BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC KINH TẾ THÀNH PHỐ HỒ CHÍ MINH

BÁO CÁO TỔNG KẾT

ĐỀ TÀI NGHIÊN CỨU KHOA HỌC THAM GIA XÉT GIẢI THƯỞNG "NHÀ NGHIÊN CỨU TRẺ UEH" NĂM 2023

ON STOCK PRICE PREDICTION: A DEEP LEARNING APPROACH USING BIDIRECTIONAL LONG-SHORT TERM MEMORY (BiLSTM)

Thuộc nhóm chuyên ngành: Khoa học dữ liệu và trí tuệ nhân tạo

TP.Hồ Chí Minh Ngày 2 tháng 2 năm 2023

Contents

List of Figures

List of Tables

List of Abbreviations

Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ARIMA	AutoRegressive Integrated Moving Average
BiLSTM	Bidirectional Long-Short Term Memory
CNN	Convolutional Neural Network
DBNs	Deep Belief Networks
FCNs	Fully Connected Networks
Fig.	Figure
GARCH	Generalized AutoRegressive Conditional Heteroskedasticity
HoSE	Ho Chi Minh City Stock Exchange
LSTM	Long-Short Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MFNN	Multi-Filters Neural Network
MSE	Mean Squared Error
NYSE	New York Stock Exchange
RBMs	Restricted Boltzmann Machines
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RMSprop	Root Mean Squared Propagation
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SSE	Shanghai Stock Exchange
SVMs	Support Vector Machines
SVR	Support Vector Regression
Tanh	Hyperbolic Tangent

1 Summary of the research

Introduction: Stock price forecasting is a difficult subject that has received substantial research in the realm of finance. Utilizing machine learning methods, such as deep learning, to model and forecast future stock values is one way to tackle this issue. Bidirectional Long Short-Term Memory (BiLSTM) is a sort of recurrent neural network (RNN) that is especially well-suited for this purpose since it can take input sequences of various lengths and keep a "memory" of prior inputs. In this work we use BiLSTM to predict the stock price of Apple Inc. from the training set that contains records of Apple stock prices in five years' time.

Objective: The objective of this experiment is to examine the advantages of using BiLSTM for stock price prediction. The aim of this experiment is to demonstrate the effectiveness of BiLSTM in stock price prediction. In specific, we compare our BiLSTM model with other machine learning and deep learning methods including SVR, KNN, Vanilla RNN, LSTM and show that BiLSTM is the best model in terms of four different performance metrics.

Methods: In this work, we started with a description of the data preparation methods and the BiLSTM model design. We then examined the use of different optimization techniques like Adam, RMSprop, and SGD, as well as activation functions including Sigmoid, Tanh, and ReLU in order to enhance the performance of the BiLSTM model. We later showed that Adam outperforms other optimization algorithms and that Tanh has the best performance among the three activation functions.

Results: Our results demonstrate that the BiLSTM model works best with Adam optimization algorithm as well as Tanh activation. The BiLSTM model was also able to capture the data's nonlinear and sequential patterns, resulting in more accurate predictions. Specifically, BiLSTM surpasses all conventional machine learning and deep learning methods taken into consideration.

2 Introduction

2.1 Stock price prediction and its significance

Stock price prediction is the technique of forecasting future stock prices using previous stock market data and other pertinent information. The importance of stock price prediction stems from the fact that it may give useful insights for investors and traders in making educated stock market decisions.

The problem of stock price prediction is one of the most challenging and important tasks in finance and economics. The stock market is a complex and dynamic system that is influenced by a variety of factors, including economic indicators, company-specific news, investor sentiment [Barsky and De Long (1993); Mitchell and Mulherin (1994); Qi and Maddala (1999); Hondroyiannis and Papapetrou (2001)] and pandemic [Mazur et al. (2021); Baker et al. (2020)]. The task of predicting stock prices is challenging because it requires understanding and modeling the relationships between these factors and the stock prices.

The importance of stock price forecasting is multifaceted. The ability to anticipate stock prices can assist individual investors and traders make educated judgments about purchasing or selling stocks. It can also help traders make winning transactions by purchasing cheap and selling high. For example, if an investor believes the value of a stock will grow, he or she may elect to acquire shares of that stock. If, on the other hand, a trader believes the value of a stock will decline, they may attempt to short-sell that asset. Accurate forecasting can also assist them in managing portfolio risk and optimizing returns.

As stock market performance is generally regarded as a reflection of an economy's overall health [Masoud (2013)], stock price prediction has a range of effects on macroeconomics. The future performance of businesses and the economy as a whole may be usefully predicted by accurate stock price predictions, which can then be used to inform investment and policy decisions.

Investment decisions can be influenced by stock price forecasts, which can then have an effect on macroeconomics. More investment in such companies might encourage

economic development in that region if stock price estimates indicate that a certain industry or group of businesses is anticipated to do well in the future. On the other hand, if stock price predictions indicate that a certain industry or set of businesses is likely to do poorly in the future, this might result in less investment in those businesses and a decline in economic activity in that industry.

Forecasts for stock prices can also affect the monetary policy decisions made by central banks [Li et al. (2010)]. In order to foster economic development and encourage borrowing and spending, the central bank may lower interest rates if stock market expectations indicate that the economy would stall. The central bank may raise interest rates in order to calm the economy and limit borrowing and expenditure if, on the other hand, stock price projections indicate that there is a danger of inflation and that the economy is overheating.

Forecasts for stock prices might also affect fiscal policy. For instance, the government could think about implementing stimulus measures like increased spending or tax cuts if stock market predictions indicate that the economy may stagnate. Additionally, since it may assist boost market efficiency and stability, accurate stock price forecast can also be advantageous for the stock market's entire operation. Stock price projections can result in more effective price discovery and less market volatility by giving market players greater information.

Overall, stock price prediction is a complex and challenging task that has significant implications for individuals, institutions and the economy as a whole.

2.2 Background information on the stock market

The stock market is a marketplace for buying and selling stocks (or shares) of publicly listed corporations. Stocks indicate a company's ownership, and their value is decided by the market's view of the company's future financial performance and development potential. Investors can purchase stocks in firms that they feel will perform well and profit, or sell stocks in companies that they believe will perform poorly and lose money.

The stock market is made up of various exchanges where equities are exchanged, such as the New York Stock Exchange (NYSE) and the NASDAQ, or, in case of the

Vietnamese market, the Ho Chi Minh City Stock Exchange (HoSE). Companies must fulfill particular listing standards in order to have their stocks listed on these exchanges. Once a company's stock is listed on an exchange, investors may buy and sell it using their brokerage accounts.

The stock market is commonly regarded as a leading predictor of an economy's overall health, since its performance is frequently utilized as a gauge of investor and trader confidence and attitude. When the stock market performs well, it is typically regarded as an indication of a healthy economy; when the stock market performs poorly, it is regarded as a symptom of a weak economy.

The stock market also plays a significant role in capital allocation. Companies that require capital to grow and expand can generate funds by issuing and selling stocks to the general public. This enables businesses to obtain the cash they want to develop and expand without having to borrow from banks or other financial organizations. Furthermore, stock exchanges enable corporations to raise cash by issuing bonds, which are debt securities.

2.3 Conventional and machine learning techniques in stock price prediction

Fundamental analysis and technical analysis are two traditional methods for predicting stock prices. Fundamental analysis is a technique for determining a company's intrinsic worth by looking at its financial and economic fundamentals such as profits, sales, assets, liabilities, and management. It is predicated on the assumption that a company's fundamental financial and economic difficulties would eventually be reflected in its stock price. It entails examining a company's financial records and other pertinent data to estimate its inherent worth and development prospects. Financial measures such as sales, profits, and debt, as well as information about the company's management and industry trends, can all be included in this data.

Technical analysis, on the other hand, is a way of appraising securities that involves monitoring market activity indicators such as historical prices, volume, and open interest. Charts and numerous technical indicators are used by technical analysts to detect patterns and trends that may be utilized to forecast future stock values. Technical analysts spot patterns and make forecasts using tools such as charts, moving averages,

and oscillators. Technical analysis is founded on the assumption that market patterns, as depicted by charts and other technical indicators, may forecast future behavior.

These strategies are not flawless and have limits of their own. Fundamental research, for example, may not always produce accurate projections of stock prices since it is dependent on the quality of financial information provided by firms and the assumptions made by analysts. Technical analysis has its own limits because it is based on previous patterns and trends that may or may not be replicated in the future. Furthermore, the stock market is influenced by a variety of other variables such as changes in government regulations, global events, geopolitical threats, and investor attitude, all of which can have an impact on stock prices and are not necessarily taken into account in these traditional methodologies.

In recent years, machine learning has emerged as a powerful tool for stock price prediction. Machine learning is a branch of artificial intelligence that focuses on creating algorithms and statistical models that let computers get better at a task over time. It includes training models to make predictions or choices without being expressly taught to do so by using a lot of data and algorithms. It has been used in a variety of stock price prediction tasks, such as predicting future prices, identifying trends, and detecting patterns in the data [Leung et al. (2014); Patel et al. (2015); Ampomah et al. (2020)]. Additionally, on the task of predicting stock price, a variety of machine learning methods may be applied.

2.4 Apple Incorporated (Apple Inc.)

Located in Cupertino, California, Apple Inc. is a global technological business. It creates, produces, and markets consumer gadgets, software, and internet services. The iPhone, iPad, MacBook, and Apple Watch are among the company's best-known hardware products. It is also recognized for its software and services, which include the iOS and macOS mobile operating systems, the iTunes media player, the Safari web browser, and other productivity tools.

Apple is renowned for its powerful brand, devoted client base, and cutting-edge, usable goods. Customers of the firm have developed a cult-like following as a result of the company's reputation for secrecy and strict control over its products. The iPhone is

Apple's best-selling device in terms of product sales, accounting for over 55% of the company's income in 2020. While the Apple Watch and AirPods have experienced considerable growth in recent years, the iPad and Mac laptops also bring in sizable sums of money. The App Store, Apple Music, iCloud, and Apple Pay are all parts of Apple's services division, which contributes significantly to the company's income.

Apple is estimated to have revenues of $274.5 billion in 2020, making it one of the greatest corporations in the world by sales and market capitalization by 2021. One of the most valuable brands in the world, its market valuation as of 2021 was over $2.4 trillion. Apple's consumers have developed a cult-like following as a result of its reputation for secrecy and strict control over its goods. The business is renowned for both its enduring brand and devoted clientele as well as for its cutting-edge and user-friendly goods. Additionally honored for its dedication to social and environmental responsibility, it has been named the most respected corporation in the world by Fortune magazine for a number of years running.

Apple Inc. is a publicly listed business, and shares of the corporation trade on the NASDAQ under the ticker AAPL. Long-term outstanding performance of the company's shares has made it one of the most valuable publicly listed businesses in the world.

The stock has been split several times over the years, with the last split in 2014, which was a 7-for-1 stock split. The stock has had a steady upward trend over the years, with some fluctuations in the short term.

In terms of dividends, Apple has a history of providing dividends to its shareholders. The company has been consistently increasing its dividends since it initiated a dividend program in 2012. As of 2021, the company's dividend yield is around 0.1%.

The debut of new products, financial performance, and general market circumstances have all had an impact on Apple's stock price. Analysts and investors actively monitor the company's financial data, and the performance of the business in terms of sales, profits, and margins can have an impact on the stock price. The most noticeable growth in the stock price of Apple throughout the years was its jump from approximately $7 per share in 2003 to around $700 per share in 2012, a return of roughly 10000%.

The stock has also had some fluctuations in the short term, for example, during the

period of the global financial crisis of 2008-2009, the stock price dropped from around $200 per share to around $80 per share, a decline of around 60%. However, the stock quickly recovered and reached a new high in 2012. In recent years, the stock has seen steady growth, reaching an all-time high of around $150 per share in August 2020 and it's still going up.

In general, Apple's stock has been considered a blue-chip stock and a safe investment, it is a consistent and stable performer, but it also can be affected by market conditions and industry trends just like any other stock.

3 Literature review

For stock price prediction, traditional machine learning approaches such as linear regression, support vector machines (SVMs), and decision trees have been frequently employed. Because of their simplicity, interpretability, and reasonably decent performance, these strategies have been widely embraced in finance. Before proceeding with further study, it is necessary to first understand the basic studies that the researchers have done and published. We present some of the earlier stock price prediction research in this section and evaluate their benefits and drawbacks.

3.1 Previous research on stock price prediction using machine learning techniques

Patel et al. (2015) analyze two ways using four distinct speculative models. The four models are Naive Bayes, Random Forest, Artificial Neural Network, and Support Vector Machine (SVM). The first approach of data input uses stock trading data to identify 10 technical limitations, whereas the second method focuses on reproducing these characteristics using static decision data. For each of these two input techniques, the correctness of each speculation model is examined.

Sun et al. (2019) suggest using an ARIMA-GARCH-NN, a machine learning technique, to identify internal patterns and forecast the stock market. They examine high-frequency stock market data in the US using explicit methodologies and procedures, sensory networks, and fundamental financial pricing models to assure variety. The results offer the first sign of a market shock. They also guarantee the appropriate

operation of ARIMA-GARCH-NN by identifying patterns in huge datasets without depending on reliable distribution estimations. Their technique successfully blends the advantages of conventional financial procedures with "data-driven" methodologies in order to reveal hidden patterns in vast volumes of financial data. In the study, they suggest and put into practice ARIMA-GARCH-NN, a data-driven approach, to address the challenging issue of stock market prediction.

Long et al. (2019) presented the Multi-Filters Neural Network (MFNN) model that eliminates aspects of financial time series and predicts stock prices. Convolutional and duplicate neurons are used in this model to provide alternative filter formats, enabling the utilization of data from diverse sources and market theories. The CSI 300 of the Chinese stock market was used to assess the MFNN model, and it was shown to have a positive prediction result of 11.47% for the best machine learning technique and 19.75% for the best mathematical method. In comparison to CNN and RNN, filters and purposeful network design increased accuracy by 7.19% and 6.28%, respectively.

Long et al. (2020) demonstrate a neural network model that forecasts stock price changes using historical data and publicly available information. In order to choose the optimal target stock for market structure and trading information, the model uses a graph of data and methodologies. The model employs a method called graph embedding to compress the data and attach it to a convolutional neural network to detect investing styles in order to manage the vast quantity of data and complexity. In order to anticipate stock price patterns and track attentiveness, the model also makes use of a short-term memory network, which can help with financial decision-making. The findings also demonstrated the model's robustness and performance. The model's claimed prediction accuracy is reported to be 70% or higher, which is greater than existing approaches.

Jing et al. (2021) propose the use of Convolutional neural networks (CNN) to categorize investor attitudes from stock forums, while long short-term memory (LSTM) neural networks are used to assess technical stock market indicators. Three different time intervals of real-world data from six important businesses on the Shanghai Stock Exchange (SSE) were used to evaluate the model. The findings demonstrate that the proposed approach outperforms previous algorithms, even ones lacking sentiment

analysis, in categorizing investor feelings and making stock price predictions.

3.2 Advantages and limitations of different methods that have been used in the past

There are several methods that have been used in the past for stock price prediction, each with its own benefits and drawbacks.

- **ARIMA and GARCH** :ARIMA (AutoRegressive Integrated Moving Average) and GARCH (Generalized AutoRegressive Conditional Heteroskedasticity) is one of the most popular methods in the use of traditional statistical and econometric techniques. These methods are based on the assumption that stock prices follow a certain pattern and can be modeled using historical data. They are simple to implement and interpret, and have been widely used in finance and economics. However, these methods have a number of limitations. They assume a linear relationship between the variables and may not be able to capture non-linear relationships or complex interactions between variables. They also assume that the data is stationary and may not be able to handle non-stationary data.
- **Time Series Decomposition** : Time series decomposition is a technique for breaking down a time series into multiple components such as trend, seasonal, and residual components. This strategy is beneficial for discovering underlying patterns in data and for making predictions. This strategy, however, is only useful when the time series is somewhat stable and contains distinct patterns. This strategy may not be as useful if the time series is noisy or includes many patterns. Time series decomposition has the advantage of providing a clear knowledge of the underlying structure of the data, which can be beneficial in spotting trends or anomalies. However, the approach cannot capture complicated patterns in the data and is vulnerable to outliers and missing values, which can influence the decomposition's accuracy.
- **Support Vector Regression (SVR)** is a supervised machine learning approach that may be used to forecast stock prices. The Support Vector Machine (SVM) algorithm, which is frequently employed for classification applications, is a version of this one. When attempting to predict a continuous variable, such as stock prices, SVR is used in regression tasks. The fundamental goal of SVR is to identify the best-fitting line, or

hyperplane, that divides the data into distinct classes. When predicting stock prices, the computer looks for the line that best fits the previous stock prices and divides them into several categories. The best fit line may then be used to forecast future stock price movements. SVR offers certain benefits over other stock price prediction techniques. It works well with tiny datasets and can manage non-linear connections in the data, for instance. SVR is also less vulnerable to outliers than conventional linear regression techniques. SVR, however, also has significant drawbacks. It may be sensitive, for instance, to the regularization parameter and kernel function selection. It might be more challenging to understand the findings since SVR doesn't always yield a singular outcome.

- **K-nearest neighbors (KNN)**, is a machine learning technique that may be applied to classification or regression applications. It would probably be employed for regression in the context of stock price prediction, as the objective would be to forecast a continuous value (the future stock price) based on a collection of input characteristics (such as past stock prices, trading volume, etc.). The algorithm would then identify the k-nearest data points (i.e. the k data points that are most similar to the current stock price based on these input features) and use these data points to make a prediction about the future stock price. The key drawbacks of KNN are that it lacks assumptions about the underlying link between the input characteristics and the output variable and is a "lazy learner," as opposed to other machine learning techniques. Because of this, it could be challenging to grasp how the model's input properties connect to the stock price. Furthermore, KNN can be sensitive to the selection of k and the distance measure employed, which can impact the success of the model.

  By averaging the predictions of various decision trees, Random Forest solves the overfitting issue and produces more reliable predictions. Furthermore, Random Forest is capable of handling large dimensional data and can locate the key features in the data. However, Random Forest may not perform well if the data is heavily linked and might be computationally demanding.

Each method has its own advantages and limitations. Traditional statistical and econometric techniques are simple to implement and interpret but may not be able to

capture non-linear relationships or complex interactions between variables. Machine learning techniques are able to model non-linear relationships and complex interactions between variables but require large amounts of data and computational resources. And deep learning methods are powerful for time series data and handle missing data, but can be difficult to interpret.

4 Data Preprocessing

Data preprocessing is the process of preparing data for use in a machine learning model. Some of the duties include data augmentation, data segmentation, data reduction, data reduction, integration, and transformation. It is a crucial step in the model-building process because by removing bias and noise from the data and preparing it for use in a machine learning model, it may help to improve model performance.

4.1 Data collection process

Data collection for stock price prediction using LSTM typically involves gathering historical stock price and trading data from a reliable source. One popular source for this data is Yahoo Finance, which provides financial data for a wide range of companies and markets. The process of collecting data from Yahoo Finance typically involves the following steps:

- **Identifying the stock or stocks of interest:** The first step is to identify the stock or stocks for which data needs to be collected. This could be a specific company or a group of companies within a specific industry or market. In this research, we choose Apple's stock price records to experiment with our model.
- **Accessing the Yahoo Finance website:** Once the stocks of interest have been identified, the next step is to access the Yahoo Finance website and navigate to the page for the stock or stocks of interest.
- **Downloading historical data:** On the stock page, there is usually a link to download historical data for the stock. This data includes information such as the stock's opening and closing prices, trading volume, and other financial metrics. The data can be downloaded in a CSV format.

- **Cleaning and preprocessing the data:** After the data is downloaded, we have the data to be cleaned and preprocessed to remove any missing or irrelevant data, and make sure that the data is in a format that can be used for analysis.
- **Saving and storing the data:** Once the data is cleaned and preprocessed, we saved and stored in a format that can be easily accessed and used for analysis.

Our dataset includes Open price (Fig. 1), High price (Fig. 3), Low price (Fig. 6), Close price (Fig. 2), Adjusted Close price (Fig. 4) as well as the volume (Fig. 5) of Apple Inc. stocks that were updated daily during the period of 5 years, ranging from 22th January 2018 to 22th January 2023, where

- **The open price** of a stock is the price at which the stock began trading on the stock market for the current trading day. The open price is decided by the exchange at the start of trade and is based on the previous day's closing price and other market considerations.

Figure 1: Apple Inc. Open stock price from 2018 to 2023

- **The closing price** of a stock is the last price at which a stock is traded on a stock exchange during a trading day. The exchange determines the closing price at the conclusion of trading and is based on the final transaction that occurred during the trading day. It is used to compute the net change and percentage change in the stock's

value during the course of the trading day, as well as the value of the stock at the conclusion of the trading day.

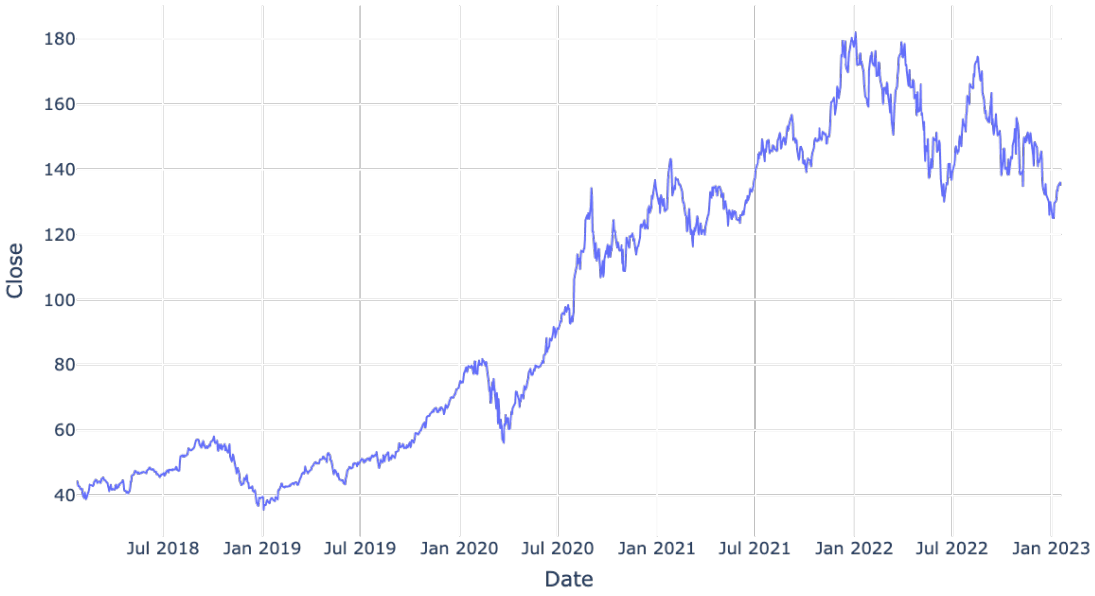

Figure 2: Apple Inc. Close stock price from 2018 to 2023

- **The highest price** at which a stock traded during a certain trading day or period of time is referred to as its high price. It is used to represent the stock's highest price throughout that time period. High prices can help traders, investors, and analysts analyze and make choices about stock performance.

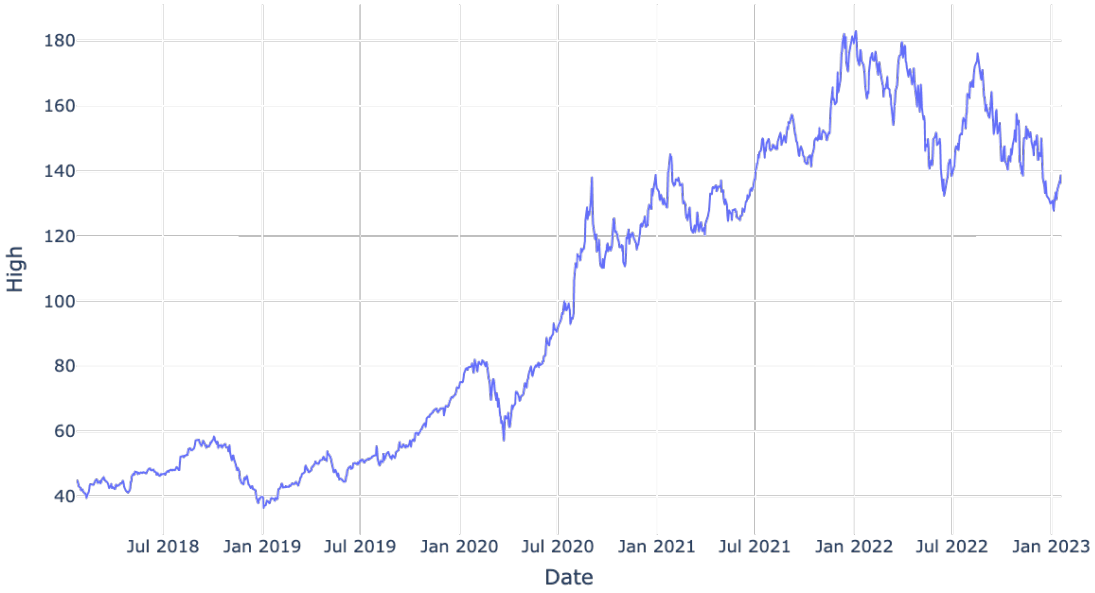

Figure 3: Apple Inc. High stock price from 2018 to 2023

- **The adjusted closing price** of stock takes into account company activities such as stock splits, dividends, and other special events that may affect the stock's price. The adjusted close price is used to calculate a stock's historical return and is said to be a more accurate representation of the stock's performance than the ordinary closing price. This price is used to correct these occurrences and provide a more realistic view of the stock's performance over time.

Figure 4: Apple Inc. Adjusted Close stock price from 2018 to 2023

- **The volume** of stock refers to the number of shares traded during a certain trading day or period of time. It is used to show a stock's degree of activity and liquidity. High volume usually implies a high degree of interest in a stock, whilst low volume may imply a lower level of interest. Stock volume may be a useful statistic for traders, investors, and analysts for evaluating stock performance, making choices, and forecasting future price movements.

- **The low price** is the lowest price that a stock reached during a trading day. It is one of several indicators of a stock's performance, along with the open price, close price, and high price. These prices provide insight into how the stock's value has changed over a specific period of time, and can be used to determine the stock's trend during the trading day and its volatility.

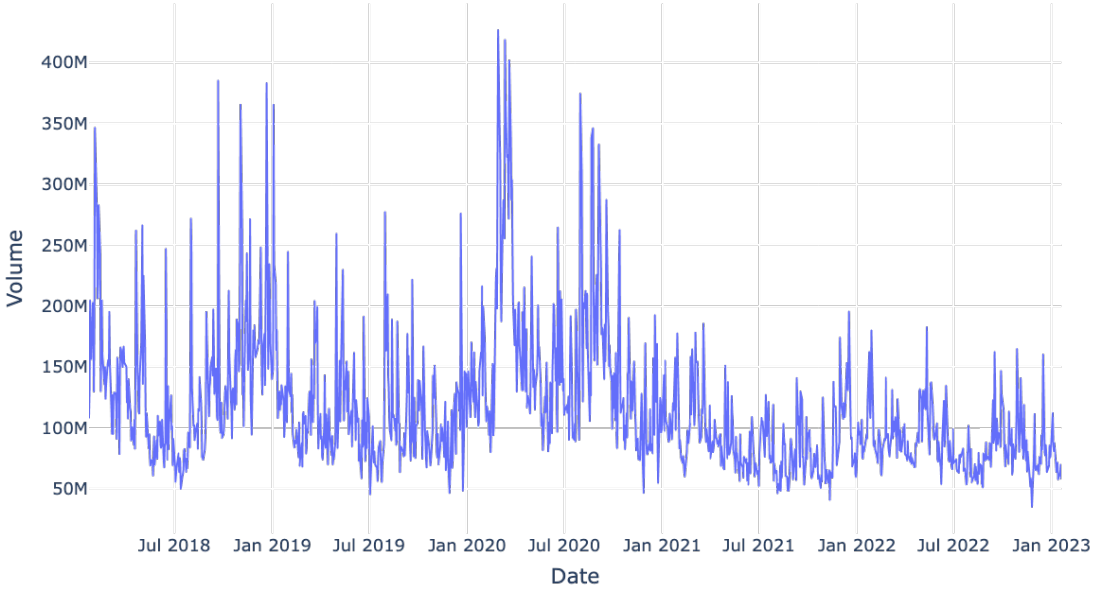

Figure 5: Apple Inc. stock price volume from 2018 to 2023

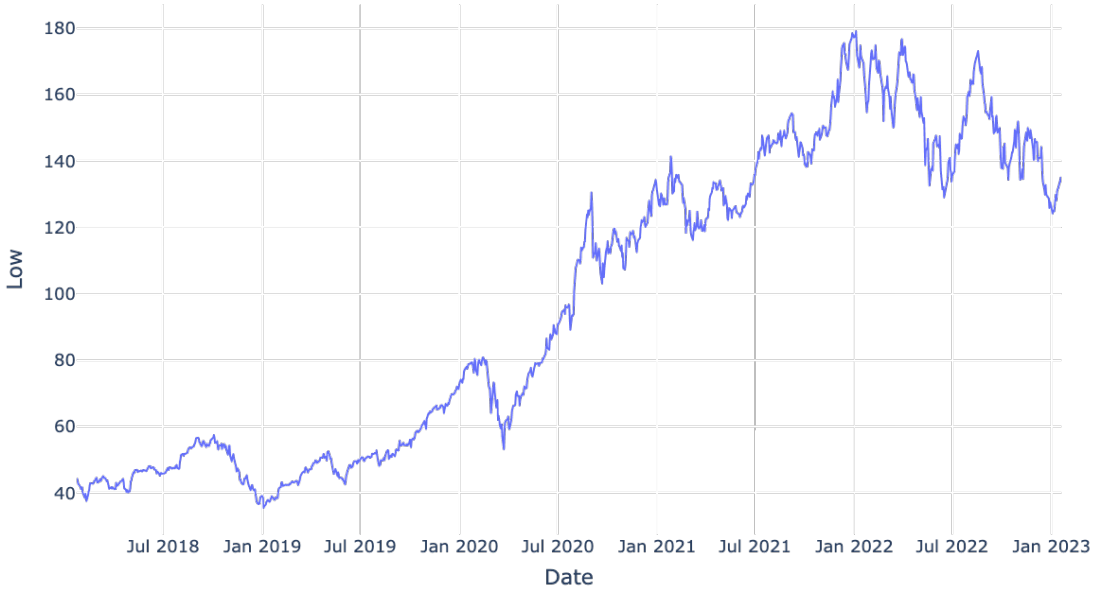

Figure 6: Apple Inc. Low stock price from 2018 to 2023

4.2 Explanatory Data Analysis

Exploratory Data Analysis (EDA) is a method for examining and condensing a dataset in order to comprehend its underlying patterns, connections, and structures. Combining visual and statistical techniques, such as producing histograms, scatter plots, and box plots, as well as computing summary statistics like mean, median, and standard deviation, are frequently used to do this. EDA seeks to develop hypotheses about the

data and find any potential outliers or abnormalities. It is a crucial first stage in any data analysis project and influences the choice of more sophisticated approaches that may be applied later.

Table 1: The first ten rows of the Apple stock price dataset.

Date	Open	High	Low	Close	Adj Close	Volume
2018-01-22	44.325001	44.445000	44.150002	44.250000	42.077320	108434400
2018-01-23	44.325001	44.860001	44.205002	44.259998	42.086819	130756400
2018-01-24	44.312500	44.325001	43.299999	43.555000	41.416439	204420400
2018-01-25	43.627499	43.737499	42.632500	42.777500	40.677120	166116000
2018-01-26	43.000000	43.000000	42.514999	42.877499	40.772205	156572000
2018-01-29	42.540001	42.540001	41.767502	41.990002	39.928280	202561600
2018-01-30	41.382500	41.842499	41.174999	41.742500	39.692936	184192800
2018-01-31	41.717499	42.110001	41.625000	41.857498	39.802284	129915600
2018-02-01	41.792500	42.154999	41.689999	41.945000	39.885494	188923200
2018-02-02	41.500000	41.700001	40.025002	40.125000	38.154842	346375200

Correlation

We calculate the correlation of the columns with each other, the result of this correlation function is a table showing the correlation between the columns. At a glance we can see that this has a matrix of size $n \times n$ where n is the number of columns in the table. And the positions on the main diagonal of the matrix represent the correlation of an attribute to itself, so the result of the correlation at the main diagonals on the matrix is 1.

Here we use Pearson method. Given the paired data $\{(x_1, y_1), ..., (x_n, y_n)\}$, the Pearson correlation is calculated as follows:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where,

- n is the sample size
- x_i, y_i are the individual sample points indexed with i
- $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$; and the same for $\bar{y}$

Pearson correlation coefficient r_{xy} fluctuates in the continuous range from -1 to 1 (i.e. $r_{xy} \in [-1, 1]$). Specifically, strong positive relationships are indicated by a coefficient

of 1, strong negative relationships are shown by a coefficient of 1, and no linear relationships are indicated by a value of 0. To ascertain the relationship between variables in data analysis, Pearson correlation is frequently utilized.

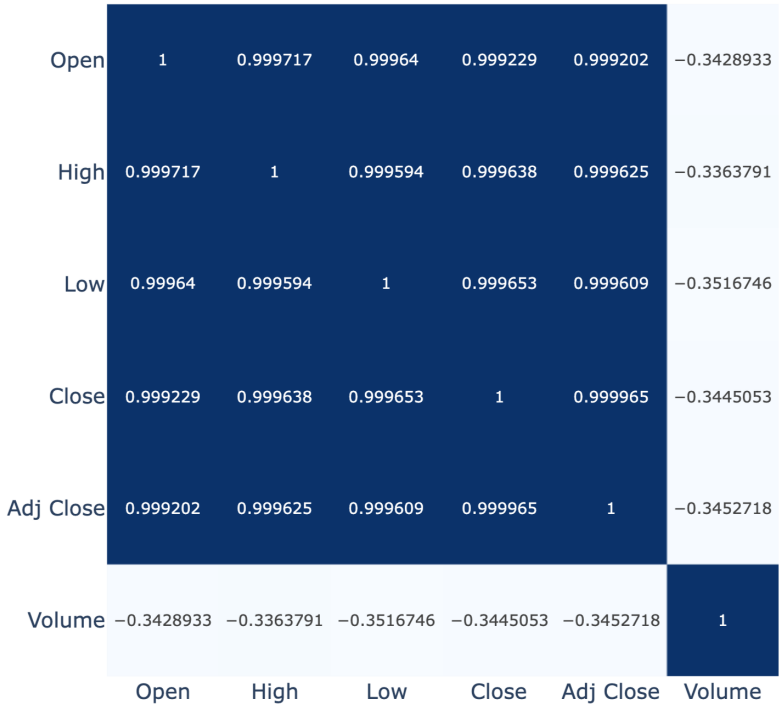

Figure 7: Correlation heatmap of the dataset

It is noteworthy that we must choose the X and y of the problem appropriately, as with any supervised learning task on tabular data. Typically, X represents the features and y is the label for the features X (usually clearly defined in the data table). The precise labels provided by the problem are not included in this problem, though. As a result, we must modify the data's input as well as the output using the Sliding Window method (later described in section 4.4).

It is important to note that the columns contained in the dataset are independent of each other, not features and target. The attributes are strongly correlated, but when forecasting the Adjusted Close Price, we will only use the data of the Adjusted Close Price in the past.

We first look for the relationship between the close price and the adjusted close price. We observe a strong correlation between these two characteristics. This implies that when the closing price rises, so will the adjusted close price. Furthermore, to rise by a nearly identical quantity. It's known as a positive correlation.

Date	Adj Close	
2018-01-22	42.077320	
2018-01-23	42.086819	
2018-01-24	41.416439	
2018-01-25	40.677120	
2018-01-26	40.772205	X
2018-01-29	39.928280	
2018-01-30	39.692936	
2018-01-31	39.802284	
2018-02-01	39.885494	
2018-02-02	38.154842	y

Figure 8: How to determine X, y based on samples of a given data table

We also discover that there is a negative correlation between the volume sold and adjusted price. We can thus infer that if the total number of shares sold throughout the day rises, the adjusted close price would fall; nevertheless, there is not a strong association between these two characteristics.

Moving Average

A statistical technique for analyzing time series data is the moving average. It is an assessment of the typical value of a collection of data points over a certain time frame. Taking the average of a predetermined number of data points, the moving average is created by "pushing" this average ahead by one data point to create the subsequent average. Up until all data points are included into the moving average, this process is repeated.

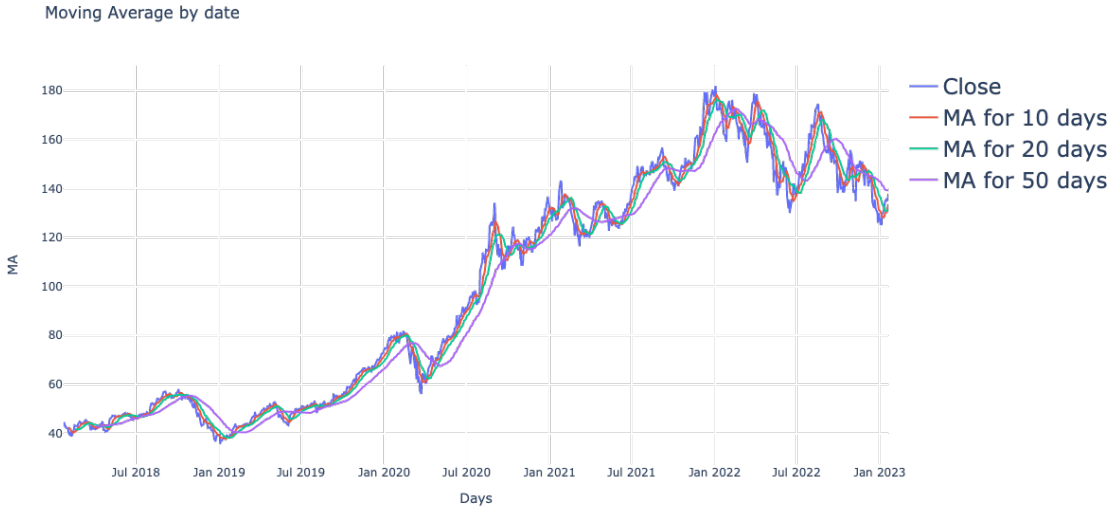

Figure 9: Moving average on Apple stock price for 10 days, 20 days, 50 days

Here, we compute the moving average values for the corresponding lengths of 10

days, 20 days, and 50 days as shown in Fig. 9. Because a moving average is a broad indication that demonstrates how data is smoothed and utilized to corroborate price patterns, it is of particular relevance to us.

4.3 Data Normalization

The practice of putting data values into a consistent scale or distribution is known as data normalization. This is frequently done to ensure that various data characteristics are on a comparable scale, which can increase the accuracy of a prediction model. There are several different types of data normalization that are commonly used in statistics and data analysis. Some of the most common types include:

Min-Max Normalization: This method scales the data so that it falls between a specified range, usually between 0 and 1. The formula for this method is $\frac{x-min}{max-min}$, where x is the original value, min is the minimum value in the dataset, and max is the maximum value in the dataset.

Logarithmic Normalization: This method scales the data by applying a logarithmic transformation. The formula for this method is $log(x+1)$, where x is the original value. This method is mostly used for datasets with a skewed distribution where the presence of outliers or extreme values can affect the analysis.

Z-Score Normalization is a way of transforming a variable's values so that they have a mean of zero and a standard deviation of one. This is often done to make sure that different features of the data are on a similar scale, which can help improve the accuracy of a prediction model.

The advantage of this method is that it makes the data have the same scale, which is useful when comparing data, and also when working with algorithms that are sensitive to the scale of the input. This is because the standardization of the data will make the algorithm less sensitive to the scale of the input, and therefore less likely to be affected by outliers or extreme values.

Another advantage of Z-score normalization is that it allows the data to be transformed into a standard normal distribution, which can be useful in certain types of statistical analysis and machine learning models that assume that the data is normally distributed.

In the case of stock price prediction, Z-score normalization may be applied to historical stock prices and other financial data, such as earnings or revenue, to make sure that these features are all on a similar scale and can be used effectively in a prediction model.

The formula for this method is $\frac{x-\mu}{\sigma}$, where x is the original value, μ is the mean of the dataset, and σ is the standard deviation of the dataset. In this work, we normalized the data using this method by default choice of the Tensorflow normalization layer.

4.4 Sliding Window method

The sliding window method is a technique used to split a sequence of data, such as a time series or a text, into smaller, overlapping segments called windows. The size of the window, or the number of data points it contains, is a user-specified parameter. Each window is then processed separately, and the results are combined to produce a final output.

For example, in a time series analysis, a window of size 10 may be used to extract features from a sequence of 100 data points. The first window would contain data points 1 through 10, the second window would contain data points 2 through 11, and so on. This process continues until the final window, which contains data points 91 through 100.

In text processing, a window of size 2 may be used to extract sequences of words called n-grams. The first window would contain the first and second words, the second window would contain the second and third words, and so on.

The overlapping nature of the sliding window method allows for the capture of temporal or sequential dependencies between data points. It is commonly used in signal processing, natural language processing, and other fields where sequential data is analyzed.

The sliding window method can also be used to predict stock prices. In this application, historical stock price data is split into windows of a user-specified size, and each window is used to train a model to predict the next stock price. The size of the window determines the number of previous stock prices that are used as input to the model.

Algorithm 1: Sliding Window Algorithm

Data: Data, Window size, Step size
Result: Subsets of data

```
for i = 0 to len(data) - window_size do
    subset = data[i:i+window_size]
    perform analysis on subset
    i = i + step_size
end
```

Figure 10: Pseudocode of Sliding window algorithm

For example, if a window size of 10 is used, the first window would contain the stock prices from the first 10 days, the second window would contain the stock prices from the second to the eleventh day, and so on. Each window is then used to train a model that predicts the stock price on the 11th day, based on the stock prices from the first 10 days.

The pseudocode of the sliding window method could be expressed as follows:

The window size and step size are assumed to be integers, the input is a one-dimensional data array, and the objective is to produce data subsets for examination in this pseudocode. The for loop extracts a portion of the data at each iteration while iterating through the data array by increasing the starting index of the window by the step size. The subset can then be handled anyway you'd like.

A final forecast may then be created by combining the models learned on each window. The predictions of all the models can be averaged, or ensemble techniques can be used.

5 Model Development

The process of building a machine learning model that can predict or decide based on incoming data is known as model development. Problem formulation, data gathering and preprocessing, model selection, model training, model assessment, and model deployment are often included. Before the model performs well enough for deployment, it may need to be modified and re-evaluated several times. This is an iterative process.

5.1 BiLSTM model for stock price prediction

Bidirectional Long Short-Term Memory (BiLSTM) [Schuster and Paliwal (1997)] is a type of Recurrent Neural Network (RNN) [Rumelhart et al. (1985); Jordan (1997)] which is particularly useful for handling sequential data. BiLSTM is particularly well-suited for stock price prediction because it can remember the historical stock prices for a long time and hence it can be used to predict future prices based on past prices.

BiLSTM is well-suited for time series data and has been used in a variety of applications, including stock price prediction. The main advantage of BiLSTM over other models, such as traditional feedforward neural networks [Rosenblatt (1958)], is its ability to retain information from previous time steps, which is important in predicting stock prices as the prices are often influenced by historical data.

Additionally, BiLSTM is able to handle the problem of vanishing gradients, which is a common issue in traditional RNNs. We describe this problem in section 5.3. This is because BiLSTM includes a memory cell, input gate, forget gate, and output gate (later described in 5.4), which allow it to selectively store, update, and access information from previous time steps. This enables the model to make more accurate predictions as it can take into account long-term dependencies in the data.

The choice of using a BiLSTM model for stock price prediction is based on the unique properties of this type of model and its ability to effectively learn from time series data. This is thanks to the following properties:

Handling temporal dependencies: BiLSTM is particularly well-suited for handling temporal dependencies, which are present in many time series datasets, including stock prices. This is due to the fact that BiLSTM has a "memory" component that can remember past information, allowing the model to effectively capture long-term dependencies in the data.

Handling noise: BiLSTM is able to handle noise in time series data, which is also a common problem when working with stock price data since BiLSTM has a "memory" component that can remember past information, allowing the model to effectively

capture the underlying patterns in the data despite the presence of noise.

Handling non-linearity: BiLSTM is able to handle non-linearity in time series data, which is a common problem when working with stock price data due to the fact that BiLSTM is a type of neural network, which are able to learn non-linear relationships between inputs and outputs.

Handling of multi-dimensional data: Stock prices are influenced by a variety of factors, such as economic indicators, news articles, and social media sentiment. BiLSTM can handle high-dimensional data and can be used with multiple input and output layers, which allows them to process and make predictions based on multiple types of data.

There are a variety of models that can be used for stock price prediction, including ARIMA, GARCH, and deep learning models such as CNNs and Autoencoders.

ARIMA (Auto-Regressive Integrated Moving Average) is a traditional time series forecasting model that is based on the assumption that the future value of a time series is a linear function of its past values and past errors. ARIMA models are widely used in finance, but they are not able to capture the non-linear dependencies in the data which may limit their performance.

GARCH (Generalized Autoregressive Conditional Heteroscedasticity) models, on the other hand, are used to model volatility in financial time series data. GARCH models are able to capture the volatility of the stock prices, but they don't take into account other factors that affect the stock prices.

CNNs (Convolutional Neural Networks) and Autoencoders are deep learning models that have been used for stock price prediction. CNNs are particularly well-suited for image and video processing, and they can be used to extract features from financial data such as stock charts. Autoencoders are unsupervised learning models that can be used to extract features from the data and to identify patterns in the data.

BiLSTM can more successfully capture the long-term relationships and patterns in financial time series data, giving it an edge over existing models for stock price prediction. Sequential data, long-term dependencies, multidimensional data, noise,

and overfitting may all be handled by it. Furthermore, BiLSTM has the capacity to analyze sequential input and preserve an internal memory state, enabling them to preserve context across time, which is crucial for stock price prediction. BiLSTM may thus be superior to other models for the aforementioned causes.

5.2 Activation functions

Activation functions are mathematical procedures that are applied to a neuron's output in a neural network. They are used to determine a neuron's output, or the signal it delivers to the next layer of the network. **The Sigmoid function** (described in 5.2.1), **the Hyperbolic Tangent (tanh)** (described in section 5.2.3) and **the Rectified Linear Unit (ReLU)** (described in section 5.2.2), and function are examples of common activation functions. The activation function used is determined by the unique problem and the desired model properties.

5.2.1 Sigmoid activation function

The sigmoid activation function is a widely used activation function in artificial neural networks and other machine learning models. It is a mathematical function that maps input values to output values between 0 and 1. The sigmoid function is defined as:

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

Where x is the input value and e is the mathematical constant (approximately 2.718). The sigmoid function produces a curve that is S-shaped, with the output value approaching 0 as the input value becomes more negative, and approaching 1 as the input value becomes more positive, as shown in Fig. 7.

When attempting to forecast one of two classes in binary classification issues, the sigmoid activation function is frequently utilized in the output layer of a neural network . The sigmoid function's output may be thought of as a probability, with near values to 0 denoting a low likelihood of the positive class and close values to 1 denoting a high

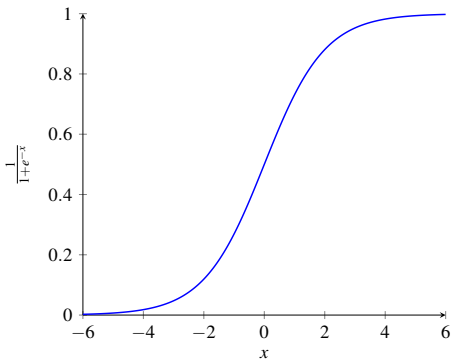

Figure 11: Sigmoid function plot

chance of the positive class.

The sigmoid activation function has several advantages, including:

It is differentiable, which allows for the use of gradient-based optimization algorithms for training the neural network. It outputs values between 0 and 1, which makes it useful for binary classification problems and probability-based predictions. It compresses the input values into a relatively small range, which can help to prevent the vanishing gradient problem during training.

However, the sigmoid activation function also has some limitations, including:

- It can produce saturating gradients (gradients close to zero) when the input value is far from zero, which can slow down or even stop the training process.
- It is not zero-centered, which can cause issues with certain types of initialization and optimization algorithms.
- It can produce output values that are not well-suited for multi-class classification problems.

5.2.2 ReLU activation function

The rectified linear unit (ReLU) activation function was first proposed in the paper "Rectified Linear Units Improve Restricted Boltzmann Machines" by Vinod Nair and Geoffrey Hinton in 2010 [Nair and Hinton (2010)].

In this paper, the authors present an experiment where they trained a deep neural network using the ReLU activation function, and compared its performance to networks

trained using traditional activation functions such as the sigmoid and hyperbolic tangent. They found that the network trained with the ReLU activation function performed better and was able to converge faster than the other networks.

The authors also showed that ReLU activation functions improve the performance of Restricted Boltzmann Machines (RBMs) and Deep Belief Networks (DBNs). The paper also discussed that using ReLU activation functions can be seen as a way of introducing non-linearity in the network.

The paper also discussed that the ReLU activation function is computationally cheaper than other activation functions, making it a more efficient choice for large-scale datasets and deep architectures.

The rectified linear unit (ReLU) activation function is a commonly used activation function in artificial neural networks. It is defined as $f(x) = max(0,x)$, which means that for any input x, the output of the function is either 0 or x.

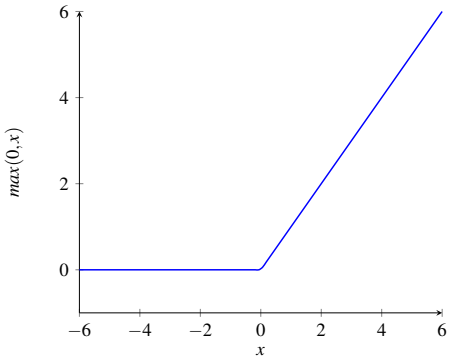

Figure 12: Rectified Linear Unit (ReLU) function plot

The ReLU function is a piecewise linear function with a slope of 1 for positive input values, and a slope of 0 for negative input values. This means that when x is positive, the output of the function is equal to the input, and when x is negative, the output is 0. This can be represented graphically as a step function, where the input and output are plotted on the x- and y-axes respectively, and the step occurs at x = 0.

The main advantage of using the ReLU activation function is its computational efficiency. Unlike other activation functions such as sigmoid and tanh, the ReLU function does not have an exponential term, which makes it computationally cheaper to compute.

Another advantage of ReLU is that it helps to prevent the vanishing gradient problem, which can occur when the weights in a network are updated during training. The vanishing gradient problem happens when the gradient of the activation function becomes very small, which makes the network unable to learn from the input data. ReLU solves this problem by having a constant gradient of 1 for positive input values, which means that the network can learn from the input data.

ReLU is commonly used in deep learning architectures, such as Convolutional Neural Networks (CNNs) and Fully Connected Networks (FCNs) as it has been shown to improve the performance and training speed of the network

5.2.3 Hyperbolic tangent (Tanh) activation function

The hyperbolic tangent (tanh) activation function is a common choice for LSTM (Long Short-Term Memory) networks and other neural network architectures. It is a non-linear function that maps its input to a value between -1 and 1, i.e. $tanh(x) \in [-1;1]$

The tanh function is defined as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

It is similar to the logistic sigmoid function, but it is shifted and scaled such that its range is between -1 and 1. This allows for more flexibility in the output of the function, making it a good choice for tasks such as stock price prediction where the output values can vary over a wide range.

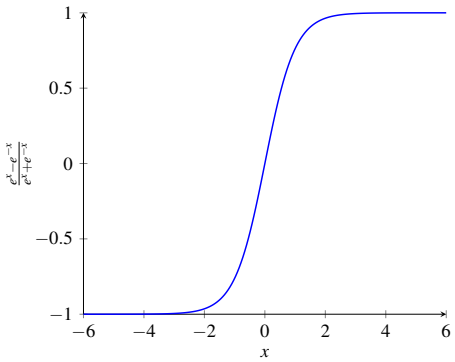

Figure 13: Hyperbolic tangent function plot

The tanh function has a number of useful properties. It is a differentiable function,

which means that it can be used in backpropagation, a technique used to train neural networks. It is also a monotonically increasing function, which means that it preserves the ordering of its input values. This can be useful in certain types of problems, such as in sorting.

The tanh function also has a "centering" effect, where the output values are centered around 0, which can be beneficial for certain types of problems. This is in contrast to the sigmoid function which centers its output around 0.5.

In LSTMs, the tanh function is used as an activation function in the "memory cell" component of the network. The memory cell is responsible for maintaining the information from previous time steps and passing it along to future time steps. The tanh function is applied to the input to the memory cell, which allows the network to learn which information to keep and which to discard.

The tanh function maps the input to a value between -1 and 1. This is useful in LSTMs because it allows the network to learn a wide range of values for the information stored in the memory cell, from very negative values to very positive values. This is important for LSTMs because it allows them to learn to handle different types of input data, such as positive and negative values.Additionally, the use of the tanh function in LSTMs also provides an additional level of non-linearity, which helps to improve the ability of the network to model complex data.

One limitation of the tanh function is that its output saturates at the extremes, meaning that for large input values the function's output will be close to 1 or -1, which may cause the gradients to become small and make it difficult for the model to learn.

Later in section 6.2, we show that Tanh outperforms both Sigmoid and ReLU activation functions in terms of all performance metrics used. .

5.3 Vanishing Gradient

The vanishing gradient problem is a common issue that can occur when training deep neural networks. It occurs when the gradients of the weights in the network become very small during backpropagation, causing the network to learn very slowly or not at all.

Backpropagation is an algorithm used to train neural networks, it works by propagating the errors from the output layer to the input layer through the network [Rumelhart et al. (1985)]. During this process, the gradients of the weights are calculated and used to update the weights with the goal of minimizing the error of the network. However, when the gradients are very small they will have a small impact in the weight update, making the training slow or even stop.

In other words, the vanishing gradient problem arises when the gradients are multiplied by the weights themselves and the derivatives of the activation functions during backpropagation. As these gradients flow backwards through the network, they can become very small and eventually disappear if the weights are very small or the activation functions have small derivatives.

This problem is particularly pronounced in deep neural networks, where the gradients must flow through many layers before reaching the input layer. Each time the gradients flow through a layer, they are multiplied by the weights and the derivatives of the activation functions. This means that by the time the gradients reach the input layer, they may have become very small and unable to update the weights effectively.

In summary, the vanishing gradient problem is a common issue that can occur when training deep neural networks. It occurs when the gradients of the weights in the network become very small during backpropagation, causing the network to learn very slowly or not at all. There are several solutions to this problem, such as using different activation functions, architectures with skip connections, gradient clipping, weight initialization, and batch normalization.

5.4 LSTM Architecture

The input layer is responsible for receiving the input data, which in the case of stock price prediction would typically be historical stock prices, trading volume, and other relevant financial indicators. The input layer can also be augmented with additional features such as technical indicators or economic indicators. X_t is passed through a linear transformation represented by a weight matrix $\mathbf{W}_i$ and a bias vector b_i, resulting in a new representation $Z_i = \mathbf{W_i} * X_t + b_i$

The hidden layers are responsible for processing the input data and passing it through the LSTM cells. Typically, there are one or more hidden layers in an LSTM model. each hidden layer i, where $i = 1, 2, ..., L-1$, contains multiple LSTM cells. Each LSTM cell in the i^{th} layer takes as input the output from the $(i-1)^{th}$ layer and the previous hidden state h_{t-1}, and produces a new hidden state h_t and an output o_t. Mathematically, this can be represented as follows:

Input gate:

$$i_t = \sigma(\mathbf{W}_{ix}x_t + \mathbf{W}_{ih}h_{t-1} + b_i)$$

Forget gate:

$$f_t = \sigma(\mathbf{W}_{fx}x_t + \mathbf{W}_{fh}h_{t-1} + b_f)$$

Output gate:

$$o_t = \sigma(\mathbf{W}_{ox}x_t + \mathbf{W}_{oh}h_{t-1} + b_o)$$

Candidate memory cell:

$$\tilde{c}_t = tanh(\mathbf{W}_{cx}x_t + \mathbf{W}_{ch}h_{t-1} + b_c)$$

Current memory cell:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

Current hidden state:

$$h_t = o_t \odot tanh(c_t)$$

where:

- x_t is the input at time step t
- h_t is the hidden state at time step t
- c_t is the memory cell at time step t
- σ is the sigmoid activation function
- $\odot$ is the Hadamard product
- $\mathbf{W}$ are weight matrices

- **b** are bias vectors.

Each LSTM cell contains several components, including a memory cell, an input gate, a forget gate, and an output gate. The memory cell is responsible for storing information from previous time steps. The input gate is responsible for controlling the flow of information into the memory cell, and the forget gate is responsible for controlling the flow of information out of the memory cell. The output gate is responsible for controlling the flow of information out of the memory cell and passing it to the next layer. Specifically:

The forget gate regulates how much of the previous cell state c_{t-1}, and hence, how much of the prior data, should be forgotten. The forget gate computes a value between 0 and 1 that is then used to weight the prior cell state using the previous hidden state h_{t-1}, current input x_t, and previous hidden state h_{t-1}. Most of the previous cell state should be retained if the value is near to 1, whereas most of the previous cell state should be forgotten if the value is close to 0.

The input gate determines how much of the new candidate cell state $\tilde{c}_t$ should be added to the current cell state (c_t) by using the previous hidden state h_{t-1}, the current input x_t, and a value between 0 and 1.

The output gate determines how much of the current cell state c_t should be passed to the output h_t by using the previous hidden state h_{t-1}, current input x_t, and a number between 0 and 1.

The cell state is the internal memory of the LSTM cell. It is calculated by first computing a candidate cell state $\tilde{c}_t$, which is a conjunction of the current input x_t and prior hidden state h_{t-1}. The Hadamard product of the input gate i_t and the candidate cell state $\tilde{c}_t$ is added to the Hadamard product of the forget gate f_t and prior cell state c_{t-1}, and this yields the current cell state.

The hidden state is the output of the LSTM cell at time step t, which is passed to the next LSTM cell in the network. It is calculated by taking the Hadamard product of the output gate o_t and the Sigmoid of the cell state c_t.

The number of neurons in each layer is a hyperparameter that needs to be determined

through a process of experimentation and trial and error. The number of neurons in each layer is typically determined based on the size of the input data and the complexity of the problem. Typically, the number of neurons in the input layer is equal to the number of features in the input data, while the number of neurons in the hidden layers and the output layer can be determined through experimentation.

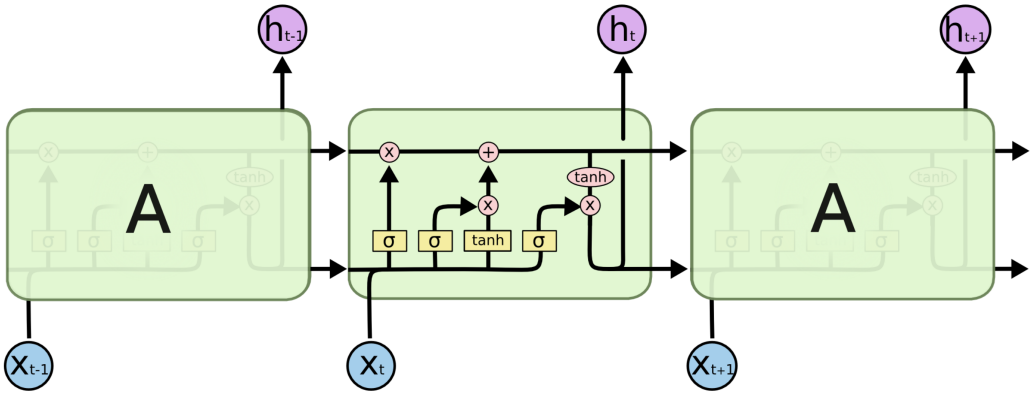

Figure 14: The repeating module in an LSTM contains four interacting layers.

In general, more neurons in the hidden layers can help to capture more complex patterns in the data, but it also increases the computational requirements and the risk of overfitting. Therefore, the number of neurons in each layer must be chosen judiciously, taking into account the size of the input data and the complexity of the problem. The number of layers in LSTM also is a hyperparameter and can be determined in a similar way.

5.5 Bidirectional LSTM

The 1997 publication "Bidirectional Recurrent Neural Networks" by Schuster and Paliwal [Schuster and Paliwal (1997)] is the first one to present the idea of Bidirectional LSTM (BiLSTM).

The BiLSTM architecture was suggested by the authors of the research as a solution to enhance voice recognition tasks. The essential concept underlying the BiLSTM is the employment of two LSTM networks, one moving ahead and the other moving backward. While the backward-facing LSTM reads the input sequence from right to left, the forward-facing LSTM reads it from left to right. The BiLSTM can make

predictions by combining the output of the two LSTMs, which allows it to include both past and future contexts.

By feeding past stock market data into the BiLSTM network as input, the same principle of BiLSTM may be used to perform stock price prediction tasks.

First, preprocessed historical stock market data is gathered, including stock prices, trade volume, and other financial indicators. After the stage of data preprocessing, the BiLSTM network then analyzes the preprocessed data in both forward and backward directions, accounting for both historical and recent data to extract meaningful features. While the backward-facing LSTM reads the input sequence from right to left, the forward-facing LSTM reads it from left to right.

The final hidden states of the forward-facing and backward-facing LSTMs are combined and utilized as the final output of the BiLSTM network after the LSTMs have digested the input data.

The output of the BiLSTM is then routed via a fully - connected layer, which generates a forecast of the stock's future price. Multiple neurons may be included in the fully connected layer, which aids in mapping the output from the LSTM to the final prediction.

5.6 Experiment

An experiment in stock price prediction using LSTM would involve collecting historical stock price data and any other relevant data, preprocessing the data, splitting the data into training, validation, and testing sets, designing the LSTM model by choosing the appropriate architecture and parameters, training the model on the training set, fine-tuning the model using the validation set and evaluating the model's performance on the testing set.

5.6.1 Glorot Uniform Initialization

Glorot uniform initialization, or Xavier uniform initialization [Glorot and Bengio (2010)], is a method for initializing the weights of a neural network, particularly deep neural networks such as LSTM (Long Short-Term Memory) networks. It's designed to

prevent the vanishing or exploding gradients problem by initializing the weights in such a way that the variance of the outputs of each layer is the same as the variance of its inputs, regardless of the number of incoming or outgoing connections.

Mathematically, the Glorot uniform initialization can be defined as:

$$W = \text{Uniform}(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}})$$

Where **W** is the weight matrix, n_{in} is the number of input units, and n_{out} is the number of output units.

This initialization method randomly samples the weights from a uniform distribution between -$\sqrt{\frac{6}{n_{in}+n_{out}}}$ and $\sqrt{\frac{6}{n_{in}+n_{out}}}$. By doing this, it ensures that the variance of the output of each neuron is the same as the variance of its inputs and this helps to prevent the vanishing or exploding gradients problem.

5.6.2 Adaptive Moment Estimation (Adam) Optimization

Adam (Adaptive Moment Estimation) [Kingma and Ba (2014)] is a popular optimization algorithm used to train deep learning models, including LSTM (Long Short-Term Memory) networks. It is an extension of the stochastic gradient descent (SGD) algorithm that uses a combination of gradient information and moving averages to adapt the learning rate on a per-parameter basis.

Adam works by maintaining an exponentially decaying average of past gradients and past squared gradients, which are used to estimate the first and second moments of the gradients. The algorithm then uses these estimates to adapt the learning rate for each parameter, with larger updates for parameters with lower first moment (mean) and smaller updates for parameters with higher second moment (variance).

The Adam algorithm has several advantages over traditional optimization algorithms such as SGD. One of the main advantages is that it requires less fine-tuning of the learning rate. Adam can also converge faster and be more robust to the choice of initial learning rate.

The Adam algorithm is typically used with a fixed learning rate and it is a combination

of two other optimization algorithms:

- Root Mean Squared Propagation (RMSProp)
- Momentum optimization

The algorithm has two main hyperparameters:

- learning rate α: controls the step size at which the optimizer makes updates to the model's parameters.
- The exponential decay rates $\beta_1$and β_2: These hyperparameters control the relative weighting of the historical gradients and historical squared gradients in the overall update.

Additionally, Adam is an optimization algorithm that uses gradient information and moving averages to adapt the learning rate on a per-parameter basis. The algorithm is defined by the following update equations for each parameter θ:

1. Initialize the parameter θ with random values, set the learning rate α, the decay rate for the first moment estimate β_1, and the decay rate for the second moment estimate β_2.
2. Initialize the first moment estimate m and the second moment estimate v to zero.
3. Compute the gradient of the loss function with respect to the parameter θ, denoted as g.
4. Update the first moment estimate m with the following equation:

$$m = \beta_1 m + (1 - \beta_1) g$$

5. Update the second moment estimate v with the following equation:

$$v = \beta_2 v + (1 - \beta_2) g^2$$

6. Compute the bias-corrected first-moment estimate:

$$\hat{m} = \frac{m}{1 - \beta_1^t}$$

Algorithm 2: Adam optimization

Result: Adam optimization

initialize: $m_0 = 0$ (Initialize 1st moment vector);

$v_0 = 0$ (Initialize 2nd moment vector);

$t = 0$ (Initialize timestep);

foreach *iteration* **do**

  $t = t + 1$ (Increment timestep);

  gradient = gradient(w_t) (Calculate gradient at current timestep);

  $m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * gradient$ (Update 1st moment estimate);

  $v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * gradient^2$ (Update 2nd moment estimate);

  $m_{\hat{t}} = \frac{m_t}{1-\beta_1^t}$ (Compute bias-corrected 1st moment estimate);

  $v_{\hat{t}} = \frac{v_t}{1-\beta_2^t}$ (Compute bias-corrected 2nd moment estimate);

  $w_t = w_t - \alpha * \frac{m_{\hat{t}}}{\sqrt{v_{\hat{t}}}+\varepsilon}$ (Update parameters);

end

Figure 15: Pseudocode of Adam optimization algorithm

7. Compute the bias-corrected second-moment estimate:

$$\hat{v} = \frac{v}{1 - \beta_2^t}$$

Some of the main advantages of using Adam include:

- **Adaptive learning rate:** Adam automatically adjusts the learning rate for each parameter, which helps to prevent oscillations and divergence during training.
- **Computationally efficient:** Adam requires less memory and is computationally more efficient than other optimization algorithms like gradient descent with momentum.
- **Robustness:** Adam is robust to the choice of the initial learning rate, which makes it easy to use.
- **Combining Momentum and RMSprop:** Adam combines the advantages of both Momentum and RMSprop, which makes it more powerful than either one alone.
- Generally it works well with most kinds of Neural Network architectures, with a good trade-off of both speed and quality.

The pseudocode of Adam Optimization algorithm could be expressed as follows:

5.6.3 Training Process

Training a BiLSTM to predict stock prices involves several steps:

Data collection: Collect historical stock market data, including the stock's opening and closing prices, trading volume, and other relevant indicators. In this work, we collect data from Yahoo Finance, a website that keeps records of multiple companies' stock prices. The data used is that of Apple's stock prices from 2018 to 2023.

Data preprocessing: Prepare the data for input into the BiLSTM model. This may include normalizing the data, removing any missing values, and creating a time series dataset by converting the data into sequences of a fixed length. We applied Z-score normalization method to our dataset via a normalization layer.

Model architecture: Design the LSTM model architecture, including the number of LSTM layers, the number of units in each layer, and any other hyperparameters such as the learning rate and batch size.

In this work, we implement a BiLSTM model that has an architecture as shown in Fig. 11. Specifically:

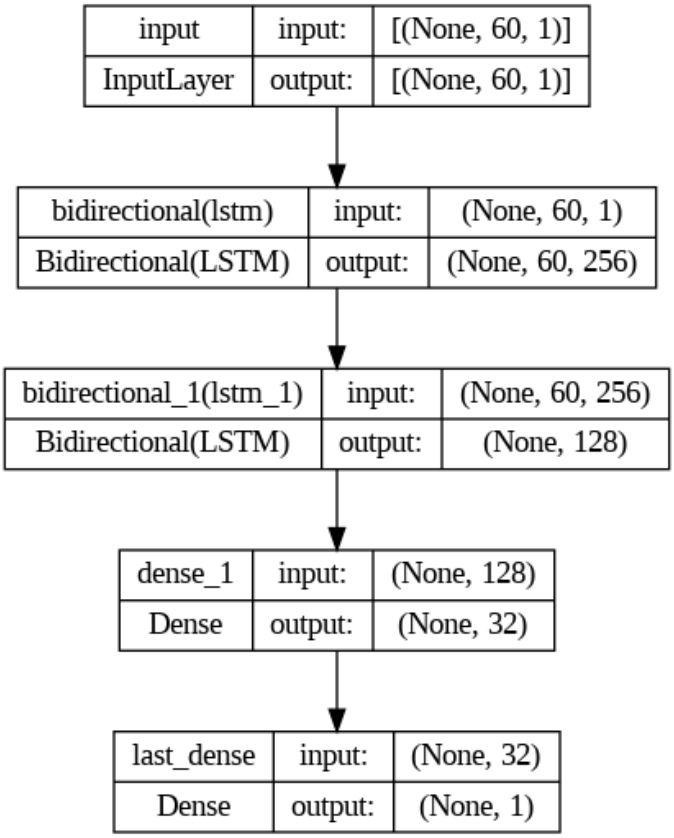

Figure 16: BiLSTM Architecture

The model architecture consists of the following layers:

I**nput Layer:** This layer defines the shape of the input data. The input is expected to be a 2-dimensional tensor with the shape `(number of samples, number of features, 1)`. The input layer is defined using the Input class in the Keras API and is named "input".

Bidirectional LSTM Layer 1: This is the first LSTM layer in the model and it is bidirectional, meaning it processes the input data in both forward and backward directions. The LSTM layer has 128 units, and the `return_sequences` parameter is set to True, which means that the LSTM layer will output a sequence rather than a single vector. The `kernel_initializer` is set to `tf.initializers.GlorotUniform` with a seed value equal to `RANDOM_SEED`. This ensures that the weights of the layer are initialized with the Glorot uniform initializer and the same values will be used every time the model is run.

Bidirectional LSTM Layer 2: This is the second LSTM layer in the model and it is also bidirectional. The LSTM layer has 64 units, and the `return_sequences` parameter is set to False, which means that the LSTM layer will output a single vector instead of a sequence. The kernel_initializer is set in the same way as the first LSTM layer.

Dense Layer 1: This is the first dense layer in the model and it has 32 units. The activation function used is the Tanh activation function, which is defined as $f(x) = \frac{e^x - e^x}{e^x + e^x}$. This layer is named `dense_1`.

Output Layer: This is the last layer in the model and it has a single unit, which means it will produce a scalar output. The layer is named `last_dense`.

The model is constructed using the Model class in the Keras API and the inputs and outputs are specified as the input layer and the output layer, respectively. The model is returned from the function and can be used for training and making predictions.

Overall, we designed a time-series prediction task, where the input data is a sequence of values and the goal is to predict a single output value. The use of bidirectional LSTMs, normalization, and fully connected layers helps to extract features from the input data and make the final prediction.

Model training: Train the LSTM model using the preprocessed data and the specified

architecture. The model will learn to predict stock prices based on the patterns in the historical data.

Model evaluation: Evaluate the trained model using a separate test dataset and metrics such as mean squared error or mean absolute error.

Model tuning: Based on the evaluation results, tune the model by adjusting the hyper-parameters, the architecture, or other aspects of the model to improve performance.

Model deployment: Once the model is trained and tuned, it can be deployed to a production environment to make predictions on new stock market data.

6 Model evaluation

Model evaluation is the process of reviewing a machine learning model's performance by testing it on fresh, previously unknown data. Splitting the data into training and testing sets, using cross-validation, evaluating performance using metrics such as Mean Absolute Error (MAE), accuracy, precision, recall, F1-score, and R-squared, creating a confusion matrix, and evaluating the performance of binary classifiers by ROC curve are all common techniques for evaluating models. Model evaluation is an iterative process, and changes to the model may be required until the desired level of performance is attained. In this research, we evaluate our model using six metrics, including MAE, MAPE, MPE, MSE, R_2, and RMSE.

6.1 Performance metrics

Performance metrics are used to evaluate the effectiveness of a model or system. They are used to compare different models or to track the performance of a single model over time. The main purpose of performance metrics is to provide a quantitative measure of how well a model is performing. They can be used to set goals for model development, track progress, and make adjustments as needed. Performance metrics also play an important role in model selection, as well as in the evaluation of model performance in real-world applications.

6.1.1 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) is a measure of the difference between two continuous variables, usually an observed and predicted value. It is calculated as the average of the absolute differences between the predicted values and the actual values. In other words, it is the average of the absolute error values.

The formula for MAE is:

$$MAE = \frac{\Sigma_{i=1}^{N} |y_i - \hat{y}_i|}{N}$$

where:

- N is the number of observations
- Y_i is the actual value for the i^{th} observation
- $\hat{Y}_i$ is the predicted value for the i^{th} observation

The absolute error values are used in the calculation to ensure that negative and positive differences are treated equally. The use of absolute error values also means that MAE is not affected by the direction of the error (i.e. whether the predicted value is greater or less than the actual value).

MAE is a commonly used metric for evaluating the performance of regression models, as it provides a simple and interpretable measure of the average error. However, it is sensitive to outliers and can be affected by the scale of the data.

Mean Absolute Percentage Error (MAPE) When the data has a consistent scale and the percentage of difference matters more than the absolute difference, the Mean Absolute Percentage Error (MAPE) measure is frequently used to assess the effectiveness of a model. The MAPE could be defined as:

$$MAE = \frac{1}{N} \frac{\Sigma_{i=1}^{N} |y_i - \hat{y}_i|}{y_i}$$

The average of the absolute percentage deviations between the anticipated and actual values is used to determine the MAPE. It is simple to comprehend because it is given

as a percentage. Higher MAPE values signify that the model is producing worse predictions, whereas lower MAPE values signify that the model is producing better predictions.

6.1.2 Mean Squared Error (MSE)

Mean Squared Error (MSE) is a commonly used loss function for regression problems in machine learning and deep learning. It measures the average of the squared differences between the predicted and actual values. The MSE is calculated by taking the average of the squared differences between the predicted and actual values for each data point in a dataset. Mathematically, it can be represented as:

$$MSE = \frac{\Sigma_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}$$

Where N is the number of samples in the dataset, $\hat{y}_i$ is the predicted values and y_i is the actual values. The MSE loss function is differentiable, which means that it can be used with optimization algorithms such as gradient descent to update the parameters of the model. The MSE is sensitive to outliers, as a single large error can significantly increase the overall MSE.

Mean Squared Error (MSE) has a number of advantages as a loss function. Some of these advantages include:

Simplicity: The MSE is a simple and easy-to-understand measure of how well a model is able to predict the target variable for a given set of input variables. It is calculated by taking the average of the squared differences between the predicted and actual values, which makes it straightforward to calculate and interpret.

Differentiability: The MSE is a differentiable loss function, which means that it can be used with optimization algorithms such as gradient descent to update the parameters of the model. The gradient of the MSE loss function with respect to the model's parameters can be calculated, and the parameters can be updated in the direction that decreases the MSE.

Penalty for large errors: The MSE is able to penalize large errors more heavily than

small errors. This is because the squared difference is used in the calculation of the MSE, which means that large errors will contribute more to the overall MSE than small errors. This can be useful for certain types of problems where large errors are particularly important to avoid.

Minimization: MSE can be minimized to find the optimal parameters of the model. Since the MSE is a scalar value, the optimization algorithms can be used to minimize it to find the best parameters that minimize the error.

Convexity: The MSE is a convex function, which means that it has only one global minimum. This makes it easier to find the optimal solution.

Widely used: MSE is widely used in many machine learning and deep learning algorithms. It is well-understood and has been well-studied, which means that there are many resources available to help practitioners understand and use it effectively.

Root Mean Squared Error

The performance of a regression model is also frequently assessed using the statistic known as Root Mean Squared Error (RMSE). It calculates the discrepancy between actual data and predictions. The RMSE formula could mathematically be derived as:

$$RMSE = \sqrt{\frac{\Sigma_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}}$$

The average of the squared discrepancies between the predicted and actual values is used to determine the RMSE. The square root is used to convert the RMSE's units to those of the true and anticipated values.

Because it is simple to comprehend and interpret, the RMSE approach is frequently used to assess a model's performance. Better forecasts are made by the model when the RMSE is lower, and worse predictions are made by the model when the RMSE is larger.

6.1.3 R^2 score

The percentage of the variation in the dependent variable—also known as the response or output variable—that can be predicted from the independent variables—also known as the predictor or input variables—in a regression model is measured by the R^2 (R-squared) statistic. It assesses how well the regression model fits the data.

The R^2 score goes from 0 to 1, with a value of 1 indicating that the regression line completely fits the data and that the independent variables are responsible for all of the variability in the dependent variable. A score of 0 on the other hand indicates that the regression line completely fails to match the data and that the independent variables offer no valuable insight into how the dependent variable varies. A higher R^2 value generally signifies that a greater percentage of the variability in the dependent variable can be explained by the independent variables, indicating a better fit of the model to the data. The R^2 score is calculated as follows:

$$R^2 = 1 - \frac{\text{sum of squared residuals}}{\text{total sum of squares}}$$

In conclusion, R^2 is a regularly used metric to assess the effectiveness of regression models and it offers a measurement of the model's goodness of fit to the data. An improved model fit to the data and a stronger capacity for the independent variables to account for the variability in the dependent variable are both indicated by higher R2 scores.

6.2 Experimental results

The loss on training data refers to the error or difference between the predicted output and the actual output of the model when it is trained on the training data. The training data is the set of data that is used to train the model. During training, the model is adjusted to minimize the loss on the training data. After training the model with 50 epochs using the architecture as described in section 5.4 with Adam optimization method and Tanh activation function, we yields the results as shown in Fig. 17 and Fig. 18

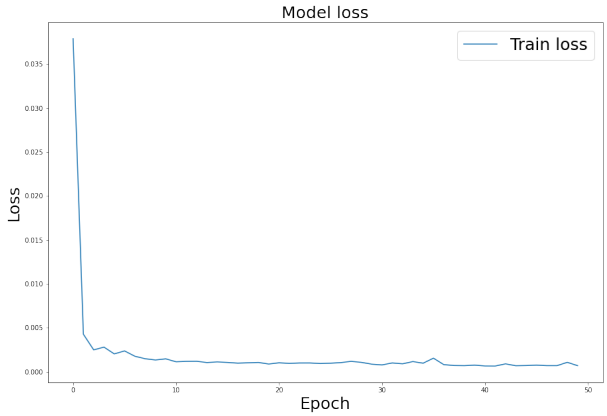

Figure 17: Loss on training data (left) and loss on test data (right) after 50 epochs

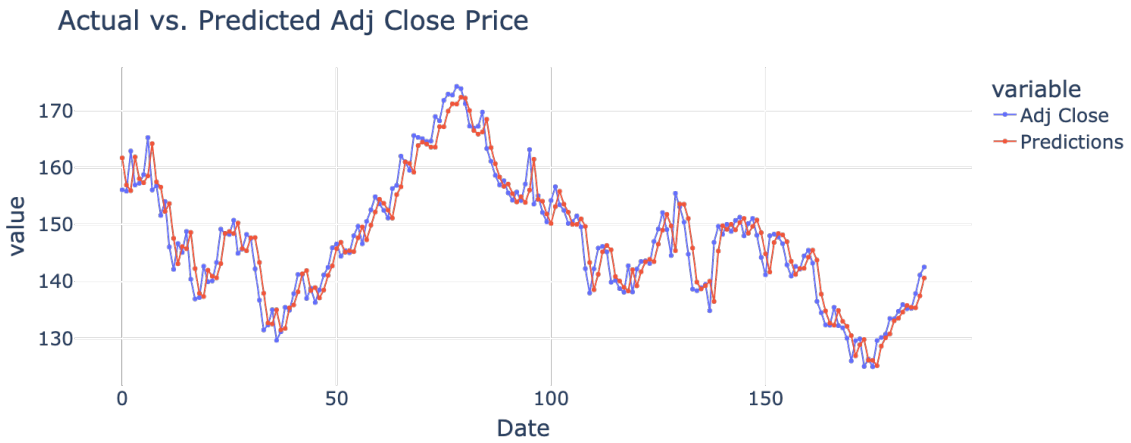

Figure 18: Actual adjusted closing stock prices (blue) vs. adjusted closing stock prices that are predicted by the model (red)

Using six evaluation metrics—root mean squared error (RMSE), mean percentage error (MPE) mean absolute percentage error (MAPE), mean squared error (MSE), mean absolute error (MAE) and R_2 score—as indicated in Table 2, we compare the performance of three optimization techniques, Adam, RMSprop, and SGD. The findings indicate that while RMSprop and SGD have inferior performance with larger values for all four measures, Adam has the highest performance with the lowest values for all four metrics.

Table 2: Comparison of different optimization methods

Model	RMSE	MPE	MAPE	MSE	MAE	R_2 score
Adam	**3.461**	**-0.0008**	**0.018**	**11.984**	**2.700**	**0.898**
RMSprop	3.627	-0.001	0.019	13.160	2.900	0.888
SGD	7.741	-0.025	0.042	59.927	6.162	0.493

In a similar vein, we evaluate the effectiveness of Tanh, ReLU, and Sigmoid as three activation functions. As shown in Table 3, Tanh performs the best and has the lowest

values for all six measures, whereas Sigmoid and ReLU perform worse and have greater values for all four metrics.

Table 3: Comparison of BiLSTM with different activation functions

Model	RMSE	MPE	MAPE	MSE	MAE	R_2 score
Tanh	**3.461**	**-0.0008**	**0.018**	**11.984**	**2.700**	**0.898**
ReLU	3.537	0.003	0.019	12.516	2.810	0.894
Sigmoid	3.563	-0.002	0.019	12.696	2.828	0.892

The tanh activation function, which transforms the values to a range between -1 and 1, is frequently applied in BiLSTM for the hidden state activations because it can assist control the size of the activations. This can aid in avoiding the vanishing gradient problem, which can happen when applying activation functions like the sigmoid, in which the gradients are so tiny that the model finds it challenging to learn.

The non-linear structure of the Tanh activation function also makes it possible for the BiLSTM to record intricate correlations between inputs and outputs. The BiLSTM may perform better by better modeling the underlying relationships and patterns in the data thanks to the non-linear nature of the Tanh function.

That being said, the choice of activation function depends on the specific problem and dataset, and it's possible that the ReLU or sigmoid activation functions could work better in other contexts. It's always important to experiment with different activation functions and compare the results to find the best configuration for a particular problem.

Finally, we compare BiLSTM to other models and as can be observed in Table 4, BiLSTM has the lowest error score regarding all of the four performance metrics, indicating the it is the best model.

Table 4: Comparison of BiLSTM and other models

Model	RMSE	MPE	MAPE	MSE	MAE	R_2 score
BiLSTM	**3.443**	**-0.001**	**0.018**	**11.854**	**2.666**	**0.899**
LSTM	3.623	0.002	0.019	13.127	2.924	0.889
Vanilla RNN	3.653	0.006	0.019	13.351	2.906	0.887
SVR	43.685	-0.260	0.515	1908.422	36.791	0.117
KNN	46.123	-0.233	0.517	2127.346	38.680	-0.047

Table 4 suggests that BiLSTM performs the best among the models listed, based on the evaluation metrics provided. The values of RMSE, MPE, MAPE, MSE, MAE and R_2 are lowest for BiLSTM, specifically 3.443, -0.001, 0.018, 11.854, 2.666 and 0.899

respectively, indicating that it has the best performance, LSTM is the second-best model, followed by Vanilla RNN, SVR, and KNN.

7 Discussion

In this section, we discuss one drawback of LSTM model and how Attention mechanism can be used to solve this problem through parallel computation. We then go on to discuss the problem of explainability in AI in the context of stock price prediction.

7.1 Sequential Computation

Sequential computation is a type of computation that processes data in a sequence, one element at a time. It is in contrast to parallel computation, which processes multiple elements at the same time. Sequential computation is often used in algorithms that process data that has an inherent sequential structure, such as time series data, natural language processing, and video processing. This might be disadvantageous because:

Training time: BiLSTM can be computationally expensive to train, especially for large datasets or long sequences. This can make it difficult to train and deploy models in a timely manner.

Difficulty in parallelizing: Because BiLSTM processes data one element at a time, it can be difficult to parallelize the computation, which can limit the speed at which the model can process the data. This is solved when Vaswani et al. (2017) proposed the use of Attention mechanisim, which allows the model to selectively focus on specific parts of the input sequence when making a prediction, rather than processing the entire sequence as a whole. This can be particularly useful for handling long sequences, where the dependencies between elements can be very distant in the sequence.

7.2 Explainability in Artificial Intelligence (AI)

The capacity to comprehend and analyze the justifications for judgments or predictions produced by an AI model is referred to as explainability in AI. Some drawbacks of explainability in AI when it comes to stock price prediction include:

- **Complexity:** Predicting stock prices is a difficult endeavor that requires studying a

lot of data and taking into consideration a variety of variables, including economic indicators, corporate performance, and market mood. It could be challenging to provide a clear and concise justification for the predictions provided by an AI model.

- **Lack of transparency:** Some AI models, such deep learning networks, may be regarded as "black boxes" because of how challenging it is to comprehend and interpret them. Because of this lack of transparency, it may be challenging to justify the model's predictions and to increase confidence in them.
- **Overfitting:** Some AI models could be overfit to the training data, which makes the model less generic to new data. The model may produce incorrect predictions as a result of overfitting, and it may be challenging to explain why.
- **Lack of domain expertise:** AI models could be able to evaluate a lot of data and find patterns that individuals might not be able to see. These models might not have the same degree of subject matter expertise as human analysts, and they might not completely comprehend the relevance or context of the patterns they find.

8 Conclusion

In conclusion, Bidirectional Long Short-Term Memory (BiLSTM) is a powerful type of Recurrent Neural Network (RNN) that is able to effectively model long-term dependencies by using gates that control the flow of information into and out of the cell state. Training BiLSTM to predict stock prices involves several steps, including data collection, data preprocessing, model architecture design, model training, model evaluation, model tuning, and model deployment. In this work we show that BiLSTM works best with Adam optimization algorithm in combination with Tanh activation function. Additionally, BiLSTM has the most outstanding performance compared to other traditional machine learning and deep learning methods. Our model is also capable of capturing complex and non-linear patterns of the Apple stock prices in the five-year period.

References

Ampomah, E. K., Qin, Z., and Nyame, G. (2020). Evaluation of tree-based ensemble machine learning models in predicting stock price direction of movement. *Information*, 11(6):332.

Ariyo, A. A., Adewumi, A. O., and Ayo, C. K. (2014). Stock price prediction using the arima model. In *2014 UKSim-AMSS 16th international conference on computer modelling and simulation*, pages 106–112. IEEE.

Baker, S. R., Bloom, N., Davis, S. J., Kost, K., Sammon, M., and Viratyosin, T. (2020). The unprecedented stock market reaction to covid-19. *The review of asset pricing studies*, 10(4):742–758.

Barsky, R. B. and De Long, J. B. (1993). Why does the stock market fluctuate? *The Quarterly Journal of Economics*, 108(2):291–311.

Beaudouin, V., Bloch, I., Bounie, D., Clémençon, S., d'Alché Buc, F., Eagan, J., Maxwell, W., Mozharovskyi, P., and Parekh, J. (2020). Flexible and context-specific ai explainability: A multidisciplinary approach. *arXiv preprint arXiv:2003.07703*.

Chang, Z., Zhang, Y., and Chen, W. (2018). Effective adam-optimized lstm neural network for electricity price forecasting. In *2018 IEEE 9th international conference on software engineering and service science (ICSESS)*, pages 245–248. IEEE.

Chang, Z., Zhang, Y., and Chen, W. (2019). Electricity price prediction based on hybrid model of adam optimized lstm neural network and wavelet transform. *Energy*, 187:115804.

Chen, K., Zhou, Y., and Dai, F. (2015). A lstm-based method for stock returns prediction: A case study of china stock market. In *2015 IEEE international conference on big data (big data)*, pages 2823–2824. IEEE.

Confalonieri, R., Coba, L., Wagner, B., and Besold, T. R. (2021). A historical perspective of explainable artificial intelligence. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(1):e1391.

DiPietro, R. and Hager, G. D. (2020). Deep learning: Rnns and lstm. In *Handbook of medical image computing and computer assisted intervention*, pages 503–519. Elsevier.

Fama, E. F. (1965). The behavior of stock-market prices. *The journal of Business*, 38(1):34–105.

Friedman, M. (1988). Money and the stock market. *Journal of Political Economy*, 96(2):221–245.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hondroyiannis, G. and Papapetrou, E. (2001). Macroeconomic influences on the stock market. *Journal of economics and finance*, 25(1):33–49.

Jing, N., Wu, Z., and Wang, H. (2021). A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction. *Expert Systems with Applications*, 178:115019.

Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier.

Khan, S. and Alghulaiakh, H. (2020). Arima model for accurate time series stocks forecasting. *International Journal of Advanced Computer Science and Applications*, 11(7).

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Leung, C. K.-S., MacKinnon, R. K., and Wang, Y. (2014). A machine learning approach for stock price prediction. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pages 274–277.

Li, Y. D., İşcan, T. B., and Xu, K. (2010). The impact of monetary policy shocks on stock prices: Evidence from canada and the united states. *Journal of international money and finance*, 29(5):876–896.

Livieris, I. E., Pintelas, E., and Pintelas, P. (2020). A cnn–lstm model for gold price time-series forecasting. *Neural computing and applications*, 32(23):17351–17360.

Long, J., Chen, Z., He, W., Wu, T., and Ren, J. (2020). An integrated framework of deep learning and knowledge graph for prediction of stock price trend: An application in chinese stock exchange market. *Applied Soft Computing*, 91:106205.

Long, W., Lu, Z., and Cui, L. (2019). Deep learning-based feature engineering for stock price movement prediction. *Knowledge-Based Systems*, 164:163–173.

Ludvigson, S. C., Steindel, C., et al. (1998). *How important is the stock market effect on consumption?*, volume 9821. Federal Reserve Bank of New York New York.

Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA.

Masoud, N. M. (2013). The impact of stock market performance upon economic growth. *International Journal of Economics and Financial Issues*, 3(4):788–798.

Mazur, M., Dang, M., and Vega, M. (2021). Covid-19 and the march 2020 stock market crash. evidence from s&p1500. *Finance research letters*, 38:101690.

Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., and Ranzato, M. (2014). Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*.

Mitchell, M. L. and Mulherin, J. H. (1994). The impact of public information on the stock market. *The Journal of Finance*, 49(3):923–950.

Morck, R., Shleifer, A., Vishny, R. W., Shapiro, M., and Poterba, J. M. (1990). The stock market and investment: is the market a sideshow? *Brookings papers on economic Activity*, 1990(2):157–215.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Icml*.

Nelson, D. M., Pereira, A. C., and De Oliveira, R. A. (2017). Stock market's price movement prediction with lstm neural networks. In *2017 International joint conference on neural networks (IJCNN)*, pages 1419–1426. Ieee.

Patel, J., Shah, S., Thakkar, P., and Kotecha, K. (2015). Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert systems with applications*, 42(1):259–268.

Qi, M. and Maddala, G. (1999). Economic factors and the stock market: a new perspective. *Journal of Forecasting*, 18(3):151–166.

Ribeiro, A. H., Tiels, K., Aguirre, L. A., and Schön, T. (2020). Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness. In *International Conference on Artificial Intelligence and Statistics*, pages 2370–2380. PMLR.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306.

Siami-Namini, S., Tavakoli, N., and Namin, A. S. (2018). A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pages 1394–1401. IEEE.

Sun, J., Xiao, K., Liu, C., Zhou, W., and Xiong, H. (2019). Exploiting intra-day patterns for market shock prediction: A machine learning approach. *Expert Systems with Applications*, 127:272–281.

Talathi, S. S. and Vartak, A. (2015). Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Zhang, J., He, T., Sra, S., and Jadbabaie, A. (2019). Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881*.