

A
Project Report
on

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

Submitted for partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

Ms. G. Pooja Reddy	(15K81A0574)
Ms. CH. Lalitha Gayatri	(15K81A0569)
Mr. Pranav Saineni	(15K81A05A5)
Mr. Yash Jain	(15K81A0565)

UNDER THE GUIDANCE OF

Mr. N. Krishnavardhan
Associate Professor
Department of CSE



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
St. MARTIN'S ENGINEERING COLLEGE
(Affiliated to JNTU, Hyderabad)
DHULAPALLY(V), QUTBULLAPUR(M), SECUNDERABAD

2018-2019



St. MARTIN'S ENGINEERING COLLEGE

Non Minority College, Permanently affiliated to JNTUH, Approved by AICTE

NBA Accredited, ISO 9001:2008 Certified

Accredited by NAAC, SIRO Recognized Institute

CERTIFICATE

This is to certify that the project work entitled “**FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK**” that is being submitted by **Ms. G. Pooja Reddy (15K81A0574)**, **Ms. CH. Lalitha Gayatri (15K81A0569)**, **Mr. Yash Jain (15K81A0565)**, **Mr. Pranav Saineni (15K81A05A5)**, in partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** is a record of bonafide work carried out by them. The results embodied in this report have been verified and found satisfactory.

Internal Guide

Mr. N. Krishnavardhan

Associate Professor

Head of the Department

Dr. P. Udaya Kumar

Professor and Head

External Examiner

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose encouragement and guidance have crowded our efforts with success.

We extend our deep sense of gratitude to Principal, **Dr. P. SANTOSH KUMAR PATRA**, St. Martin's Engineering College, Dhulapally, for permitting us to undertake this project.

We are also thankful to **Dr. P. UDAYA KUMAR**, Head of the Department, Computer Science and Engineering, St. Martin's Engineering College, Dhulapally, for his support and guidance throughout our project as well as our Project Co-Ordinator **Dr. RASHMI SONI**, Professor, Computer Science and Engineering department for her valuable support.

We would like to express our sincere gratitude and indebtedness to our project supervisor **Mr. N. Krishnavardhan** Associate Professor, Computer Science and Engineering, St. Martin's Engineering College, Dhulapally, for his support and guidance throughout our project

Finally, we express thanks to all those who have helped us in successfully completing this project. Furthermore, we would like to thank our family and friends for their moral support and encouragement. We express thanks to all those who have helped us in successfully completing this project.

G. POOJA REDDY	(15K81A0574)
CH. LALITHA GAYATRI	(15K81A0569)
PRANAV SAINENI	(15K81A05A5)
YASH JAIN	(15K81A0565)

ABSTRACT

When is the right time to invest in a company? Can we use recurrent neural networks for time series analysis? Since stock prices are a sequence, we can use recurrent neural networks to make predictions. This software allows a user to predict the stock of company with a certain accuracy based on the dataset taken from yahoo finance. In this project, we will use python, jupyter notebook and anaconda navigator to test out a prediction model for Apple stock.

Recurrent neural networks allow computers to see, in other words, RNN are used to recognize images by transforming the original image through layers to a class scores and were inspired by the visual cortex. Every time we see something, a series of layers of neurons gets activated, and each layer will detect a set of features such as lines, edges. The high level of layers will detect more complex features in order to recognize what we saw.

Most of the models used for financial forecasting are not open-source and alternate model to be use is the ARIMA (Auto Regressive Integrated Moving Average) model which does not provide as much performance as a recurrent neural network.

The application does require a Wi-Fi connection and requires the user to install Jupyter Notebook with all the required dependencies. There has been an unprecedented growth in the number of devices being connected to the Internet since past few years. A lot of what we used to do in the real world has been shifted to the internet such as the stock market. This gives us a valuable insight to the vast amount of data which we can use to accurately predict the stock of the coming years.

INDEX

S.NO	CHAPTER NAME	PAGE.NO
1	INTRODUCTION	01
1.1	Objectives	01
1.2	Problem Specification	02
1.3	Methodology	03
1.4	Contributions	04
2	LITERATURE SURVEY	05
3	SYSTEM ANALYSIS	07
3.1	The Study of the System	07
3.2	Input & Output Representation	10
3.3	Process Model Used with Justification	11
4	SYSTEM DESIGN	15
4.1	System Requirements	15
4.2	UML Diagrams	16
4.3	Modules	18
5	IMPLEMENTATION	26
5.1	About Language and tools	26
5.2	Flow Chart	48
5.3	Sample Code	51
5.4	Screenshots	55
6	TESTING	60
6.1	Types of Test	60
6.2	Testing Objectives	63
6.3	Test cases	64
6.4	Test Results	65
7	CONCLUSION & FUTURE SCOPE	66
	REFERENCES	

LIST OF IMPORTANT FIGURES

FIG.NO	FIG TITLE	PAGE NO
3.1	Model of Feed-Forward Neural Network	07
3.2	Recurrent Neural Network & Feed Forward Network	08
3.3	Gradient Descent Graph with Back Propagation	12
3.4	Forward Propagation and Backward Propagation	13
3.5	Recurrent Neural Network as a sequence of Neural Network	13
4.1	Activity Diagram for Long Short-Term Memory	16
4.2	Activity Diagram for Neural Network	17
4.3	Long Short-Term Memory Diagram	18
4.4	Plotting graph using matplotlib	21
4.5	Recurrent Neural Network Structure in Keras	24
5.1	Creating environments in Anaconda Navigator	28
5.2	Activating environments in Anaconda Navigator	28
5.3	Installed Package Toolbar	29
5.4	Package Installation	30
5.5	Jupyter Notebook Example for Plotting	34
5.6	Jupyter Notebook Interface	35
5.7	Python Type Hierarchy	39
5.8	Python Sample Kernel Code	40
5.9	Long Short-Term Memory Model implemented using Keras	41
5.10	tanh Function Graph	42
5.11	Sigmoid Function Graph	43
5.12	VSCode Interface	47
5.13	Supervised Learning Flow Chart	48
5.14	Recurrent Neural Network with shared inputs	49
5.15	Recurrent Neural Network with multiple inputs	50
5.16	Anaconda Navigator	55
5.17	Apple Inc. stock taken from yahoo finance	56
5.18	Dataset values after fixing random values	57
5.19	Normalized Values	57
5.20	Dataset values after applying LSTM Model	58
5.21	Plotted graph for trained vs test set	59

LIST OF IMPORTANT TABLES

TAB.NO	TAB TITLE	PAGE NO
6.1	Test Results	65

LIST OF ACRONYMS AND DEFINITONS

S.NO	ACRONYM	DEFINITION
1	LSTM	Long Short-Term Memory
2	NN	Neural Network
3	RNN	Recurrent Neural Network
4	WIFI	Wireless Fidelity
5	Inc.	Incorporation
6	CTC	Connectionist Temporal Classification
7	IDE	Integrated Development Environment
8	UML	Unified Modeling Language
9	SDK	Software Development Kit
10	ANN	Artificial Neural Networks
11	GD	Gradient Descent
12	A.I	Artificial Intelligence
13	IFC	Internet Foundation Classes
14	GUI	Graphical User Interface

1. INTRODUCTION

1.1 Objectives

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

In this project, you will see how you can use a time-series model known as Long Short-Term Memory. LSTM models are powerful, especially for retaining a long-term memory, by design, as you will see later. You'll tackle the following topics in this tutorial:

- Understand why would you need to be able to predict stock price movements;
- Download the data - You will be using stock market data gathered from Yahoo finance
- Split train-test data and also perform some data normalization;
- Go over and apply a few averaging techniques that can be used for one-step ahead predictions;
- Motivate and briefly discuss an LSTM model as it allows to predict more than one-step ahead;
- Predict and visualize future stock market with current data

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

1.2 Problem Specification

If you understand the problem clearly, you should be able to list some potential solutions to test in order to generate the best model. Understand that you will likely have to try out a few solutions before you land on a good working model.

Exploratory data analysis can help you understand your data, but you can't yet claim that patterns you find generalize until you check those patterns against previously unseen data. Failure to check could lead you in the wrong direction or reinforce stereotypes or bias.

Data collected specifically for your task is going to be the most useful. In practice, you may not be able to do this, and you'll rely on whatever data you can get that's close enough. That's fine as long as you're aware of the cost, and as you can eventually get product logs, you can use those to build something more targeted to your task.

Recurrent Neural Networks (RNN) are a powerful and robust type of neural networks and belong to the most promising algorithms out there at the moment because they are the only ones with an internal memory. RNN's are relatively old, like many other deep learning algorithms. They were initially created in the 1980's, but can only show their real potential since a few years, because of the increase in available computational power, the massive amounts of data that we have nowadays and the invention of LSTM in the 1990's. Because of their internal memory, RNN's are able to remember important things about the input they received, which enables them to be very precise in predicting what's coming next. This is the reason why they are the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more because they can form a much deeper understanding of a sequence and its context, compared to other algorithms.

You would like to model stock prices correctly, so as a stock buyer you can reasonably decide when to buy stocks and when to sell them to make a profit. This is where time series modelling comes in. You need good machine learning models that can look at the history of a sequence of data and correctly predict what the future elements of the sequence are going to be.

This project uses Recurrent Neural Networks to calculate the stock prices of a company (in our case we use Apple Inc.) to predict the stock of the company for the upcoming years based on the data we currently have upon it.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

1.3 Methodology

Before we begin our journey of understanding how recurrent neural network enhances personalization across various businesses, let us try to get a little idea about machine learning first. Machine learning mainly focuses on the development of computer programs which can teach themselves to grow and change when exposed to new data. Machine learning studies algorithms for self-learning to do stuff. It can process massive data faster with the learning algorithm.

Data is growing day by day, and it is impossible to understand all of the data with higher speed and higher accuracy. More than 80% of the data is unstructured that is audios, videos, photos, documents, graphs, etc. Finding patterns in data on planet earth is impossible for human brains. The data has been very massive and the time taken to compute would increase only. This is where Machine Learning comes into action, to help people with significant data in minimum time.

The term "recurrent neural network" is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that cannot be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional stored state, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memory networks (LSTMs) and gated recurrent units.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

1.4 Contributions

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. As a scientific endeavor, machine learning grew out of the quest for artificial intelligence. Already in the early days of AI as an academic discipline, some researchers were interested in having machines learn from data. They attempted to approach the problem with various symbolic methods, as well as what were then termed "neural networks"; these were mostly perceptron and other models that were later found to be reinventions of the generalized linear models of statistics. Probabilistic reasoning was also employed, especially in automated medical diagnosis.

However, an increasing emphasis on the logical, knowledge-based approach caused a rift between AI and machine learning. Probabilistic systems were plagued by theoretical and practical problems of data acquisition and representation. By 1980, expert systems had come to dominate AI, and statistics was out of favor. Work on symbolic/knowledge-based learning did continue within AI, leading to inductive logic programming, but the more statistical line of research was now outside the field of AI proper, in pattern recognition and information retrieval. Neural networks research had been abandoned by AI and computer science around the same time. This line, too, was continued outside the AI/CS field, as "connectionism", by researchers from other disciplines including Hopfield, Rumelhart and Hinton. Their main success came in the mid-1980s with the reinvention of backpropagation.

Long short-term memory (LSTM) networks were discovered by Hochreiter and Schmidhuber in 1997 and set accuracy records in multiple applications domains. Around 2007, LSTM started to revolutionize speech recognition, outperforming traditional models in certain speech applications. In 2009, a Connectionist Temporal Classification (CTC)-trained LSTM network was the first RNN to win pattern recognition contests when it won several competitions in connected handwriting recognition. In 2014, the Chinese search giant Baidu used CTC-trained RNNs to break the Switchboard Hub5'00 speech recognition benchmark without using any traditional speech processing methods.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

2. LITERATURE SURVEY

Artificial neural networks (ANNs) are made from layers of connected units called artificial neurons. A “shallow network” refers to an ANN with one input layer, one output layer, and at most one hidden layer without a recurrent connection. As the number of layers increases, the complexity of network increases too. More number of layers or recurrent connections generally increases the depth of the network and empowers it to provide various levels of data representation and feature extraction, referred to as “deep learning”. In general, these networks are made from nonlinear but simple units, where the higher layers provide a more abstract representation of data and suppresses unwanted variability.

Due to optimization difficulties caused by composition of the nonlinearity at each layer, not much work occurred on deep network architectures before significant advances in 2006. ANNs with recurrent connections are called recurrent neural networks (RNNs), which are capable of modelling sequential data for sequence recognition and prediction. RNNs are made of high dimensional hidden states with non-linear dynamics. The structure of hidden states work as the memory of the network and state of the hidden layer at a time is conditioned on its previous state. This structure enables the RNNs to store, remember, and process past complex signals for long time periods. RNNs can map an input sequence to the output sequence at the current timestep and predict the sequence in the next timestep.

The development of back-propagation using gradient descent (GD) has provided a great opportunity for training RNNs. This simple training approach has accelerated practical achievements in developing RNNs. However, it comes with some challenges in modelling long-term dependencies such as vanishing and exploding gradient problems. RNNs are a class of supervised machine learning models, made of artificial neurons with one or more feedback loops. The feedback loops are recurrent cycles over time or sequence (we call it time throughout this paper). Training a RNN in a supervised fashion requires a training dataset of input-target pairs. The objective is to minimize the difference between the output and target pairs (i.e., the loss value) by optimizing the weights of the network.

A RNN refers to a network of artificial neurons with recurrent connections among them. The recurrent connections learn the dependencies among input sequential or time-series data. The ability to learn sequential dependencies has allowed RNNs to gain popularity in applications such

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

as speech recognition, speech synthesis, machine vision, and video description generation. One of the main challenges in training RNNs is learning long-term dependencies in data. It occurs generally due to the large number of parameters that need to be optimized during training in RNN over long periods of time. This paper discusses several architectures and training methods that have been developed to tackle the problems associated with training of RNNs.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

3. SYSTEM ANALYSIS

3.1 The Study of the System

We will first discuss some important facts about the normal “Feed Forward Neural Networks”, that you need to know, to understand Recurrent Neural Networks properly. But it is also important that you understand what sequential data is. It basically is just ordered data, where related things follow each other. Examples are financial data or the DNA sequence. The most popular type of sequential data is perhaps Time series data, which is just a series of data points that are listed in time order.

Model:

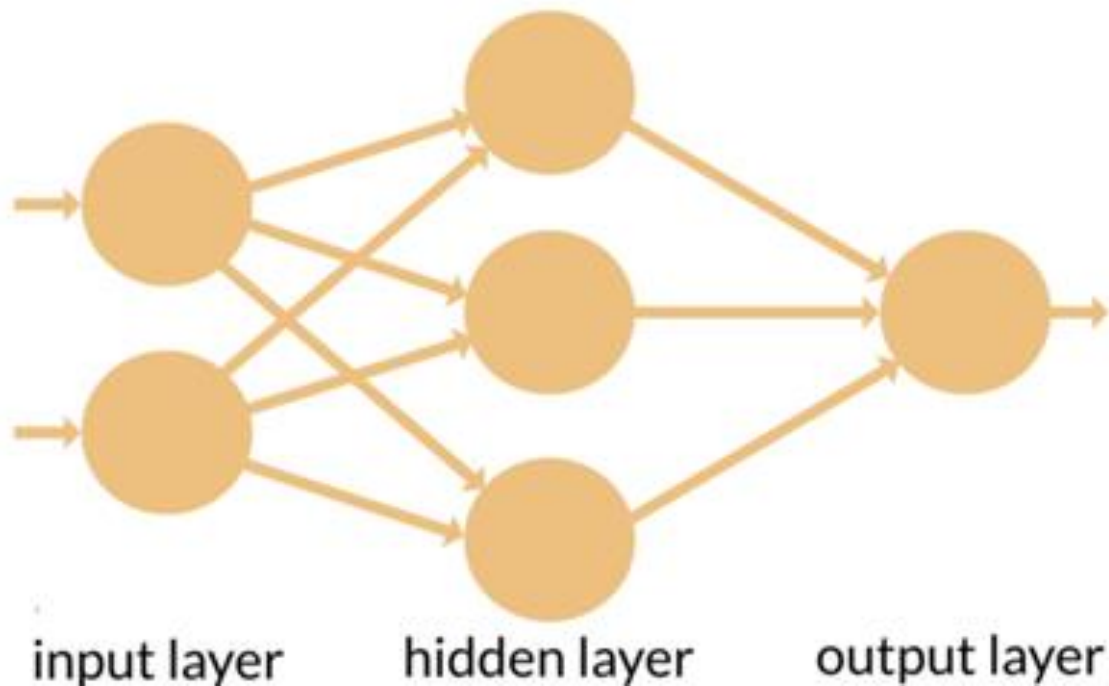


Fig 3.1: Model of Feed-Forward Neural Network

RNN's and Feed-Forward Neural Networks are both named after the way they channel information. In a Feed-Forward neural network, the information only moves in one direction, from the input layer, through the hidden layers, to the output layer. The information moves straight through the network. Because of that, the information never touches a node twice. Feed-Forward Neural Networks, have no memory of the input they received previously and are therefore bad in predicting what's coming next. Because a feedforward network only considers the current input,

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

it has no notion of order in time. They simply can't remember anything about what happened in the past, except their training.

In a RNN, the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously. The two images below illustrate the difference in the information flow between a RNN and a Feed-Forward Neural Network.

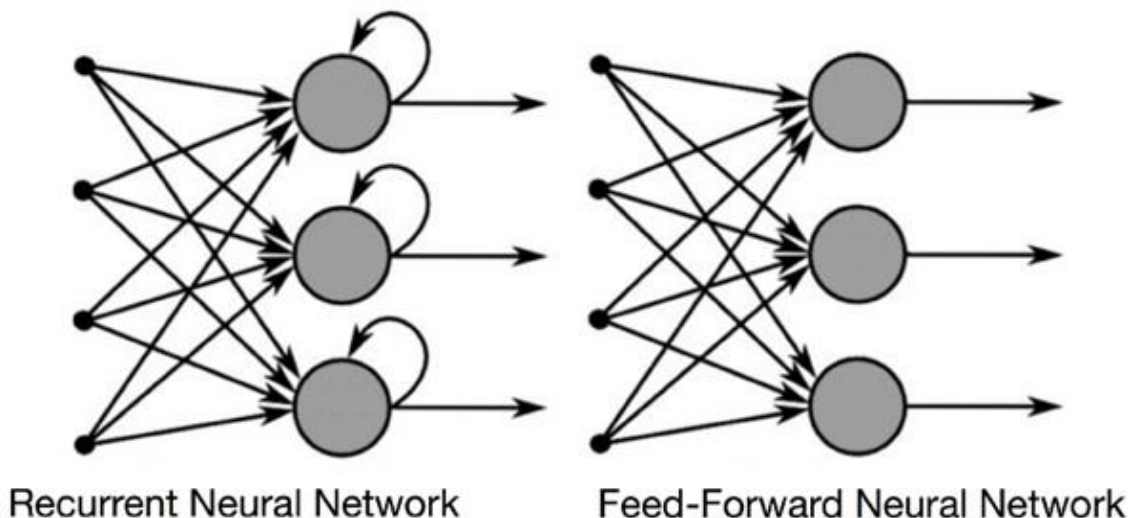


Fig 3.2: Recurrent Neural Network & Feed-Forward Neural Network

A usual RNN has a short-term memory. In combination with a LSTM they also have a long-term memory, but we will discuss this further below. Another good way to illustrate the concept of a RNN's memory is to explain it with an example:

Imagine you have a normal feed-forward neural network and give it the word "neuron" as an input and it processes the word character by character. At the time it reaches the character 'r', it has already forgotten about 'n', 'e' and 'u', which makes it almost impossible for this type of neural network to predict what character would come next.

A Recurrent Neural Network is able to remember exactly that, because of its internal memory. It produces output, copies that output and loops it back into the network. Recurrent Neural Networks add the immediate past to the present. Therefore, a Recurrent Neural Network has two inputs, the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

A Feed-Forward Neural Network assigns, like all other Deep Learning algorithms, a weight matrix to its inputs and then produces the output. Note that RNN's apply weights to the current and also to the previous input. Furthermore, they also tweak their weights for both through gradient descent and Backpropagation Through Time, which we will discuss in the next section below.

Also note that while Feed-Forward Neural Networks map one input to one output, RNN's can map one to many, many to many (translation) and many to one (classifying a voice).

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

3.2 Input & Output Representation

To provide following features-

- A. Download stock of any company listed in yahoo finance
- B. The ability to control the splitting ratio of the test & training sets
- C. Storing the predicted values in xlxs format
- D. High Performance
- E. Can run on any windows, osx and linux device
- F. Storing the predicted values in csv format
- G. Plot the graph according to the trained and test set

Desktop app must be lightweight and must not consume excess resources (memory, CPU, power etc.).

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

3.3 Process Model Used with Justification

This Project consists of following characteristics-

- A. It has been developed using Python
- B. It developed in Jupyter Notebook so has access to all Tensorflow Directories
- C. It does not require much resources and can be run on a laptop with less than 2GB of RAM.
- D. It is platform independent.

It's tough at first to wrap our minds around the fact that a computer can predict the future but this can be accurately done if we have enough data train our computer.

Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore, it is well suited to learn from important experiences that have very long-time lags in between.

To understand the concept of Backpropagation Through Time you definitely have to understand the concepts of Forward and Back-Propagation first. I will not go into the details here because that would be way out of the limit of this blog post, so I will try to give you a definition of these concepts that is as simple as possible but allows you to understand the overall concept of backpropagation through time.

In neural networks, you basically do Forward-Propagation to get the output of your model and check if this output is correct or incorrect, to get the error. Now you do Backward-Propagation, which is nothing but going backwards through your neural network to find the partial derivatives of the error with respect to the weights, which enables you to subtract this value from the weights. You can view an RNN as a sequence of Neural Networks that you train one after another with backpropagation

Those derivatives are then used by Gradient Descent, an algorithm that is used to iteratively minimize a given function. Then it adjusts the weights up or down, depending on which decreases the error.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

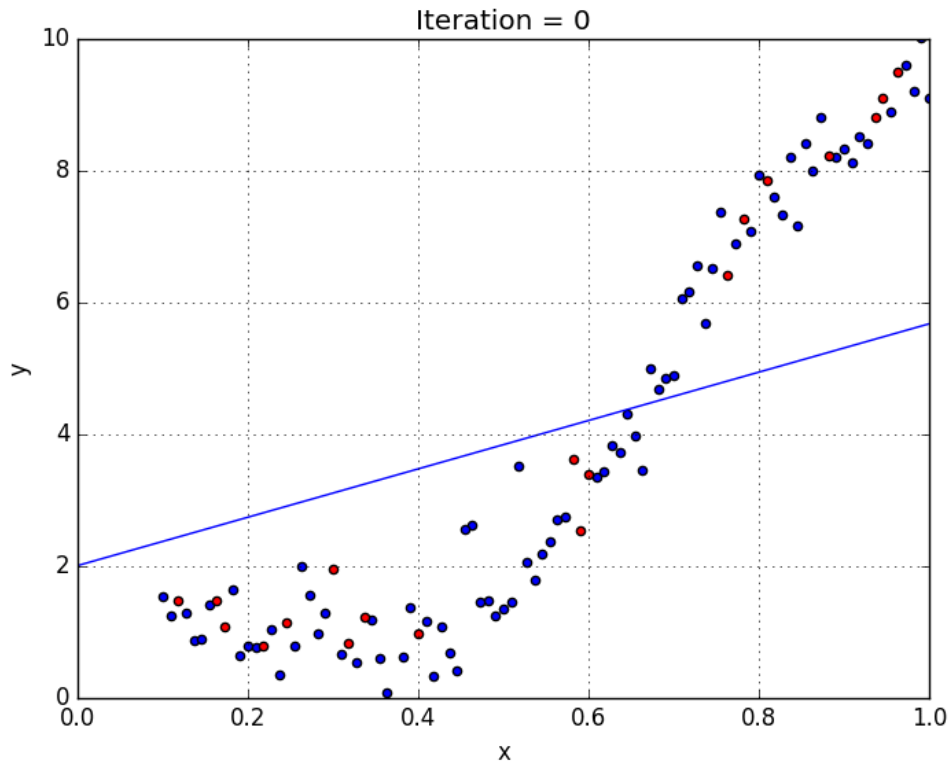


Fig 3.3: Gradient Descent Graph with Back Propagation

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent to update the parameters of our model. Parameters refer to coefficients in Linear Regression and weights in neural networks.

A Loss Functions tells us “how good” our model is at making predictions for a given set of parameters. The cost function has its own curve and its own gradients. The slope of this curve tells us how to update our parameters to make the model more accurate. Formula for cost function:

$$f(m,b)=1/N_n\sum_{i=1}(y_i-(mx_i+b))^2$$

That is exactly how a Neural Network learns during the training process. So, with Backpropagation you basically try to tweak the weights of your model, while training.

The image below illustrates the concept of Forward Propagation and Backward Propagation perfectly at the example of a Feed Forward Neural Network:

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

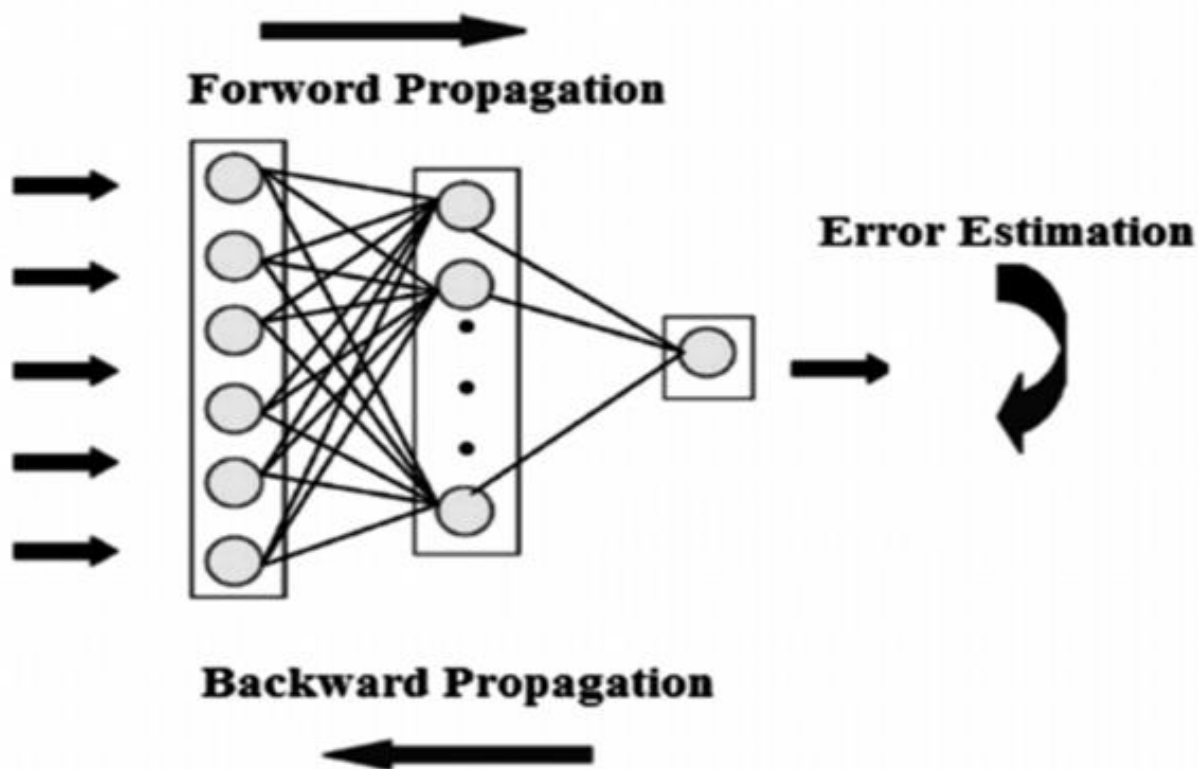


Fig 3.4: Forward Propagation and Backward Propagation

The image below illustrates an unrolled RNN. On the left, you can see the RNN, which is unrolled after the equal sign. Note that there is no cycle after the equal sign since the different timesteps are visualized and information gets passed from one timestep to the next. This illustration also shows why an RNN can be seen as a sequence of Neural Networks.

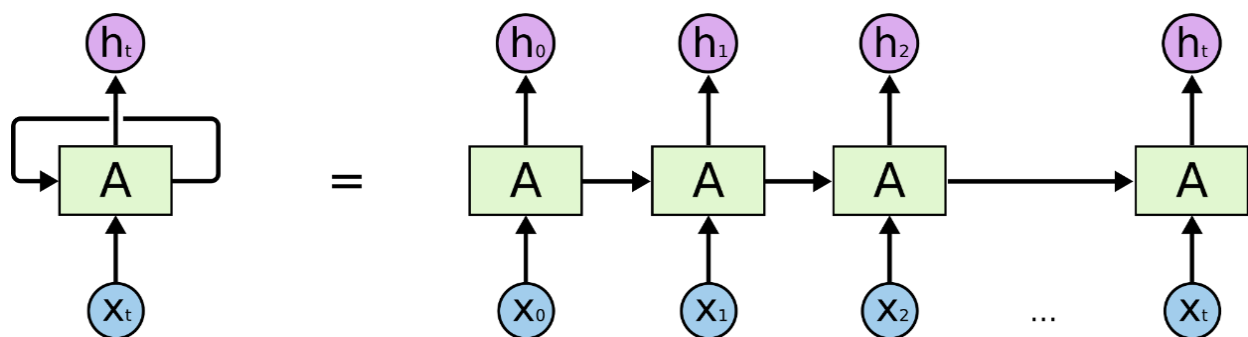


Fig 3.5: RNN as a sequence of Neural Networks

If you do Backpropagation Through Time, it is required to do the conceptualization of unrolling, since the error of a given timestep depends on the previous timestep.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

Within BPTT the error is back-propagated from the last to the first timestep, while unrolling all the timesteps. This allows calculating the error for each timestep, which allows updating the weights. Note that BPTT can be computationally expensive when you have a high number of timesteps. Some of the formulas used are:

- Formula for calculating current state:

$$h_t = f(h_{t-1}, x_t)$$

h_t -> current state

h_{t-1} -> previous state

x_t -> input state

- Formula for applying Activation function(tanh):

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

w_{hh} -> weight at recurrent neuron

w_{xh} -> weight at input neuron

- Formula for calculating output:

$$y_t = W_{hy}h_t$$

Y_t -> output

W_{hy} -> weight at output layer

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

4. SYSTEM DESIGN

4.1 System Requirements

Software Requirement:

- Python
- Anaconda Navigator
- VScode
- Jupyter Notebook
- Chrome

Hardware Requirements:

- Processor – i3
- Hard Disk – 5 GB
- Memory – 2GB RAM

4.2 UML Diagrams

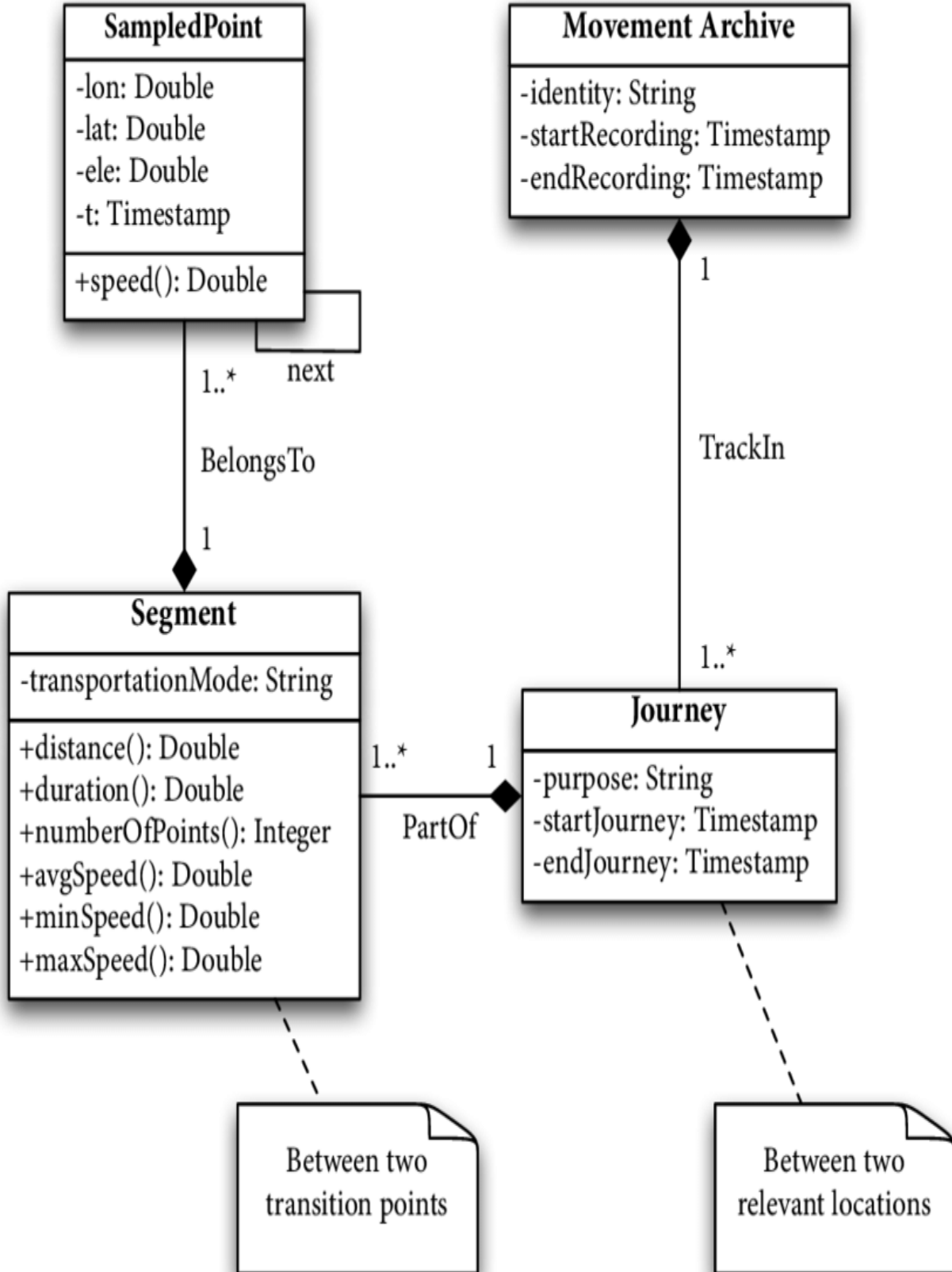


Fig 4.1: Activity Diagram for LSTM

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

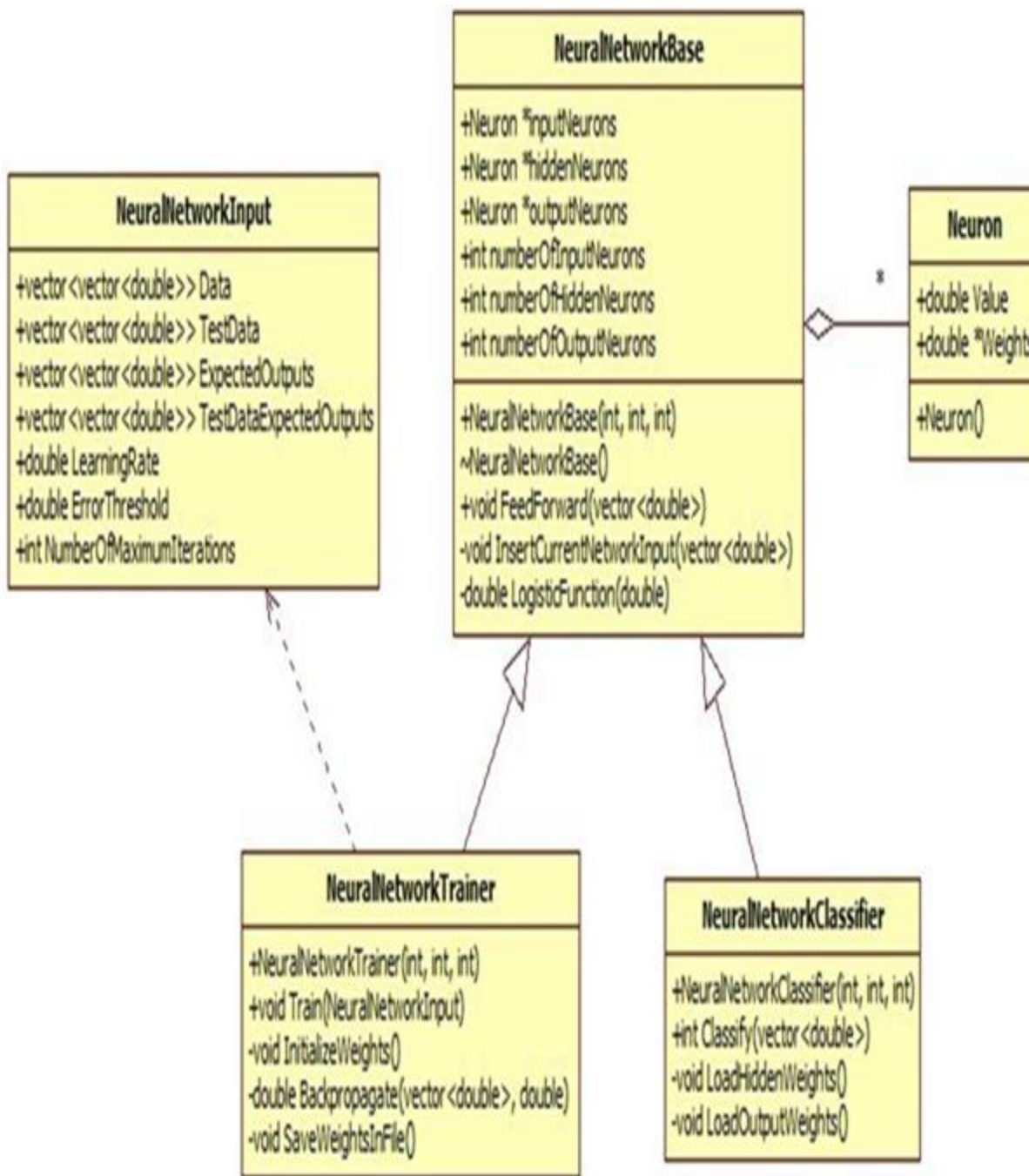


Fig 4.2: Activity Diagram for Neural Network

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

4.3 Modules

4.3.1 LONG-SHORT TERM MEMORY:

Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore, it is well suited to learn from important experiences that have very long-time lags in between. The units of an LSTM are used as building units for the layers of a RNN, which is then often called an LSTM network. LSTM's enable RNN's to remember their inputs over a long period of time. This is because LSTM's contain their information in a memory, that is much like the memory of a computer because the LSTM can read, write and delete information from its memory.

This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information (e.g. if it opens the gates or not), based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which not. In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn't important (forget gate) or to let it impact the output at the current time step (output gate). You can see an illustration of a RNN with its three gates below:

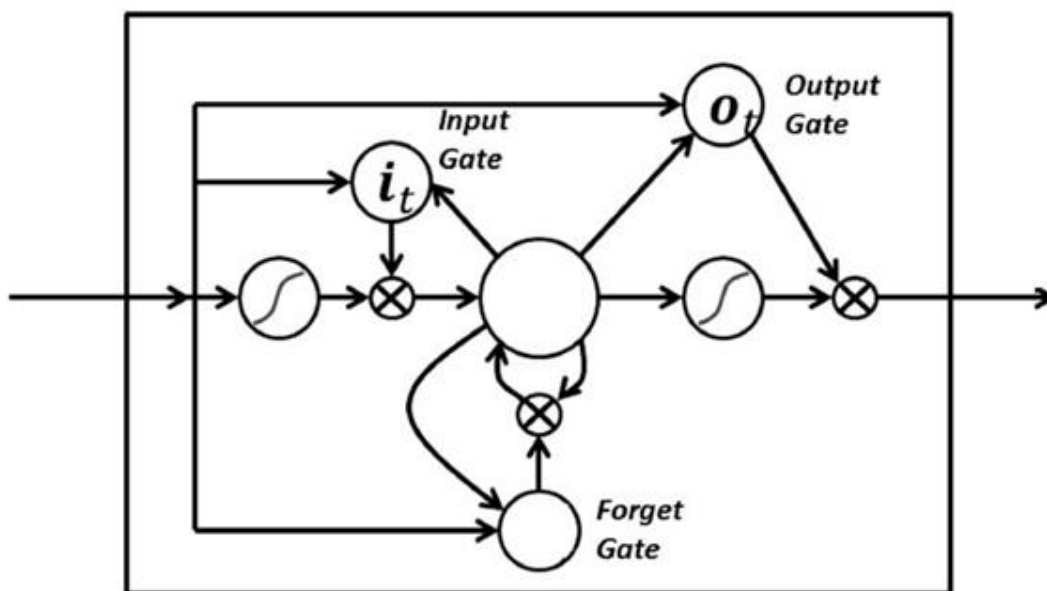


Fig 4.3: Long-Short Term Memory Diagram

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

4.3.2 NUMPY:

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n -dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. Some examples are

Array creation:

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x
array([1, 2, 3])
>>> y = np.arange(10) # like Python's range, but returns an array
>>> y
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Basic operations:

```
>>> a = np.array([1, 2, 3, 6])
>>> b = np.linspace(0, 2, 4) # create an array with four equally spaced
points starting with 0 and ending with 2.
>>> c = a - b
>>> c
array([ 1.          ,  1.33333333,  1.66666667,  4.          ])
>>> a**2
array([ 1,  4,  9, 36])
```

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

4.3.3 MATPLOTLIB:

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).

Generating visualizations with pyplot is very quick:

```
>>> a = np.array([1, 2, 3, 6])
>>> b = np.linspace(0, 2, 4) # create an array with four equally spaced
points starting with 0 and ending with 2.
>>> c = a - b
>>> c
array([ 1.          ,  1.33333333,  1.66666667,  4.          ])
>>> a**2
array([ 1,  4,  9, 36])
```

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

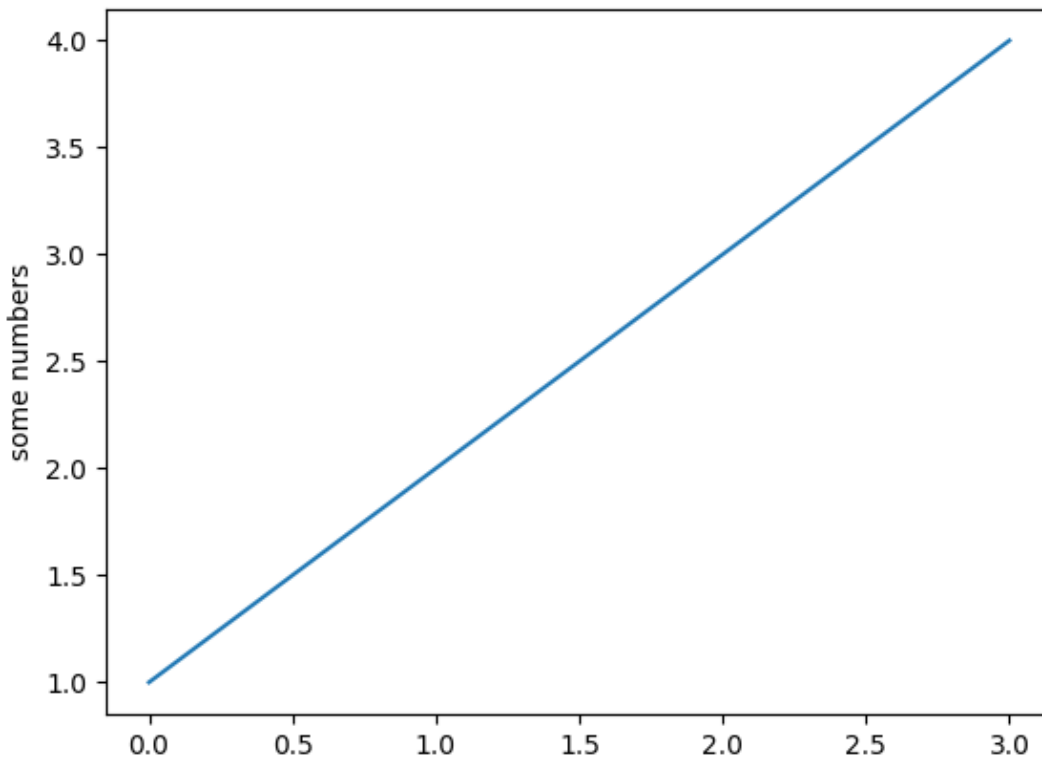


Fig 4.4: Plotting graph using matplotlib

You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4. If you provide a single list or array to the `plot()` command, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you. Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are `[0,1,2,3]`. `plot()` is a versatile command, and will take an arbitrary number of arguments.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

4.3.4 PANDAS:

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language.

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet.
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure.

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R’s data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

The best way to think about the pandas data structures is as flexible containers for lower dimensional data. For example, DataFrame is a container for Series, and Series is a container for scalars. We would like to be able to insert and remove objects from these containers in a dictionary-like fashion.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

4.3.5 KERAS:

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the Xception deep neural network model.

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than a standalone machine-learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine.[10] It also allows use of distributed training of deep-learning models on clusters of Graphics Processing Units (GPU) and Tensor processing units (TPU).

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU

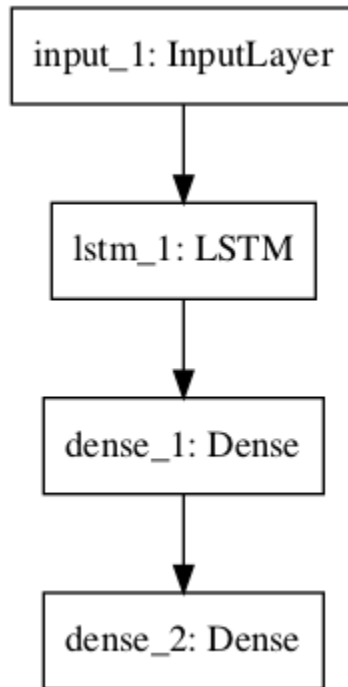


Fig 4.5: Recurrent Neural Network Structure in Keras

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

4.3.6 SCIKIT-LEARN:

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy. The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from the French Institute for Research in Computer Science and Automation in Rocquencourt, France, took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012. As of 2018, scikit-learn is under active development.

Scikit-learn is largely written in Python, with some core algorithms written in Cython to achieve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR.

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007. Later Matthieu Brucher joined the project and started to use it as a part of his thesis work. In 2010 INRIA, the French Institute for Research in Computer Science and Automation, got involved and the first public release (v0.1 beta) was published in late January 2010.

- July 2017. scikit-learn 0.19.0
- September 2016. scikit-learn 0.18.0
- November 2015. scikit-learn 0.17.0
- March 2015. scikit-learn 0.16.0
- July 2014. scikit-learn 0.15.0
- August 2013. scikit-learn 0.14

5. SYSTEM IMPLEMENTATION

5.1 About Language and Tools

5.1.1 ANACONDA NAVIGATOR:

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It is available for Windows, macOS, and Linux.

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages, and use multiple environments to separate these different versions.

The command line program conda is both a package manager and an environment manager, to help data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages and update them, all inside Navigator.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- QTConsole
- Spyder
- VSCode
- Glueviz
- Orange 3 App
- Rodeo
- RStudio

Advanced conda users can also build your own Navigator applications. The simplest way is with Spyder. From the Navigator Home tab, click Spyder, and write and execute your code.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

You can also use Jupyter Notebooks the same way. Jupyter Notebooks are an increasingly popular system that combine your code, descriptive text, output, images and interactive interfaces into a single notebook file that is edited, viewed and used in a web browser.

Normally Navigator is used online, so that it can download and install packages. In online mode, Navigator must be able to reach these sites, so they may need to be whitelisted in your network's firewall settings.

- <https://repo.anaconda.com> (or for older versions of Navigator and Conda, <https://repo.anaconda.com>)
- <https://conda.anaconda.org> for conda-forge and other channels on Anaconda Cloud ([anaconda.org](https://conda.anaconda.org))
- <https://vscode-update.azurewebsites.net/> for updating Visual Studio Code
- google-public-dns-a.google.com (8.8.8.8:53) to check internet connectivity with Google Public DNS

If Navigator detects that internet access is not available, it automatically enables offline mode and displays this message:

Offline mode: Some of the functionality of Anaconda Navigator will be limited. Conda environment creation will be subject to the packages currently available on your package cache. Offline mode is indicated to the left of the login/logout button on the top right corner of the main application window. Offline mode will be disabled automatically when internet connectivity is restored. You can also manually force Offline mode by enabling the setting on the application preferences. In the Preferences dialog, select "Enable offline mode" to enter offline mode even if internet access is available. Using Navigator in offline mode is equivalent to using the command line conda commands create, install, remove, and update with the flag `--offline` so that conda does not connect to the internet.

Managing Environments: Navigator uses conda to create separate environments containing files, packages, and their dependencies that will not interact with other environments. Create a new environment named snowflakes and install a package in it:

1. In Navigator, click the Environments tab, then click the Create button. The Create new environment dialog box appears.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

2. In the Environment name field, type a descriptive name for your environment.

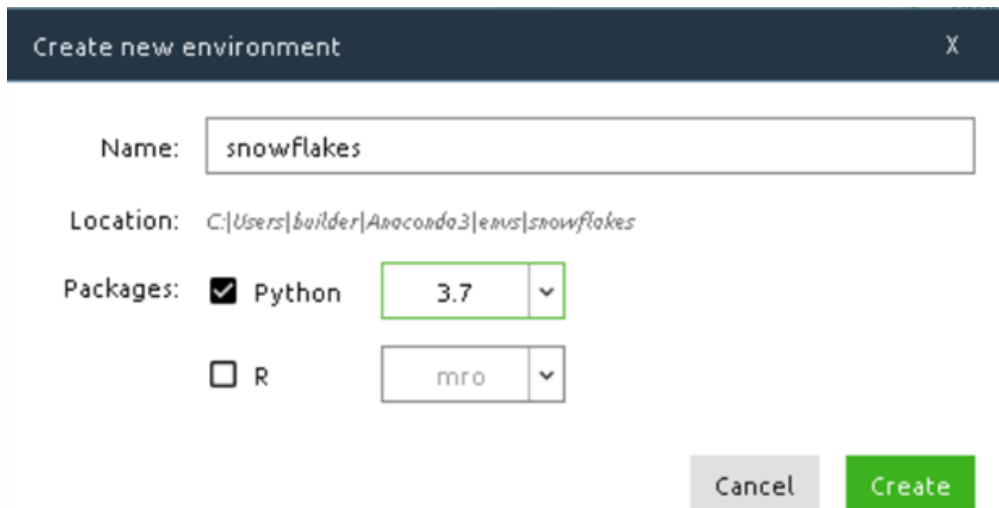


Fig 5.1: Creating Environment

3. Click Create. Navigator creates the new environment and activates it.

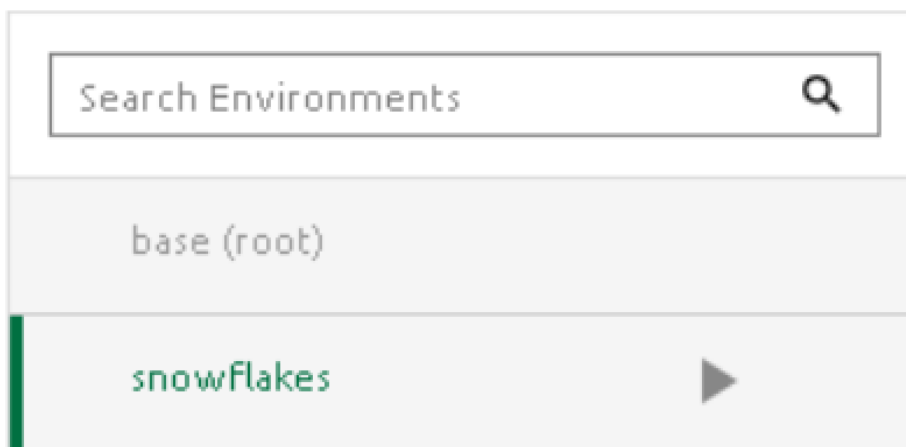


Fig 5.2: Activating Environment

4. Switch between them (activate and deactivate environments) by clicking the name of the environment you want to use.
5. Return to the other environment by clicking its name.

Managing Python: When you create a new environment, Navigator installs the same Python version you used when you downloaded and installed Anaconda. If you want to use a different version of Python, for example Python 3.5, simply create a new environment and specify the version of Python that you want in that environment. Create a new environment named “snakes” that contains Python 3.5:

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

1. In Navigator, click the Environments tab, then click the Create button. The Create new environment dialog box appears.
2. In the Environment name field, type the descriptive name “snakes” and select the version of Python you want to use from the Python Packages box (3.6, 3.5 or 2.7). Select a different version of Python than is in your other environments, base or snowflakes.
3. Click the Create button.
4. Activate the version of Python you want to use by clicking the name of that environment.

Managing Packages: In this section, you check which packages you have installed, check which are available, and look for a specific package and install it.

1. To find a package you have already installed, click the name of the environment you want to search. The installed packages are displayed in the right pane.
2. You can change the selection of packages displayed in the right pane at any time by clicking the drop-down box above it and selecting Installed, Not Installed, Updateable, Selected, or All.

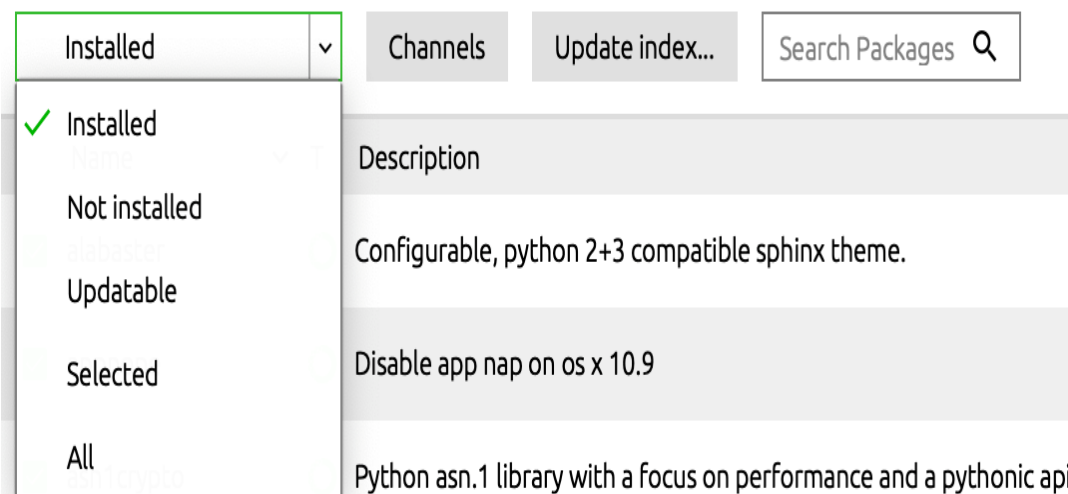
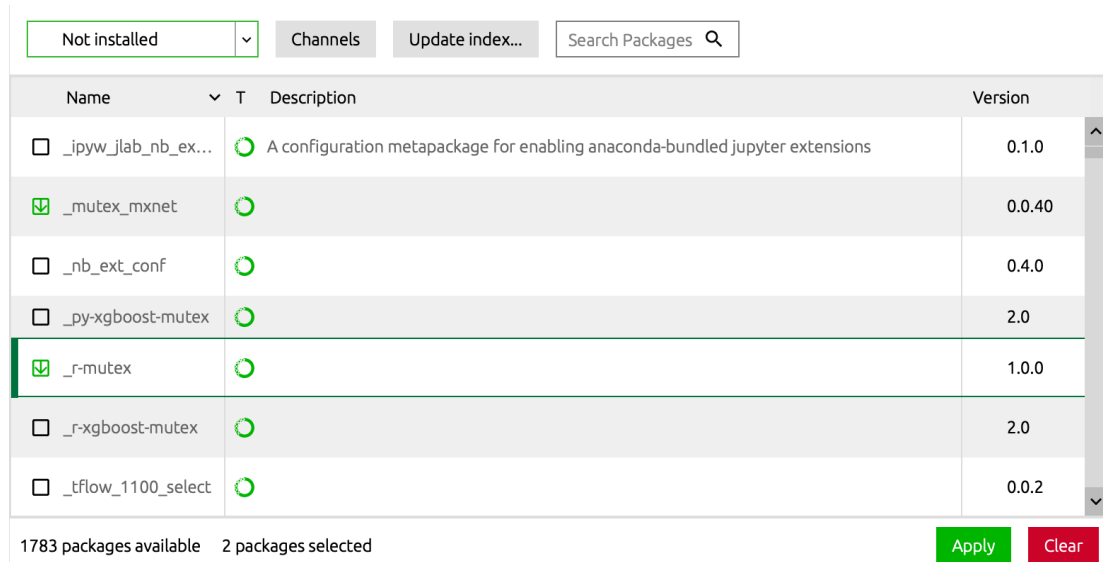


Fig 5.3: Installed Package Toolbar

3. Check to see if a package you have not installed named “beautifulsoup4” is available from the Anaconda repository (must be connected to the Internet). On the Environments tab, in the Search Packages box, type beautifulsoup4, and from the Search Subset box select All or Not Installed.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

- To install the package into the current environment, check the checkbox next to the package name, then click the bottom Apply button.



The screenshot shows a package manager interface. At the top, there is a dropdown menu set to 'Not installed', a 'Channels' button, an 'Update index...' button, and a search bar labeled 'Search Packages'. Below this is a table with columns for 'Name', 'Description', and 'Version'. The table lists several packages, with two selected (checkboxes checked): '_mutex_mxnet' (version 0.0.40) and '_r-mutex' (version 1.0.0). At the bottom, it shows '1783 packages available' and '2 packages selected', along with 'Apply' and 'Clear' buttons.

Name	Description	Version
<input type="checkbox"/> _ipyw_jlab_nb_ex...	A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
<input checked="" type="checkbox"/> _mutex_mxnet		0.0.40
<input type="checkbox"/> _nb_ext_conf		0.4.0
<input type="checkbox"/> _py-xgboost-mutex		2.0
<input checked="" type="checkbox"/> _r-mutex		1.0.0
<input type="checkbox"/> _r-xgboost-mutex		2.0
<input type="checkbox"/> _tflow_1100_select		0.0.2

1783 packages available 2 packages selected Apply Clear

Fig 5.4: Package Installation

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

5.1.2 JUPYTER NOTEBOOK:

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

A Jupyter Notebook can be converted to a number of open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python) through "Download As" in the web interface, via the nbconvert library or "jupyter nbconvert" command line interface in a shell.

To simplify visualisation of Jupyter notebook documents on the web, the nbconvert library is provided as a service through NbViewer which can take a URL to any publicly available notebook document, convert it to HTML on the fly and display it to the user.

Jupyter Notebook provides a browser-based REPL built upon a number of popular open-source libraries:

- IPython
- ØMQ
- Tornado (web server)
- jQuery
- Bootstrap (front-end framework)
- MathJax

Jupyter Notebook can connect to many kernels to allow programming in many languages. By default, Jupyter Notebook ships with the IPython kernel. As of the 2.3 release (October 2014), there are currently 49 Jupyter-compatible kernels for as many programming languages, including Python, R, Julia and Haskell.

The Notebook interface was added to IPython in the 0.12 release (December 2011), renamed to Jupyter notebook in 2015 (IPython 4.0 – Jupyter 1.0). Jupyter Notebook is similar to the notebook interface of other programs such as Maple, Mathematica, and SageMath, a computational interface style that originated with Mathematica in the 1980s. According to The Atlantic, Jupyter interest overtook the popularity of the Mathematica notebook interface in early 2018.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

1. Jupyter Kernels: A Jupyter kernel is a program responsible for handling various types of request (code execution, code completions, inspection), and providing a reply. Kernels talk to the other components of Jupyter using ZeroMQ over the network, and thus can be on the same or remote machines. Unlike many other Notebook-like interfaces, in Jupyter, kernels are not aware that they are attached to a specific document, and can be connected to many clients at once. Usually kernels allow execution of only a single language, but there are a couple of exceptions.

By default, Jupyter ships with IPython as a default kernel and a reference implementation via the ipykernel wrapper. Kernels for many languages having varying quality and features are available.

2. JupyterHub: JupyterHub is a multi-user server for Jupyter Notebooks. It is designed to support many users by spawning, managing, and proxying many singular Jupyter Notebook servers. While JupyterHub requires managing servers, third-party services like JupyterLab provide an alternative to JupyterHub by hosting and managing multi-user Jupyter notebooks in the cloud.
3. JupyterLab: JupyterLab is the next-generation user interface for Project Jupyter. It offers all the familiar building blocks of the classic Jupyter Notebook (notebook, terminal, text editor, file browser, rich outputs, etc.) in a flexible and powerful user interface. The first stable release was announced on February 20, 2018, and in December 2018 it was adopted as the primary interface for the cloud-based Jupyter service JupyterLab.

Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

The Jupyter Notebook is not included with Python, so if you want to try it out, you will need to install Jupyter. There are many distributions of the Python language. This article will focus on just two of them for the purposes of installing Jupyter Notebook. The most popular is CPython, which is the reference version of Python that you can get from their website. It is also assumed that you are using Python 3.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

When you are working with Jupyter Notebooks, you will find that you need to share your results with non-technical people. When that happens, you can use the nbconvert tool which comes with Jupyter Notebook to convert or export your Notebook into one of the following formats:

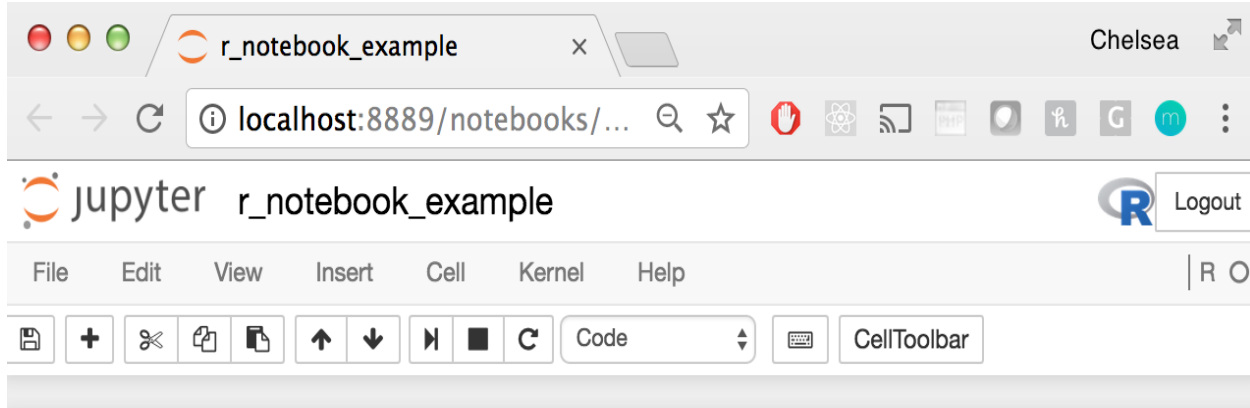
- HTML
- LaTeX
- PDF
- RevealJS
- Markdown
- ReStructured Text
- Executable script

The nbconvert tool uses Jinja templates under the covers to convert your Notebook files (.ipynb) into these other formats.

Jinja is a template engine that was made for Python. Also note that nbconvert also depends on Pandoc and TeX to be able to export to all the formats above. If you don't have one or more of these, some of the export types may not work. For more information, you should check out the documentation.

The Notebook Dashboard is the component which is shown first when you launch Jupyter Notebook App. The Notebook Dashboard is mainly used to open notebook documents, and to manage the running kernels (visualize and shutdown).

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK



```
In [5]: library(plotly)
set.seed(100)
d <- diamonds[sample(nrow(diamonds), 1000), ]
plot_ly(d, type = 'scatter', mode = 'markers',
        x = ~carat, y = ~price,
        color = ~carat, size = ~carat,
        text = ~paste("Clarity: ", clarity))
```

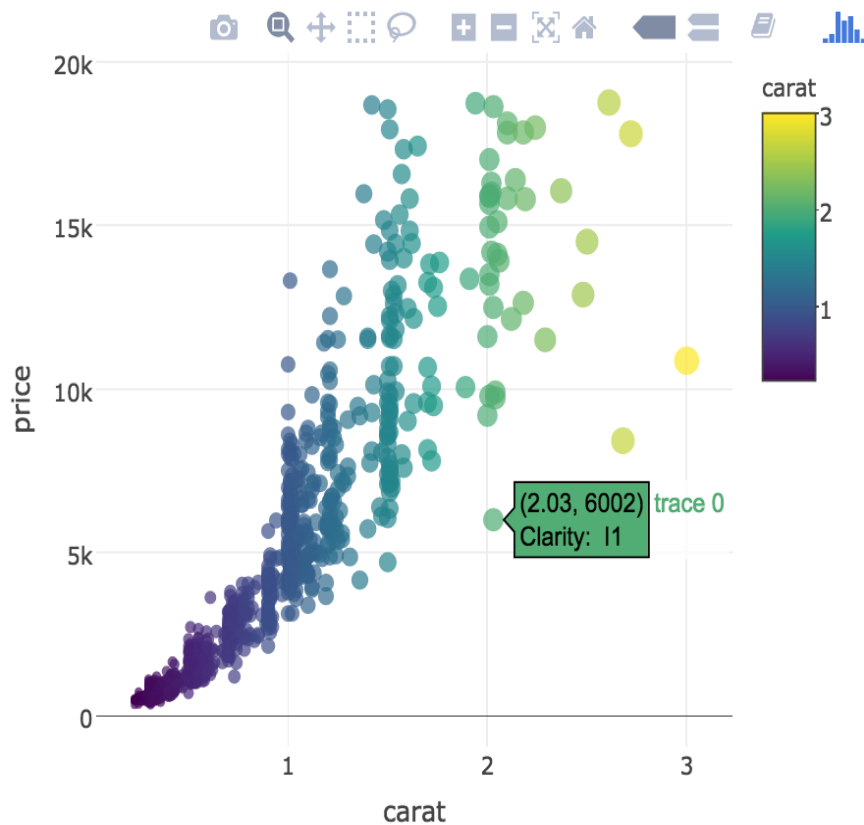


Fig 5.5: Jupyter Notebook Example For Plotting

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

The Notebook Dashboard has other features similar to a file manager, namely navigating folders and renaming/deleting files.

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

A web application: a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.

Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

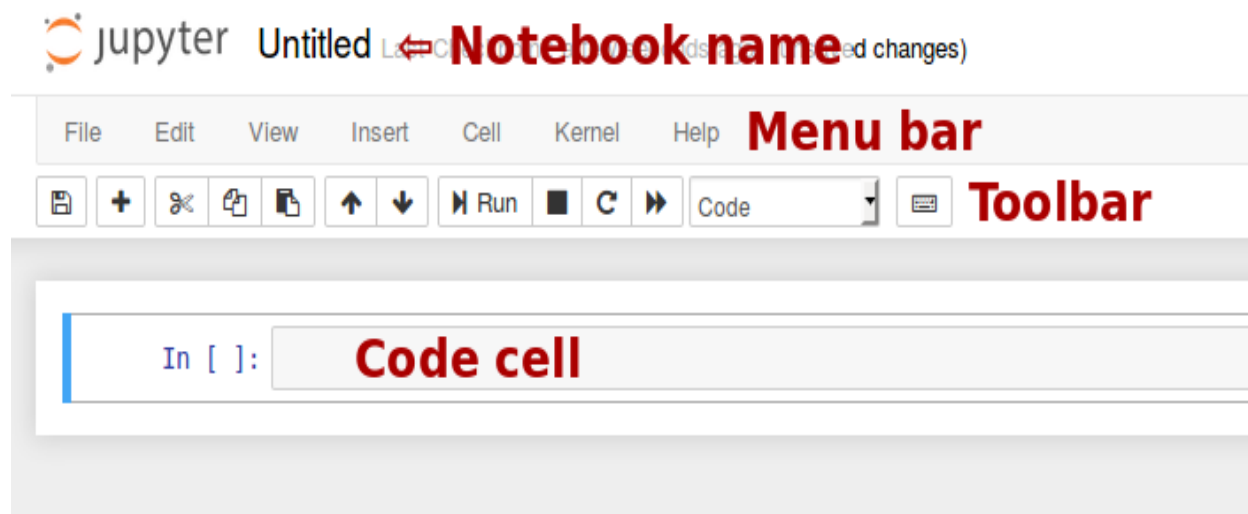


Fig 5.6: Jupyter Notebook Interface

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

5.1.3 CORE PYTHON:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural. It also has a comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter, map, and reduce functions; list comprehensions, dictionaries, sets and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

- Simple is better than complex
- Complex is better than complicated
- Readability counts.

Python's development is conducted largely through the Python Enhancement Proposal (PEP) process, the primary mechanism for proposing major new features, collecting community input on issues and documenting Python design decisions. Python coding style is covered in PEP 8. Outstanding PEPs are reviewed and commented on by the Python community and Guido Van Rossum, Python's Benevolent Dictator for Life.

Enhancement of the language corresponds with development of the CPython reference implementation. The mailing list python-dev is the primary forum for the language's development. Specific issues are discussed in the Roundup bug tracker maintained at python.org. Development originally took place on a self-hosted source-code repository running Mercurial, until Python moved to GitHub in January 2017.

CPython's public releases come in three types, distinguished by which part of the version number is incremented:

- Backward-incompatible versions, where code is expected to break and need to be manually ported. The first part of the version number is incremented. These releases happen infrequently—for example, version 3.0 was released 8 years after 2.0.
- Major or "feature" releases, about every 18 months, are largely compatible but introduce new features. The second part of the version number is incremented. Each major version is supported by bugfixes for several years after its release.
- Bugfix releases, which introduce no new features, occur about every 3 months and are made when a sufficient number of bugs have been fixed upstream since the last release. Security vulnerabilities are also patched in these releases. The third and final part of the version number is incremented.

Many alpha, beta, and release-candidates are also released as previews and for testing before final releases. Although there is a rough schedule for each release, they are often delayed if the code is not ready. Python's development team monitors the state of the code by running the large unit test suite during development, and using the BuildBot continuous integration system.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

The community of Python developers has also contributed over 86,000 software modules (as of 20 August 2016) to the Python Package Index (PyPI), the official repository of third-party Python libraries. The major academic conference on Python is PyCon. There are also special Python mentoring programme, such as Pyladies.

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, `SpamClass()` or `EggsClass()`), and the classes are instances of the metaclass type (itself an instance of itself), allowing metaprogramming and reflection.

Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long-term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named `mypy` supports compile-time type checking.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

Python 3 The standard type hierarchy

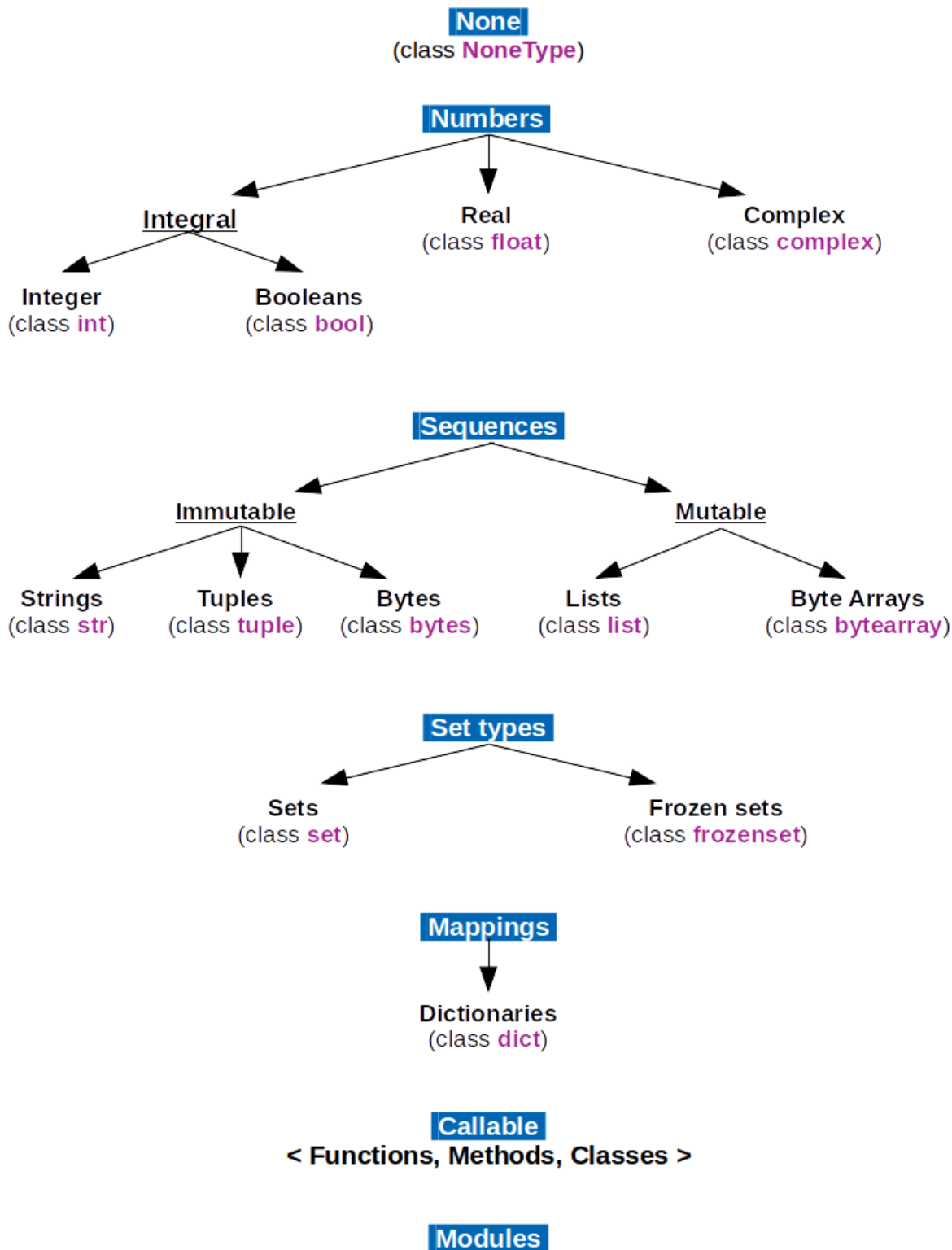


Fig 5.7: Python Type Hierarchy

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

```
1 # This Python 3 environment comes with many helpful analytics libraries installed
2 # It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
3 # For example, here's several helpful packages to load in
4
5 import numpy as np # linear algebra
6 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
7
8 # Input data files are available in the "../input/" directory.
9 # For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory
10
11 import os
12 print(os.listdir("../input"))
13
14 # Any results you write to the current directory are saved as output.
15 from sklearn import preprocessing
16 from sklearn_pandas import CategoricalImputer
17
18 titanic_train = pd.read_csv("../input/train.csv")
19 titanic_train.shape
20 titanic_train.info()
21
22 titanic_test = pd.read_csv("../input/test.csv")
23 titanic_test.shape
24
25 titanic_all = pd.concat([titanic_train, titanic_test])
26 titanic_all.shape
27 titanic_all.info()
28
29 #impute missing values for continuous features
30 imputable_cont_features = ['Age', 'Fare']
31 cont_imputer = preprocessing.Imputer()
32 cont_imputer.fit(titanic_all[imputable_cont_features])
33 titanic_all[imputable_cont_features] = cont_imputer.transform(titanic_all[imputable_cont_features])
34
35 #impute missing values for categorical features
36 cat_imputer = CategoricalImputer()
37 cat_imputer.fit(titanic_all['Embarked'])
38 titanic_all['Embarked'] = cat_imputer.transform(titanic_all['Embarked'])
39
40 titanic_all['FamilySize'] = titanic_all['SibSp'] + titanic_all['Parch'] + 1
```

Fig 5.8: Python Sample Kernel Code

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

5.1.4 KERAS LSTM MODEL:

The above drawback of RNN pushed the scientists to develop and invent a new variant of the RNN model, called Long Short-Term Memory. LSTM can solve this problem, because it uses gates to control the memorizing process.

Let's understand the architecture of LSTM and compare it with that of RNN:

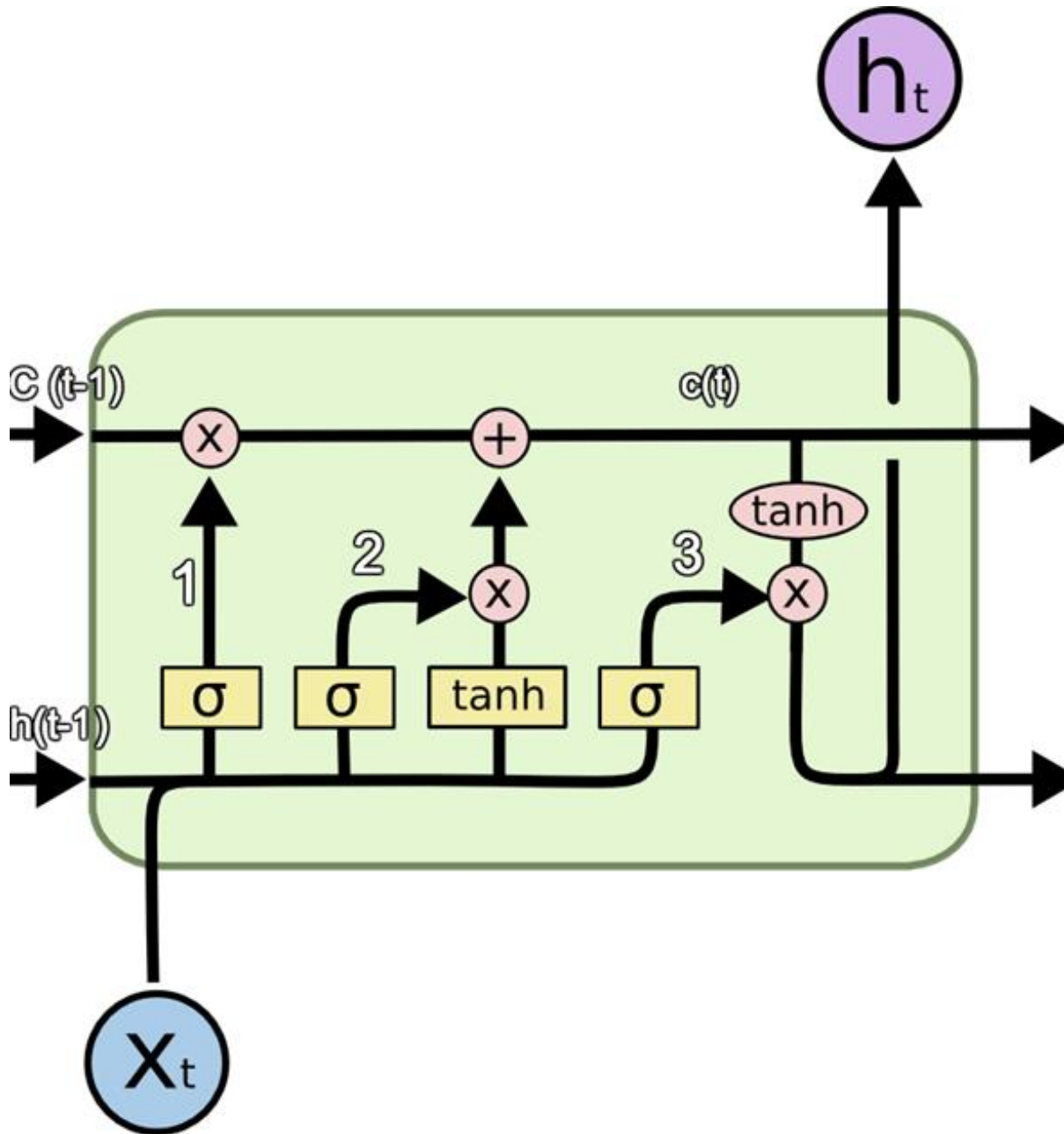


Fig 5.9: LSTM Model implemented using Keras

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

The symbols used here have following meaning:

- X : Scaling of information
- $+$: Adding information
- σ : Sigmoid layer
- \tanh : tanh layer
- $h(t-1)$: Output of last LSTM unit
- $c(t-1)$: Memory from last LSTM unit
- $X(t)$: Current input
- $c(t)$: New updated memory
- $h(t)$: Current output

Why tanh? To overcome the vanishing gradient problem, we need a function whose second derivative can sustain for a long range before going to zero. tanh is a suitable function with the above property.

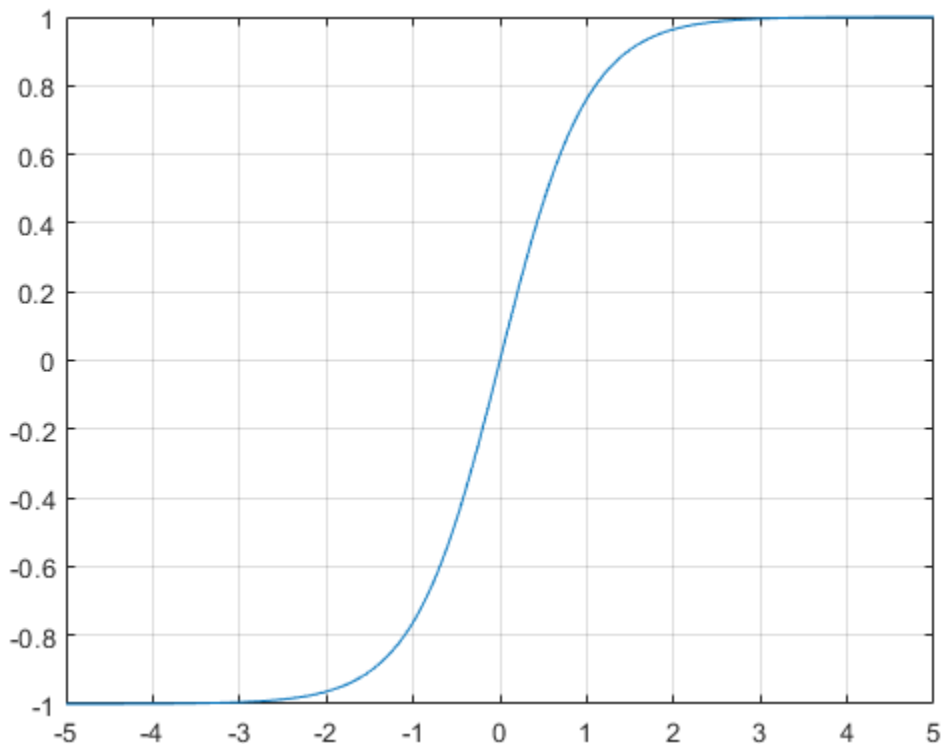


Fig 5.10: tanh Function Graph

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

Why Sigmoid? As Sigmoid can output 0 or 1, it can be used to forget or remember the information. Information passes through many such LSTM units.

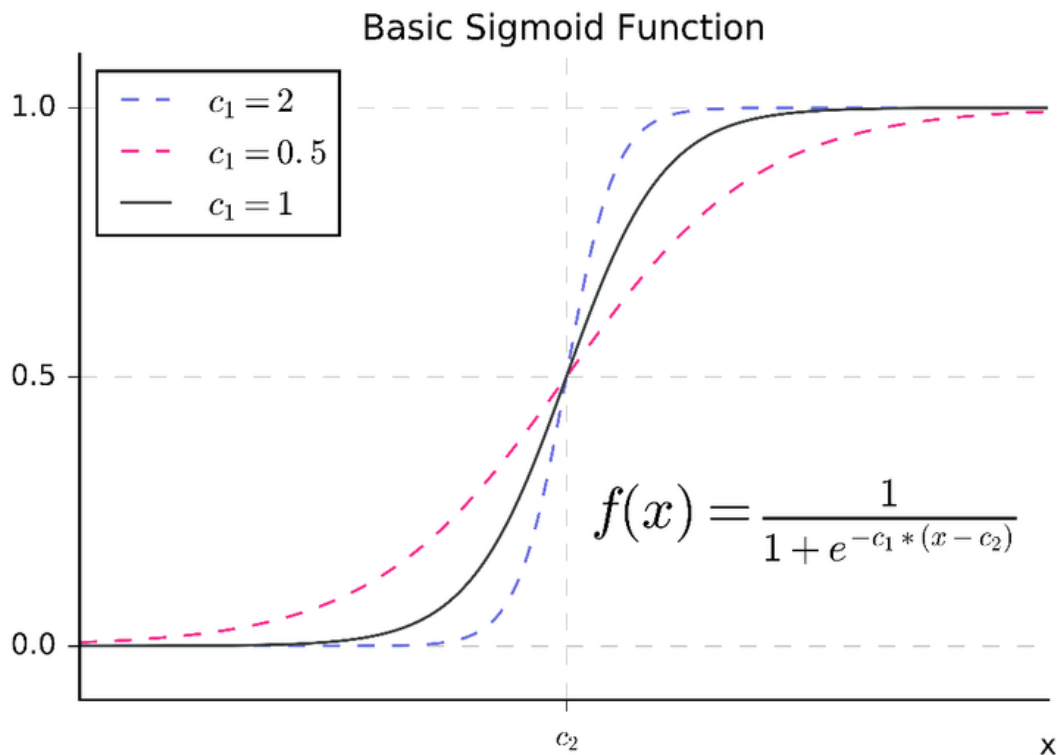


Fig 5.11: Sigmoid Function Graph

There are three main components of an LSTM unit:

1. LSTM has a special architecture which enables it to forget the unnecessary information. The sigmoid layer takes the input $X(t)$ and $h(t-1)$ and decides which parts from old output should be removed (by outputting a 0). In our example, when the input is 'He has a female friend Maria', the gender of 'David' can be forgotten because the subject has changed to 'Maria'. This gate is called forget gate $f(t)$. The output of this gate is $f(t)*c(t-1)$
2. The next step is to decide and store information from the new input $X(t)$ in the cell state. A Sigmoid layer decides which of the new information should be updated or ignored. A tanh layer creates a vector of all the possible values from the new input. These two are multiplied to update the new cell state. This new memory is then added to old memory $c(t-1)$ to give $c(t)$. In our example, for the new input 'He has a female friend Maria', the gender of Maria will be updated. When the input is 'Maria works as a cook in a famous restaurant in New York whom he met recently in a school alumnus meet', the words like 'famous',

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

‘school alumni meet’ can be ignored and words like ‘cook, ‘restaurant’ and ‘New York’ will be updated.

3. Finally, we need to decide what we’re going to output. A sigmoid layer decides which parts of the cell state we are going to output. Then, we put the cell state through a tanh generating all the possible values and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to. In our example, we want to predict the blank word, our model knows that it is a noun related to ‘cook’ from its memory, it can easily answer it as ‘cooking’. Our model does not learn this answer from the immediate dependency, rather it learnt it from long term dependency.

We just saw that there is a big difference in the architecture of a typical RNN and a LSTM. In LSTM, our model learns what information to store in long term memory and what to get rid of.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

5.1.5 VISUAL STUDIO CODE

Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences. The source code is free and open source and released under the permissive MIT License. The compiled binaries are freeware and free for private or commercial use.

Visual Studio Code is based on Electron, a framework which is used to deploy Node.js applications for the desktop running on the Blink layout engine. Although it uses the Electron framework, the software does not use Atom and instead employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services).

In the Stack Overflow 2019 Developer Survey, Visual Studio Code was ranked the most popular developer environment tool, with 50.7% of 87,317 respondents claiming to use it.

Visual Studio Code was announced on April 29, 2015, by Microsoft at the 2015 Build conference. A Preview build was released shortly thereafter.

On November 18, 2015, Visual Studio Code was released under the MIT License and its source code posted to GitHub. Extension support was also announced.

On April 14, 2016, Visual Studio Code graduated the public preview stage and was released to web.

Visual Studio Code is a source code editor that can be used with a variety of programming languages. Instead of a project system it allows users to open one or more directories, which can then be saved in workspaces for future reuse. This allows it to operate as a language-agnostic code editor for any language, contrary to Microsoft Visual Studio which uses the proprietary .sln solution file and project-specific project files. It supports a number of programming languages and a set of features that differs per language. Unwanted files and folders can be excluded from the project tree via the settings. Many of Visual Studio Code features are not exposed through menus or the user interface, but can be accessed via the command palette.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

Visual Studio Code can be extended via plug-ins, available through a central repository. This includes additions to the editor and language support. A notable feature is the ability to create extensions that add support for new languages, themes, debuggers, perform static code analysis, add code linters, using the Language Server Protocol and connect to additional services.

Visual Studio Code includes multiple extensions for FTP, allowing the software to be used as a free alternative for web development. Code can be synced between the editor and the server, without downloading any extra software.

Visual Studio Code allows users to set the code page in which the active document is saved, the newline character for Windows/Linux, and the programming language of the active document. This allows it to be used on any platform, in any locale, and for any given programming language.

Visual Studio Code has out-of-the-box support for almost every major programming language. Several are included by default, for example, JavaScript, TypeScript, CSS, and HTML but other language extensions can be found and downloaded for free from the VS Code Marketplace.

In the 2016 Developers Survey of Stack Overflow, Visual Studio Code ranked #13 among the top popular development tools, with only 7.2% of the 46,613 respondents using it. However, in the 2019 Developers Survey, Visual Studio Code was ranked #1, with 50.7% of the 87,317 respondents using it.

Visual Studio Code is widely reviewed to be fast and lightweight, and is considered to be flexible across various domains such as Java, JavaScript, Go, Node.js and even C++.

Visual Studio Code collects usage data and sends it to Microsoft, although this telemetry reporting can be disabled. The data is shared among Microsoft-controlled affiliates and subsidiaries and with law enforcement, per the privacy statement. Because of the open-source nature of the app, it is known exactly what is collected. Up stream's binary is shipped under a proprietary license.

VSCodium is an alternative binary distribution of the software which uses only the open-source parts and omits Microsoft's trademarks and the telemetry component, while remaining fully functional and compatible in all other regards.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

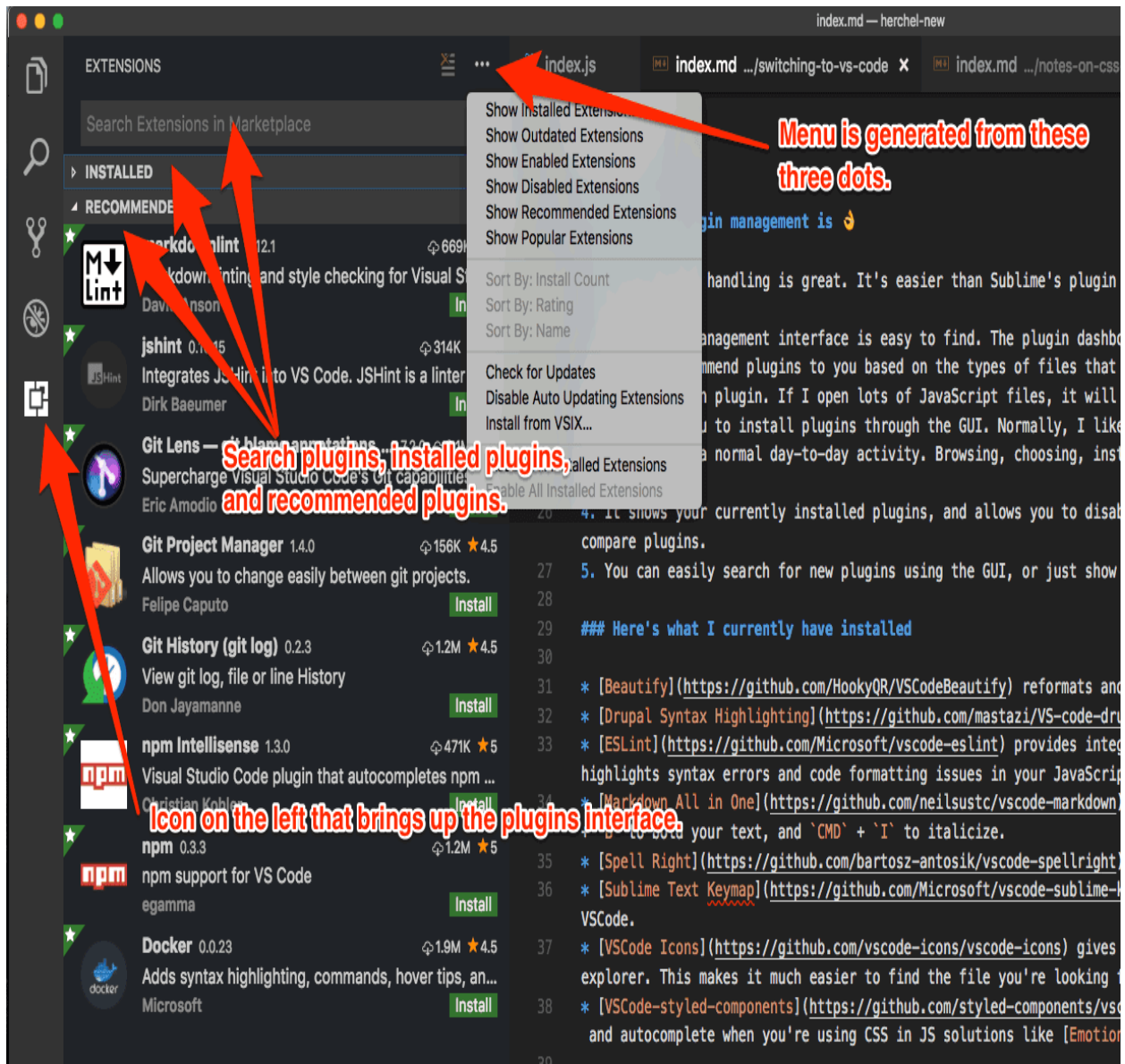


Fig 5.12: VSCode Interface

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

5.2 Flow Chart

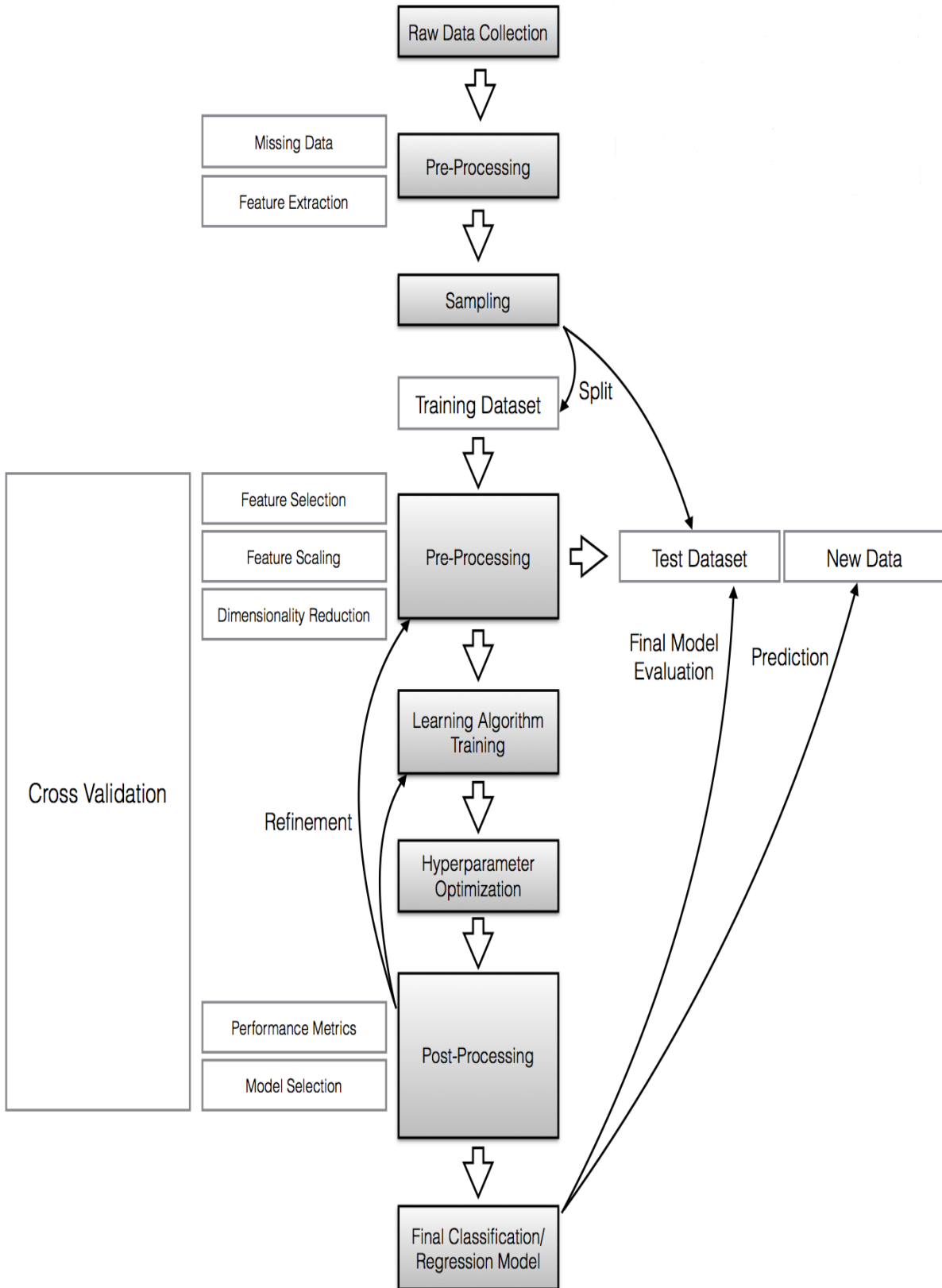


Fig 5.13: Supervised Learning Flow Chart

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

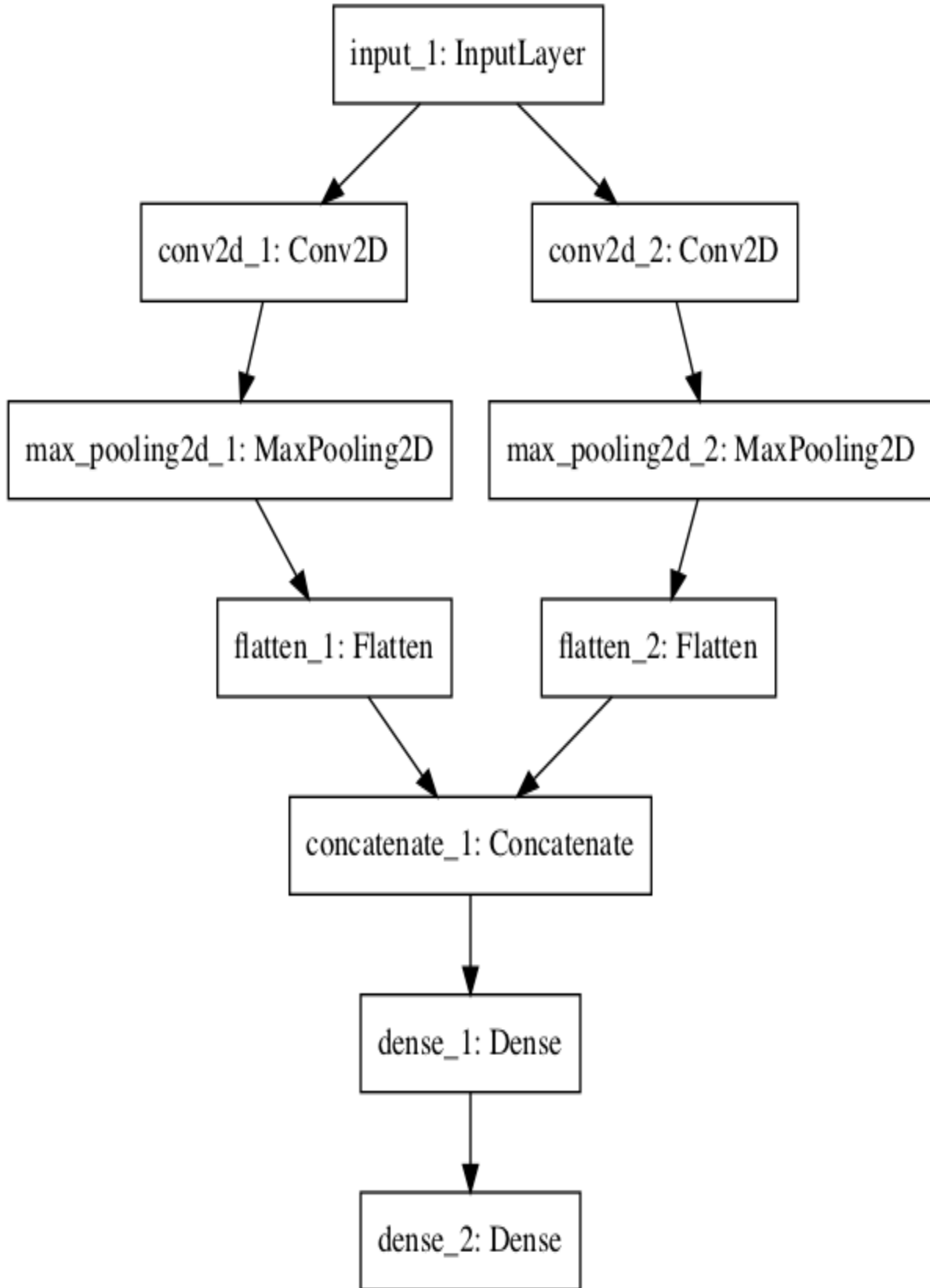


Fig 5.14: Recurrent Neural Network with shared inputs

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

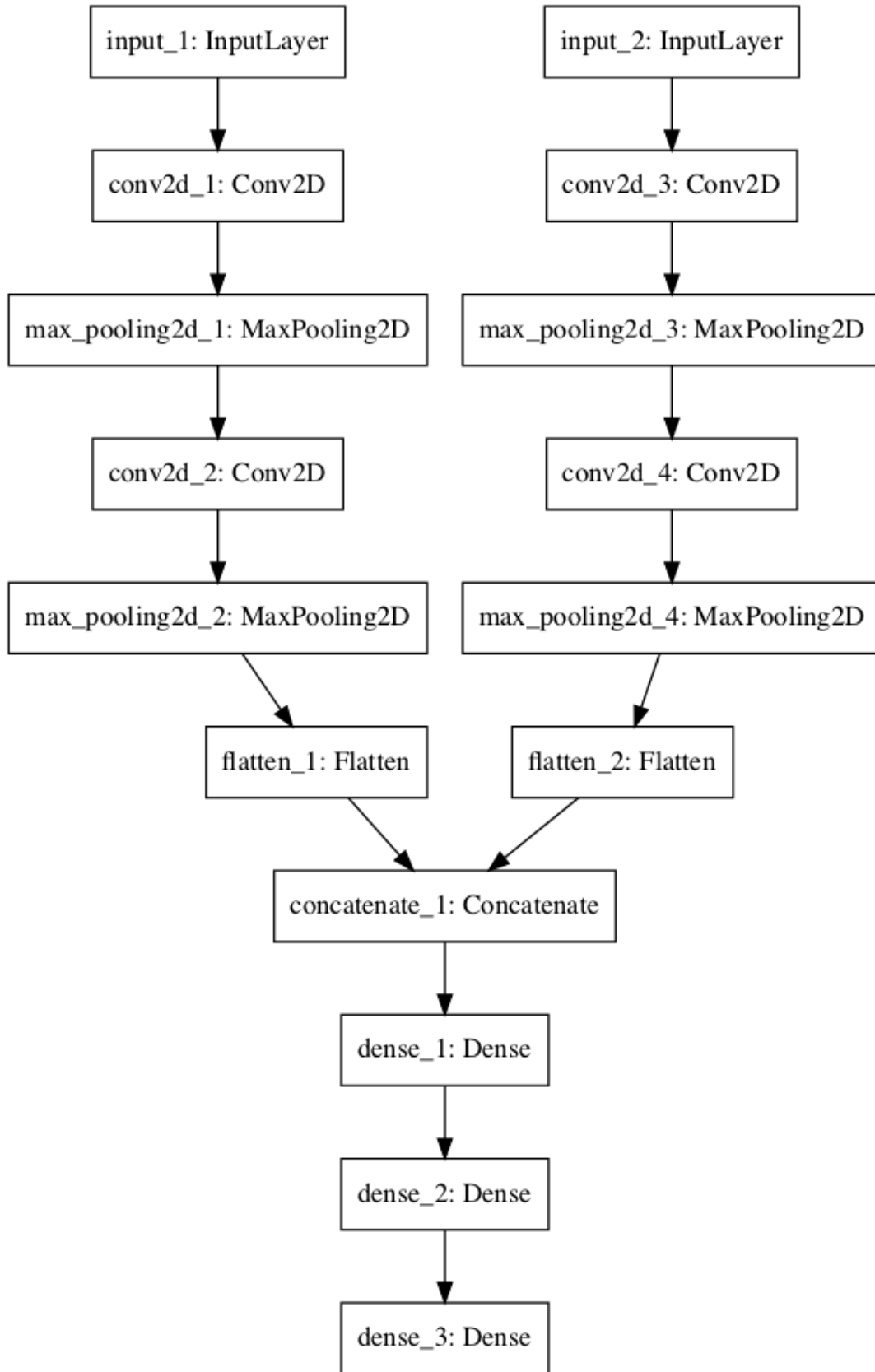


Fig 5.15: Recurrent Neural Network with multiple inputs

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

5.3 Sample Code

5.3.1 Dependencies to be imported

```
import numpy
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader.data as web
import datetime as dt
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

5.3.2 Gathering data from yahoo finance

```
start = dt.datetime(2018,1,29)
end = dt.datetime.today()
stock = 'AAPL' #Id for Apple Inc.
df = web.DataReader(stock, 'yahoo', start, end)
data_source = r'C:\Users\saineni\Desktop\Projects\Anaconda\stock
Estimation\Applestock.csv' #input your path
df.to_csv(data_source)
```

5.3.3 Converting this dataset matrix into an array of values

```
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a) #inputs
        dataY.append(dataset[i+look_back, 0]) #labels
    return numpy.array(dataX), numpy.array(dataY)
```

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

```
#Fixing Randoms
numpy.random.seed(7)

#load saved data set
dataframe = pd.read_csv('Applestock.csv', usecols=[1],engine='python', skipfooter=3)
dataset = dataframe.values
dataset=dataset.astype('float32')
```

5.3.4 Normalizing dataset and splitting

```
#normalize dataset
scaler = MinMaxScaler(feature_range=(0,1))
dataset = scaler.fit_transform(dataset)

#splitting data sets
splitting_ratio=0.64
train_size = int(len(dataset)*splitting_ratio)
test_size= len(dataset) - train_size
train,test = dataset[0:train_size,:],dataset[train_size:len(dataset),:]
```

5.3.5 Reshaping the data

```
#Reshaping
#reshape into x=t and y=t+1
look_back=3
trainX,trainY = create_dataset(train,look_back)
testX, testY = create_dataset(test,look_back)

#reshape input to be [sample,time steps, feature]
trainX= numpy.reshape(trainX,(trainX.shape[0],1,trainX.shape[1]))
testX = numpy.reshape(testX,(testX.shape[0],1,testX.shape[1]))
```

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

5.3.6 LSTM network for accuracy improvement

```
#LSTM network for accuracy improvement using keras
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2) #epoch
h is a unix timestamp
```

5.3.7 Making Predictions and improving accuracy

```
#making predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

#increasing accuracy

#invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

#calculating RMS error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[: , 0]))
print('Train Score: %.2f RMSE' % (trainScore))
```

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

```
testScore = math.sqrt(mean_squared_error(testY[0], testPredict
[:,0]))
print('Test Score: %.2f RMSE' %(testScore))

#shifting train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

#shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-
1, :] = testPredict
```

5.3.8 Plotting the data

```
#plotting
accuracy = 1-((trainScore)/(trainScore+testScore))
print(accuracy)
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

5.4 Screenshots

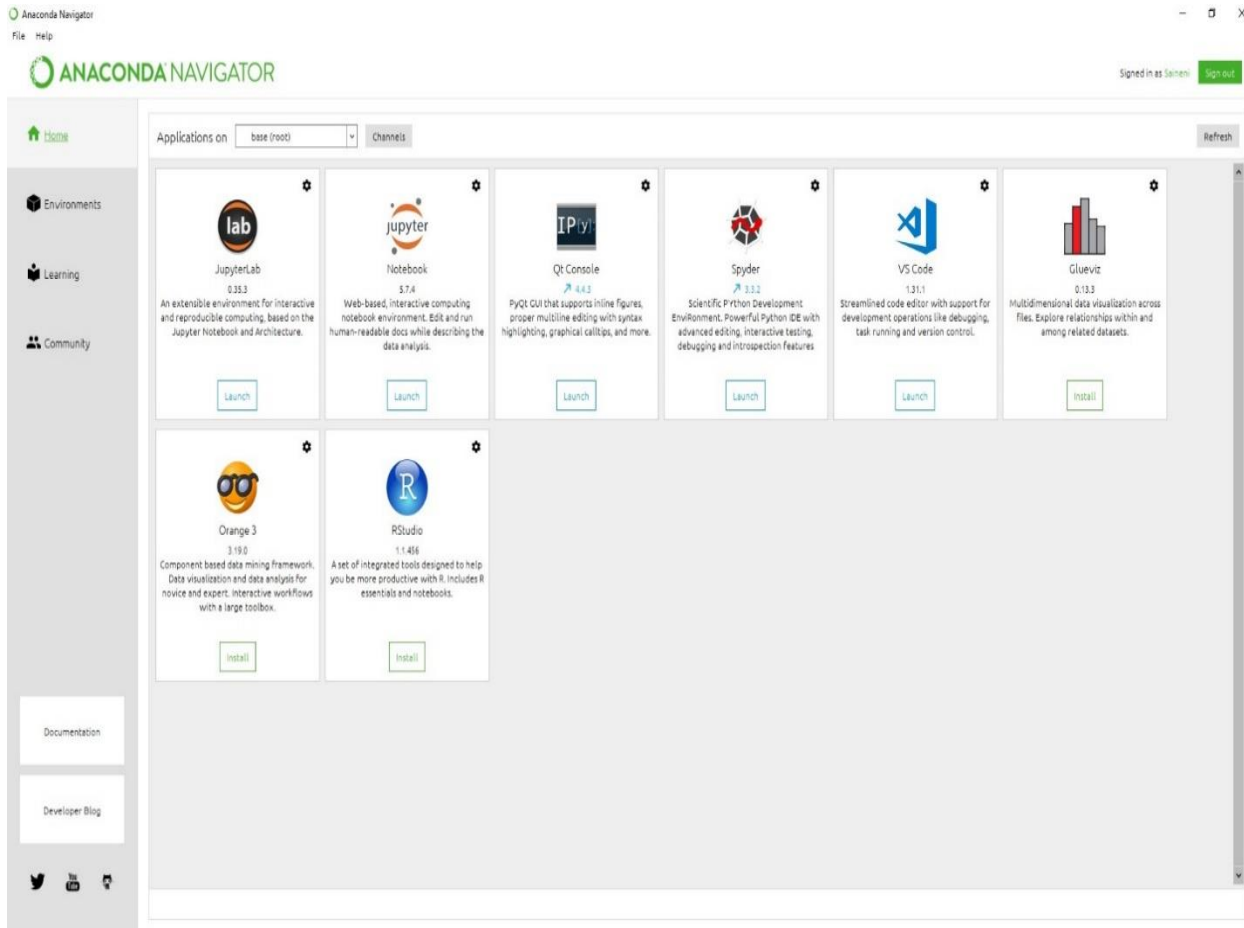


Fig 5.16: Anaconda Navigator

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

Date	High	Low	Open	Close	Volume	Adj Close
1986-01-28 00:00:00	0.399554	0.392857	0.395089	0.397321	55574400	0.017882
1986-01-29 00:00:00	0.435268	0.392857	0.397321	0.421875	1.47E+08	0.018987
1986-01-30 00:00:00	0.419643	0.408482	0.419643	0.410714	59220000	0.018485
1986-01-31 00:00:00	0.415179	0.408482	0.410714	0.412946	36926400	0.018585
1986-02-03 00:00:00	0.428571	0.408482	0.412946	0.426339	87505600	0.019188
1986-02-04 00:00:00	0.435268	0.424107	0.426339	0.424107	65044000	0.019087
1986-02-05 00:00:00	0.426339	0.419643	0.424107	0.424107	49291200	0.019087
1986-02-06 00:00:00	0.433036	0.421875	0.424107	0.430804	33555200	0.019389
1986-02-07 00:00:00	0.430804	0.419643	0.430804	0.428571	32351200	0.019288
1986-02-10 00:00:00	0.4375	0.424107	0.428571	0.426339	27960800	0.019188
1986-02-11 00:00:00	0.428571	0.419643	0.426339	0.426339	38365600	0.019188
1986-02-12 00:00:00	0.428571	0.424107	0.426339	0.428571	33264000	0.019288
1986-02-13 00:00:00	0.428571	0.424107	0.428571	0.426339	27344800	0.019188
1986-02-14 00:00:00	0.430804	0.424107	0.426339	0.424107	34378400	0.019087
1986-02-18 00:00:00	0.428571	0.415179	0.424107	0.426339	37027200	0.019188
1986-02-19 00:00:00	0.455357	0.426339	0.426339	0.446429	89919200	0.020092
1986-02-20 00:00:00	0.453125	0.444196	0.446429	0.448661	34479200	0.020192
1986-02-21 00:00:00	0.459821	0.448661	0.448661	0.450893	47269600	0.020293
1986-02-24 00:00:00	0.459821	0.446429	0.450893	0.459821	61779200	0.020695
1986-02-25 00:00:00	0.470982	0.448661	0.459821	0.470982	56184800	0.021197
1986-02-26 00:00:00	0.477679	0.464286	0.470982	0.464286	41182400	0.020896
1986-02-27 00:00:00	0.466518	0.455357	0.464286	0.457589	27031200	0.020594
1986-02-28 00:00:00	0.462054	0.444196	0.457589	0.446429	31281600	0.020092
1986-03-03 00:00:00	0.448661	0.4375	0.446429	0.439732	27204800	0.019791
1986-03-04 00:00:00	0.446429	0.4375	0.439732	0.439732	22276800	0.019791
1986-03-05 00:00:00	0.455357	0.433036	0.439732	0.450893	44256800	0.020293
1986-03-06 00:00:00	0.459821	0.448661	0.450893	0.453125	25334400	0.020393
1986-03-07 00:00:00	0.453125	0.441964	0.453125	0.441964	24046400	0.019891
1986-03-10 00:00:00	0.444196	0.439732	0.441964	0.439732	18872000	0.019791
1986-03-11 00:00:00	0.444196	0.4375	0.439732	0.444196	25765600	0.019991
1986-03-12 00:00:00	0.448661	0.441964	0.444196	0.441964	21420000	0.019891
1986-03-13 00:00:00	0.446429	0.435268	0.441964	0.441964	28991200	0.019891
1986-03-14 00:00:00	0.46875	0.441964	0.441964	0.466518	96213600	0.020996
1986-03-17 00:00:00	0.464286	0.453125	0.464286	0.464286	29680000	0.020896
1986-03-18 00:00:00	0.486607	0.462054	0.464286	0.479911	62339200	0.021599
1986-03-19 00:00:00	0.486607	0.470982	0.479911	0.473214	47471200	0.021297
1986-03-20 00:00:00	0.529018	0.5	0.5	0.504464	2.26E+08	0.022704

Fig 5.17: Apple Inc. stock taken from yahoo finance

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

[158.16]
[151.55]
[157.23]
[156.77]
[158.52]
[159.36]
[158.85]
[145.72]
[148.55]
[148.83]
[151.82]
[154.53]
[153.97]
[153.7]
[151.27]
[153.39]
[155.88]
[157.66]
[157.88]
[156.73]
[155.14]
[154.48]
[158.13]
[156.33]
[158.13]]

Fig 5.18: Dataset values after fixing random values

[0.30598295]
[0.2662108]
[0.25789177]
[0.24854696]
[0.24763525]
[0.18678069]
[0.14176643]
[0.06643879]
[0.13116801]
[0.12592602]
[0.14586902]
[0.15544164]
[0.14962971]
[0.]
[0.03225076]
[0.03544164]
[0.06951571]
[0.1003989]
[0.09401715]
[0.09094012]
[0.06324792]
[0.08740735]
[0.11578357]
[0.13606846]
[0.13857555]
[0.12547004]
[0.10735047]
[0.09982896]
[0.14142454]
[0.12091172]
[0.14142454]]

Fig 5.19: Normalized Values

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

```
Epoch 75/100
- 0s - loss: 8.8429e-04
Epoch 76/100
- 0s - loss: 8.2828e-04
Epoch 77/100
- 0s - loss: 7.8846e-04
Epoch 78/100
- 0s - loss: 8.2018e-04
Epoch 79/100
- 0s - loss: 8.7865e-04
Epoch 80/100
- 0s - loss: 8.3552e-04
Epoch 81/100
- 0s - loss: 8.7452e-04
Epoch 82/100
- 0s - loss: 8.1587e-04
Epoch 83/100
- 0s - loss: 8.5981e-04
Epoch 84/100
- 0s - loss: 9.2640e-04
Epoch 85/100
- 0s - loss: 7.9208e-04
Epoch 86/100
- 0s - loss: 7.8768e-04
Epoch 87/100
- 0s - loss: 8.1855e-04
Epoch 88/100
- 0s - loss: 8.1655e-04
Epoch 89/100
- 0s - loss: 8.0088e-04
Epoch 90/100
- 0s - loss: 8.4861e-04
Epoch 91/100
- 0s - loss: 8.0525e-04
Epoch 92/100
- 0s - loss: 8.2976e-04
Epoch 93/100
- 0s - loss: 8.2400e-04
Epoch 94/100
- 0s - loss: 8.4806e-04
Epoch 95/100
- 0s - loss: 8.3568e-04
Epoch 96/100
- 0s - loss: 8.0393e-04
Epoch 97/100
- 0s - loss: 8.0153e-04
Epoch 98/100
- 0s - loss: 8.2722e-04
Epoch 99/100
- 0s - loss: 8.5977e-04
Epoch 100/100
- 0s - loss: 8.2172e-04
```

```
Out[11]: <keras.callbacks.History at 0x2042a161978>
```

Fig 5.20: Dataset values after applying LSTM Model

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

0.6292503906939901

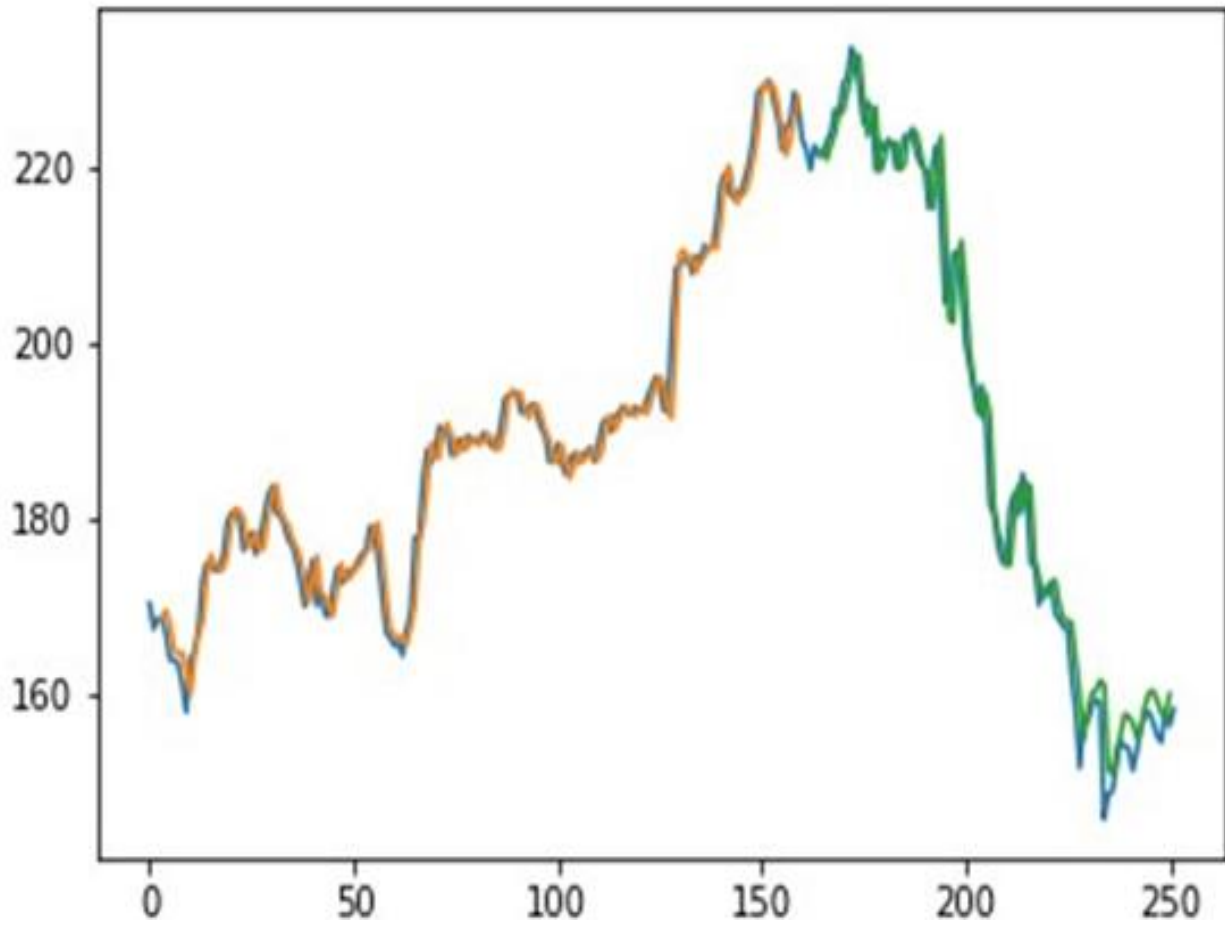


Fig 5.21: Plotted graph for trained vs test sets

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

6. Testing

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data should be chosen such that it passed through all possible condition. The following is the description of the testing strategies, which were carried out during the testing period.

6.1 Types of Test

System testing is normally carried out in a planned manner according to the system test plan document. The system test plan identifies all testing-related activities that must be performed, specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case. Here the modules are integrated in a planned manner.

6.1.1 FUNCTIONAL TESTING:

Functional testing refers to tests that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work". Some examples of functional testing done in our project:

1. By checking all the Connection modules, it is ensured that connection is success full every time when its connected.
2. All the modules are ensured to work properly after connection is successful

6.1.2 STRUCTURAL TESTING:

Structural testing is also called White box testing. This means a testing technique whereby explicit knowledge of the internal workings of the item being tested is used to select the test data. White box testing uses specific knowledge of programming code to examine outputs. The test is accurate only if the tester knows what the program is supposed to do. He or she can then see if the program diverges from its intended goal. White box testing does not account for errors caused by omission, and all visible code must also be readable.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

6.1.3 MODULE TESTING:

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus, all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example, the matplotlib module is tested separately. This module is tested with different sets of data and its approximate execution time and the result of the test is compared with the results that are prepared manually. Each module in the system is tested separately. In this system the modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

6.1.4 INTEGRATION TESTING:

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct.

6.1.5 SYSTEM TESTING

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limiting type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

Usually, software is only one element of a larger computer-based system. Ultimately, software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system. That is a very basic description of what is involved in system testing. You need to build detailed test cases and test

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

suites that test each aspect of the application as seen from the outside without looking at the actual source code. Various commercial and open source tools help QA teams perform and review the results of system testing. These tools can create, manage and automate tests or test cases, and they might also offer features beyond system testing, such as requirements management capabilities.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing is an investigatory testing phase, where the focus is to have almost a destructive attitude and tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s).

6.1.6 ACCEPTANCE TESTING

When that user find no major problems with its accuracy, the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

6.2 Testing Objectives

Software Testing has different goals and objectives. The major objectives of Software testing are as follows:

- Finding defects which may get created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of quality.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product.

Software testing helps in finalizing the software application or product against business and user requirements. It is very important to have good test coverage in order to test the software application completely and make it sure that it's performing well and as per the specifications.

While determining the test coverage the test cases should be designed well with maximum possibilities of finding the errors or bugs. The test cases should be very effective. This objective can be measured by the number of defects reported per test cases. Higher the number of the defects reported the more effective are the test cases.

Once the delivery is made to the end users or the customers, they should be able to operate it without any complaints. In order to make this happen the tester should know as how the customers are going to use this product and accordingly, they should write down the test scenarios and design the test cases. This will help a lot in fulfilling all the customer's requirements.

Software testing makes sure that the testing is being done properly and hence the system is ready for use. Good coverage means that the testing has been done to cover the various areas like functionality of the application, compatibility of the application with the OS, hardware and different types of browsers, performance testing to test the performance of the application and load testing to make sure that the system is reliable and should not crash or there should not be any blocking issues. It also determines that the application can be deployed easily to the machine and without any resistance. Hence the application is easy to install, learn and use.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

6.3 Test Cases

A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not.

Unit Test Cases: The software is being divided into different components and unit testing is performed on each of these modules. This section is repeated for all components.

Integration Test Cases: Integration testing is a part of stress testing which involves integrating the components to create a system or sub-system. It may involve testing an increment to be delivered to the customer. In integration testing, the test team has access to the system source code. The system is tested as components are integrated.

Validation Test Cases: This testing is done to see whether the integrated software is valid according to the user needs.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

6.4 Test Results

Following table will give us glimpse on the test results

S.no	Type of Test Case	Result
1	Unit Test Cases	SUCCESS
2	Integration Test cases	SUCCESS
3	Validation Test cases	SUCCESS

Fig 6.1 Test results table

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

7. CONCLUSION & FUTURE SCOPE

Now you have proper understanding of how a Recurrent Neural Network works, which enables you to decide if it is the right algorithm to use for a given Machine Learning problem.

Specifically, you learned what's the difference between a Feed-Forward Neural Network and a RNN, when you should use a Recurrent Neural Network, how Backpropagation and Backpropagation Through Time works, what the main issues of a RNN are and how a LSTM works.

One of the main challenges is training RNNs is learning long-term dependencies in data. It occurs generally due to the large number of parameters that need to be optimized during training in RNN over long periods of time. This paper discusses several architectures and training methods that have been developed to tackle the problems associated with training of RNNs.

Several regularization methods such as dropout, activation stabilization, and activation preservation have been adapted for RNNs to avoid overfitting. While these methods have shown to improve performance, there is no standard for regularizing RNNs. Further research into RNNs regularization can help introduce potentially better regularization methods.

RNNs have a great potential to learn features from 3- dimensional medical images, such as head MRI scans, lung computed tomography (CT), and abdominal MRI. In such modalities, the temporal dependency between images is very important, particularly for cancer detection and segmentation

The popularity of stock market trading is growing rapidly, which is encouraging researchers to find out new methods for the prediction using new techniques. The forecasting technique is not only helping the researchers but it also helps investors and any person dealing with the stock market. In order to help predict the stock indices, a forecasting model with good accuracy is required. In this work, we have used one of the most precise forecasting technologies using Recurrent Neural Network and Long Short-Term Memory unit which helps investors, analysts or any person interested in investing in the stock market by providing them a good knowledge of the future situation of the stock market.

FINANCIAL FORECASTING USING RECURRENT NEURAL NETWORK

REFERENCES

- [1] P. Enyindah and Onwuachu Uzochukwu C, “A Neural Network Approach to Financial Forecasting” International Journal of Computer Applications, Vol. 135 – No.8, February 2016, Page No 28 - 32
- [2] <https://www.datacamp.com/community/tutorials/lstm-python-stock-market>
- [3] <https://blog.usejournal.com/stock-market-prediction-by-recurrent-neural-network-on-lstm-model-56de700bff68>
- [4] <https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5>
- [5] <https://docs.scipy.org/doc/numpy-1.13.0/user/whatisnumpy.html>
- [6] <https://arxiv.org/pdf/1801.01078.pdf>
- [7] <https://en.wikipedia.org/wiki/Scikit-learn>
- [8] <https://keras.io/>
- [9] https://pandas.pydata.org/pandas-docs/stable/getting_started/overview.html
- [10] <https://matplotlib.org/>
- [11] <https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47>