with its higher complexity and a larger number of parameters, demonstrated superior performance. It achieved lower values for MSE, MAE, and RMSE, as well as a positive R2 score, indicating better predictive accuracy and an ability to capture the underlying patterns in the stock market data.

The RNN and GRU models, although having fewer parameters compared to LSTM, exhibited relatively higher values for MSE, MAE, RMSE, and negative R2 scores. These results suggest that these models may not effectively capture the stock market trends and perform less accurately in prediction.

Additionally, the CNN model, despite having a lower number of parameters, demonstrated strong performance. It achieved low values for MSE, MAE, RMSE, and a high R2 score, indicating accurate predictions and the ability to identify interrelationships within the data.

Based on these findings, it can be concluded that for stock market analysis, both the LSTM and CNN models show promise. The LSTM model, with its higher complexity,

proves effective in capturing the trends and patterns in the data. Meanwhile, the CNN model, despite its simplicity, exhibits strong performance, highlighting its ability to accurately predict stock market trends.

Ultimately, the choice between the LSTM and CNN models will depend on factors such as model complexity, computational resources, interpretability, and specific requirements of the stock market analysis task. Further research and experimentation can be conducted to validate these findings and explore additional factors that may influence the selection of the most suitable model for stock market analysis.

The scope of market trend prediction is vast, and in the future, the trend analysis for a particular type of industry can be analyzed. The production and the revenue of the company can be machine-learned. Though much emphasis is given to experimenting with different hidden learning architecture, less time is dedicated to hyperparameter optimization of the models. For this limitation of the work, it is assumed that the reported performance of the models might not accurately represent their full potential. Neural Architectural Search (NAS) can be utilized to find out better performing models. This study considered one stock data. Using a more diverse training set comprising closely correlated companies could benefit future work predictions.

Bibliography

[1] Overfitting vs underfitting. [Online]. Available: https://vitalflux.com/overfitting-underfitting-concepts-interview-questions/ x, 10

[2] A. Hayes. Financial markets. [Online]. Available: https://www.investopedia.com/terms/f/financial-market.asp 1

[3] ——. What are stocks. [Online]. Available: https://www.investopedia.com/terms/s/stock.asp 1

[4] ——. What is the stock market? [Online]. Available: https://www.investopedia.com/terms/s/stockmarket.asp 1

[5] A. Turing, "Computing machinery and intelligence in "mind", vol," 1950. 2

[6] Machine learning. [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning 2

[7] 11

[8] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine learning practice and the bias-variance trade-off," *arXiv preprint arXiv:1812.11118*, 2018. 11

[9] Double descent curve. [Online]. Available: https://www.researchgate.net/figure/Double-descent-curve_fig3_359244250 11

[10] Overfitting vs underfitting. [Online]. Available: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/ 11, 12

[11] Arima forecast straight line. [Online]. Available: https://stats.stackexchange.com/questions/286900/arima-forecast-straight-line 13

[12] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961. 13

[13] https://www.javatpoint.com/artificial-neural-network. [Online]. Available: https://www.javatpoint.com/artificial-neural-network 14

[14] An artificial neural network. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network 15

[15] Tensorflow - single layer perceptron. [Online]. Available: https://www.tutorialspoint.com/tensorflow/tensorflow_single_layer_perceptron.htm 17

[16] Architecture of mlp. [Online]. Available: https://towardsdatascience.com/mlp-mixer-is-all-you-need-20dbc7587fe4 18

[17] Recurrent neural networks. [Online]. Available: https://www.ibm.com/cloud/learn/recurrent-neural-networks 19, 21, 22

[18] [Online]. Available: https://stackoverflow.com/questions/57122727/understanding-pytorch-vanilla-rnn-architectures 19

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 23

[20] [Online]. Available: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/ 23

[21] [Online]. Available: https://datascience-enthusiast.com/DL/Building_a_Recurrent_Neural_Network-Step_by_Step_v1.html 24

[22] [Online]. Available: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be 24

[23] [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 25

[24] Support vector machine algorithm. [Online]. Available: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm 26

[25] Z. Pawlak, "Information systems theoretical foundations," *Information systems*, vol. 6, no. 3, pp. 205–218, 1981. 25

[26] When to use and how to read the macd indicator. [Online]. Available: https://commodity.com/technical-analysis/macd/ 27

[27] Simple moving average. [Online]. Available: https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/sma 27

[28] J. W. Wilder, *New concepts in technical trading systems.* Trend Research, 1978. 28

[29] Relative strength index (rsi). [Online]. Available: https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/RSI 29

[30] Chaikin money flow. [Online]. Available: https://school.stockcharts.com/doku.php?id=technical_indicators:chaikin_money_flow_cmf 28

[31] Chaikin money flow. [Online]. Available: https://school.stockcharts.com/doku.php?id=technical_indicators:chaikin_money_flow_cmf 30

[32] D. Farrar and R. Glauber, "Multicollinearity in regression analysis: The problem revisited," *The Review of Economics and Statistics*, vol. 49, no. 1, pp. 92–107, 1967. 32

[33] D. Belsley, E. Kuh, and R. Welsch, *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity.* John Wiley & Sons, 1980. 32

[34] J. Hair, W. Black, B. Babin, and R. Anderson, *Multivariate Data Analysis*, 7th ed. Pearson Education Limited, 2014. 32

[35] A. Hoerl and R. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970. 32

[36] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996. 32

[37] S. Makridakis, M. Hibon, and C. Moser, "Accuracy of forecasting: An empirical investigation," *Journal of the Royal Statistical Society: Series A (General)*, vol. 142, no. 2, pp. 97–145, 1979. 33, 34

[38] N. Draper and H. Smith, *Applied Regression Analysis*. John Wiley & Sons, 1998. 34

[39] A. Sachdeva, G. Jethwani, C. Manjunath, M. Balamurugan, and A. V. Krishna, "An effective time series analysis for equity market prediction using deep learning model," in *2019 International Conference on Data Science and Communication (IconDSC)*. IEEE, 2019, pp. 1–5. 35, 39

[40] A. K. Khan, M. Anwer, and S. Banik, "Market timing decisions by hybrid machine learning technique: A case study for dhaka stock market," *Journal of Computations & Modelling*, vol. 3, no. 2, pp. 183–202, 2013. 36, 39

[41] M. Usmani, S. H. Adil, K. Raza, and S. S. A. Ali, "Stock market prediction using machine learning techniques," in *2016 3rd international conference on computer and information sciences (ICCOINS)*. IEEE, 2016, pp. 322–327. 36, 37, 40

[42] K. Khare, O. Darekar, P. Gupta, and V. Attar, "Short term stock price prediction using deep learning," in *2017 2nd IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)*. IEEE, 2017, pp. 482–486. 38, 41

[43] W. Li and J. Liao, "A comparative study on trend forecasting approach for stock price time series," in *2017 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, 2017, pp. 74–78. 38, 41

[44] A. Porshnev, I. Redkin, and A. Shevchenko, "Machine learning in prediction of stock market indicators based on historical data and data from twitter sentiment analysis," in *2013 IEEE 13th International Conference on Data Mining Workshops*, 2013, pp. 440–444. 38, 41

[45] A. Bhowmick, A. Rahman, and R. M. Rahman, "Performance analysis of different recurrent neural network architectures and classical statistical model for financial forecasting: A case study on dhaka stock exchange," in *Computer science on-line conference*. Springer, 2019, pp. 277–286. 40

[46] S. Selvin, R. Vinayakumar, E. Gopalakrishnan, V. K. Menon, and K. Soman, "Stock price prediction using lstm, rnn and cnn-sliding window model," in *2017 international conference on advances in computing, communications and informatics (icacci)*. IEEE, 2017, pp. 1643–1647. 41

[47] Dhaka stock exchange's csv data. [Online]. Available: https://dsecsv.simdif.com/ 43

[48] Display company information — dhaka stock exchange. [Online]. Available: https://www.dse.com.bd/displayCompany.php?name=BEXIMCO 43

[49] pandas - python data analysis library.". [Online]. Available: ttps://pandas.pydata.org/ 45, 51

[50] Numpy. [Online]. Available: https://numpy.org/ 45, 51

[51] Welcome to python.org. [Online]. Available: https://www.python.org/ 45, 51

[52] Project jupyter — home. [Online]. Available: https://jupyter.org/ 51

[53] Mashfiqur-rahman/stock-market-analysis-using-ml: stock-market-analysis-using-ml. [Online]. Available: https://github.com/Mashfiqur-Rahman/stock-market-analysis-using-ml 51

[54] Tensorflow. [Online]. Available: ttps://www.tensorflow.org/ 51, 52

[55] Keras: the python deep learning api. [Online]. Available: https://keras.io/ 51, 52

[56] J. D. Hunter and the Matplotlib Development Team, "Matplotlib," 2020, accessed: 2023-04-09. [Online]. Available: https://matplotlib.org/ 51

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn:

Machine learning in Python," pp. 2825–2830, 2011. [Online]. Available: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html 51

Appendix A

Appendix Title

A.1 Section Title

Write section here.

A.1.1 Sub-section here

Write sub-section here.

Appendix B

Appendix Title

Add appendix here.