MACHINE LEARNING

(Stock Price Prediction)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

B.LAXMAN REDDY

221710302011

Under the Guidance of

Department Of Computer Science Engineering GITAM School of Technology

GITAM (Deemed to be University)

Hyderabad-502329

July 2020

i

DECLARATION

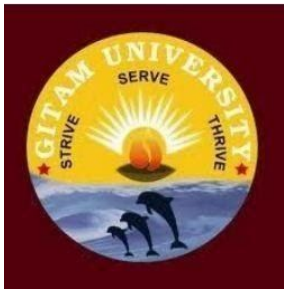
I submit this industrial training work entitled **"STOCK PRICE PREDICTION**" to GITAM (Deemed To Be University), Hyderabad in partial
fulfillment of the requirements for the award of the degree of "**Bachelor of Technology**" in "**Computer Science Engineering**". I declare that it was carried out independently by me under the guidance of,Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD B.LAXMAN REDDY

Date :22th July,2020 221710302011

ii



GITAM (DEEMED TO BE UNIVERSITY)
Hyderabad-502329, India
Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **"STOCK PRICE PREDICTION"** is being submitted by **B.LAXMAN REDDY(221710302011)** in partial fulfillment of the requirement for the award oF **Bachelor of Technology in Computer Science Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2019-20

It is faithful record work carried out by him at the Computer Science Engineering, GITAM University Hyderabad Campus under my guidance and supervision.

Dr.S.Phani Kumar

Assistant Professor

Department of CSE

Professor and HOD

Department of CSE

iv

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this Internship.

I would like to thank respected **Dr. N. Siva Prasad,** Pro Vice Chancellor, GITAM Hyderabad and **N. Seetaramaiah,** Principal, GITAM Hyderabad

I would like to thank respected **Dr.S.Phani Kumar ,** Head of the Department of **Computer Science Engineering** for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present an internship report. It helped me a lot to realize what we study for.

I would like to thank the respected faculties who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

B.LAXMAN REDDY

221710302011

v

ABSTRACT

As financial institutions begin to embrace artificial intelligence,machine learning is increasingly utilized to help make trading decisions.Although there is an abundance of stock data for machine learning models to train on, a high noise to signal ratio and the multitude of factors that affect stock prices are among the several reasons to predict the market difficulties. At the same time, these models don't need to reach high levels of accuracy because even 60% accuracy can deliver solid returns. One method for predicting stock prices is using a long short-term memory neural network (LSTM) for times series forecasting.

LSTMs are an improved version of recurrent neural networks (RNNs). RNNs are analogous to human learning. When humans think, we don't start our thinking from scratch each second. For example, in the sentence "Bob plays basketball", we know that Bob is the person who plays basketball because we retain information about past words while reading sentences. Similarly, RNNs are networks with loops in them, which allow them to use past information before arriving at a final output. However, RNNs can only connect recent previous information and cannot connect information as the time gap grows. This is where LSTMs come into play; LSTMs are a type of RNN that remember information over long periods of time, making them better suited for predicting stock prices.

Table of Contents:

ix

LIST OF FIGURES:

CHAPTER 1 MACHINE LEARNING

INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence(AI).

IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and "more items to consider" and "get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of .big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that

have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

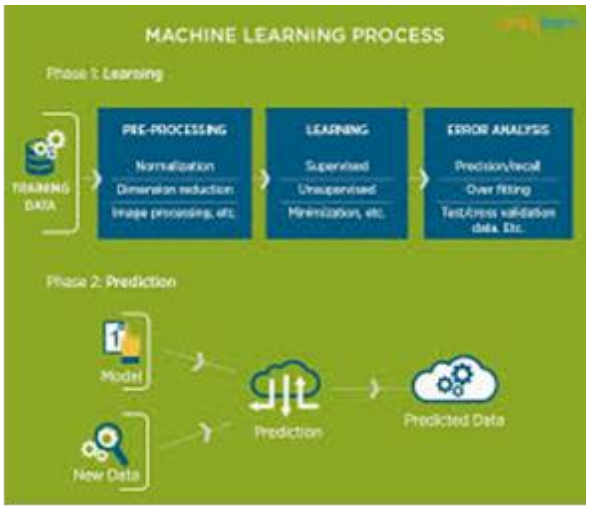


Figure 1 : The Process Flow

USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

Supervised Learning :

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based

on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

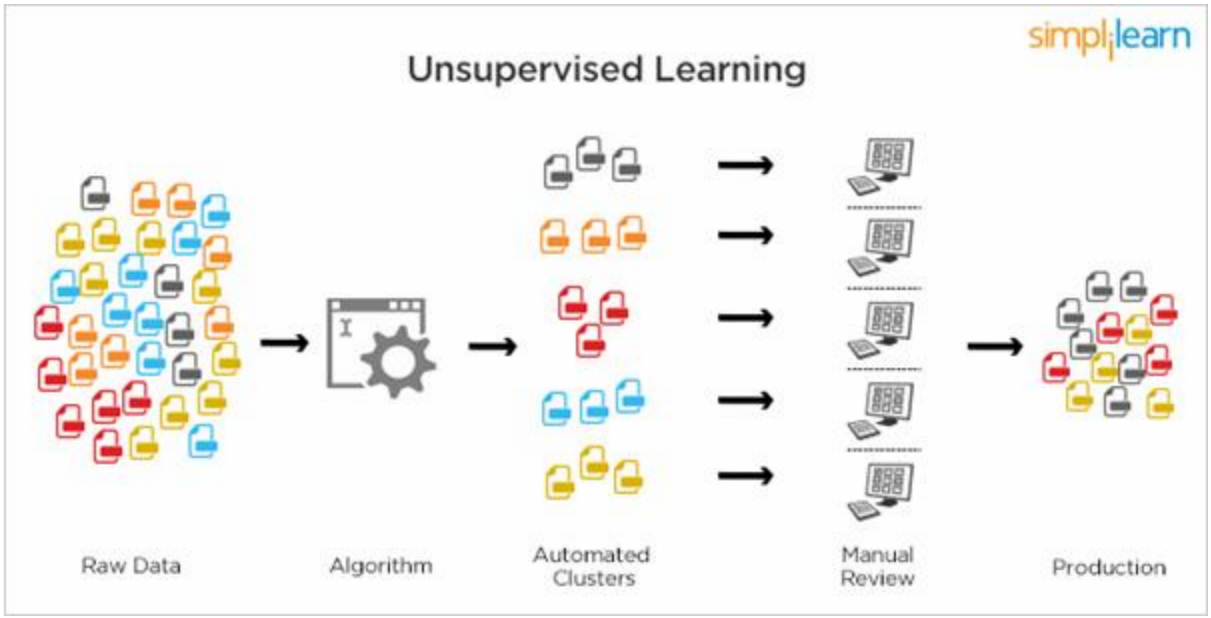


Figure 2 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

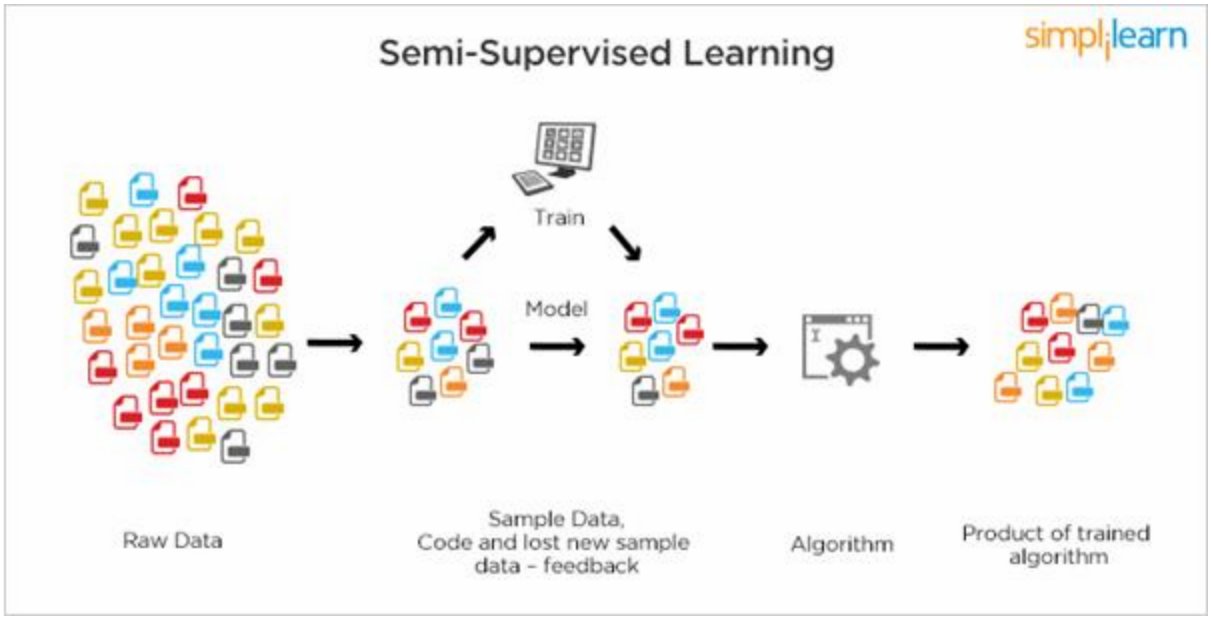


Figure 3 : Semi Supervised Learning

RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovered previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special types of neural networks and

applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

PYTHON

Basic programming language used for machine learning is : PYTHON

INTRODUCTION TO PYTHON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.

- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

Installation(using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

Figure 4 : Python download

Installation(using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager that quickly installs and manages packages.

In WINDOWS:

- In windows
  - Step 1: Open Anaconda.com/downloads in a web browser.
  - Step 2: Download python 3.4 version for (32-bits graphic installer/64 -bit graphic installer)
  - Step 3: select installation type(all users)
  - Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
  - Step 5: Open jupyter notebook (it opens in default browser)

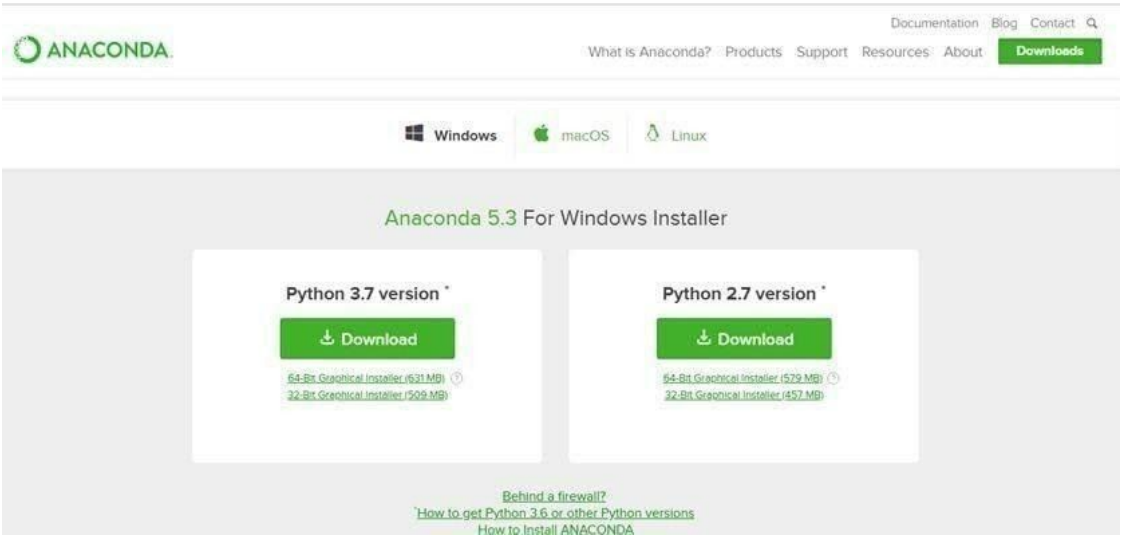


Figure 5 : Anaconda download

Figure 6 : Jupyter notebook

PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.

- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
    - Numbers
    - Strings
    - Lists
    - Tuples
    - Dictionary

Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets-([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data types.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays
- or hashes found in Perl and consist of key-value pairs. A dictionary key can be

almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

PYTHON FUNCTION:

Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

PYTHON USING OOPs CONCEPTS:

Class:

- Class: A user-defined prototype for an object that defines a set of attributes that

characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.
- Defining a Class:

o We define a class in a very similar way how we define a function.

  o Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Figure 7 : Defining a Class

init method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores: init ().

CHAPTER 3
CASE STUDY: STOCK PRICE PREDICTION

PROBLEM STATEMENT:

"To predict the stock prices of a particular-company using the stock market dataset- Long Short Term Memory networks".

DATA SET:

- The data is presented in a couple of formats to suit different individual's needs or computational limitations.
- I have included files containing 5 years of stock data (in the all*stocks*5yr.csv and corresponding folder).

  - The folder *stock* contains files of data for individual stocks, labelled by their stock ticker name.
  - The all*stocks*5yr.csv contains the same data, presented in a merged .csv file.
  - Depending on the intended use (graphing, modelling etc.) the user may prefer one of these given formats .
  - All the files have the following columns:
  - Date - in format: yy-mm-dd
  - Open - price of the stock at market open (this is NYSE data so all in USD)
  - High - Highest price reached in the day
  - Low Close - Lowest price reached in the day
  - Volume - Number of shares traded

- Name - the stock's ticker name.

OBJECTIVE OF THE CASE STUDY:

The Main Aim of this project is to predict the stock prices of a particular-company using the stock market dataset. In this project, I am going to use LSTM networks to predict stock prices. It's important to note that there are always other factors that affect the prices of stock, such as the political atmosphere and the market. However, we won't focus on those factors for this project. This type of problem comes under time series prediction. You may use a similar approach for language and electricity demand.

CHAPTER 4

DATA PREPROCESSING

READING DATA AND LIBRARIES:

Preparing text using following steps:

Getting the dataset:

We can get the data set from stock by scraping the web.

Importing the libraries:

```
[ ]  # Importing Required libraries
     import pandas as pd
     import numpy as np
     import seaborn as sns
     import re
```

Figure 8: Importing libraries

Importing the data :

We read the training data and printing all 5 year stocks data(head)

```
#@title Default title text
# all_stocks_5yr.csv
import pandas as pd
data = pd.read_csv('My Drive/all_stocks_5yr.csv')
data.head()
```

	date	open	high	low	close	volume	Name
0	2013-02-08	15.07	15.12	14.63	14.75	8407500	AAL
1	2013-02-11	14.89	15.01	14.26	14.46	8882000	AAL
2	2013-02-12	14.45	14.51	14.10	14.27	8126000	AAL
3	2013-02-13	14.30	14.94	14.25	14.66	10259500	AAL
4	2013-02-14	14.94	14.96	13.16	13.99	31879900	AAL

Figure 9: Reading 5year stocks data

Number of entries there in the data:

```
[ ] data.shape

(619040, 7)
```

Figure 10:number of entries in data

Displaying top five rows:

```
[ ] # Top five rows
    data.head()
```

	date	open	high	low	close	volume	Name
0	2013-02-08	15.07	15.12	14.63	14.75	8407500	AAL
1	2013-02-11	14.89	15.01	14.26	14.46	8882000	AAL
2	2013-02-12	14.45	14.51	14.10	14.27	8126000	AAL
3	2013-02-13	14.30	14.94	14.25	14.66	10259500	AAL
4	2013-02-14	14.94	14.96	13.16	13.99	31879900	AAL

Figure 11:displaying top 5 rows

Total number of columns :

```
[ ] data.columns

Index(['date', 'open', 'high', 'low', 'close', 'volume', 'Name'], dtype='object')
```

Observation : Total we have 619040 entries and 7 columns which indicates high price, low price and close column which means predicting the final price of the stock

Checking how many company's data we have in Data Set:

```
[ ] ### How many company's data we have here?
    data.Name.nunique()

505
```

Figure 12: Number of company's data in dataset

Selecting single company data(MCK):

```
[ ] ## Selecting the data XRX
    df = data[data['Name']=='XRX']
    df.shape
```

```
(1259, 7)
```

```
#select the columns date and close price
df = df[['date','close']]
df
```

	date	close
367536	2013-02-08	103.80
367537	2013-02-11	103.53
367538	2013-02-12	104.83
367539	2013-02-13	104.91
367540	2013-02-14	104.21
...	...	...
368790	2018-02-01	166.27
368791	2018-02-02	159.21
368792	2018-02-05	152.39
368793	2018-02-06	152.52
368794	2018-02-07	152.83

1259 rows × 2 columns

Figure 13: selecting the data of MCK company.

STATISTICAL ANALYSIS :

Visualising Output Column:

checking up trend and down trend using line plot.

```
[ ] # Line plot
    import matplotlib.pyplot as plt
    plt.plot(df['date'],df['close'])
    plt.show()
```

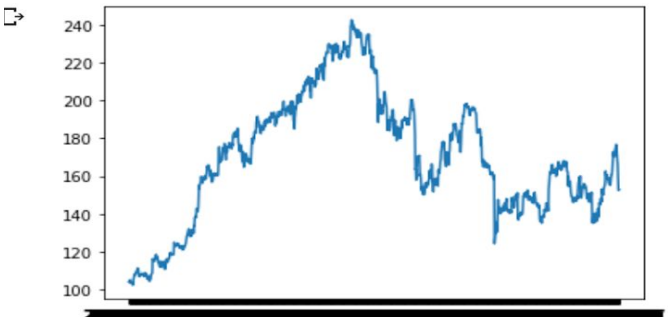


Figure 14: visualizing label counts

HANDLING MISSING VALUES:

```
[ ]  ### Missing values in the data
     df.isnull().sum()
```

```
date     0
close    0
dtype: int64
```

Figure 15: Checking missing values

- No missing values found in the dataset .
- Hence, no need for imputation .

CHAPTER 5

FEATURE SELECTION:

Select relevant features for the analysis

```
## Preparing the data
## Features and target
## Input and output
## last 8 days data as input
## X -- [[d1-d8],[d10,d18],[d20,28]....]
## y -- [d9,d19,d29,...]

## X = [[d1-d8],[d2-d9],]
## y = [d9,d10]
X = [] ## input
y = [] ## ouput
for i  in range(df.shape[0]-8-1):
  X.append(df['close'].iloc[i:i+8])# 0 -->[0:8], 1-->[1:9]
  y.append(df['close'].iloc[i+8])# 8,9,10
print(X[:2])
```

```
[611486    31.84
611487    31.96
611488    31.84
611489    32.00
611490    32.12
611491    31.88
611492    32.00
611493    31.44
Name: close, dtype: float64, 611487    31.96
611488    31.84
611489    32.00
611490    32.12
611491    31.88
611492    32.00
611493    31.44
611494    31.36
Name: close, dtype: float64]
```

Figure 16: feature and target the data

```
[ ]  import numpy as np
     X = np.array(X)
```

```
[ ]  df1  = pd.DataFrame(X,columns=['d1','d2','d3','d4','d5','d6','d7','d8'])
     df1.head()
```

	d1	d2	d3	d4	d5	d6	d7	d8
0	31.84	31.96	31.84	32.00	32.12	31.88	32.00	31.44
1	31.96	31.84	32.00	32.12	31.88	32.00	31.44	31.36
2	31.84	32.00	32.12	31.88	32.00	31.44	31.36	32.48
3	32.00	32.12	31.88	32.00	31.44	31.36	32.48	31.56
4	32.12	31.88	32.00	31.44	31.36	32.48	31.56	32.08

Figure 17: display the data in the columns

```
[ ] df1['d9'] = y
    df1.head()
```

	d1	d2	d3	d4	d5	d6	d7	d8	d9
0	31.84	31.96	31.84	32.00	32.12	31.88	32.00	31.44	31.36
1	31.96	31.84	32.00	32.12	31.88	32.00	31.44	31.36	32.48
2	31.84	32.00	32.12	31.88	32.00	31.44	31.36	32.48	31.56
3	32.00	32.12	31.88	32.00	31.44	31.36	32.48	31.56	32.08
4	32.12	31.88	32.00	31.44	31.36	32.48	31.56	32.08	32.72

Figure 18: displaying the outcome data

Train-Test-Split

```
[ ] ## Train test split
    X_train = df1.iloc[0:1000,0:8]
    X_test = df1.iloc[1000:,0:8]
    y_train =  df1['d9'][:1000]
    y_test  = df1['d9'][1000:]
    print(X_train.shape)
    print(y_train.shape)
    print(X_test.shape)
    print(y_test.shape)
```

```
(1000, 8)
(1000,)
(250, 8)
(250,)
```

Figure 19: dividing the data into train and test

Train/Test is a method to measure the accuracy of your model.

It is called Train/Test because you split the data set into two sets: a training set and a testing set.

Train the model means *create* the model.

Test the model means test the accuracy of the model.

split into train and test set:

- The *training* set should be a random selection of 80% of the original data.
- The *testing* set should be the remaining 20%.

Feature Scaling:

```
[ ]  ## Sacling the data
     from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     sc.fit(X_train)
     X_train_sc = pd.DataFrame(sc.transform(X_train),columns = X_train.columns)
     X_test_sc = pd.DataFrame(sc.transform(X_test),columns = X_train.columns)
     X_train_sc.describe()
```

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
count	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03	1.000000e+03
mean	9.814372e-17	-1.131317e-15	2.577938e-16	5.423439e-16	2.582379e-16	1.865175e-16	3.151923e-16	8.606449e-16	-6.782352e-16	1.587619e-16
std	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00	1.000500e+00
min	-2.020329e+00	-2.024109e+00	-2.028119e+00	-2.032038e+00	-2.035876e+00	-2.039817e+00	-2.043973e+00	-2.048092e+00	-2.052383e+00	-2.056681e+00
25%	-5.901951e-01	-5.919603e-01	-5.938406e-01	-5.956880e-01	-5.974916e-01	-5.993380e-01	-6.012849e-01	-6.032166e-01	-6.052305e-01	-6.072360e-01
50%	1.557702e-01	1.550559e-01	1.542864e-01	1.535196e-01	1.527772e-01	1.520232e-01	1.512284e-01	1.504379e-01	1.496119e-01	1.488016e-01
75%	6.215174e-01	6.214591e-01	6.213832e-01	6.212910e-01	6.212113e-01	6.211394e-01	6.210638e-01	6.209859e-01	6.209015e-01	6.208375e-01
max	1.916442e+00	1.918207e+00	1.920060e+00	1.921843e+00	1.923606e+00	1.925430e+00	1.927355e+00	1.929258e+00	1.931235e+00	1.933246e+00

Figure 20: scaling the data

CHAPTER 6

BUILDING MODEL AND EVALUATION

LSTM

Brief about the algorithms used:

Sequence prediction problems have been around for a long time. They are considered as one of the hardest problems to solve in the data science industry. These include a wide range of problems; from predicting sales to finding patterns in stock markets' data, from understanding movie plots to recognizing your way of speech, from language translations to predicting your next word on your iPhone's keyboard.

With the recent breakthroughs that have been happening in data science, it is found that for almost all of these sequence prediction problems, Long short Term Memory networks, a.k.a LSTMs have been observed as the most effective solution.

LSTMs have an edge over conventional feed-forward neural networks and RNN in many ways. This is because of their property of selectively remembering patterns for long durations of time. The purpose of this article is to explain LSTM and enable you to use it in real life problems. Let's have a look!

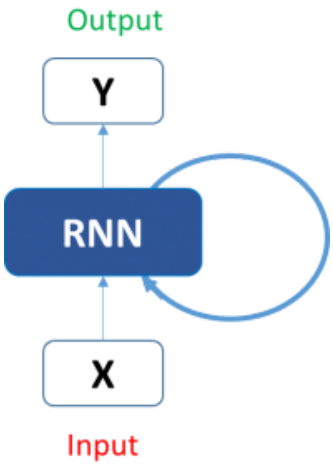
Table of Contents

1. A look into Recurrent Neural Networks (RNN)

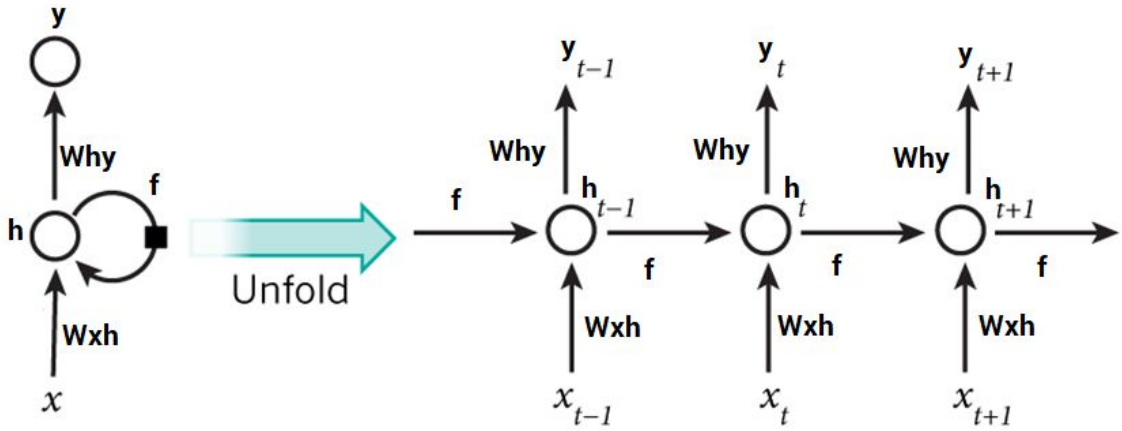
Take an example of sequential data, which can be the stock market's data for a particular stock. A simple machine learning model or an Artificial Neural Network may learn to predict the stock prices based on a number of features: the volume of the stock, the opening value etc. While the price of the stock depends on these features, it is also largely dependent on the stock values in the previous days. In fact for a trader, these values in the previous days (or the trend) is one major deciding factor for predictions.

In the conventional feed-forward neural networks, all test cases are considered to be independent. That is when fitting the model for a particular day, there is no consideration for the stock prices on the previous days.

This dependency on time is achieved via Recurrent Neural Networks. A typical RNN looks like:



This may be intimidating at first sight, but once unfolded, it looks a lot simpler:



Now it is easier for us to visualize how these networks are considering the trend of stock prices, before predicting the stock prices for today. Here every prediction at time t (h_t) is dependent on all previous predictions and the information learned from them.

RNNs can solve our purpose of sequence handling to a great extent but not entirely. We want our computers to be good enough to write Shakespearean sonnets. Now RNNs are great when it comes to short contexts,

but in order to be able to build a story and remember it, we need our models to be able to understand and remember the context behind the sequences, just like a human brain. This is not possible with a simple RNN.

Why? Let's have a look.

2. Limitations of RNNs

Recurrent Neural Networks work just fine when we are dealing with short-term dependencies. That is when applied to problems like:

The colour of the sky is _____.

RNNs turn out to be quite effective. This is because this problem has nothing to do with the context of the statement. The RNN need not remember what was said before this, or what was its meaning, all they need to know is that in most cases the sky is blue. Thus the prediction would be:

The colour of the sky is blue.

However, vanilla RNNs fail to understand the context behind an input. Something that was said long before, cannot be recalled when making predictions in the present. Let's understand this as an example:

I spent 20 long years working for the under-privileged kids in Spain. I then moved to Africa.

.....

I can speak fluent _____.

Here, we can understand that since the author has worked in Spain for 20 years, it is very likely that he may possess a good command over Spanish. But, to make a proper prediction, the RNN needs to remember this context. The relevant information may be separated from the point where it is needed, by a huge load of irrelevant data. This is where a Recurrent Neural Network fails!

The reason behind this is the problem of **Vanishing Gradient.** In order to understand this, you'll need to have some knowledge about how a feed-forward neural network learns. We know that for a conventional

feed-forward neural network, the weight updating that is applied on a particular layer is a multiple of the learning rate, the error term from the previous layer and the input to that layer. Thus, the error term for a particular layer is somewhere a product of all previous layers' errors. When dealing with activation functions like the sigmoid function, the small values of its derivatives (occurring in the error function) get multiplied multiple times as we move towards the starting layers. As a result of this, the gradient almost vanishes as we move towards the starting layers, and it becomes difficult to train these layers.

A similar case is observed in Recurrent Neural Networks. RNN remembers things for just small durations of time, i.e. if we need the information after a small time it may be reproducible, but once a lot of words are fed in, this information gets lost somewhere. This issue can be resolved by applying a slightly tweaked version of RNNs – the Long Short-Term Memory Networks.

3. Improvement over RNN: LSTM (Long Short-Term Memory) Networks

When we arrange our calendar for the day, we prioritize our appointments right? If in case we need to make some space for anything important we know which meeting could be canceled to accommodate a possible meeting.

Turns out that an RNN doesn't do so. In order to add new information, it transforms the existing information completely by applying a function. Because of this, the entire information is modified, on the whole, i. e. there is no consideration for *'important'* information and *'not so important'* information.

LSTMs on the other hand, make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things. The information at a particular cell state has three different dependencies.

We'll visualize this with an example. Let's take the example of predicting stock prices for a particular stock. The stock price of today will depend upon:

1. The trend that the stock has been following in the previous days, maybe a downtrend or an uptrend.
2. The price of the stock on the previous day, because many traders compare the stock's previous day price before buying it.
3. The factors that can affect the price of the stock for today. This can be a new company policy that is being criticized widely, or a drop in the company's profit, or maybe an unexpected change in the senior leadership of the company.

These dependencies can be generalized to any problem as:

1. The previous cell state *(i.e. the information that was present in the memory after the previous time step)*
2. The previous hidden state *(i.e. this is the same as the output of the previous cell)*
3. The input at the current time step *(i.e. the new information that is being fed in at that moment)*

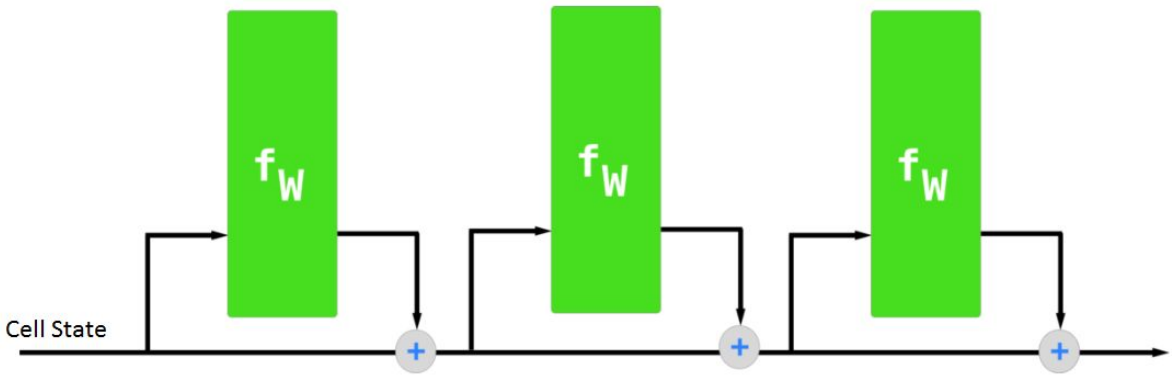
Another important feature of LSTM is its analogy with conveyor belts!

That's right!

Industries use them to move products around for different processes. LSTMs use this mechanism to move information around.

We may have some addition, modification or removal of information as it flows through the different layers, just like a product may be molded, painted or packed while it is on a conveyor belt.

The following diagram explains the close relationship of LSTMs and conveyor belts.



Source

Although this diagram is not even close to the actual architecture of an LSTM, it solves our purpose for now.

Just because of this property of LSTMs, where they do not manipulate the entire information but rather modify them slightly, they are able to *forget* and *remember* things selectively. How do they do so, is what we are going to learn in the next section?

4. Architecture of LSTMs

The functioning of LSTM can be visualized by understanding the functioning of a news channel's team covering a murder story. Now, a news story is built around facts, evidence and statements of many people. Whenever a new event occurs you take either of the three steps.

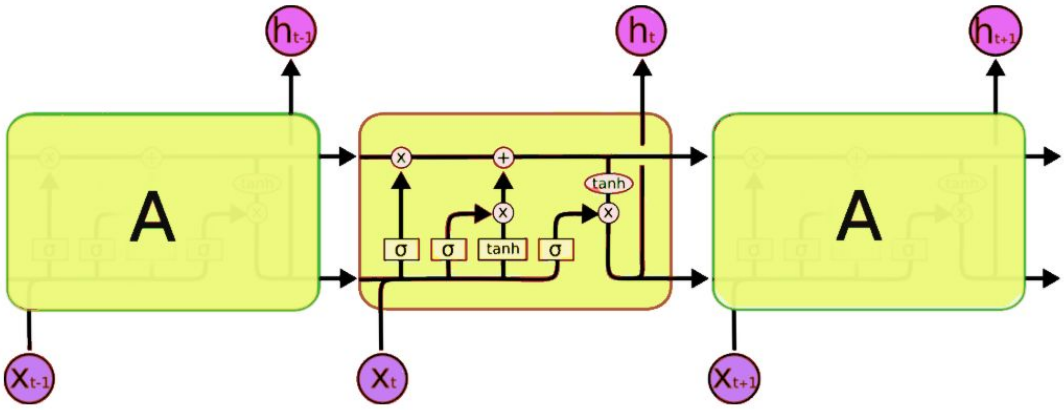
Let's say, we were assuming that the murder was done by 'poisoning' the victim, but the autopsy report that just came in said that the cause of death was 'an impact on the head'. Being a part of this news team what do you do? You immediately **forget** the previous cause of death and all stories that were woven around this fact.

What, if an entirely new suspect is introduced into the picture. A person who had grudges with the victim and

could be the murderer? You **input** this information into your news feed, right?

Now all these broken pieces of information cannot be served on mainstream media. So, after a certain time interval, you need to summarize this information and **output** the relevant things to your audience. Maybe in the form of "*XYZ turns out to be the prime suspect.*".

Now let's get into the details of the architecture of LSTM network:



Source

Now, this is nowhere close to the simplified version which we saw before, but let me walk you through it. A typical LSTM network is comprised of different memory blocks called **cells**

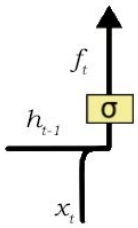
(the rectangles that we see in the image)**.** There are two states that are being transferred to the next cell; the **cell state** and the **hidden state**. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called **gates.** Each of them is being discussed below.

4.1 Forget Gate

Taking the example of a text prediction problem. Let's assume an LSTM is fed in, the following sentence:

Bob is a nice person. Dan on the other hand is evil.

As soon as the first full stop after "*person*" is encountered, the forget gate realizes that there may be a change of context in the next sentence. As a result of this, the *subject* of the sentence is *forgotten* and the place for the subject is vacated. And when we start speaking about "*Dan*" this position of the subject is allocated to "*Dan*". This process of forgetting the subject is brought about by the forget gate.



A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via multiplication of a filter. This is required for optimizing the performance of the LSTM network.

This gate takes in two inputs; h_t-1 and x_t.

h_t-1 is the hidden state from the previous cell or the output of the previous cell and x_t is the input at that particular time step. The given inputs are multiplied by the weight matrices and a bias is added. Following this, the sigmoid function is applied to this value. The sigmoid function outputs a vector, with values ranging from 0 to 1, corresponding to each number in the cell state. Basically, the sigmoid function is responsible for deciding which values to keep and which to discard. If a '0' is output for a particular value in the cell state, it means that the forget gate wants the cell state to forget that piece of information completely. Similarly, a '1' means that the forget gate wants to remember that entire piece of information. This vector output from the sigmoid function is multiplied to the cell state.

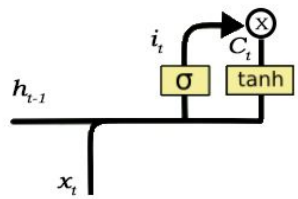
4.2 Input Gate

Okay, let's take another example where the LSTM is analyzing a sentence:

Bob knows swimming. He told me over the phone that he had served the navy for 4 long years.

Now the important information here is that "Bob" knows swimming and that he has served the Navy for four years. This can be added to the cell state, however, the fact that he told all this over the phone is a less important fact and can be ignored. This process of adding some new information can be done via the **input** gate.

Here is its structure:



The input gate is responsible for the addition of information to the cell state. This addition of information is basically a three-step process as seen from the diagram above.

1. Regulating what values need to be added to the cell state by involving a sigmoid function. This is basically very similar to the forget gate and acts as a filter for all the information from h_t-1 and x_t.
2. Creating a vector containing all possible values that can be added (as perceived from h_t-1 and x_t) to the cell state. This is done using the **tanh** function, which outputs values from -1 to +1.
3. Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the tanh function) and then adding this useful information to the cell state via addition operation.

Once this three-step process is done with, we ensure that only that information is added to the cell state that is *important* and is not *redundant*.

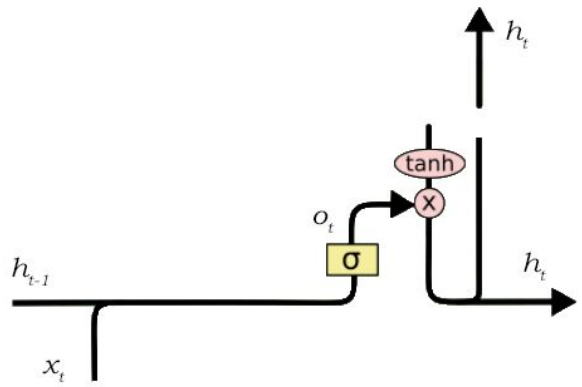
4.3 Output Gate

Not all information that runs along the cell state, is fit for being output at a certain time. We'll visualize this with an example:

Bob fought single handedly with the enemy and died for his country. For his contributions brave _____.

In this phrase, there could be a number of options for the empty space. But we know that the current input of *'brave'*, is an adjective that is used to describe a noun. Thus, whatever word follows, has a strong tendency of being a noun. And thus, Bob could be an apt output.

This job of selecting useful information from the current cell state and showing it out as an output is done via the output gate. Here is its structure:



The functioning of an output gate can again be broken down to three steps:

1. Creating a vector after applying **tanh** function to the cell state, thereby scaling the values to the range -1 to +1.
2. Making a filter using the values of h_t-1 and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
3. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as output and also to the hidden state of the next cell.

The filter in the above example will make sure that it diminishes all other values but *'Bob'*. Thus the filter needs to be built on the input and hidden state values and be applied on the cell state vector.

```
[ ] from keras.models import Sequential
    from keras.layers import Dense,Dropout,LSTM
    model = Sequential()
    # 250 cells
    model.add(LSTM(250,input_shape=(8,1)))
    # single neuron
    model.add(Dense(1))
    model.summary()
```

```
Using TensorFlow backend.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 250)               252000
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 251
=================================================================
Total params: 252,251
Trainable params: 252,251
Non-trainable params: 0
_________________________________________________________________
```

Figure19:Defining Parameters of the LSTM and Regression layer

Train the Models

```
[ ] X_test.shape
```

```
(250, 8)
```

```
[ ] # number of records , time steps,Features
    X_train_sc = X_train_sc.values.reshape(1000,8,1)
    X_test_sc = X_test_sc.values.reshape(250,8,1)
```

```
[ ] history = model.fit(X_train_sc,y_train,epochs=100,validation_data=(X_test_sc,y_test))
```

```
Train on 1000 samples, validate on 248 samples
Epoch 1/100
1000/1000 [==============================] - 2s 2ms/step - loss: 29000.2627 - val_loss: 16891.0290
Epoch 2/100
1000/1000 [==============================] - 1s 1ms/step - loss: 22798.3309 - val_loss: 14547.0924
Epoch 3/100
1000/1000 [==============================] - 1s 1ms/step - loss: 20311.1314 - val_loss: 12656.4364
Epoch 4/100
1000/1000 [==============================] - 1s 1ms/step - loss: 18219.2256 - val_loss: 11032.1005
Epoch 5/100
1000/1000 [==============================] - 1s 1ms/step - loss: 16369.5284 - val_loss: 9604.6550
Epoch 6/100
1000/1000 [==============================] - 1s 1ms/step - loss: 14715.3570 - val_loss: 8332.2738
Epoch 7/100
1000/1000 [==============================] - 1s 1ms/step - loss: 13227.1128 - val_loss: 7199.7944
Epoch 8/100
1000/1000 [==============================] - 1s 1ms/step - loss: 11880.4016 - val_loss: 6193.0473
Epoch 9/100
1000/1000 [==============================] - 1s 1ms/step - loss: 10660.7346 - val_loss: 5295.8550
Epoch 10/100
1000/1000 [==============================] - 1s 1ms/step - loss: 9555.1436 - val_loss: 4499.3977
Epoch 11/100
1000/1000 [==============================] - 1s 1ms/step - loss: 8556.6074 - val_loss: 3796.0257
Epoch 12/100
1000/1000 [==============================] - 1s 1ms/step - loss: 7658.5772 - val_loss: 3180.3693
Epoch 13/100
1000/1000 [==============================] - 1s 1ms/step - loss: 6852.2103 - val_loss: 2639.8718
Epoch 14/100
1000/1000 [==============================] - 1s 1ms/step - loss: 6129.3505 - val_loss: 2171.5547
Epoch 15/100
1000/1000 [==============================] - 1s 1ms/step - loss: 5486.6481 - val_loss: 1768.0369
Epoch 16/100
1000/1000 [==============================] - 1s 1ms/step - loss: 4911.9398 - val_loss: 1424.8893
Epoch 17/100
1000/1000 [==============================] - 1s 1ms/step - loss: 4405.0904 - val_loss: 1134.0065
Epoch 18/100
1000/1000 [==============================] - 1s 1ms/step - loss: 3955.3893 - val_loss: 888.8353
```

... ...
... ...

```
Epoch 92/100
1000/1000 [==============================] - 1s 1ms/step - loss: 61.6670 - val_loss: 28.6288
Epoch 93/100
1000/1000 [==============================] - 1s 1ms/step - loss: 55.7979 - val_loss: 26.2299
Epoch 94/100
1000/1000 [==============================] - 1s 1ms/step - loss: 51.3606 - val_loss: 25.1744
Epoch 95/100
1000/1000 [==============================] - 1s 1ms/step - loss: 56.4960 - val_loss: 38.4993
Epoch 96/100
1000/1000 [==============================] - 1s 1ms/step - loss: 49.5574 - val_loss: 31.3359
Epoch 97/100
1000/1000 [==============================] - 1s 1ms/step - loss: 44.7178 - val_loss: 18.3547
Epoch 98/100
1000/1000 [==============================] - 1s 1ms/step - loss: 39.0520 - val_loss: 23.1997
Epoch 99/100
1000/1000 [==============================] - 1s 1ms/step - loss: 38.3125 - val_loss: 17.9267
Epoch 100/100
1000/1000 [==============================] - 1s 1ms/step - loss: 35.2149 - val_loss: 20.6779
```

Figure21: Running the LSTM

Validate the Model

```
[ ] tr_loss = history.history['loss']
    val_loss = history.history['val_loss']
    ep = list(range(1,101))
    plt.plot(ep,tr_loss,color='r')
    plt.plot(ep,val_loss,color='b')
```

```
[<matplotlib.lines.Line2D at 0x7f6a17726630>]
```

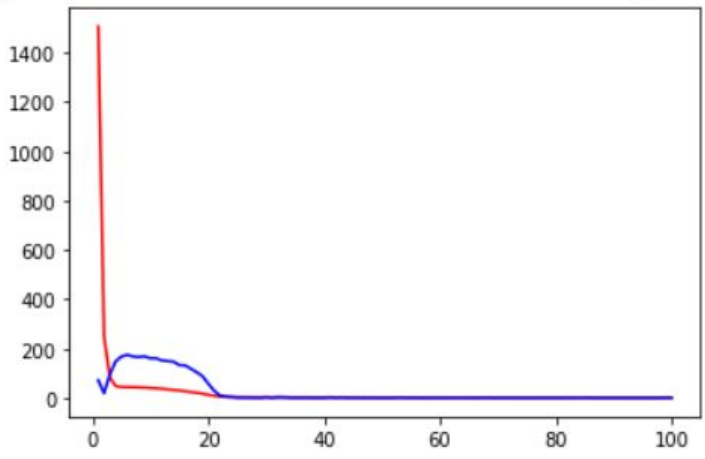


Figure22: Validate the Model

```
model.predict(X_test_sc)
```

```
array([[29.71954 ],
       [29.795204],
       [29.913649],
       [30.00442 ],
       [30.289612],
       [30.45243 ],
       [30.502684],
       [30.610018],
       [30.681866],
       [30.724665],
       [30.827076],
       [30.91192 ],
       [30.857706],
       [30.836216],
       [30.765888],
       [30.586956],
       [30.531645],
       [30.57952 ],
       [30.498362],
       [30.55208 ],
       [30.55098 ],
       [30.521479],
       [30.447777],
       [30.404919],
       [30.326199],
       [30.390844],
       [30.383293],
       [30.462788],
       [30.295288],
       [30.169695],
       [30.088388],
       [29.978638],
       [29.83427 ],
       [29.950829],
       [29.967518],
```

```
[30.10419 ],
[30.19788 ],
[30.179533],
[30.122475],
[30.02218 ],
[29.88971 ],
[29.767769],
[29.641245],
[29.55053 ],
[29.444567],
[29.372942],
[29.312428],
[29.268267],
[29.168371],
[29.130064],
[29.055244],
[29.115229],
[29.095816],
[29.193012],
[29.29432 ],
[29.43897 ],
[29.432598],
[29.426607],
[29.339127],
[29.220371],
[29.237402],
[29.27321 ],
[29.249699],
[29.323313],
[29.370409],
[29.361387],
[29.446547],
[29.495378],
[29.291553],
[29.265009],
[29.231937],
```

```
[29.147837],
[29.077099],
[29.114475],
[29.10323 ],
[29.106445],
[29.193762],
[29.243633],
[29.309755],
[29.353407],
[29.297958],
[29.27035 ],
[29.271368],
[29.216131],
[29.130463],
[29.18856 ],
[29.173315],
[29.094124],
[29.073416],
[29.203968],
[29.337463],
[29.352268],
[29.311863],
[29.306612],
[29.362715],
[29.541187],
[29.638983],
[29.838112],
[29.972872],
[29.9676  ],
[29.979538],
```

```
[29.867115],
[29.735676],
[29.736181],
[29.750372],
[29.768587],
[29.851574],
[29.978165],
[30.110062],
[30.19289 ],
[30.248327],
[30.308502],
[30.3476  ],
[30.677807],
[30.910213],
[31.22968 ],
[31.331331],
[31.433123],
[31.444952],
[31.49974 ],
[32.00458 ],
[32.45908 ],
[32.599316],
[32.759117],
[32.988087],
[33.03544 ],
[33.023613],
[32.719734],
[32.600986],
[32.715588],
```

```
[32.82764 ],
[32.92152 ],
[32.7975  ],
[32.58411 ],
[32.394623],
[32.36841 ],
[32.454803],
[32.567398],
[32.71383 ],
[32.83881 ],
[32.898098],
[32.954243],
[32.97811 ],
[33.08815 ],
[33.00179 ],
[32.94066 ],
[32.826008],
[32.640915],
[32.695953],
[33.00576 ],
[33.163017],
[33.272663],
[33.30157 ],
[33.190296],
[33.16679 ],
[33.323975],
[33.438175],
[33.561264],
[33.69113 ],
[33.948315],
```

```
[30.550594],
[30.601683],
[30.545477],
[30.482153],
[30.530851],
[30.725912],
[30.772017],
[30.82911 ],
[30.821524],
[30.770466],
[30.66807 ],
[30.60206 ],
[30.534163],
[30.402834],
[30.38108 ],
[30.355297],
[30.425903],
[30.54201 ],
[30.73522 ],
[30.866394],
[31.020376],
[31.588017],
[32.274666],
[32.591908],
[32.66252 ],
[32.534218],
[32.582672],
[32.89201 ],
[33.04225 ],
[33.031155],
[33.049976],
[33.244884],
[33.369495],
[33.359455],
[33.89501 ],
[33.75356 ],
[33.149326],
[32.736057]], dtype=float32)
```

Figure23: predicting the x-test value.

```
[ ] plt.plot(range(len(X_test_sc)),model.predict(X_test_sc).flat)
    plt.plot(range(len(X_test_sc)),y_test)
    plt.show()
```

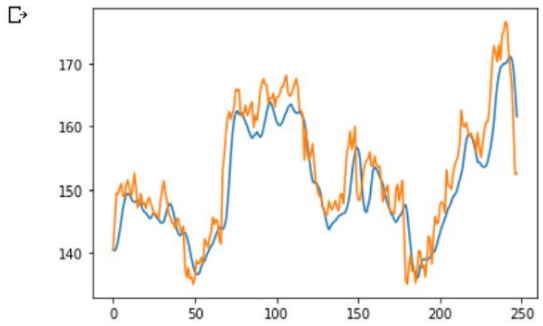


Figure24:Visualizing the Predictions

```
[ ] X_test_sc[0].reshape(1,8,1)
```

```
array([[[-2.34092839],
        [-2.34954981],
        [-2.26086844],
        [-2.19726994],
        [-2.20673369],
        [-2.25233077],
        [-2.24936232],
        [-2.07122018]]])
```

+ Code + Text

```
[ ] model.predict(X_test_sc[0].reshape(1,8,1))
```

```
array([[29.719545]], dtype=float32)
```

Figure25: reshaping the x-test data and stock price after 10 days

```
[ ] y_test[1000]
```

```
29.04
```

Figure26: predict stock price after 20 days

```
[ ] from keras.models import Sequential
    from keras.layers import Dense,Dropout,LSTM
    model2 = Sequential()
    # 250 cels
    model2.add(LSTM(250,input_shape=(8,1)))
    model2.add(Dropout(0.3))
    # single neuron
    model2.add(Dense(1))
    model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 250)               252000
_________________________________________________________________
dropout_1 (Dropout)          (None, 250)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
=================================================================
Total params: 252,251
Trainable params: 252,251
Non-trainable params: 0
_________________________________________________________________
```

Figure27:Defining Parameters of the LSTM and Regression layer with Dropout.

```
[ ] model2.compile(optimizer='adam',loss='mse')
```

```
[ ] history2 = model2.fit(X_train_sc,y_train,epochs=100,validation_data=(X_test_sc,y_test))
```

```
[ ] tr_loss = history2.history['loss']
    val_loss = history2.history['val_loss']
    ep = list(range(1,101))
    plt.plot(ep,tr_loss,color='r')
    plt.plot(ep,val_loss,color='b')
```

Figure28: reshaping the x-test data

```
[ ] model.predict(X_test_sc[0].reshape(1,10,1))
```

```
array([[140.43369]], dtype=float32)
```

Figure29:: stock price after 20 days

```
[ ] y_test[1000]
```

```
140.49
```

```
[ ]
    X_test_sc[0].reshape(1,8,1)
```

```
array([[[-2.34092839],
        [-2.34954981],
        [-2.26086844],
        [-2.19726994],
        [-2.20673369],
        [-2.25233077],
        [-2.24936232],
        [-2.07122018]]])
```

Figure30: predict stock price after 20 days

```
[ ]  plt.plot(range(len(X_test_sc)),model.predict(X_test_sc).flat)
     plt.plot(range(len(X_test_sc)),y_test)
     plt.show()
```

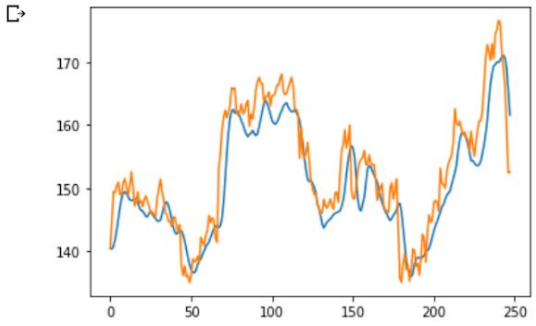


Figure 31: visualizing predicted and real graphs.

CONCLUSION

The LSTM part of the model allows us to build an RNN model with improved learning of long-term dependencies i.e., better memory which facilitates an improved performance for those words that we will generate.

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.

REFERENCES

1.https://www.kaggle.com/camnugent/sandp500
2.https://en.wikipedia.org/wiki/Long_short-term_memory

3.https://medium.com/datathings/the-magic-of-lstm-neural-networks-6775e8b540cd