A

MINI PROJECT REPORT

on

STOCK PRICE PREDICTION USING LONG SHORT-TERM MEMORY NETWORKS

Submitted to the Faculty of Engineering and Technology

B.Tech. - VI Semester

By

Rohan Aditya Thirumala

B20CS161

Under the Guidance of

Dr. Syed Abdul Moeed

Assistant Professor

Department of Computer Science and Engineering

Kakatiya Institute of Technology & Science

(An Autonomous Institute under Kakatiya University)

Warangal – Telangana

2022-2023

NAAC - 'A' Grade Institute (CGPA : 3.21) ISO 9001:2015 NIRF - 2020 Rank Band : 201-250

KAKATIYA INSTITUTE OF TECHNOLOGY & SCIENCE
Opp : Yerragattu Gutta, Hasanparthy (Mandal), WARANGAL - 506 015, Telangana, INDIA.
काकतीय प्रैद्योगिकी एवं विज्ञान संस्थान, वरंगल - ५०६ ०१५ तेलंगाना, भारत
కాకతీయ సాంకేతిక విజ్ఞాన శాస్త్ర విద్యాలయం, వరంగల్ - ౫౦౬ ౦౧౫ తెలంగాణ, భారతదేశము
(An Autonomous Institute under Kakatiya University, Warangal)
(Approved by AICTE, New Delhi; Recognised by UGC under 2(f) & 12(B); Sponsored by EKASILA EDUCATION SOCIETY)
Estd-1980 KITSW
website: www.kitsw.ac.in E-mail: principal@kitsw.ac.in ✆ : +91 9392055211, +91 7382564888

C E R T I F I C A T E

This is to certify that **ROHAN ADITYA THIRUMALA** bearing Roll No. **B20CS161** of the VI Semester B.Tech. Computer Science and Engineering has satisfactorily completed the mini project report dissertation work entitled, **"STOCK PRICE PREDICTION USING LONG SHORT-TERM MEMORY NETWROKS",** in partial fulfillment of the requirements of B. Tech Degree during this academic year 2022-2023.

Supervisor
Dr. Syed Abdul Moeed
Assistant Professor

Coordinator
Sri. P. Sreenivas
Assistant Professor

Convenor
Sri. S. NAGA RAJU
Associate Professor

Head of the Department
Prof. P. NIRANJAN
Professor

ACKNOWLEDGEMENT

I extend my sincere and heartfelt thanks to our esteemed counselor/guide **Dr. Syed Abdul Moeed, Assistant Professor** for his exemplary guidance, monitoring and consistent encouragement throughout the course at crucial junctures and for showing us the right way.

It is a great privilege for me to here by deep sense of gratitude towards mini-project Coordinator **Sri. P. SREENIVAS, Assistant Professor** who guided and permitting me in successful completion of my work on time.

I am grateful to respected mini project convenor **Sri. S. NAGA RAJU, Associate Professor** for guiding and permitting me to utilize all the necessary facilities of the Institute.

I would like to extend thanks to **Prof. P. NIRANJAN, Professor & Head Department of CSE** for allowing me to use the facilities available.

I express my immense delight to **Prof. M. VEERA REDDY, Dean, Research & Development** for his kind cooperation and elderly suggestions through out of this Mini-Project report. I am very much thankful for issuing similarity report through Turnitin Anti Plagiarism Software.

I take this opportunity to express my sincere and heartfelt gratitude to honorable Principal **Prof. K. ASHOKA REDDY** of our institute for granting me permission to do this Mini-Project.

I am very happy in expressing my sincere thanks to faculty & staff, Department of Computer Science and Engineering, friends and family for the support and encouragement that they have given us during the Mini-Project.

ROHAN ADITYA

B20CS161

ABSTRACT

The job of predicting stock prices has proven difficult for financial analysts. Because they can manage vast volumes of data and recognize complicated patterns in the data, machine learning techniques have recently become more popular for stock prediction.

Stock prediction using Long Short-Term Memory (LSTM) is one of the popular topics in financial forecasting. LSTM is a type of recurrent neural network that has been shown to be effective in capturing long-term dependencies in sequential data, such as time series. LSTM model is trained on historical stock data and evaluation of trained model is done using root mean squared error and mean squared error. Then future stock prices are predicted and a prediction graph will be plotted.

CONTENTS

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

A stock of a company refers to the ownership in that company. It is also known as share or equity. A company generally issues stocks to public in order to raise funds that help in expansion or development of that company. In today's world, stock market became an important aspect in the field of economy.

Stock prices can be predicted using different machine learning algorithms like,

- Linear regression,
- Decision trees,
- Random forests,
- Support Vector Machines,
- Artificial Neural Networks and
- LSTM networks

1.2 NEED OF STOCK PRICE PREDICTION

- **Investment decision making:** Stock prediction models help users to take better decisions regarding investments and sales by providing insights into the future price trends.
- **Risk management:** As the model gets trained on historical data, it is capable of identifying and managing risks related to investments.
- **Market efficiency:** Problems in estimating stocks like over estimation or under estimation can be reduced by using machine learning models.
- **Financial analysis:** A better understanding and better conclusions can be drawn about a stock and about a company.

1.3 OBJECTIVES

- To develop a deep learning model that uses LSTM networks to predict stock prices for a given time period.
- To compare the performance of the LSTM model in predicting stock prices across different companies or industries.

1.4 METHODOLOGY

General methodology that is followed for stock price prediction using LSTM networks:

- **Data collection**

  Obtain historical stock price information as well as any other pertinent information, such as trade volume, economic indicators, and news stories about the stock or sector, from online financial data providers like Yahoo Finance and Alpha Vantage.

- **Data preparation**

  The data is preprocessed by being cleaned, normalized, and put into a format that will work for LSTM model training. To do this, the data may need to be cleaned up, scaled to a common range, and divided into training, validation, and test sets.

- **Feature engineering**

  To increase the accuracy of the LSTM model, add more features to it. These characteristics can be gleaned from various sources, such as sentiment analysis of news stories, or through technical analysis indicators like moving averages.

- **Model training**

  Using a deep learning framework like TensorFlow or Keras, create an LSTM model, then train it with the preprocessed data. To increase the model's accuracy, test out various network designs and hyperparameters.

- **Model evaluation**

  Utilize suitable measures, such as mean squared error (MSE), mean absolute error (MAE), and root mean squared error (RMSE), to assess the performance of the LSTM model. Compared to other established time series forecasting models like ARIMA or exponential smoothing, the model's performance is evaluated.

- **Trading strategy**

  Use the predictions of the LSTM model to guide your trading decisions, such as purchasing or selling stocks based on anticipated price changes. Analyze the strategy's profitability through paper trading or back testing.

- **Model visualization**

  Use interactive graphs and charts to visualize the LSTM model's performance and predictions in order to better comprehend the data patterns and the model's precision.

- **Model deployment**

  Use an API or web application to integrate the LSTM model and give users access to real-time stock price prediction.

CHAPTER 2

NEURAL NETWORKS

2.1 RECURRENT NEURAL NETWORKS

Recurrent Neural Networks are a special kind of artificial networks that are mainly used to handle sequential data like time series data. This feature separates Recurrent Neural Networks from other available machine learning algorithms in predicting stock prices. RNNs are intended to operate on sequences of input data by preserving an internal state or memory of prior inputs, in contrast to standard feedforward neural networks, which analyze input data in a single pass.

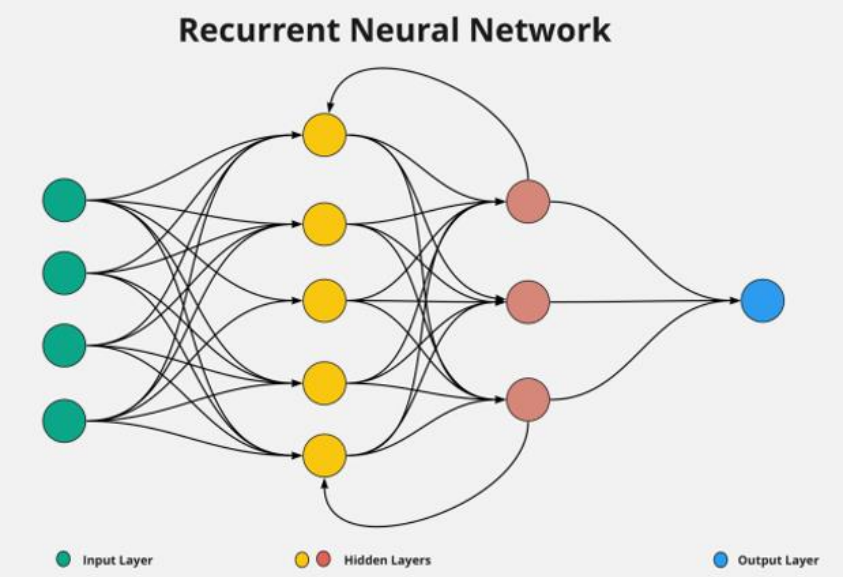

Fig.2.1 Recurrent Neural Networks

Drawbacks of Recurrent Neural Networks

- Vanishing and exploding gradients
- Memory limitations
- Difficulty with capturing long-term dependencies

2.2 LONG SHORT-TERM MEMORY NETWORKS

DEFINITION

Recurrent neural network (RNN) architectures called Long Short-Term Memory (LSTM) networks were created to solve the vanishing gradient issue that regular RNNs frequently experience.

The most well-known name for this special class of RNNs is "LSTMs," or long-short-term memory networks, which are able to identify long-term dependencies. Hochreiter & Schmidhuber (1997) first introduced them, and many authors refined and popularised them in subsequent works.1 They are currently in wide use and work amazingly well when applied to a variety of problems.

LSTMs are designed specifically to avoid the long-term reliance problem. They don't work hard to learn; instead, they naturally tend to retain information for long periods of time.

All recurrent neural networks take the form of a succession of repeating neural network modules. For instance, the recurring module in a typical RNN will consist of just one tanh layer.

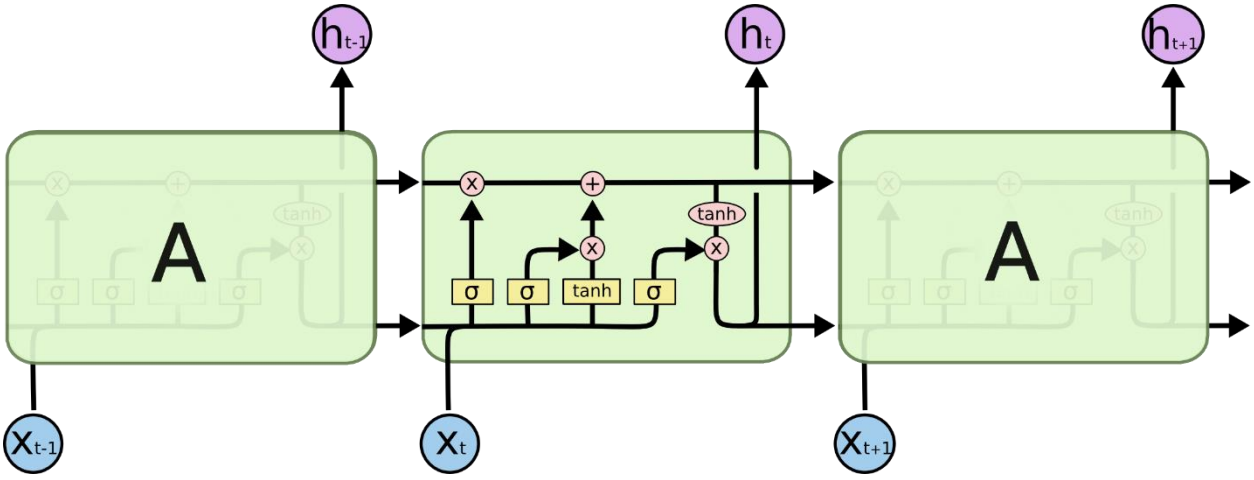

Fig.2.2.1 The repeating module in an LSTM contains four interacting layers

2.3 THE VANISHING GRADIENT PROBLEM

The problem

An issue that frequently arises during the training of artificial neural networks, particularly those with many layers (i.e., deep neural networks), is the vanishing gradient problem. During training, the gradient is passed through the layers backwards and becomes very small. Due to this, the network's early layers may learn very slowly, making it challenging to effectively train the network.

Why

For instance, the sigmoid function condenses a large input space between 0 and 1 into a small input region. As a result, even when the input changes dramatically, the sigmoid function's output will not change much. As a result, the derivative diminishes.

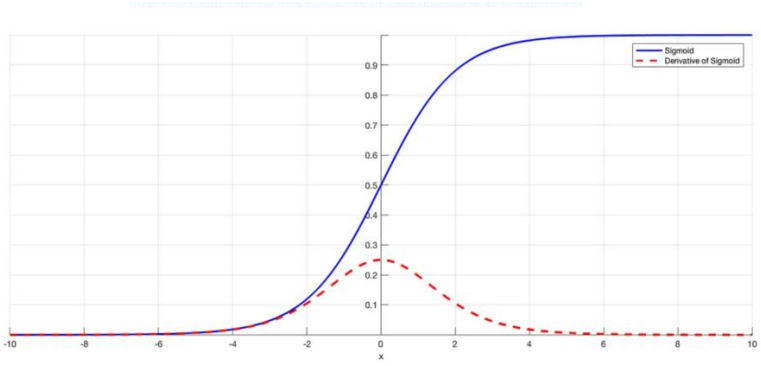

Fig.2.2.2 The sigmoid function and its derivative

Solution

Utilizing alternative activation functions, like ReLU, which doesn't result in a small derivative, is the simplest solution. This residual link bypasses activation functions, which "squashes" the derivatives, resulting in a greater block-wide derivative.

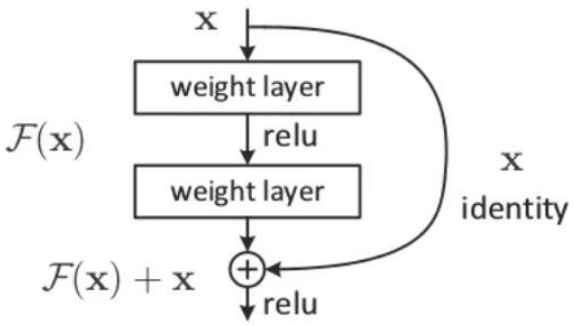

Fig.2.2.3 Solution for vanishing gradient problem

Advantages of LSTM over Recurrent Neural Networks

- **Ability to handle long-term dependencies**

  LSTMs are designed to address the issue by using a memory cell to store and update information over long periods of time
- **Resistance to vanishing gradient problem**

  Gating mechanism is used to control the flow of information through the network and migrate the vanishing gradient problem
- **Improved performance on complex tasks**

  LSTMs have been shown to outperform traditional RNNs on a range of complex tasks, including language modeling, speech recognition, and hand writing recognition.
- **Flexibility in handling input and output sequences**

  LSTMs are capable of handling input and output sequences of arbitrary length and produce variable-length output sequences, making them well-suited for a wide range of applications.

Disadvantages of LSTM over Recurrent Neural Networks

- **Complexity**

  Because LSTM networks are more complicated than conventional neural networks, it may be more challenging to comprehend, train, and optimize them. With additional parameters needed because to the many gates and memory units in LSTMs, overfitting and longer training times may result.

- **Overfitting**

  As previously indicated, LSTM networks' increased complexity can result in overfitting, which happens when the network learns to fit the training data too closely and struggles to generalize to new data.

- **Computational Cost**

  LSTM networks can be computationally expensive because to their complexity, especially when processing huge volumes of data. As a result, developing and deploying LSTM models on devices with low resources may be challenging.

- **Difficulty in training**

CHAPTER 3

ARCHITECTURE

LSTMs use a number of 'gates' to control how information in a sequence of data enters, is stored in, and leaves the network. Three gates make up an LSTM:

1. Forget Gate

2. Input Gate

3. Output Door

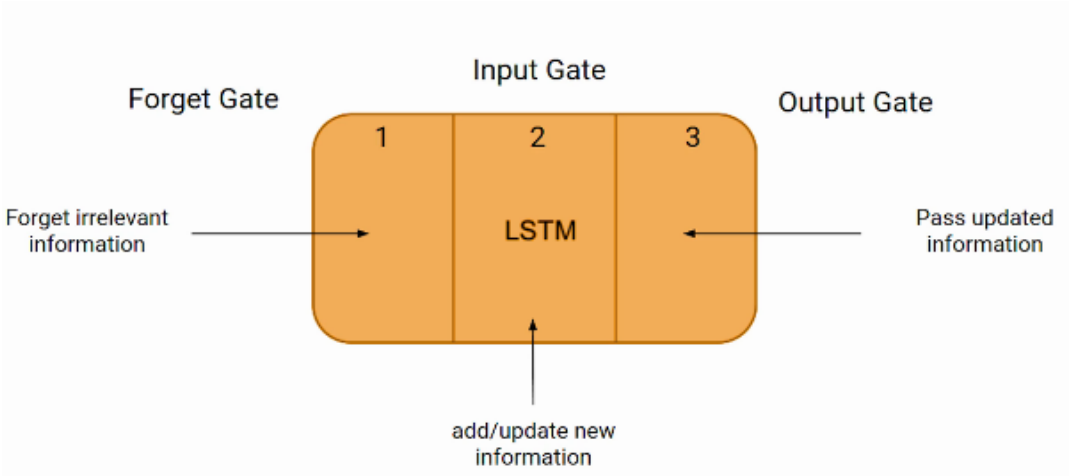

Fig.3.1 Architecture of LSTM

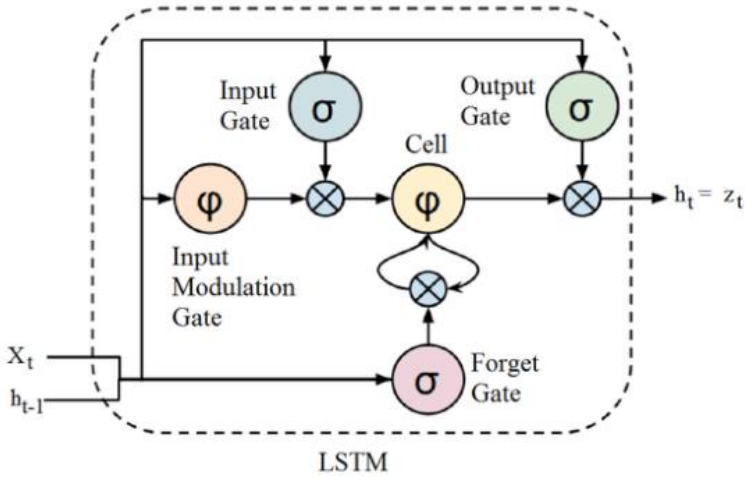

Fig.3.2 Gates in LSTM

3.1 INPUT GATE

This gate takes input from the previous hidden state and the current input and decides which information is worth adding to the memory cell. It produces an activation vector that ranges between 0 and 1, with 1 indicating that all the input should be added to the memory cell, and 0 indicating that no input should be added. The input gate is used to rate the significance of fresh data brought in by the input. The input gate's equation is shown below.

Input Gate:

- $i_t = \sigma (x_t * U_i + H_{t-1} * W_i)$

- X t: Input at the current timestamp t
- Ui: weight matrix of input
- Ht-1: A hidden state at the previous timestamp
- Wi: Weight matrix of input associated with hidden state

3.2 OUTPUT GATE

Which data from the cell state to be output as the hidden state is decided by the output gate. It returns a vector of values between 0 and 1 that denote the significance of each value in the cell state after receiving the current input and the prior concealed state as inputs

Output Gate:

- $o_t = \sigma (x_t * U_o + H_{t-1} * W_o)$

$$H_t = o_t * \tanh(C_t)$$

3.3 FORGET GATE

This gate decides which information should be removed from the memory cell. It takes the previous hidden state and the current input as input and produces an activation vector that ranges between 0 and 1, with 1 indicating that all the information should be retained and 0 indicating that all the information should be removed.

Forget Gate:

- $f_t = \sigma(x_t * U_f + H_{t-1} * W_f)$

$C_{t-1} * f_t = 0$ ...if $f_t = 0$ (forget everything)

$C_{t-1} * f_t = C_{t-1}$ ...if $f_t = 1$ (forget nothing)

- Xt: input to the current timestamp.
- Uf: weight associated with the input
- Ht-1: The hidden state of the previous timestamp
- Wf: It is the weight matrix associated with the hidden state

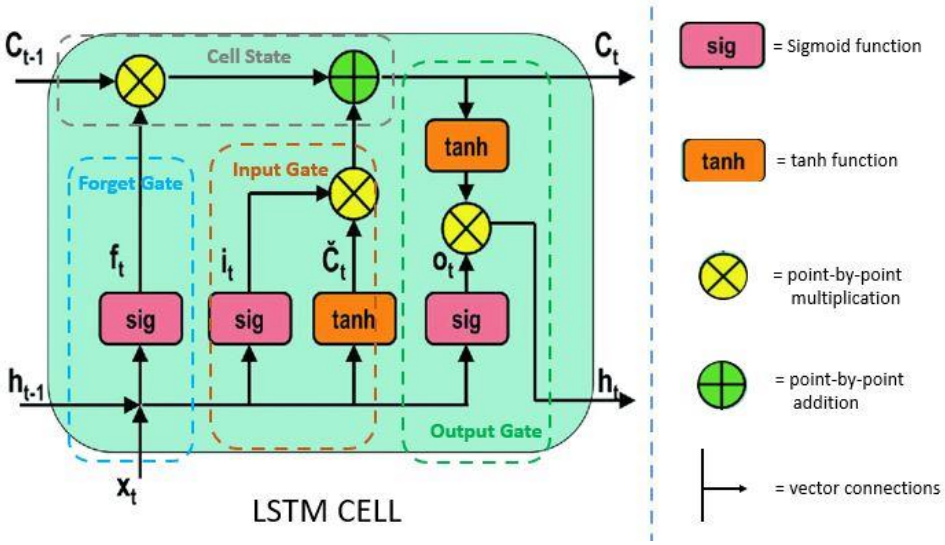

Fig.3.3 LSTM cell states

CHAPTER 4

IMPLEMENTATION

4.1 ALGORITHM

In order to generate an output, the LSTM method performs a number of mathematical operations on the input data and the internal state of the network, including matrix multiplications, element-wise operations, and activation functions.

4.2 METHODOLOGY

The steps you can take to create an LSTM network are as follows,

- **Define the problem**

  Identify the issue you're trying to address, such as anticipating the next word in a sentence or creating new text using the input that has been provided.

- **Gather and pre-process the data**

  Collect the data you need to train the LSTM network, then divide it into training, validation, and testing sets after cleaning, normalising, and pre-processing it.

- **Define the architecture**

  Choose the LSTM network's design, including the number of layers, neurons per layer, activation function, and gate types (input, output, and forget).

- **Define the loss function**

  In order to optimise its parameters during training, the LSTM network will employ the loss function that you define. This could be mean squared error for regression issues or cross-entropy for classification issues.

- **Train the network**

  Utilising an optimisation algorithm like stochastic gradient descent, train the LSTM network with the training set, and iteratively update the network weights and biases to reduce the loss function

- **Validate the network**

  Use the validation set to test the LSTM network's performance on data that wasn't utilised during training. If necessary, make network architecture and configuration changes to boost performance.

- **Test the network**

  To assess the LSTM network's final performance on unobserved data, test it using the testing set.

- **Iterate and improve**

  Repeat steps 3–8, modifying the network architecture and parameters as necessary, until the performance is acceptable.

4.3 SOFTWARE REQUIREMENT SPECIFICATIONS(SRS)

HARDWARE REQIUREMENTS:

PROCESSOR: Minimum Core i3 processor

RAM: 2GB(Min) or 8GB(Recommended)

Hard Disk Space: 500GB+

SOFTWARE REQUIREMENTS:

Programming Language: Python

Operating System: Windows 7 or later versions of Windows

4.3.1 TECHNOLOGY DESCRIPTION

PYTHON

A well-liked high-level programming language, Python, was initially introduced in 1991. It is well-liked by developers, data scientists, and researchers because of its readability, simplicity, and ease of use. Python supports a variety of programming paradigms, including imperative, functional, and object-oriented programming.

Python is an interpreted language, which means that the Python interpreter runs the code line by line. This facilitates speedy code writing as well as code testing and debugging.

The extensive Python standard library offers modules for everything from web development and data processing to scientific computing and machine learning. Additionally, Python has access to a wide range of third-party libraries, including NumPy, Pandas, and Scikit-learn, making it a potent tool for data analysis.

Additionally, Python has a sizable and vibrant developer community with a wealth of resources accessible to both novice and seasoned programmers. Due to its widespread use, it is now a top choice for creating artificial intelligence, scientific computing, and web applications.

Overall, Python is a popular and effective programming language that is used in many different fields and sectors.

PYTHON FEATURES

Python is a strong programming language with many features, some of which are as follows:

- Python is a straightforward, simple, and easy-to-learn syntax that enables programmers to write code fast and effectively.
- Python offers object-oriented programming, making it possible for programmers to create reusable and maintainable code.

- Large standard library: The Python language comes with a sizable standard library that contains modules for many different activities, including web development, data processing, and scientific computing.
- Cross-platform compatibility: Python is a cross-platform language, allowing code created on one platform to execute without any changes on another.
- Python uses dynamic typing, which enables programmers to alter the type of a variable while it is being used.
- Language that can be directly executed without the need for compilation is Python, which is an interpreted language.
- Python is a high-level language, which implies that it is more similar to human language and that it enables developers to concentrate on the issue at hand rather than the specifics of the machine language.
- Large community: Python has a sizable and vibrant developer community that actively contributes to the language's development and produces a wide range of libraries and frameworks.

PYTHON SYNTAX

Python's straightforward syntax makes it simple to learn and enables developers to write code quickly and effectively. Python uses several fundamental syntaxes, some of which are:

- Data types and variables: In Python, variables are made by giving names a value. Python supports a wide range of data types, including dictionaries, lists, tuples, strings, lists, and floating-point numbers.
- Code documentation and function explanation are done using comments. The # symbol is used to denote comments in Python, and the interpreter ignores them.
- Python offers a number of control structures, including functions, for- and while-loops, and if expressions. These structures give programmers the ability to regulate how a program is executed.
- Indentation: Unlike many other programming languages, Python uses indentation to define code blocks rather than curly brackets. The code is easier to read and less prone to errors because to this functionality.

- Python offers modules and packages, which are tools for organizing code and producing reusable parts.

- Python comes with built-in input and output methods like print() and input() that let programmers communicate with users and show information on the screen.

- Python's built-in exception handling system enables programmers to handle errors and exceptions politely and prevent program crashes.

PACKAGES REQUIRED

Numpy

NumPy is an essential Python tool for scientific computing. It offers support for large matrices and multidimensional arrays, as well as a sizable library of mathematical operations that may be performed on these arrays. In data science and machine learning applications where effective manipulation of huge datasets is necessary, NumPy is extensively employed.

Pandas

Pandas is a well-known Python data manipulation toolkit. It offers data structures like data frames and series for effectively managing and analyzing big datasets. Additionally, Pandas has tools for filtering, merging, and visualizing data.

Yfinance

A Python package called yfinance allows users to get financial information from Yahoo Finance. For downloading historical stock prices, financial statements, and other financial data, it offers a simple user interface. Applications in finance and investing, such portfolio management and quantitative analysis, frequently employ yfinance.

Datetime

A Python library for working with dates and times is the Date Time module. Along with utilities for modifying and displaying date and time data, it offers classes for expressing dates, times, and

time intervals. In financial applications, web development, and scientific computing, the Date Time module is frequently utilized.

Matplotlib

Matplotlib is a Python plotting package. It offers an extensive selection of visualization tools for producing 2D and 3D charts, histograms, scatter plots, and other kinds of visualizations. Applications for scientific computing and data science frequently utilize Matplotlib.

Scikit-learn

A machine learning library for Python is called Scikit-learn. Along with a variety of algorithms for classification, regression, clustering, and dimensionality reduction, it offers a wide range of tools for preprocessing data, choosing models, and evaluating them. Data science, machine learning, and artificial intelligence applications frequently employ Scikit-learn.

CHAPTER 5

EXPERIMENTATION AND RESULTS

5.1 EXPERIMENTAL WORK

Implementation of Long short-term memory network is done on the data set of 'Apple' company's stock data which is downloaded dynamically using yfinance.

5.2 IMPLEMENTATION OF LSTM NETWORKS

importing dependencies

```
import numpy as np
import pandas as pd
import yfinance as yf
import datetime
from datetime import date, timedelta
import matplotlib.pyplot as plt
import pandas_datareader as data
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential
```

Scaler

```
from sklearn.preprocessing import MinMaxScaler
```

Data Collection and processing

#downloading the csv data using yfinance

```
today = date.today()
d1 = today.strftime("%Y-%m-%d")
end_date = d1
d2 = date.today() - timedelta(days=1825)
d2 = d2.strftime("%Y-%m-%d")
start_date = d2

df= yf.download('AAPL',start=start_date,end=end_date,progress=False)
```

#Data Analysis

#printing first five rows of the dataset

df.head()

Date	Open	High	Low	Close	Adj Close	Volume
2018-04-30	40.532501	41.814999	40.459999	41.314999	39.386463	169709600
2018-05-01	41.602501	42.299999	41.317501	42.275002	40.301655	214277600
2018-05-02	43.807499	44.437500	43.450001	44.142502	42.081970	266157600
2018-05-03	43.970001	44.375000	43.610001	44.222500	42.158249	136272800
2018-05-04	44.562500	46.062500	44.542500	45.957500	43.812248	224805200

Fig.5.2.1. First five rows of dataset

#printing last five rows of the dataset

df.tail()

Date	Open	High	Low	Close	Adj Close	Volume
2023-04-21	165.050003	166.449997	164.490005	165.020004	165.020004	58311900
2023-04-24	165.000000	165.600006	163.889999	165.330002	165.330002	41949600
2023-04-25	165.190002	166.309998	163.729996	163.770004	163.770004	48714100
2023-04-26	163.059998	165.279999	162.800003	163.759995	163.759995	45498800
2023-04-27	165.190002	168.559998	165.190002	168.410004	168.410004	64728500

Fig.5.2.2. Last five rows of dataset

#printing number of rows and columns in the dataset

df.shape

```
In [9]: df.shape
Out[9]: (1258, 6)
```

Fig.5.2.3. Number of rows and columns in dataset

#getting some info about the data

df.info()

```
In [11]: df.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1258 entries, 2018-04-30 to 2023-04-27
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       1258 non-null   float64
 1   High       1258 non-null   float64
 2   Low        1258 non-null   float64
 3   Close      1258 non-null   float64
 4   Adj Close  1258 non-null   float64
 5   Volume     1258 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 68.8 KB
```

Fig.5.2.4. Info about data

#checking for missing values

df.isnull().sum()

```
In [13]: df.isnull().sum()

Out[13]: Open         0
         High         0
         Low          0
         Close        0
         Adj Close    0
         Volume       0
         dtype: int64
```

Fig.5.2.5. Checking missing values

#stastical measures about the data

```
df.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	1258.000000	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03
mean	104.932502	106.215425	103.735266	105.037428	103.739391	1.117948e+08
std	45.687836	46.268010	45.130734	45.724948	46.057204	5.457882e+07
min	35.994999	36.430000	35.500000	35.547501	34.257290	3.519590e+07
25%	54.741249	55.343124	54.204374	54.707501	52.855941	7.587110e+07
50%	118.625000	119.744999	116.445000	118.457500	116.773426	9.643985e+07
75%	146.597496	148.174999	145.257496	146.755005	145.701271	1.302981e+08
max	182.630005	182.940002	179.119995	182.009995	180.683853	4.265100e+08

Fig.5.2.6. Description about data

#reseting the index of the Pandas Data Frame df to a range index, and makes the previous index into a new column in the Data Frame

```
df=df.reset_index()
df.head()
```

	index	Date	Open	High	Low	Close	Adj Close	Volume
0	0	2018-04-30	40.532501	41.814999	40.459999	41.314999	39.386456	169709600
1	1	2018-05-01	41.602501	42.299999	41.317501	42.275002	40.301662	214277600
2	2	2018-05-02	43.807499	44.437500	43.450001	44.142502	42.081974	266157600
3	3	2018-05-03	43.970001	44.375000	43.610001	44.222500	42.158245	136272800
4	4	2018-05-04	44.562500	46.062500	44.542500	45.957500	43.812260	224805200

Fig.5.2.7. Resetting index of data frame

#calculating the 100-day moving average and 200-day moving average of the "Close" column

```
ma100 = df.Close.rolling(100).mean()
ma200=df.Close.rolling(200).mean()

ma100
ma200
```

```
0             NaN
1             NaN
2             NaN
3             NaN
4             NaN
          ...
1253    150.38955
1254    150.48445
1255    150.56810
1256    150.66255
1257    150.77530
Name: Close, Length: 1258, dtype: float64
```

Fig.5.2.8. Output of moving averages

#plotting moving averages and against closing price

```
plt.figure(figsize=(12,6))
plt.plot(df.Close)
plt.plot(ma100,'r')
plt.plot(ma200,'g')
plt.show()
```

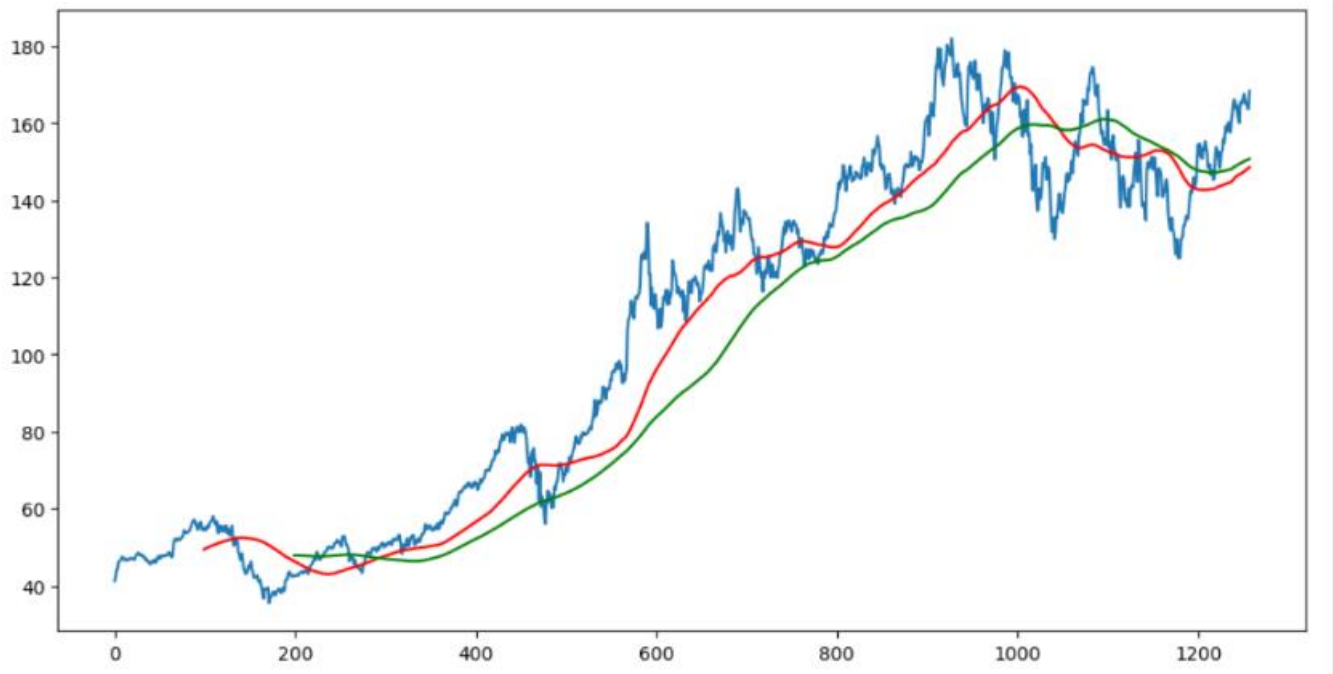

Fig.5.2.9. Plot of moving average and closing prices against closing price

#splitting the data into training and testing data

```
data_training=pd.DataFrame(df['Close'][0:int(len(df)*0.70)])
data_testing=pd.DataFrame(df['Close'][int(len(df)*0.70):int(len(df))])
print(data_training.shape)
print(data_testing.shape)
print(data_testing)
```

#scaling the training data using the MinMaxScaler from the scikit-learn library within a range of (0,1)

```
scaler = MinMaxScaler(feature_range=(0,1))
data_training_array=scaler.fit_transform(data_training)
print(data_training_array)
```

Fig.5.2.10. Scaling the training data

#initializing two empty list to store the training data for a machine learning model.

```
In [27]: x_train=[]
         y_train=[]

         for i in range(100,data_training_array.shape[0]):
             x_train.append(data_training_array[i-100:i])
             y_train.append(data_training_array[i,0])

         x_train, y_train = np.array(x_train),np.array(y_train)
         x_train
         y_train
```

```
Out[27]: array([0.16063726, 0.15574633, 0.16220565, 0.16509482, 0.16144209,
                0.17079058, 0.17242091, 0.17555769, 0.17972635, 0.18548404,
                0.1770642 , 0.16942854, 0.16835545, 0.17475286, 0.15306353,
                0.14912189, 0.16492973, 0.15512722, 0.16501226, 0.16303114,
                0.15236189, 0.1591514 , 0.16191673, 0.1662092 , 0.15044265,
                0.16016262, 0.15293972, 0.14456116, 0.14674866, 0.15822275,
                0.16515673, 0.134738  , 0.12258289, 0.12708174, 0.1398353 ,
                0.13682234, 0.12852632, 0.10727035, 0.1032668 , 0.092061  ,
                0.10157459, 0.1059496 , 0.09012113, 0.0717956 , 0.07138287,
                0.06211691, 0.0669253 , 0.06614112, 0.07996781, 0.07709928,
                0.07509751, 0.08797491, 0.07119714, 0.06713168, 0.05427493,
                0.05656562, 0.05456384, 0.05553378, 0.05935158, 0.04806322,
                0.04488516, 0.04928081, 0.03859091, 0.03021235, 0.01762386,
                0.0095755 , 0.030914  , 0.02880903, 0.02897413, 0.03209031,
                0.03246176, 0.        , 0.01252655, 0.01184553, 0.01766514,
                0.02294817, 0.02395939, 0.0208432 , 0.01611738, 0.02245291,
```

Fig.5.2.11. Building training sets

#building sequential model

```
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential

model=Sequential()

model.add(LSTM(units=50,activation='relu',return_sequences=True,input_shape=(x_train.shape[1],1)))
model.add(Dropout(0.2))

model.add(LSTM(units=60,activation='relu',return_sequences=True))
model.add(Dropout(0.3))

model.add(LSTM(units=80,activation='relu',return_sequences=True))
model.add(Dropout(0.4))

model.add(LSTM(units=120,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=1))
```

Fig.5.2.12. Sequential Model

#printing model summary

```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 dropout (Dropout)           (None, 100, 50)           0

 lstm_1 (LSTM)               (None, 100, 60)           26640

 dropout_1 (Dropout)         (None, 100, 60)           0

 lstm_2 (LSTM)               (None, 100, 80)           45120

 dropout_2 (Dropout)         (None, 100, 80)           0

 lstm_3 (LSTM)               (None, 120)               96480

 dropout_3 (Dropout)         (None, 120)               0

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 178,761
Trainable params: 178,761
Non-trainable params: 0
_________________________________________________________________
None
```

Fig.5.2.13. Model Summary

#compiling the model

```
model.compile(optimizer='adam',loss='mean_squared_error')
model.fit(x_train,y_train,epochs=50)
model.save('my_model.h5')
```

#preparing the input data for the machine learning model

```
print(data_testing.head())
print(data_training.tail(100))

past_100_days=data_training.tail(100)
final_df=past_100_days.append(data_testing,ignore_index=True)
print(final_df.head())

input_data=scaler.fit_transform(final_df)
print(input_data.shape)
```

two empty lists called "x_test" and "y_test" are created to store the input and output data for the testing set

```
x_test=[]
y_test=[]

for i in range(100,input_data.shape[0]):
      x_test.append(input_data[i-100:i])
      y_test.append(input_data[i,0])

x_test, y_test = np.array(x_test), np.array(y_test)
#print(x_test.shape)
#print(y_test.shape)
```

#The input_data array's rows are then iterated through using a for loop, beginning #with the 100th row (because the model's window size is 100). The input_data #array is divided into windows of 100 consecutive days of data for each iteration, #beginning with the current row (i-100) and terminating with the row before it (i-1). The "x_test" list is expanded to include this window of data.

#Additionally, the "y_test" list, which represents the corresponding output value #for the input data window, is supplemented with the stock's closing price for the #current day (input_data[i,0]).

#predicting value of y for input x_test

```
y_predicted=model.predict(x_test)
#print(y_predicted.shape)
```

#performing inverse transformation to get back original values

```
#print(scaler.scale_)
scale_factor=1/(scaler.scale_)
y_predicted=y_predicted*scale_factor
y_test=y_test*scale_factor
```

#creating a plot to visualize the predicted closing prices of the stock compared to the actual closing prices

```
plt.figure(figsize=(12,6))
plt.plot(y_test,'b',label='Original Price')
plt.plot(y_predicted,'r', label='Predicted Price')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```

#ouput of previous snippet

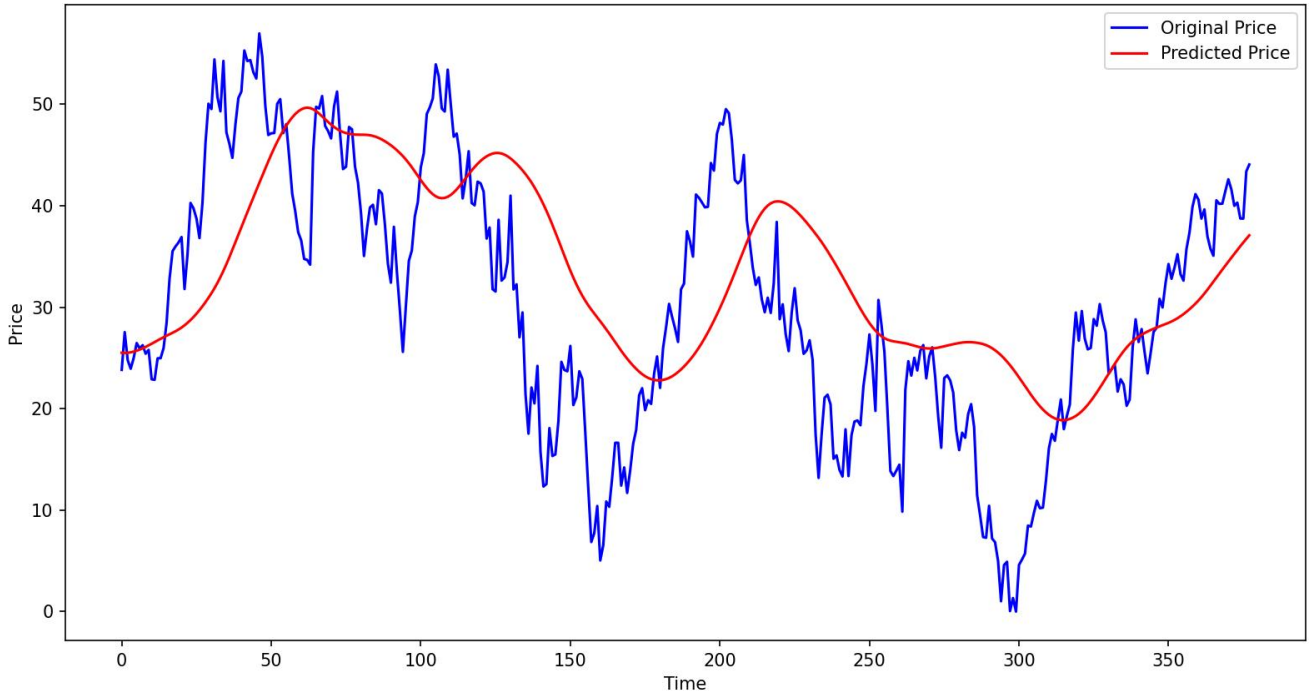

Fig.5.2.14. Plot of Original prices and predicted prices

5.3 RESULTS

The predicted stock prices of ‘Apple’ company are plotted with year on X-axis and Stock price on Y-axis

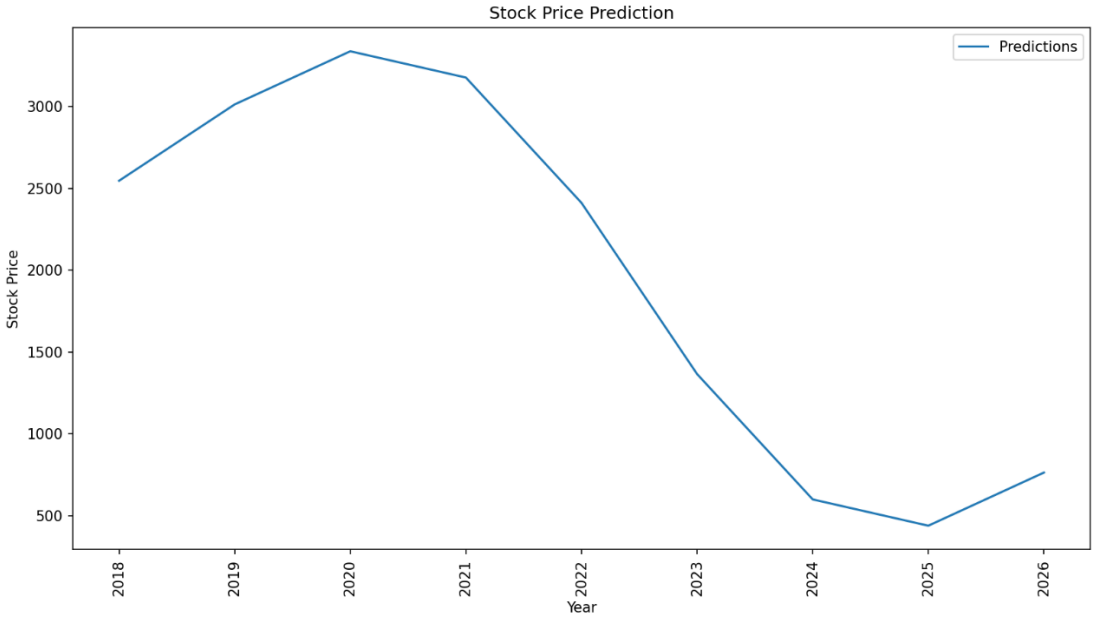

Fig.5.2.15. Plot of predicted stock prices

CHAPTER 6

CONCLUSIONS AND FUTURE SCOPE

6.1 CONCLUSIONS

LSSTM (Long Short-Term Memory Networks) have demonstrated great promise in stock price prediction, to sum up. LSSTMs are a subset of recurrent neural networks that are suitable for predicting stock prices because they can learn and remember long-term dependencies in time-series data.

LSSTMs can use historical data to identify patterns and trends that can be used to forecast future prices. It is crucial to remember that the stock market is incredibly unpredictable and is influenced by a variety of factors, including political developments, economic indicators, and world news. Therefore, even with the use of LSSTMs, predicting stock prices accurately is still difficult.

6.2 FUTURE SCOPE

Despite these difficulties, stock price prediction accuracy using LSSTMs has significantly outperformed that of conventional statistical models. LSSTMs may benefit from more research and development to produce more reliable stock price prediction models and forecasts with higher forecasting accuracy.

REFERENCES

- Pasala Sandhya, Raswitha Bandi, & D. Dakshayani Himabindu. (2022, April 13). *Stock price prediction using recurrent neural network and LSTM*. IEEE Xplore.

  https://ieeexplore.ieee.org/abstract/document/9753764

- Nasirtafreshi I. (2022, May). *Forecasting cryptocurrency prices using Recurrent Neural Network and Long Short-term Memory*. sciencedirect.

  https://www.sciencedirect.com/science/article/abs/pii/S0169023X22000234

- Mohammad Diqi. (2022, November). StockTM: Accurate stock price prediction model using LSTM. International Journal of Informatics and Computation.

  https://ijicom.respati.ac.id/index.php/ijicom/article/view/50

Turnitin Originality Report

Processed on: 30-Apr-2023 15:48 CST
ID: 2079617953
Word Count: 3297
Submitted: 1

MINOR By b20cs161

Similarity Index	Similarity by Source	
12%	Internet Sources:	4%
	Publications:	4%
	Student Papers:	8%

1% match (student papers from 16-Sep-2022)
Submitted to University of Greenwich on 2022-09-16

1% match (student papers from 15-Mar-2023)
Submitted to Liverpool John Moores University on 2023-03-15

1% match (student papers from 12-Apr-2023)
Submitted to St Xaviers University Kolkata on 2023-04-12

1% match (student papers from 31-Jul-2022)
Submitted to The Robert Gordon University on 2022-07-31

1% match (student papers from 16-May-2022)
Submitted to Kakatiya Institute of Technology and Science on 2022-05-16

1% match (student papers from 14-Oct-2021)
Submitted to University of Technology, Sydney on 2021-10-14

1% match (student papers from 19-Apr-2023)
Submitted to Trafford College Group on 2023-04-19

1% match (student papers from 23-Apr-2023)
Submitted to Barnet and Southgate College on 2023-04-23

1% match (Internet from 16-Jan-2023)
https://computingconf.com/2017/technical-papers.php

< 1% match (student papers from 26-Apr-2023)
Submitted to Liverpool John Moores University on 2023-04-26

< 1% match (Internet from 23-Nov-2021)
https://www.mdpi.com/2673-7426/1/3/11/htm

< 1% match ("Computer Networks and Inventive Communication Technologies", Springer Science and Business Media LLC, 2023)
"Computer Networks and Inventive Communication Technologies", Springer Science and Business Media LLC, 2023

< 1% match (Sepp Hochreiter, Jürgen Schmidhuber. "Long Short-Term Memory", Neural Computation, 1997)
Sepp Hochreiter, Jürgen Schmidhuber. "Long Short-Term Memory", Neural Computation, 1997

< 1% match (student papers from 14-Mar-2023)
Submitted to Tower Hamlets College on 2023-03-14

< 1% match (student papers from 31-Dec-2022)
Submitted to University of Birmingham on 2022-12-31

< 1% match (student papers from 02-Sep-2019)
Submitted to University of Warwick on 2019-09-02

< 1% match (Internet from 11-Apr-2023)
https://www.ijraset.com/best-journal/college-nirf-rank-predictor-using-ml

< 1% match (Yuanhang Su, C.-C. Jay Kuo. "On extended long short-term memory and dependent bidirectional recurrent neural network", Neurocomputing, 2019)
Yuanhang Su, C.-C. Jay Kuo. "On extended long short-term memory and dependent bidirectional recurrent neural network", Neurocomputing, 2019

< 1% match (Internet from 05-May-2022)
https://skewfabricator.com/yiem/gated-recurrent-unit-python

< 1% match (Internet from 14-Aug-2020)
https://amazingcrypto.com/bitcoin-unfazed-by-fud-as-it-breaks-through-6000/?share=facebook

< 1% match (Internet from 24-Dec-2022)
https://archive.org/stream/MastersThesisOfP.SureshKumar/M_Tech_Thesis_P_Suresh_Kumar_djvu.tx

< 1% match (Charuku, Bharat. "Machine Learning-Based Model Predictive Control with Koopman Linearization", Iowa State University, 2023)
Charuku, Bharat. "Machine Learning-Based Model Predictive Control with Koopman Linearization", Iowa State University, 2023

< 1% match (Gyula Dorgo, Janos Abonyi. "Learning and predicting operation strategies by sequence mining and deep learning", Computers & Chemical Engineering, 2019)
Gyula Dorgo, Janos Abonyi. "Learning and predicting operation strategies by sequence mining and deep learning", Computers & Chemical Engineering, 2019

< 1% match (Internet from 16-Apr-2023)
https://www.researchsquare.com/article/rs-777010/v1.pdf%3Fc%3D1632252273000

< 1% match ("Intelligent Computing", Springer Science and Business Media LLC, 2021)
"Intelligent Computing", Springer Science and Business Media LLC, 2021

< 1% match (Duc Huu Dat Nguyen, Loc Phuoc Tran, Vu Nguyen. "Chapter 15 Predicting Stock Prices Using Dynamic LSTM Models", Springer Science and Business Media LLC, 2019)
Duc Huu Dat Nguyen, Loc Phuoc Tran, Vu Nguyen. "Chapter 15 Predicting Stock Prices Using Dynamic LSTM Models", Springer Science and Business Media LLC, 2019

< 1% match (Sarnıç, Taha Eren. "Presumption of Complex Appendicitis in Children and Predicting With Machine Learning Algorithms", TED University (Turkey), 2023)
Sarnıç, Taha Eren. "Presumption of Complex Appendicitis in Children and Predicting With Machine Learning Algorithms", TED University (Turkey), 2023

CHAPTER 1 INTRODUCTION 1.1 INTRODUCTION A stock of a company refers to the ownership in that company. It is also known as share or equity. A company generally issues stocks to public in order to raise funds that help in expansion or development of that company. In today's world, stock market became an important aspect in the field of economy. Stock prices can be predicted using different machine learning algorithms like, ‣ Linear regression, ‣ Decision trees, ‣ Random forests, ‣ Support Vector Machines, ‣ Artificial Neural Networks and ‣ LSTM networks 1.2 NEED OF STOCK PRICE PREDICTION ‣ Investment decision making: Stock prediction models help users to take better decisions regarding investments and sales by providing insights into the future price trends. ‣ Risk management: As the model gets trained on historical data, it is capable of identifying and managing risks related to investments. ‣ Market efficiency: Problems in estimating stocks like over estimation or under estimation can be reduced by using machine learning models. ‣ Financial analysis: A better understanding and better conclusions can be drawn about a stock and about a company. 1.3 OBJECTIVES ‣ To develop a deep learning model that uses LSTM networks to predict stock prices for a given time period. ‣ To compare the performance of the LSTM model in predicting stock prices across different companies or industries. 1.4 METHODOLOGY General methodology that is followed for stock price prediction using LSTM networks: ‣ Data collection Obtain historical stock price information as well as any other pertinent information, such as trade volume, economic indicators, and news stories about the stock or sector, from online financial

data providers like Yahoo Finance and Alpha Vantage. ‣ Data preparation The data is preprocessed by being cleaned, normalized, and put into a format that will work for LSTM model training. To do this, the data may need to be cleaned up, scaled to a common range, and divided into training, validation, and test sets. ‣ Feature engineering To increase the accuracy of the LSTM model, add more features to it. These characteristics can be gleaned from various sources, such as sentiment analysis of news stories, or through technical analysis indicators like moving averages. ‣ Model training Using a deep learning framework like TensorFlow or Keras, create an LSTM model, then train it with the preprocessed data. To increase the model's accuracy, test out various network designs and hyperparameters. ‣ Model evaluation Utilize suitable measures, such as mean squared error (MSE), mean absolute error (MAE), and root mean squared error (RMSE), to assess the performance of the LSTM model. Compared to other established time series forecasting models like ARIMA or exponential smoothing, the model's performance is evaluated. ‣ Trading strategy Use the predictions of the LSTM model to guide your trading decisions, such as purchasing or selling stocks based on anticipated price changes. Analyze the strategy's profitability through paper trading or back testing. ‣ Model visualization Use interactive graphs and charts to visualize the LSTM model's performance and predictions in order to better comprehend the data patterns and the model's precision. ‣ Model deployment Use an API or web application to integrate the LSTM model and give users access to real-time stock price prediction. CHAPTER 2 NEURAL NETWORKS 2.1 RECURRENT NEURAL NETWORKS Recurrent Neural Networks are a special kind of artificial networks that are mainly used to handle sequential data like time series data. This feature separates Recurrent Neural Networks from other available machine learning algorithms in predicting stock prices. RNNs are intended to operate on sequences of input data by preserving an internal state or memory of prior inputs, in contrast to standard feedforward neural networks, which analyze input data in a single pass. Fig. 2.1 Recurrent Neural Networks Drawbacks of Recurrent Neural Networks ‣ Vanishing and exploding gradients ‣ Memory limitations ‣ Difficulty with capturing long-term dependencies 2.2 LONG SHORT-TERM MEMORY NETWORKS DEFINITION Recurrent neural network (RNN) architectures called Long Short-Term Memory (LSTM) networks were created to solve the vanishing gradient issue that regular RNNs frequently experience. The most well-known name for this special class of RNNs is "LSTMs," or long-short-term memory networks, which are able to identify long-term dependencies. Hochreiter & Schmidhuber (1997) first introduced them, and many authors refined and popularised them in subsequent works.1 They are currently in wide use and work amazingly well when applied to a variety of problems. LSTMs are designed specifically to avoid the long-term reliance problem. They don't work hard to learn; instead, they naturally tend to retain information for long periods of time. All recurrent neural networks take the form of a succession of repeating neural network modules. For instance, the recurring module in a typical RNN will consist of just one tanh layer. Fig.2.2.1 The repeating module in an LSTM contains four interacting layers 2.3 THE VANISHING GRADIENT PROBLEM The problem An issue that frequently arises during the training of artificial neural networks, particularly those with many layers (i.e., deep neural networks), is the vanishing gradient problem. During training, the gradient is passed through the layers backwards and becomes very small. Due to this, the network's early layers may learn very slowly, making it challenging to effectively train the network. Why For instance, the sigmoid function condenses a large input space between 0 and 1 into a small input region. As a result, even when the input changes dramatically, the sigmoid function's output will not change much. As a result, the derivative diminishes. Fig.2.2.2 The sigmoid function and its derivative Solution Utilizing alternative activation functions, like ReLU, which doesn't result in a small derivative, is the simplest solution. This residual link bypasses activation functions, which "squashes" the derivatives, resulting in a greater block-wide derivative. Fig.2.2.3 The sigmoid function and its derivative Advantages of LSTM over Recurrent Neural Networks ‣ Ability to handle long-term dependencies LSTMs are designed to address the issue by using a memory cell to store and update information over long periods of time ‣ Resistance to vanishing gradient problem Gating mechanism is used to control the flow of information through the network and migrate the vanishing gradient problem ‣ Improved performance on complex tasks LSTMs have been shown to outperform traditional RNNs on a range of complex tasks, including language modeling, speech recognition, and hand writing recognition. ‣ Flexibility in handling input and output sequences LSTMs are capable of handling input and output sequences of arbitrary length and produce variable-length output sequences, making them well-suited for a wide range of applications. Disadvantages of LSTM over Recurrent Neural Networks ‣ Complexity Because LSTM networks are more complicated than conventional neural networks, it may be more challenging to comprehend, train, and optimize them. With additional parameters needed because to the many gates and memory units in LSTMs, overfitting and longer training times may

result. ‣ Overfitting As previously indicated, LSTM networks' increased complexity can result in overfitting, which happens when the network learns to fit the training data too closely and struggles to generalize to new data. ‣ Computational Cost LSTM networks can be computationally expensive because to their complexity, especially when processing huge volumes of data. As a result, developing and deploying LSTM models on devices with low resources may be challenging. ‣ Difficulty in training CHAPTER 3 ARCHITECTURE LSTMs use a number of 'gates' to control how information in a sequence of data enters, is stored in, and leaves the network. Three gates make up an LSTM: 1. Forget Gate 2. Input Gate 3. Output Door Fig.3.1 Architecture of LSTM Fig.3.2 Gates in LSTM 3.1 INPUT GATE This gate takes input from the previous hidden state and the current input and decides which information is worth adding to the memory cell. It produces an activation vector that ranges between 0 and 1, with 1 indicating that all the input should be added to the memory cell, and 0 indicating that no input should be added. The input gate is used to rate the significance of fresh data brought in by the input. The input gate's equation is shown below. 3.2 OUTPUT GATE Which data from the cell state to be output as the hidden state is decided by the output gate. It returns a vector of values between 0 and 1 that denote the significance of each value in the cell state after receiving the current input and the prior concealed state as inputs 3.3 FORGET GATE This gate decides which information should be removed from the memory cell. It takes the previous hidden state and the current input as input and produces an activation vector that ranges between 0 and 1, with 1 indicating that all the information should be retained and 0 indicating that all the information should be removed. Fig.3.3 LSTM cell states CHAPTER 4 IMPLEMENTATION 4.1 ALGORITHM In order to generate an output, the LSTM method performs a number of mathematical operations on the input data and the internal state of the network, including matrix multiplications, element- wise operations, and activation functions. 4.2 METHODOLOGY The steps you can take to create an LSTM network are as follows, ‣ Define the problem Identify the issue you're trying to address, such as anticipating the next word in a sentence or creating new text using the input that has been provided. ‣ Gather and pre-process the data Collect the data you need to train the LSTM network, then divide it into training, validation, and testing sets after cleaning, normalising, and pre-processing it. ‣ Define the architecture Choose the LSTM network's design, including the number of layers, neurons per layer, activation function, and gate types (input, output, and forget). ‣ Define the loss function In order to optimise its parameters during training, the LSTM network will employ the loss function that you define. This could be mean squared error for regression issues or cross- entropy for classification issues. ‣ Train the network Utilising an optimisation algorithm like stochastic gradient descent, train the LSTM network with the training set, and iteratively update the network weights and biases to reduce the loss function ‣ Validate the network Use the validation set to test the LSTM network's performance on data that wasn't utilised during training. If necessary, make network architecture and configuration changes to boost performance. ‣ Test the network To assess the LSTM network's final performance on unobserved data, test it using the testing set. ‣ Iterate and improve Repeat steps 3–8, modifying the network architecture and parameters as necessary, until the performance is acceptable. 4.3 SOFTWARE REQUIREMENT SPECIFICATIONS(SRS) HARDWARE REQIUREMENTS: PROCESSOR: Minimum Core i3 processor RAM: 2GB(Min) or 8GB(Recommended) Hard Disk Space: 500GB+ SOFTWARE REQUIREMENTS: Programming Language: Python Operating System: Windows 7 or later versions of Windows 4.3.1 TECHNOLOGY DESCRIPTION PYTHON A well-liked high-level programming language, Python, was initially introduced in 1991. It is well- liked by developers, data scientists, and researchers because of its readability, simplicity, and ease of use. Python supports a variety of programming paradigms, including imperative, functional, and object-oriented programming. Python is an interpreted language, which means that the Python interpreter runs the code line by line. This facilitates speedy code writing as well as code testing and debugging. The extensive Python standard library offers modules for everything from web development and data processing to scientific computing and machine learning. Additionally, Python has access to a wide range of third-party libraries, including NumPy, Pandas, and Scikit-learn, making it a potent tool for data analysis. Additionally, Python has a sizable and vibrant developer community with a wealth of resources accessible to both novice and seasoned programmers. Due to its widespread use, it is now a top choice for creating artificial intelligence, scientific computing, and web applications. Overall, Python is a popular and effective programming language that is used in many different fields and sectors. PYTHON FEATURES Python is a strong programming language with many features, some of which are as follows: ‣ Python is a straightforward, simple, and easy-to-learn syntax that enables programmers to write code fast and effectively. ‣ Python offers object-oriented programming, making it possible for programmers to create reusable and maintainable code. ‣ Large standard library: The Python language comes with a

sizable standard library that contains modules for many different activities, including web development, data processing, and scientific computing. ‣ Cross-platform compatibility: Python is a cross-platform language, allowing code created on one platform to execute without any changes on another. ‣ Python uses dynamic typing, which enables programmers to alter the type of a variable while it is being used. ‣ Language that can be directly executed without the need for compilation is Python, which is an interpreted language. ‣ Python is a high-level language, which implies that it is more similar to human language and that it enables developers to concentrate on the issue at hand rather than the specifics of the machine language. ‣ Large community: Python has a sizable and vibrant developer community that actively contributes to the language's development and produces a wide range of libraries and frameworks. PYTHON SYNTAX Python's straightforward syntax makes it simple to learn and enables developers to write code quickly and effectively. Python uses several fundamental syntaxes, some of which are: ‣ Data types and variables: In Python, variables are made by giving names a value. Python supports a wide range of data types, including dictionaries, lists, tuples, strings, lists, and floating-point numbers. ‣ Code documentation and function explanation are done using comments. The # symbol is used to denote comments in Python, and the interpreter ignores them. ‣ Python offers a number of control structures, including functions, for- and while-loops, and if expressions. These structures give programmers the ability to regulate how a program is executed. ‣ Indentation: Unlike many other programming languages, Python uses indentation to define code blocks rather than curly brackets. The code is easier to read and less prone to errors because to this functionality. 15 ‣ Python offers modules and packages, which are tools for organizing code and producing reusable parts. ‣ Python comes with built-in input and output methods like print() and input() that let programmers communicate with users and show information on the screen. ‣ Python's built-in exception handling system enables programmers to handle errors and exceptions politely and prevent program crashes. PACKAGES REQUIRED Numpy NumPy is an essential Python tool for scientific computing. It offers support for large matrices and multidimensional arrays, as well as a sizable library of mathematical operations that may be performed on these arrays. In data science and machine learning applications where effective manipulation of huge datasets is necessary, NumPy is extensively employed. Pandas Pandas is a well-known Python data manipulation toolkit. It offers data structures like data frames and series for effectively managing and analyzing big datasets. Additionally, Pandas has tools for filtering, merging, and visualizing data. Yfinance A Python package called yfinance allows users to get financial information from Yahoo Finance. For downloading historical stock prices, financial statements, and other financial data, it offers a simple user interface. Applications in finance and investing, such portfolio management and quantitative analysis, frequently employ yfinance. Datetime A Python library for working with dates and times is the Date Time module. Along with utilities for modifying and displaying date and time data, it offers classes for expressing dates, times, and time intervals. In financial applications, web development, and scientific computing, the Date Time module is frequently utilized. Matplotlib Matplotlib is a Python plotting package. It offers an extensive selection of visualization tools for producing 2D and 3D charts, histograms, scatter plots, and other kinds of visualizations. Applications for scientific computing and data science frequently utilize Matplotlib. Scikit-learn A machine learning library for Python is called Scikit-learn. Along with a variety of algorithms for classification, regression, clustering, and dimensionality reduction, it offers a wide range of tools for preprocessing data, choosing models, and evaluating them. Data science, machine learning, and artificial intelligence applications frequently employ Scikit-learn. CHAPTER 5 EXPERIMENTATION AND RESULTS 5.1 EXPERIMENTAL WORK Implementation of Long short-term memory network is done on the data set of 'Apple' company's stock data which is downloaded dynamically using yfinance. 5.2 IMPLEMENTATION OF LSTM NETWORKS # importing dependencies # Scaler # Data Collection and processing #downloading the csv data using yfinance #Data Analysis #printing first five rows of the dataset df.head() Fig.5.2.1. First five rows of dataset #printing last five rows of the dataset df.tail() Fig.5.2.2. Last five rows of dataset #printing number of rows and columns in the dataset df.shape Fig.5.2.3. Number of rows and columns in dataset 19 #getting some info about the data df.info() Fig.5.2.4. Info about data #checking for missing values df.isnull().sum() Fig.5.2.5. Checking missing values #stastical measures about the data Fig.5.2.6. Description about data #reseting the index of the Pandas Data Frame df to a range index, and makes the previous index into a new column in the Data Frame Fig.5.2.7. Resetting index of data frame #calculating the 100-day moving average and 200-day moving average of the "Close" column Fig.5.2.8. Output of moving averages #plotting moving averages and against closing price Fig.5.2.9. Plot of moving average and closing prices against closing price #splitting the data into training and testing data #scaling the training

data using the MinMaxScaler from the scikit-learn library within a range of (0,1) Fig.5.2.10. Scaling the training data #initializing two empty list to store the training data for a machine learning model. Fig.5.2.11. Building training sets #building sequential model Fig.5.2.12. Sequential Model #printing model summary Fig.5.2.13. Model Summary #compiling the model #preparing the input data for the machine learning model # two empty lists called "x_test" and "y_test" are created to store the input and output data for the testing set #The input_data array's rows are then iterated through using a for loop, beginning #with the 100th row (because the model's window size is 100). The input_data #array is divided into windows of 100 consecutive days of data for each iteration, #beginning with the current row (i-100) and terminating with the row before it (i-1). The "x_test" list is expanded to include this window of data. #Additionally, the "y_test" list, which represents the corresponding output value #for the input data window, is supplemented with the stock's closing price for the #current day (input_data[i,0]). #predicting value of y for input x_test #performing inverse transformation to get back original values #creating a plot to visualize the predicted closing prices of the stock compared to the actual closing prices #ouput of previous snippet Fig.5.2.14. Plot of Original prices and predicted prices 5.3 RESULTS The predicted stock prices of 'Apple' company are plotted with year on X-axis and Stock price on Y-axis Fig.5.2.15. Plot of predicted stock prices CHAPTER 6 CONCLUSIONS AND FUTURE SCOPE 6.1 CONCLUSIONS LSSTM (Long Short-Term Memory Networks) have demonstrated great promise in stock price prediction, to sum up. LSSTMs are a subset of recurrent neural networks that are suitable for predicting stock prices because they can learn and remember long-term dependencies in time- series data. LSSTMs can use historical data to identify patterns and trends that can be used to forecast future prices. It is crucial to remember that the stock market is incredibly unpredictable and is influenced by a variety of factors, including political developments, economic indicators, and world news. Therefore, even with the use of LSSTMs, predicting stock prices accurately is still difficult. 6.2 FUTURE SCOPE Despite these difficulties, stock price prediction accuracy using LSSTMs has significantly outperformed that of conventional statistical models. LSSTMs may benefit from more research and development to produce more reliable stock price prediction models and forecasts with higher forecasting accuracy. REFERENCES ? Pasala Sandhya, Raswitha Bandi, & D. Dakshayani Himabindu. (2022, April 13). Stock price prediction using recurrent neural network and LSTM. IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/9753764 ? Nasirtafreshi I. (2022, May). Forecasting cryptocurrency prices using Recurrent Neural Network and Long Short-term Memory. sciencedirect. https://www.sciencedirect.com/science/article/abs/pii/S0169023X22000234 ? Mohammad Diqi. (2022, November). StockTM: Accurate stock price prediction model using LSTM. International Journal of Informatics and Computation. https://ijicom.respati.ac.id/index.php/ijicom/article/view/50 1 2 3 4 5 6 7 8 9 10 11 12 13 14 16 17 18 20 21 22 23 24 25 26 27 28 29 30