PROJECT REPORT

A MINOR PROJECT REPORT

Submitted by

SUHAIB AHMED 2020-310-219

in partial fulfilment for the award of the degree of
BACHELORS OF TECHNOLOGY

Under the supervision of
Ms. Pooja Gupta

Department of Computer Science & EngineeringSchool

of Engineering Sciences & Technology

JAMIA HAMDARD

New Delhi - 110062

(2023)

ACKNOWLEDGEMENTS

I am very thankful to my supervisor, Ms. Pooja Gupta, for her invaluable assistance throughout the semester; her direction on this research-based capstone was highly welcomed and appreciated. I am also thankful to all the other instructors I had during my bachelor's degree in computer science; they imparted vital skills that aided me throughout this project and more. In addition, I would like to thank everyone who provided me with research papers, tutorials, figures, and pre-built models in order to complete this study. In addition, I would like to thank my entourage, particularly my family and friends, for the emotional support they provided me through both good and difficult times. Finally, I would want to convey my heartfelt appreciation to everyone who has taken the time to read this report, which I hope you will find fascinating.

CONTENTS

TABLE OF FIGURES

System Requirements

Recommended System Requirements

- **Processors:**
    - Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM
- **Disk space:** 2 to 3 GB
- **Graphics card:** Nvidia 960
- **Operating systems:** Windows® 10, macOS®, and Linux®

Minimum System Requirements

- **Processors:** Intel Atom® processor or Intel® Core™ i3 processor
- **Graphics card:** Intel HD 640
- **Disk space:** 1.5 GB
- **Operating systems:** Windows® 7 or later, macOS®, and Linux®
- **Python versions:** 3.X
- **Included development tools:** Conda, Conda-Env, Jupyter Notebook (IPython)
- **Compatible tools:** Microsoft Visual Studio®, PyCharm
- **Included Python packages:** NumPy, SciPy, scikit-learn, pandas, Matplotlib, Numba, Intel® Threading Building Blocks, pyDAAL, Jupyter, mpi4py, PIP, and others.

Modules

- ***Pandas: -*** Pandas is an open source library in Python. It provides ready to use high-performance data structures and data analysis tools. Pandas module runs on top of NumPy and it is popularly used for data science and data analytics.
- ***NumPy***: - NumPy is a Python package which stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc
- ***Matplotlib***: - Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.
- ***Scikit-learn: -*** Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.
- ***Keras: -***Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow. It was developed to make implementing deep learning models as fast and easy as possible for research and development
- ***TensorFlow****:* - It is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.
- ***Pandas-datareader****:* - Up to date remote data access for **pandas**, works for multiple versions of **pandas**. Warning. v0. 8.0 is the last version which officially supports Python 2.7.
- ***Math: -*** **Python math module** is defined as the most popular mathematical functions, which includes trigonometric functions, representation functions, logarithmic functions, etc. Furthermore, it also defines two mathematical constants, i.e., Pie and Euler number, etc.

ABSTRACT

Predicting the value of shares is one of the most complex machine learning challenges. It relies on a range of supply and demand-affecting variables. This study investigates several methodologies for predicting future stock prices and presents an example using a pre-built model customized for the Moroccan stock exchange. Stock prices are represented by time series data, and neural networks are taught to discover patterns from existing data trends. In this study, we also examine the outcomes of a basic single LSTM model, a stacked LSTM model customized to the Moroccan market by using BMCE BANK stock price data, and a hybrid model that combines stacked auto-encoders with sentiment analysis.

1. INTRODUCTION

Stock market volatility, randomness, and unpredictability are well-known characteristics. It is a chaotic environment with an unfathomably large, continually changing stream of data, which makes it very difficult to forecast and capitalize on such predictions. In reality, it is one of the most difficult problems in times series forecasting.

The primary objective of this capstone is to research and apply deep learning methods to the stock market in order to forecast stock behavior, hence avoiding investing risk and generating profit. Transfer learning will be used to take use of pre-built neural network models to accomplish the objective. The predictions are then compared to real historical stock price information. The goal of this initiative is to assist novice traders in making better judgments.

To achieve the goals of this project with precision, several instruments will be used. Deep learning Studio (software) is an excellent beginning point for novices in the subject, since it facilitates the creation of many neural network models to see which performs best for times series forecasting. As for the model and languages to be used, extensive research has led to the conclusion that Python will be the programming language used for implementation due to its flexibility and the availability of pre-built models and open-source libraries that can assist us in achieving our objective and possibly improve results.

In addition, this article will provide a basic illustration of the best-fitting model (the one that produces the best results) for time series forecasting, which is unquestionably the LSTM (Long Short Term Memory) model. Compared to a standard deep neural

The incorporation of a vital component in time series predictions, the memory component, contributes to the network's efficiency.
In addition, this paper will present a sophisticated example of LSTM models using layered LSTM networks, stacked auto-encoders, and a Twitter sentiment analysis of the targeted equities. We will also examine and discuss how this model outperforms its predecessor and achieves promising outcomes.

2. STEEPLE ANALYSIS

2.1. Social

Utilizing Deep Learning to analyze and extract information from previous stock market data in order to estimate the future value of stocks for the aim of making a profit. The objective is to comprehend deep learning models and tailor them to the Moroccan market. People outside the fields of finance and statistics have traditionally seen the stock market as a perilous arena. Some, unable to comprehend its fundamental complexity, even compare it to gambling. This is clearly not a game of pure chance, and the significance of this capstone resides in its potential to provide the typical trader/normal citizen with an educated perspective on the stock market so that they may at least make better judgments than random ones.

2.2. Technological

In addition, by providing LSTM models with the capacity to preserve long-term dependencies, Deep Learning considerably aided in tackling the challenge of stock market forecasting. This immediately implies the development of all relevant technologies, which might result in a significant improvement of all Time Series Forecasting issues. These new technologies might potentially improve the quality of life for novice, ordinary, and uninformed traders.

2.3. Economic

As for the economic ramifications of this endeavor, there are various benefits to using this technology. In the event of a very accurate model, this instrument would be a highly valuable asset for directing stock market investments. In addition, the development of models capable of learning long-term relationships might be beneficial in a variety of other domains, including voice recognition and music creation. Thus making it an economically intriguing path to investigate.

2.4. Environmental

There seems to be no direct link between the environmental factor of our steeple analysis and this project.

2.5. Political

There does not seem to be a link between politics and projecting the future stock price. Nonetheless, individuals or businesses offering an upgraded version of this service might utilise the resulting income for political purposes. As this is a new topic, particularly in the Moroccan setting, we do not have any specific proof to support this claim.

2.6. Legal & Ethical

Legally and morally, the sector is mostly uncontrolled, which has two significant ramifications. First, it provides an opportunity since it is mostly undiscovered ground and, as with any legal grey zones, the potential earnings are enormous. Secondly, it would also pose an unknown danger, as is the case with all technological advancements. In an attempt to maintain justice and equality, this project should also be governed by the correct ethics and morals while dealing with stock market forecasts.

3. THEORITICAL BACKGROUND

3.1. Deep Learning

Deep Learning is an AI technique that teaches robots to learn through analogy, much as the human brain does naturally. Deep Learning is a significant invention that enabled autonomous automobiles, enabling them to interpret traffic signs, identify a person on foot, and even determine whether a driver is aware or not in order to park the vehicle securely and prevent a tragedy. It is also responsible for voice-controlled devices such as smartphones, tablets, televisions, and wireless speakers [2]. Deep learning has recently received a great deal of attention, which is well-deserved in view of the present state of affairs, since it is producing outcomes that were previously deemed impossible. The diagram below illustrates the distinctions between Artificial Intelligence, Machine Learning, and Deep Learning. [1]

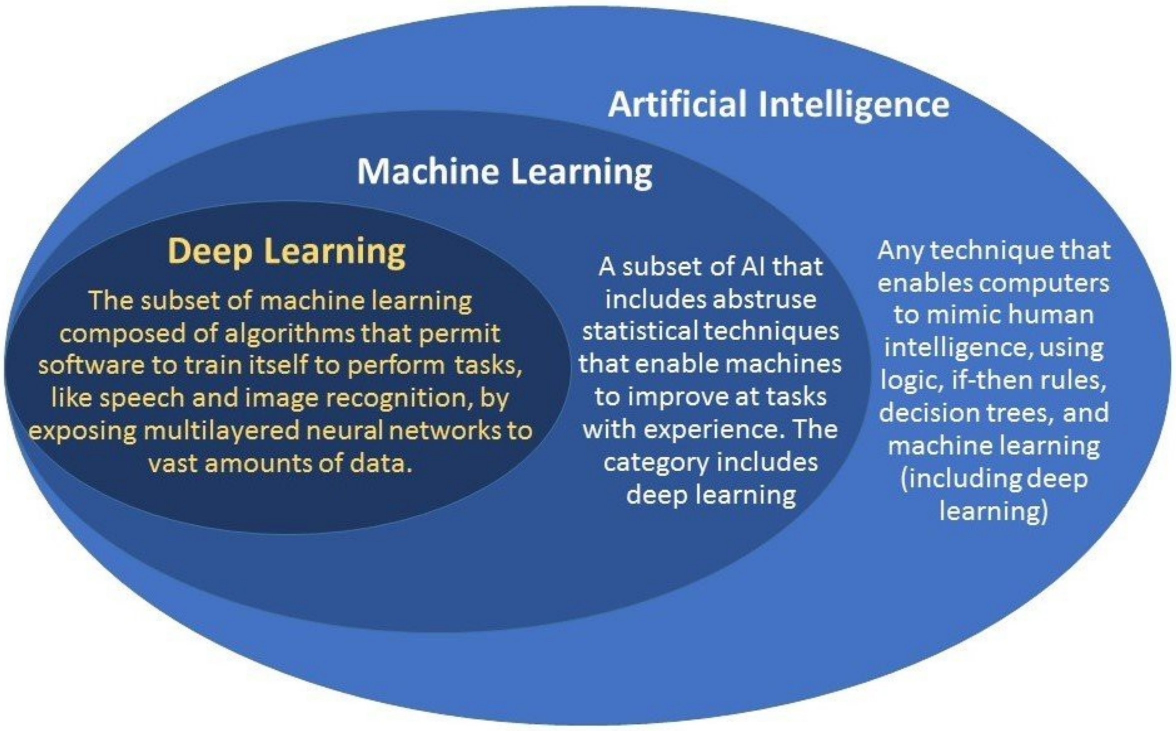

Artificial Intelligence
Machine Learning
Deep Learning
The subset of machine learning composed of algorithms that permit software to train itself to perform tasks, like speech and image recognition, by exposing multilayered neural networks to vast amounts of data.
A subset of AI that includes abstruse statistical techniques that enable machines to improve at tasks with experience. The category includes deep learning
Any technique that enables computers to mimic human intelligence, using logic, if-then rules, decision trees, and machine learning (including deep learning)

Figure 1: AI vs Machine Learning vs Deep Learning [1]

A computer model learns to do categorization tasks directly from pictures, text, or voice through Deep Learning. Deep learning models may achieve cutting-edge precision, sometimes exceeding human execution [5]. Typically, models with a neural network architecture consisting of several layers are fed and trained on huge quantities of categorized data, which makes Deep Learning extremely expensive in terms of necessary computational time and power. [2]

Deep learning improves recognition accuracy at a faster pace. This is essential for safety-focused applications, such as driverless cars, and helps consumer devices meet user demands. Recent advancements in deep learning have enabled it to beat humans in some tasks, such as detecting objects in images[1]. The figure below depicts a basic neural network and a Deep Learning neural network.

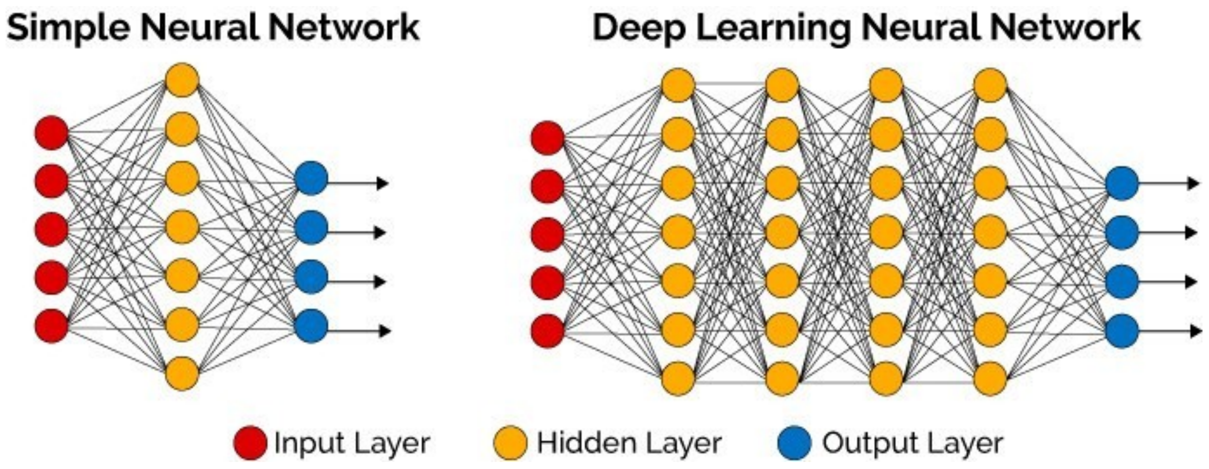


Figure 2: Conceptual illustration of a simple neural network and a Deep Learning neuralnetwork. [10]

- As described in the preceding section, a deep learning neural network consists of numerous hidden layers, comparable to stacking multiple basic neural networks. This begs the issue of how deep learning achieves higher performance.
-
-
- Since the introduction of the first theoretical deep learning algorithm in the 1980s, deep learning has only recently begun to approach its true potential for two main reasons.
-
- Deep Learning requires a vast quantity of labelled information or data, which has just been accessible in the previous decade. For instance, implementing autonomous vehicles requires millions of photographs and hundreds of hours of video. [1] [1]

3.2. Transfer Learning

Transfer learning is a Machine Learning approach that re-applies a trained model to a second task that is linked to the first. Once the second job has been represented, this optimization allows for rapid progress or enhanced performance. Transfer learning is also strongly associated with difficulties such as multitasking learning and is not only a topic of interest for Deep Learning. Given the huge resources and complex data sets required for training Deep Learning models, Transfer Learning has become very popular and is mostly used for Deep Learning applications.

There are two primary ways to implement Transfer Learning:

1. Model Development Approach:

First, a source job with a big data set is selected. Then, we create a source model for the first task in order to discover its characteristics. Next, we utilise the model as a starting point for a second job that is closely related. A few more layers may be added to the model for the second job to increase its performance. [11]

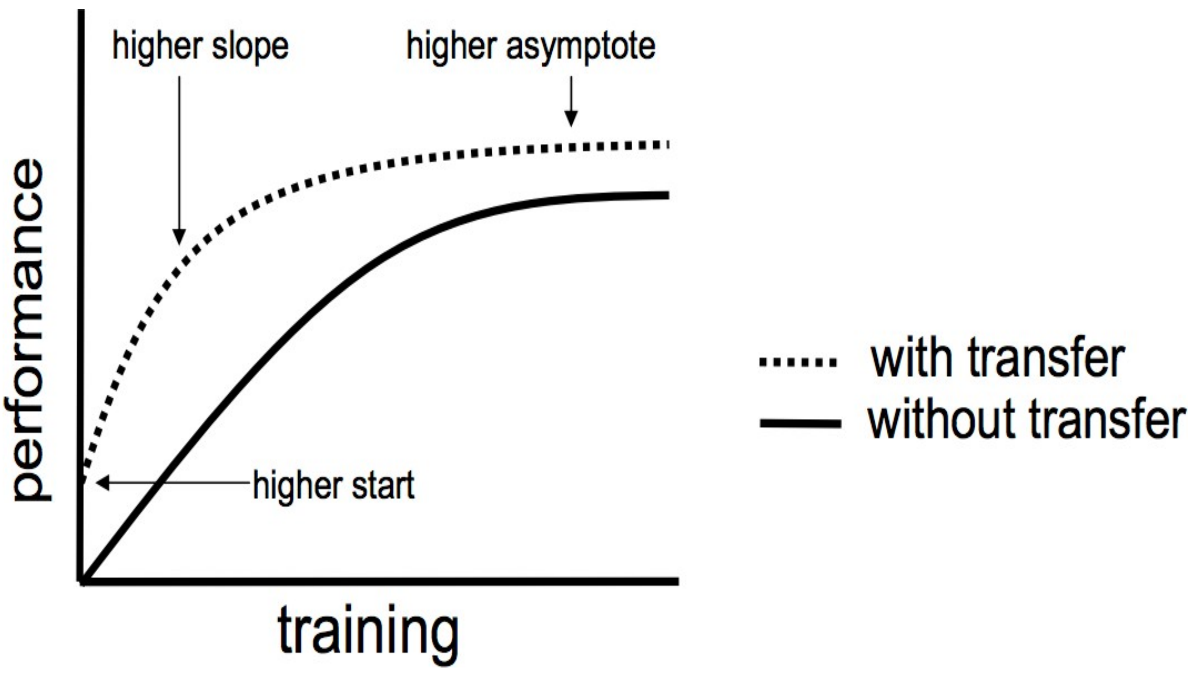


Figure 3: Performance with Transfer Learning & without it. [11]

Below are the major advantages to using Transfer Learning[11]:

- Higher Start: refers to the initial performance of a model before refining it.
- Higher Slope: refers to the rate at which the model's performance improves.
- Higher asymptote: refers to the peak performance reached by a model.

3.3. Time Series Analysis

Before discussing time series analysis, let's first define time series: It is a series of data points or values that are ordered, enumerated, or graphed in relation to the passage of time. In the majority of instances, a time series sequence will consist of continuous data points uniformly spaced in time.

The primary objectives of time-series analysis are to (a) determine the nature of the observation sequence phenomena and (b) forecast future events (accurately predict time-series variable). These two aims involve the detection and formal description of observable patterns in time series. Once the pattern has been detected, it may be interpreted and embedded among additional data. Regardless of the complexity and accuracy of our comprehension of the subject, we may derive the recognized pattern to forecast events using reasoning. [12]

Time series analysis entails strategies for dissecting information from a succession of data points in order to extract relevant measures and distinct characteristics. Utilizing a model to forecast future values by analyzing patterns in past value records or datasets is time series prediction. In addition, a random model for time series forecasting will always be constrained by the fact that data points closer in time will be more correlated than data points with a greater separation in time. This is precisely what makes predicting time series difficult: maintaining long-term correlation without ignoring short-term connections between data points. [12]

In this way, a variety of data mining approaches may be used with varying degrees of success to tackle the issue. Next, we will examine the most promising algorithms for predicting time series data and how LSTM neural networks currently dominate the area.

3.4. Long Short Term Memory model (LSTM)

LSTM, which stands for Long Short Term Memory, is a neural network type that is very good for time series forecasting. According to Srivastava's paper on LSTMs and the fundamentals of deep learning, an LSTM network is the most successful method for time series analysis and, by extension, stock market forecasting. [7]

Srivastava asserts, "Sequence prediction issues have existed for a very long time. They are regarded one of the most difficult issues to address in the field of data science. From estimating sales to identifying trends in stock market data, from comprehending movie storylines to recognizing your manner of speaking, from language translations to anticipating your next word on your iPhone's keypad, these difficulties span a broad spectrum.

Long short-term memory networks have been demonstrated to be the most successful solution for almost all of these sequence prediction challenges, as a result of recent developments in data science.

In several aspects, LSTMs outperform traditional feed-forward neural networks and Recurrent Neural Networks. This is due to their ability to memorise patterns selectively over extended periods of time. [7]

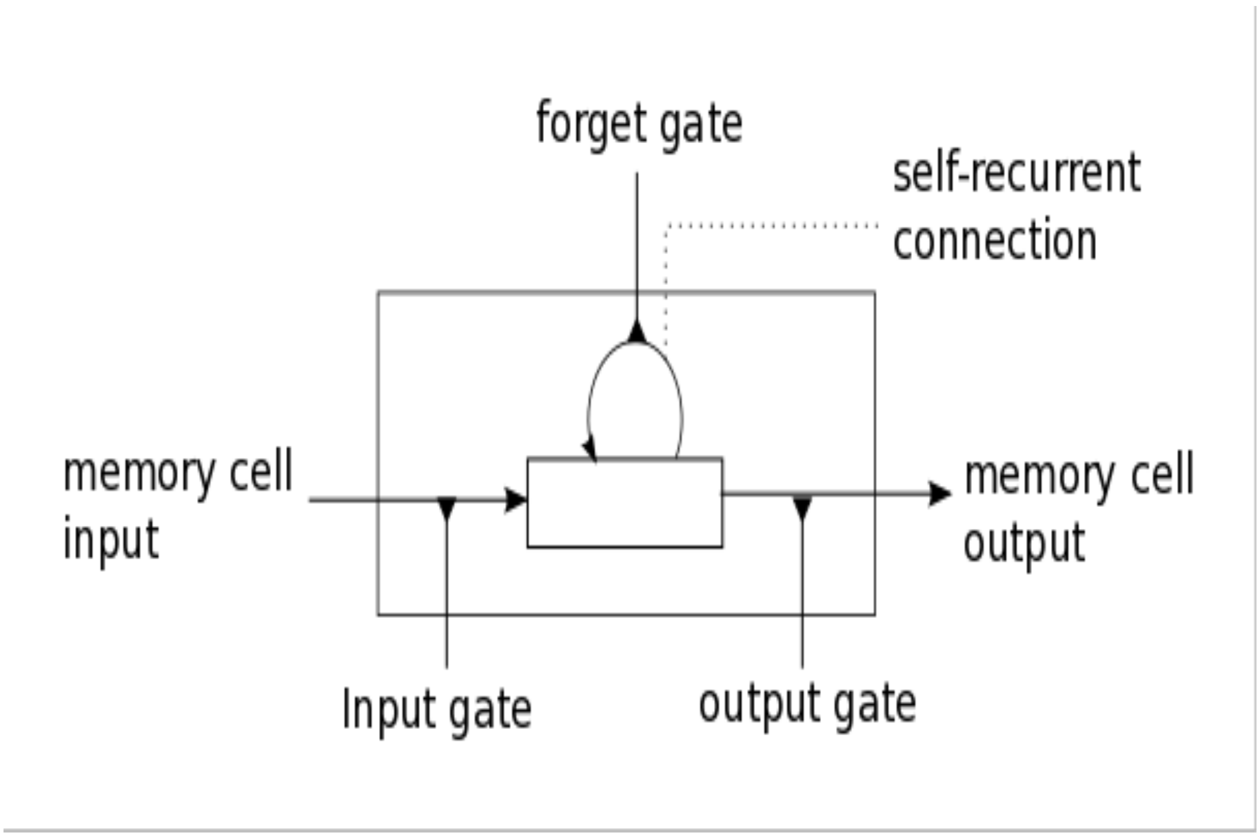


Figure 4: Simplified look at an LSTM cell. [7]

In order to add new information to a rudimentary neural network, the old information is entirely transformed using a sigmoid function. As a result, the whole of the material is updated. Thus, there is no distinction between 'important' and 'less important' information. LSTMs, on the other hand, do little adds and multiplications on the information. Information flows via a process known as cell states in LSTMs. Thus, LSTMs are able to selectively recall or forget information. [7]

The following figure, represents a more detailed view at the internal architecture of anLSTM network:

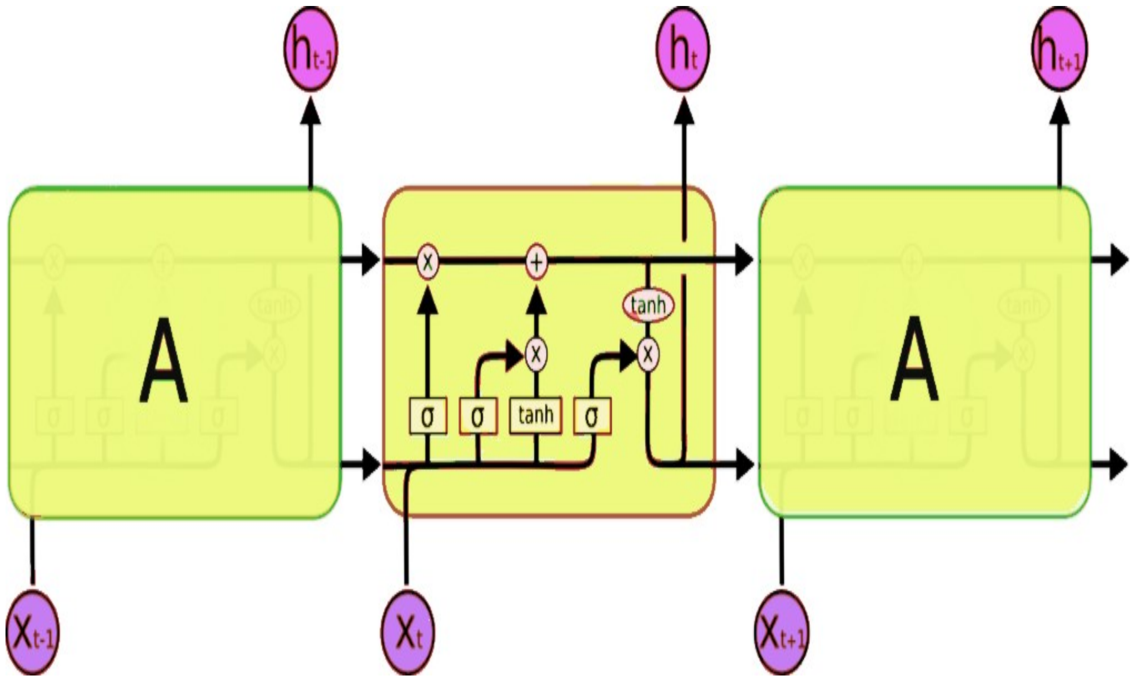


Figure 5: LSTM Cell Architecture [7]

A typical LSTM network is comprised of different memory blocks called cells. Thereare two states that are being transferred to the next cell; the cell state and the hidden state.

The memory blocks are responsible for remembering things, and manipulations to this memory is done through three major mechanisms called gates:

(a) Forget Gate:

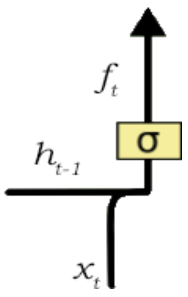


Figure 6: Forget Gate, Internal Architecture [7]

"A forget gate is responsible for erasing information from the cell state," says Srivastava. The information that is no longer necessary for the LSTM to comprehend things or is of lesser value is eliminated. This gate takes in two inputs; h_t-1 and x_t.

h t-1 is the hidden state or output from the previous cell, whereas x t is the input for the current time step. Multiplying the inputs by the weight matrices and adding a bias. Then, this number is subjected to the sigmoid function. The sigmoid function returns a vector with values ranging from 0 to 1 that correspond to each cell state number. Essentially, the sigmoid function determines which data to retain and which to reject. If a '0' is generated for a specific value in the cell state, it indicates that the forget gate wishes for the cell state to entirely forget that piece of information. Similarly, a value of '1' indicates that the forget gate want to retain the whole of the information. This sigmoid function output vector is multiplied by the cell state." [7]

(b) Input Gate:

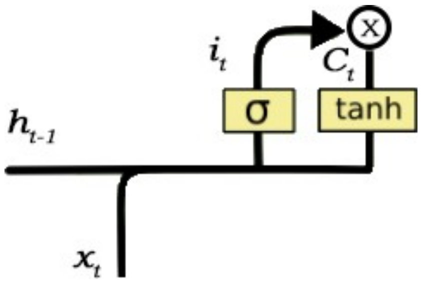


Figure 7: Input Gate, Internal Architecture [7]

"The input gate is responsible for adding information to the cell state," says Srivastava. This information addition is essentially a three-step procedure, as seen in the figure.

Utilizing a sigmoid function to determine which values must be added to the cell's state. This is essentially identical to the forget gate and functions as a filter for all the data from h t-1 and x t.

Creating a vector containing all potential values that may be added to the cell state (based on h t-1 and x t) This is accomplished using the tanh function, which returns values between -1 and 1.

multiplying the value of the regulatory filter (the sigmoid gate) by the vector formed (the tanh function) and then adding this beneficial information to the cell's state through addition.

After completing these three steps, we guarantee that only the relevant information is uploaded to the database.

(c) Output Gate:

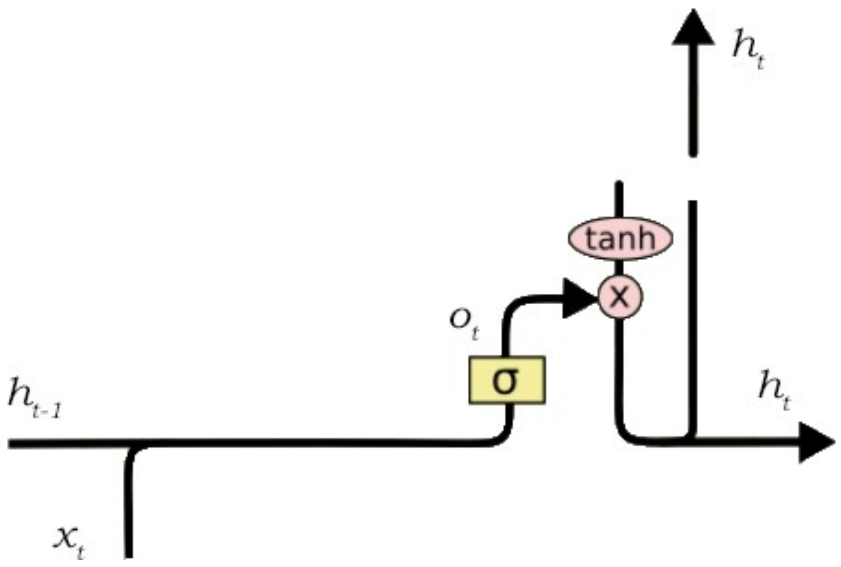


Figure 8: Output Gate, Internal Architecture [7]

Citing Srivastava: "The output gate is responsible for picking and displaying important information from the present cell state." Again, the operation of an output gate may be broken down into three steps:

1. Creating a vector by applying the tanh function on the cell state and scaling the values to the range of -1 to +1.
2. Using the values of h t-1 and x t to design a filter that may control the output values from the vector established earlier. This filter utilizes a sigmoid function once again.
3. Multiplying the value of this regulatory filter by the vector established in Step 1 and transmitting it as an output as well as to the concealed state of the next cell." [7]

3.5. Stacked Auto-Encoders

Multiplying the value of this regulatory filter by the vector established in Step 1 and transmitting it as an output as well as to the concealed state of the next cell." [7]

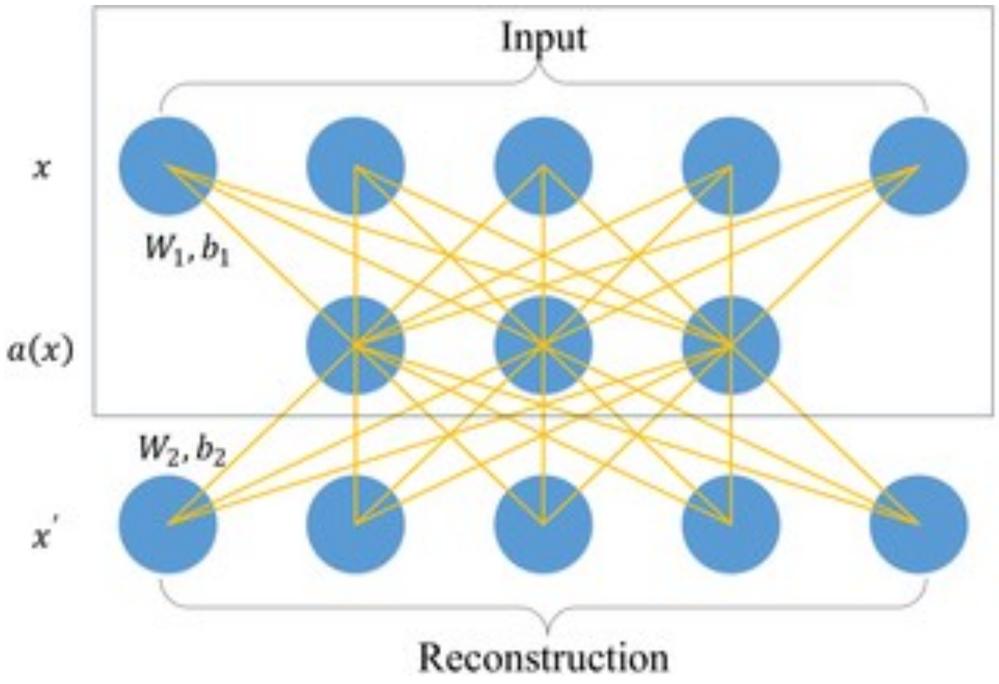


Figure 9: Structure of a Single Layer Autoencoder [4]

A single layer autoencoder, as seen in the diagram above, is a three-layer neural network containing an input layer, a hidden layer, and a reconstruction layer [5].

These autoencoders are meant to create deeper or more abstract hidden layer characteristics. We attempt to reduce the error between the input layer and the reconstruction layer while training these autoencoders. [4]

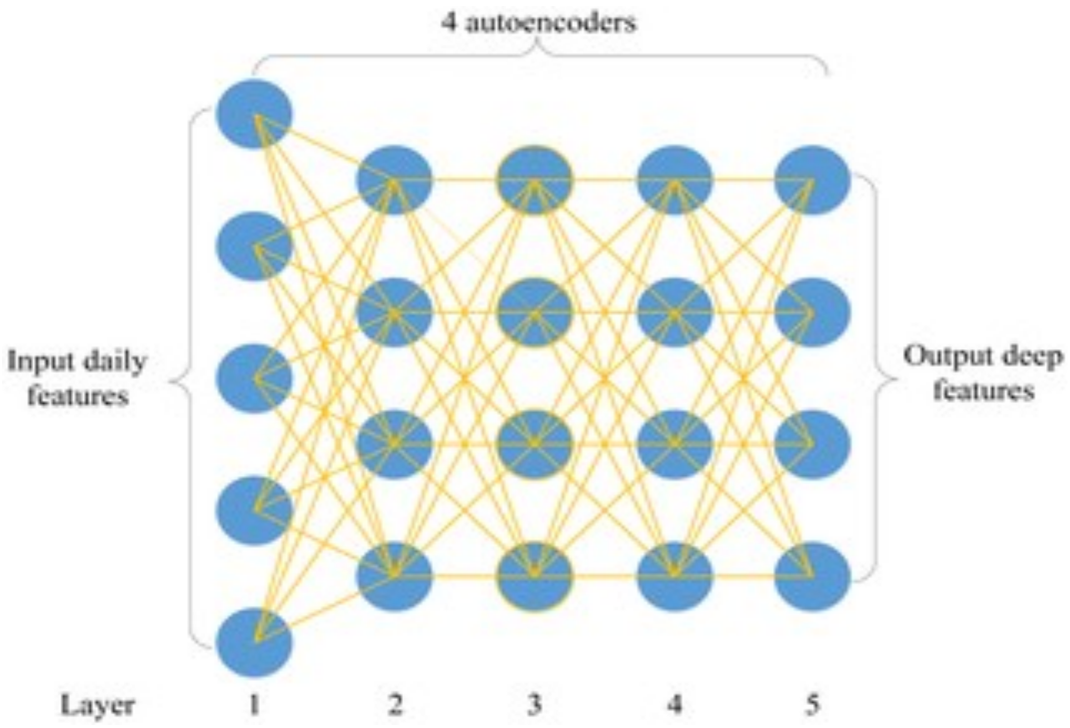


Figure 10: Stacked Autoencoder, Structure [4]

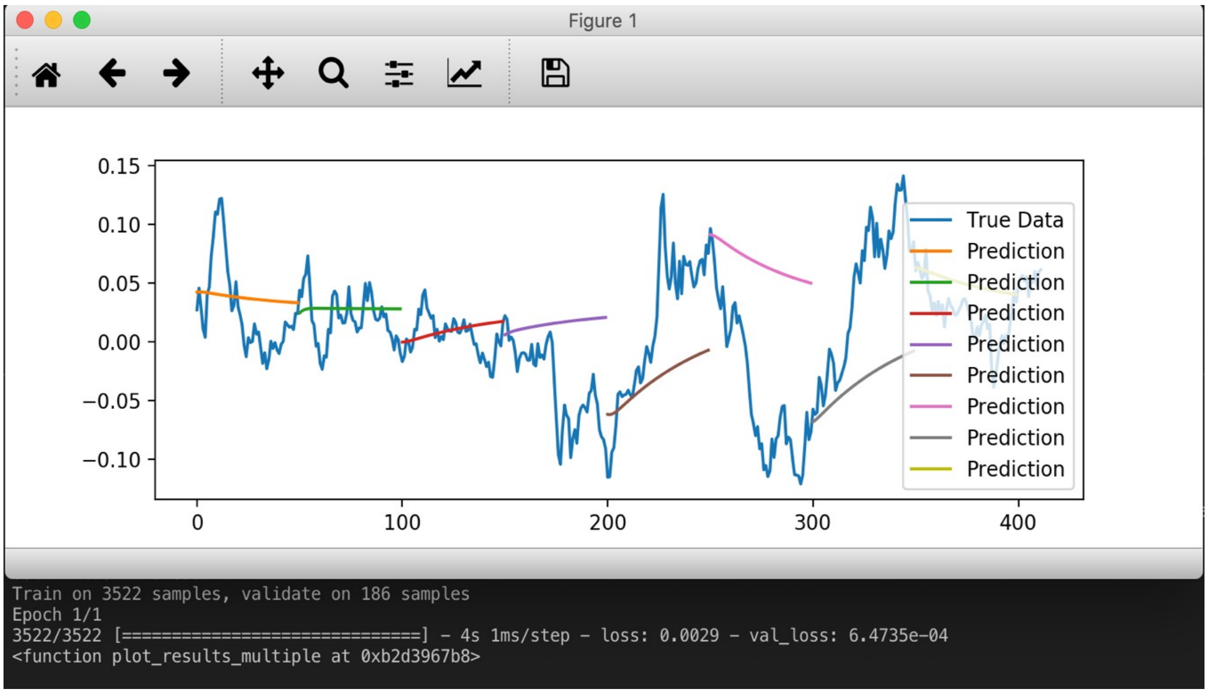
In this case of a stacked autoencoder (see above), it consists of four autoencoders and five layers, and its purpose is to produce high-level features. According to a paper entitled A deep learning framework for financial time series utilising stacked autoencoders and long-short term memory: "The autoencoder with a single layer transfers the daily input variables to the first hidden vector. Following the training of the first single-layer autoencoder, the reconstruction layer is removed and the hidden layer is retained as the input layer for the second single-layer autoencoder. The input layer of the succeeding autoencoder is typically the hidden layer of the preceding autoencoder. [4]

The initial layer of a stacked auto encoder tends to learn first order features (e.g. edges in an image). The second layer of a stacked autoencoder tends to learn second-order features, which correspond to appearance patterns of the second order (i.e., contours or corners) Higher layers of the stacked autoencoder have a tendency to acquire even higher-order features. [4]

4. LSTM MODELS & RESULTS

4.1. Single LSTM Model example
4.2.

Khaled Tinubu's [8] python-based model is an excellent example for understanding the fundamentals of realistically constructing a single LSTM network that utilizes historical S&P 500 stock data to train an LSTM in order to predict the future closing price of the stock at a particular period. This model consists of two successively stacked LSTM layers and a dense layer, using linear regression at the level of the final layer. Additionally, it employs MSE (mean squared error) for loss calculations and RMSProp as an optimizer. This model's outcomes are shown in the image below:



```
Train on 3522 samples, validate on 186 samples
Epoch 1/1
3522/3522 [==============================] - 4s 1ms/step - loss: 0.0029 - val_loss: 6.4735e-04
<function plot_results_multiple at 0xb2d3967b8>
```

Figure 11: Single LSTM Model, Results. [8]

Clearly, based on the graph, the model's predictions are not as accurate as we would want. This is owing to the model's simplicity, since it is just intended for experimenting and learning how to forecast the future value of stocks using an LSTM deep neural network. Appendix A contains the fragments of code required to construct this reasonably simple project.

4.3. Hybrid LSTM Model example

In an attempt to accurately anticipate the future value of stocks, Vivek Palaniappan's [3] hybrid model is an advanced version of an LSTM network using both stacked autoencoders and sentiment analysis via Twitter and news filtering. Even with this degree of sophistication, this model is not intended to be used as a trading tool without changes and the application of the proper methods.

In contrast to most machine learning projects, where it is typical to first collect the data, then preprocess it, then train the model, and finally test it, this project begins with model training. This model requires a substantially more intricate construction plan. A flowchart outlining the procedure is shown below.

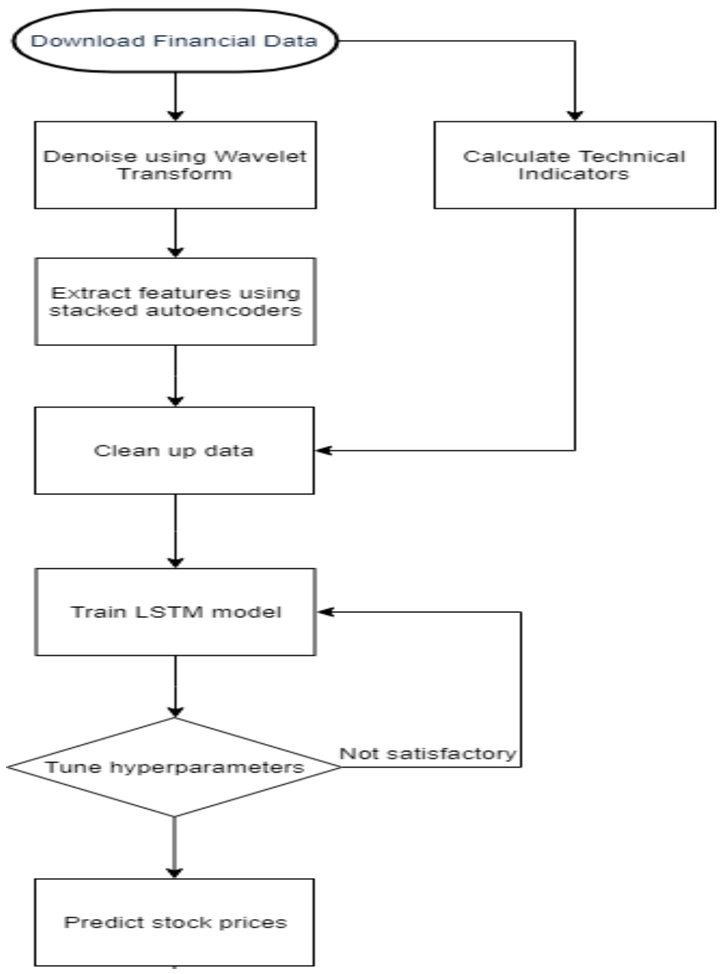


Figure 12: Building a Hybrid LSTM Model, Flowchart. [3]

This model utilizes the Keras package to build 2 dense layers with 20 as the input dimension, a tanh activation function, and an activity regularize to penalize layer activity for performance and optimization purposes. As for the optimizer, it employs SGD (Stochastic gradient descent) for its efficiency and speed, which are intriguing characteristics for this 2000-epoch-trained model. Below is a graph of the results:

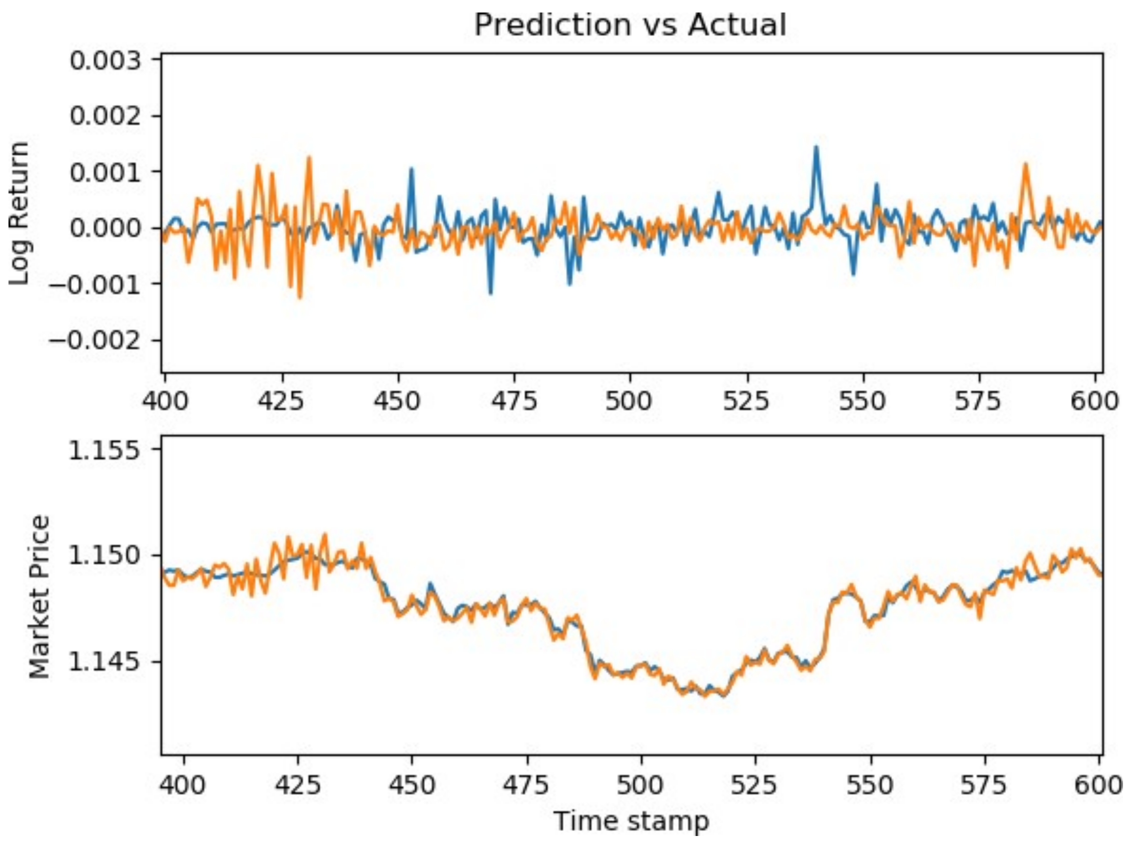


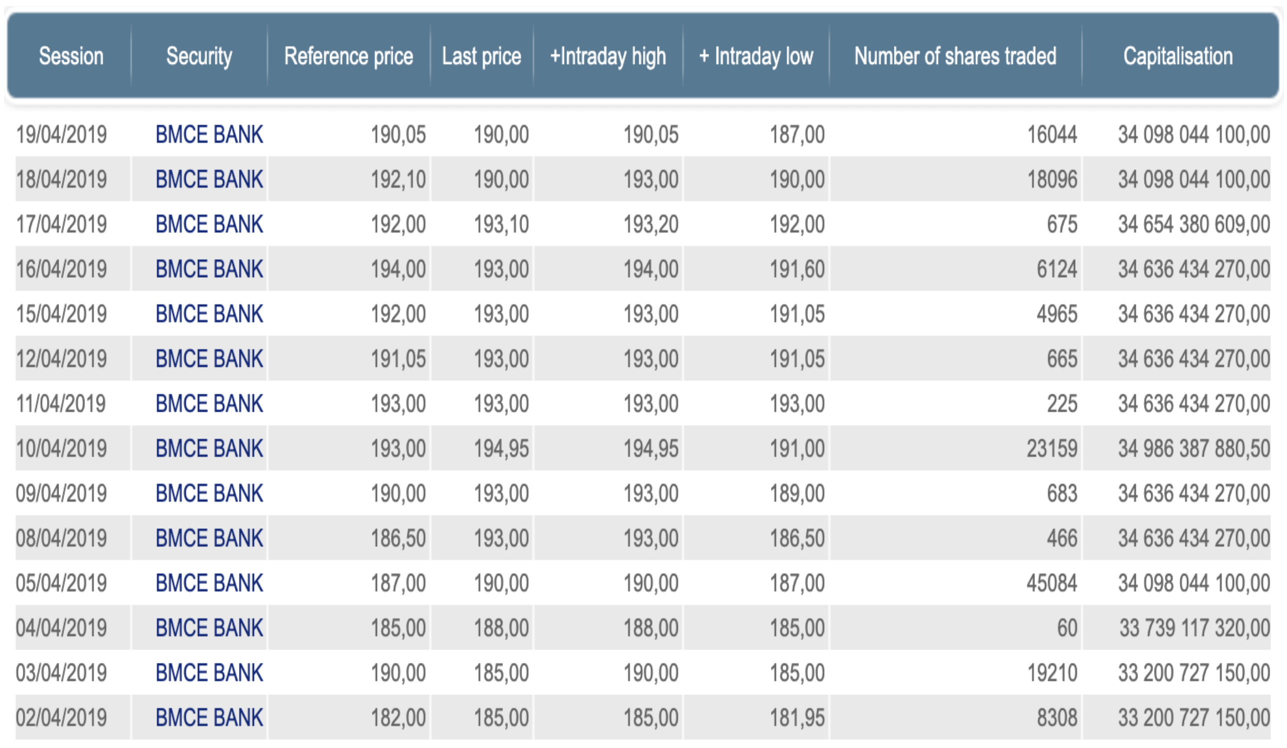
Figure 13: Hybrid LSTM Model, Results. [3]

Evidently, based on the plot, the model's predictions are sufficiently accurate, giving this model an excellent foundation for developing a successful deep learning model for stock prediction. This is owing to the model's increased complexity and hybrid nature, which combines news scrubbing for sentiment analysis with stacked autoencoders.

5. STOCK PREDICTION FOR BMCE BANK

In an attempt to apply what I have learnt from the research portion of my capstone, I based this example on Nouroz Rahman's [9] model. In order to make the projections, I used a three-year-old Moroccan data collection that I compiled for BMCE shares.

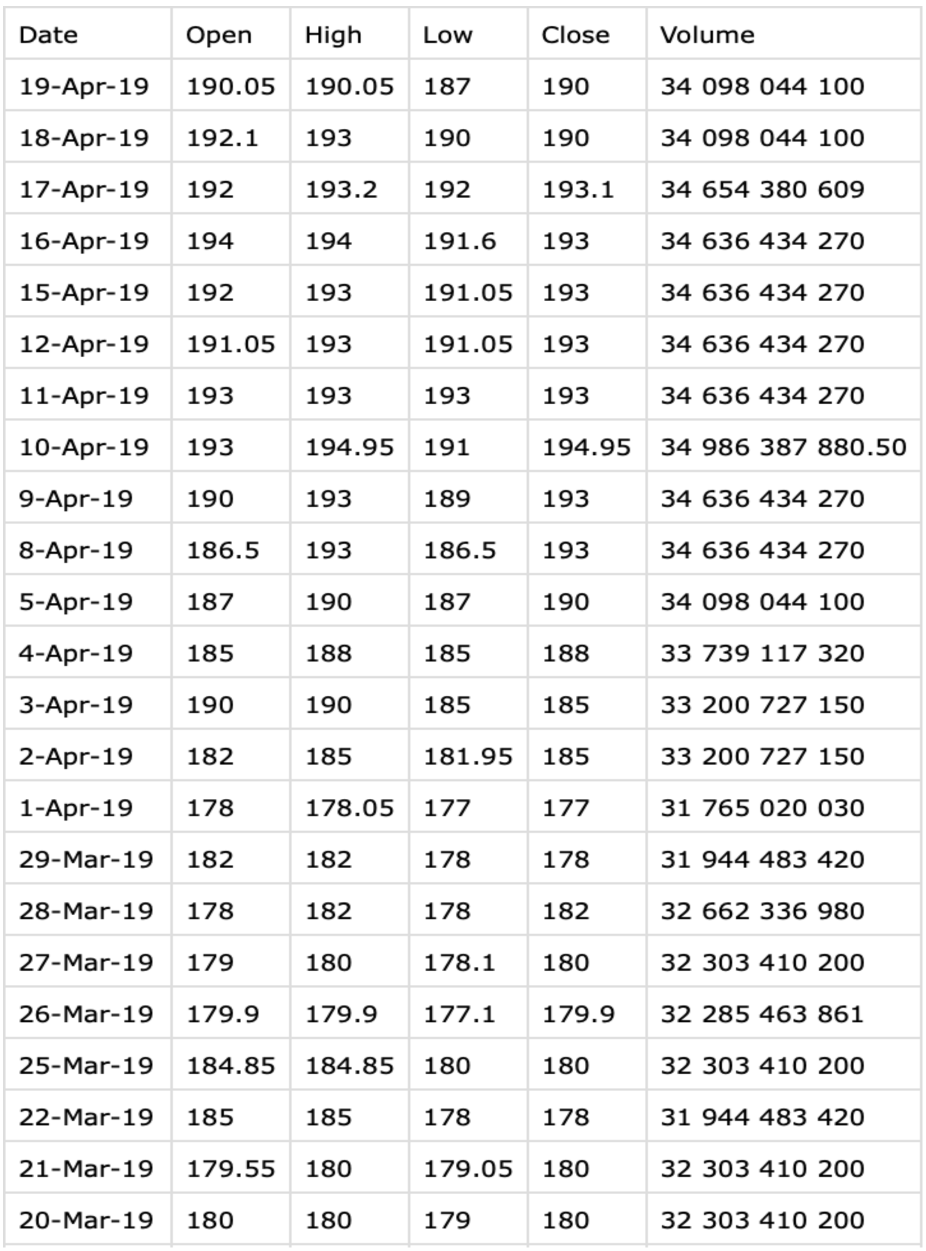
The first data set retrieved from the Casablanca-bourse website appeared as follows:



Session	Security	Reference price	Last price	+Intraday high	+ Intraday low	Number of shares traded	Capitalisation
19/04/2019	BMCE BANK	190,05	190,00	190,05	187,00	16044	34 098 044 100,00
18/04/2019	BMCE BANK	192,10	190,00	193,00	190,00	18096	34 098 044 100,00
17/04/2019	BMCE BANK	192,00	193,10	193,20	192,00	675	34 654 380 609,00
16/04/2019	BMCE BANK	194,00	193,00	194,00	191,60	6124	34 636 434 270,00
15/04/2019	BMCE BANK	192,00	193,00	193,00	191,05	4965	34 636 434 270,00
12/04/2019	BMCE BANK	191,05	193,00	193,00	191,05	665	34 636 434 270,00
11/04/2019	BMCE BANK	193,00	193,00	193,00	193,00	225	34 636 434 270,00
10/04/2019	BMCE BANK	193,00	194,95	194,95	191,00	23159	34 986 387 880,50
09/04/2019	BMCE BANK	190,00	193,00	193,00	189,00	683	34 636 434 270,00
08/04/2019	BMCE BANK	186,50	193,00	193,00	186,50	466	34 636 434 270,00
05/04/2019	BMCE BANK	187,00	190,00	190,00	187,00	45084	34 098 044 100,00
04/04/2019	BMCE BANK	185,00	188,00	188,00	185,00	60	33 739 117 320,00
03/04/2019	BMCE BANK	190,00	185,00	190,00	185,00	19210	33 200 727 150,00
02/04/2019	BMCE BANK	182,00	185,00	185,00	181,95	8308	33 200 727 150,00

Figure 14: Original BMCE BANK Dataset [6]

After formatting, scrubbing, and converting the dataset to.csv format. Our original dataset now appears as follows:



Date	Open	High	Low	Close	Volume
19-Apr-19	190.05	190.05	187	190	34 098 044 100
18-Apr-19	192.1	193	190	190	34 098 044 100
17-Apr-19	192	193.2	192	193.1	34 654 380 609
16-Apr-19	194	194	191.6	193	34 636 434 270
15-Apr-19	192	193	191.05	193	34 636 434 270
12-Apr-19	191.05	193	191.05	193	34 636 434 270
11-Apr-19	193	193	193	193	34 636 434 270
10-Apr-19	193	194.95	191	194.95	34 986 387 880.50
9-Apr-19	190	193	189	193	34 636 434 270
8-Apr-19	186.5	193	186.5	193	34 636 434 270
5-Apr-19	187	190	187	190	34 098 044 100
4-Apr-19	185	188	185	188	33 739 117 320
3-Apr-19	190	190	185	185	33 200 727 150
2-Apr-19	182	185	181.95	185	33 200 727 150
1-Apr-19	178	178.05	177	177	31 765 020 030
29-Mar-19	182	182	178	178	31 944 483 420
28-Mar-19	178	182	178	182	32 662 336 980
27-Mar-19	179	180	178.1	180	32 303 410 200
26-Mar-19	179.9	179.9	177.1	179.9	32 285 463 861
25-Mar-19	184.85	184.85	180	180	32 303 410 200
22-Mar-19	185	185	178	178	31 944 483 420
21-Mar-19	179.55	180	179.05	180	32 303 410 200
20-Mar-19	180	180	179	180	32 303 410 200

Figure 15: BMCE BANK, Modified Data Set.

- Typically, stock traders use two indicators to create forecasts [9]:
-
- OHLC (average of Open, High, Low and Closing Prices)
- HLC (average of High, Low and Closing Prices)

For this example, OHLC average has been used as shown in the following graph:

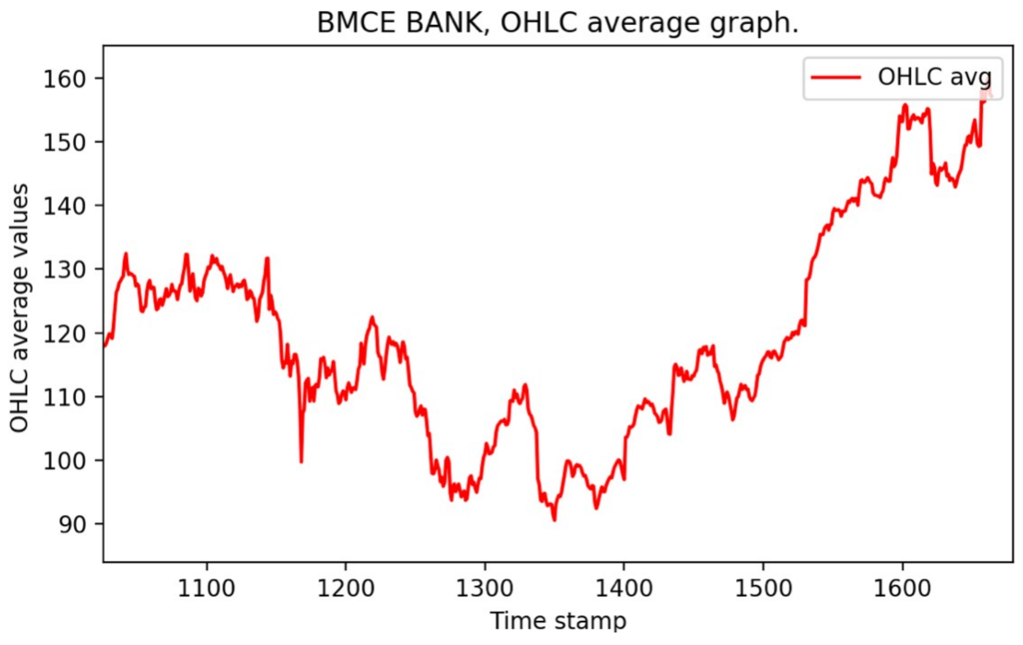


Figure 16: BMCE BANK, OHLC Average Graph

In addition, the BMCE BANK modified data set has been pre-processed as such:

- The data set is converted into OHLC average (see graph below)

  ➔ The data set has now only one column data.

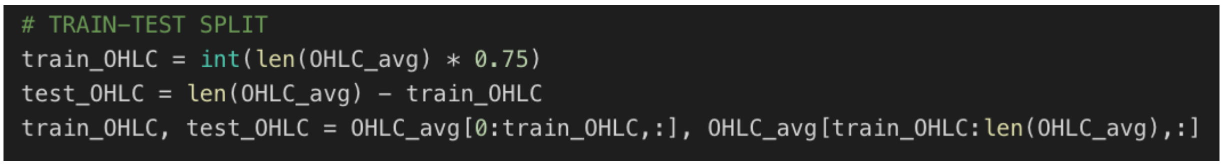
- The data set is then converted into two column time series data.

  ➔ 1st column represents the stock price of time t,

  ➔ The second column represents the stock price of time t+1.

- All values are then normalized between 1 and 0.

- The data set is split into training and testing data with respective percentages of75% and 25% as shown in the code below:



```
# TRAIN-TEST SPLIT
train_OHLC = int(len(OHLC_avg) * 0.75)
test_OHLC = len(OHLC_avg) - train_OHLC
train_OHLC, test_OHLC = OHLC_avg[0:train_OHLC,:], OHLC_avg[train_OHLC:len(OHLC_avg),:]
```

Figure 17: Code Snippet, Training & Testing Data [9].

Now that the dataset is complete, let's examine Nouroz Rahman's model.

Using the Keras deep learning package, this model employs two successive LSTM layers layered together. Due to the nature of the challenge, linear regression is used for the final layer.

After modifying the code, mostly for the data pre-processing function, and experimenting with several optimizers (RMSProp, AdaGrad, Adam), I have chosen to utilize the Adam optimizer owing to its ability to get decent results more quickly than the two previously mentioned optimizers. The acquired findings are shown in the following figures:

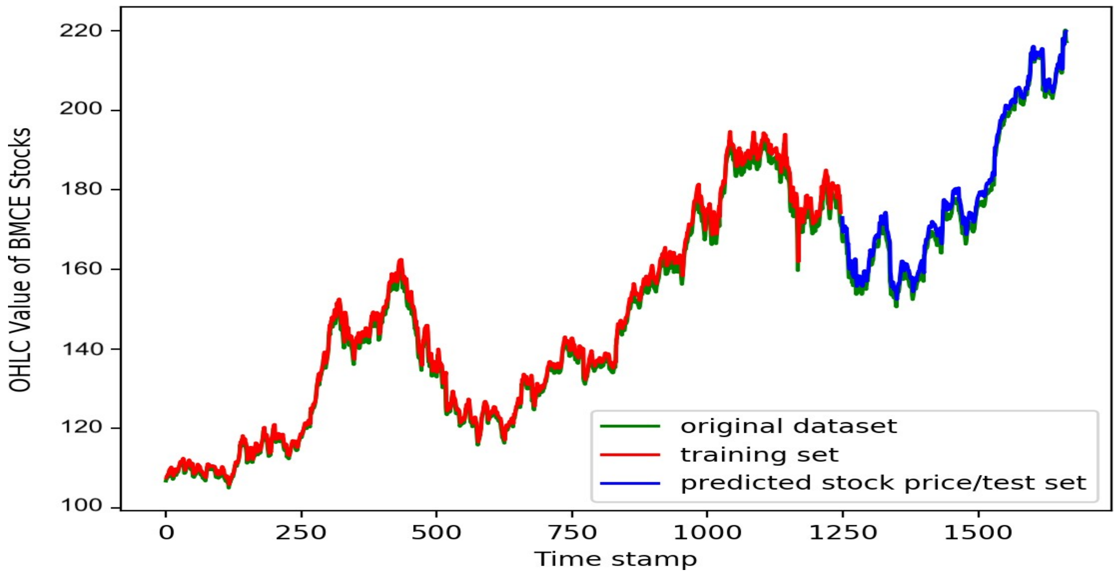
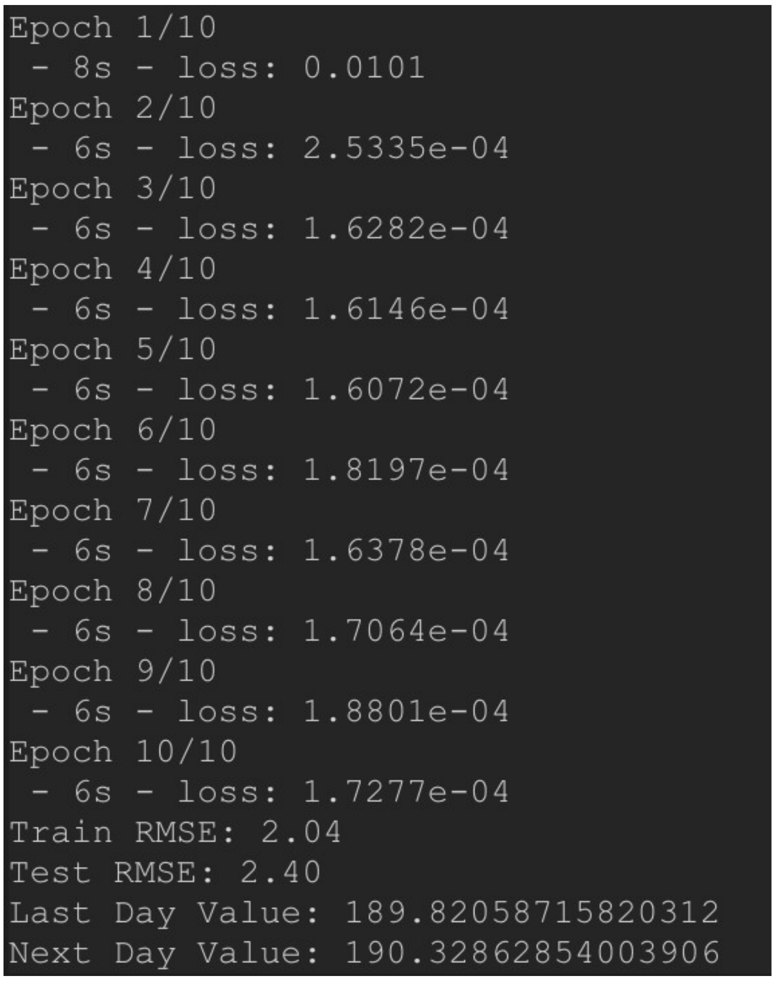


Figure 18: Results Graph.



```
Epoch 1/10
 - 8s - loss: 0.0101
Epoch 2/10
 - 6s - loss: 2.5335e-04
Epoch 3/10
 - 6s - loss: 1.6282e-04
Epoch 4/10
 - 6s - loss: 1.6146e-04
Epoch 5/10
 - 6s - loss: 1.6072e-04
Epoch 6/10
 - 6s - loss: 1.8197e-04
Epoch 7/10
 - 6s - loss: 1.6378e-04
Epoch 8/10
 - 6s - loss: 1.7064e-04
Epoch 9/10
 - 6s - loss: 1.8801e-04
Epoch 10/10
 - 6s - loss: 1.7277e-04
Train RMSE: 2.04
Test RMSE: 2.40
Last Day Value: 189.82058715820312
Next Day Value: 190.32862854003906
```

Figure 19: Training for 10 Epochs.

As shown before, the model is trained for ten epochs prior to providing predictions; here, loss refers to the root mean squared error. Thus, we can observe that the training RMSE (root mean squared error) is 2.04 and the test RMSE is 2.40, which is neither ideal nor inadequate in comparison to other models consulted. The previous day's value is around 189, and the forecast value for the next day is approximately 190, which is close to the actual figure of 189.278 received from the Casablanca-bourse website.

6. CONCLUSION & FUTURE WORKS

After completing this capstone project, it is obvious that the combination of LSTM networks, stacked autoencoder, and sentiment analysis gives adequate results for live trading. Future work on this project will involve and need a more in-depth study effort to adapt the model that yields the greatest results, the hybrid model, to the Moroccan market. The sheer quantity of accessible data, which is insufficient for an optimum and lucrative deep learning model, might be a significant obstacle in this instance. This might be remedied by using data augmentation methods to the current data sets to increase their size and make them suitable for a deep learning project.

7. REFERENCES

1. "What Is Deep Learning?," mathworks, [Online]. Available:https://www.mathworks.com/discovery/deep-learning.html.

2. A. Tipirisetty, "Stock Price Prediction using Deep Learning," 2018. [Online]. Available: https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1639&context=etd_projects.

3. V. Palaniappan, 2018. [Online]. Available: https://github.com/VivekPa/AIAlpha.

4. "Stacked Autoencoders," *Ufldl,Stanford*. [Online]. Available:http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders.

5. W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PLOS ONE*. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0180944#pone-0180944-g002.

6. *Casablanca Stock Exchange ::. MARKETS >Market data > Historical data*. [Online]. Available: http://www.casablanca-bourse.com/bourseweb/en/Negociation-History.aspx?Cat=24&IdLink=225.

7. P. Srivastava, “Essentials of Deep Learning : Introduction to Long Short TermMemory,” *Analytics Vidhya*, 23-Dec-2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/.

8. K. tinubu, “ktinubu/Predict-Stock-With-LSTM,” *GitHub*, 14-Oct-2017. [Online]. Available: https://github.com/ktinubu/Predict-Stock-With-LSTM.

9. N. Rahman , “NourozR/Stock-Price-Prediction-LSTM,” *GitHub*, 08-Nov-2017. [Online]. Available: https://github.com/NourozR/Stock-Price-Prediction-LSTM.

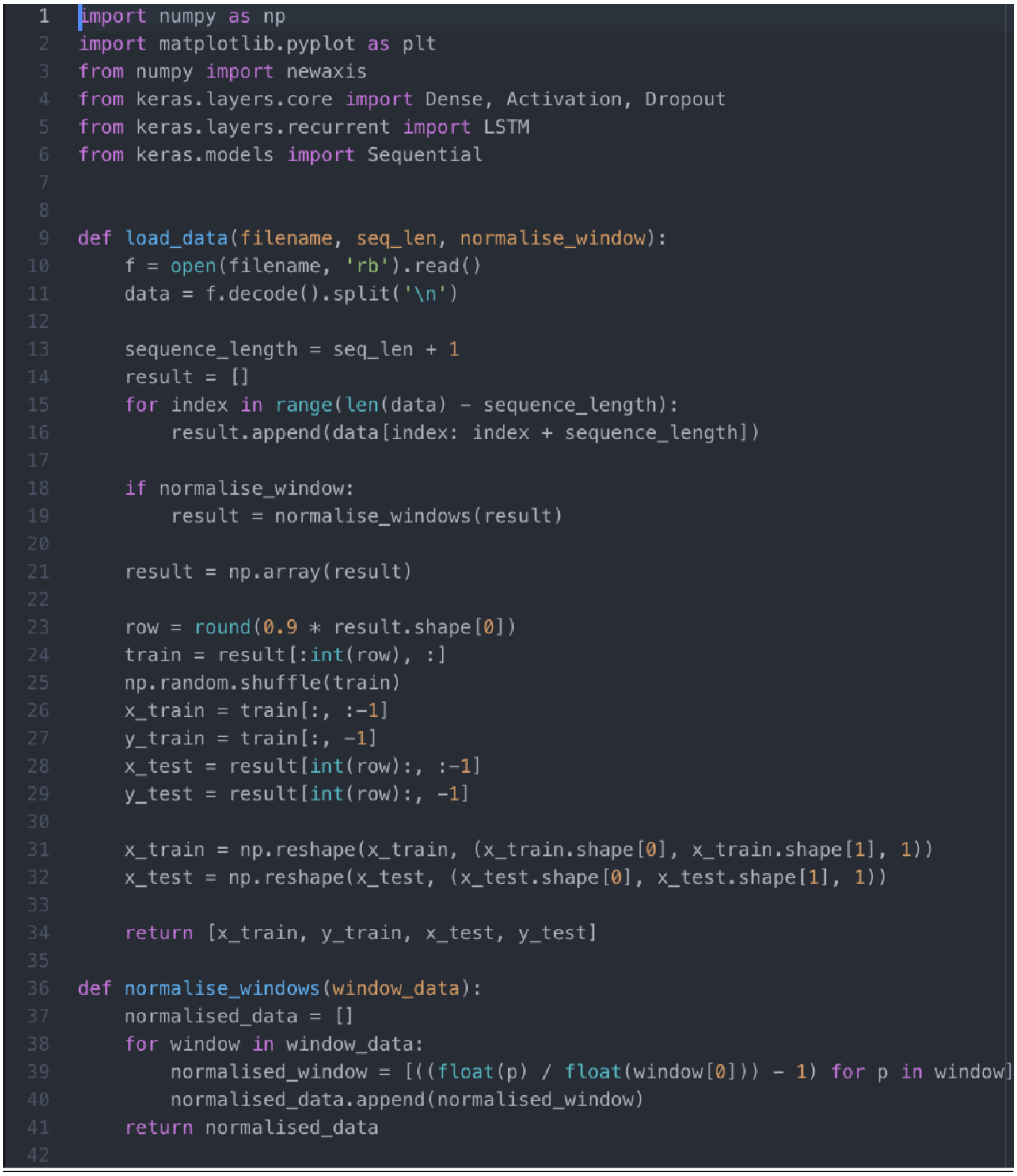
10. “What deep learning is and isn't,” *The Data Scientist*, 28-Aug-2018. [Online]. Available:https://thedatascientist.com/what-deep-learning-is-and-isnt/.

11. “A Gentle Introduction to Transfer Learning for Deep Learning,” *Machine Learning Mastery*, 12-Dec-2018. [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/.

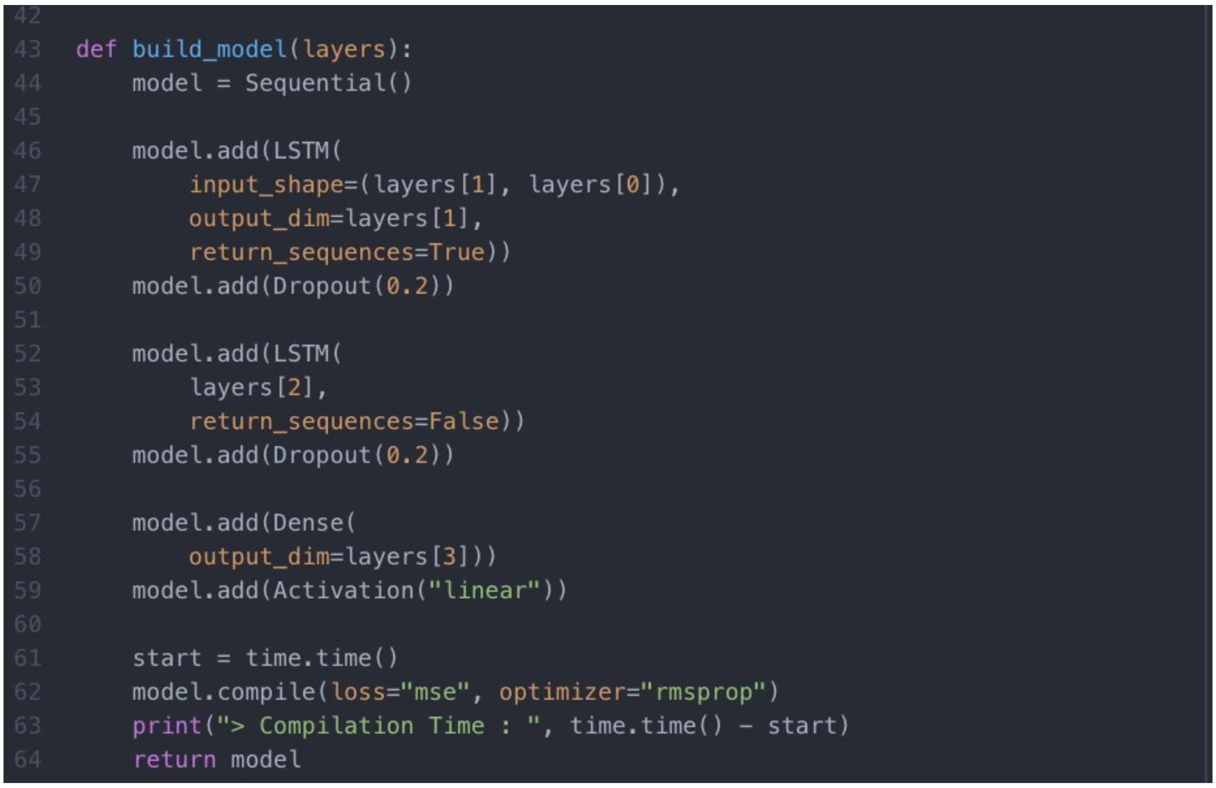
12. “How To Identify Patterns in Time Series Data: Time Series Analysis,” *statsoft*. [Online]. Available: http://www.statsoft.com/Textbook/Time-Series-Analysis.

Appendix A – Single LSTM model code snippets

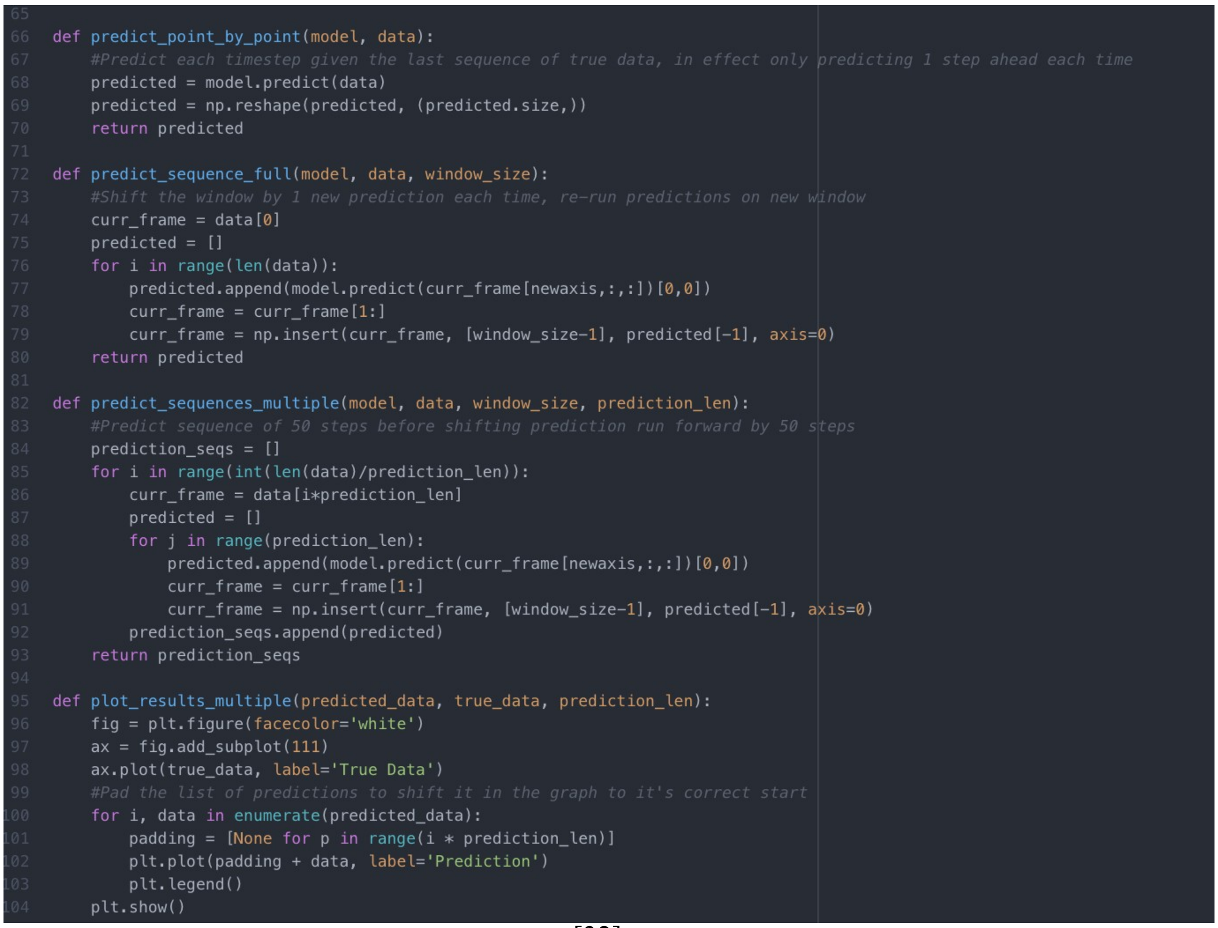
<u>Building the model:</u>



```python
import numpy as np
import matplotlib.pyplot as plt
from numpy import newaxis
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential


def load_data(filename, seq_len, normalise_window):
    f = open(filename, 'rb').read()
    data = f.decode().split('\n')

    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])

    if normalise_window:
        result = normalise_windows(result)

    result = np.array(result)

    row = round(0.9 * result.shape[0])
    train = result[:int(row), :]
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = result[int(row):, :-1]
    y_test = result[int(row):, -1]

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

    return [x_train, y_train, x_test, y_test]

def normalise_windows(window_data):
    normalised_data = []
    for window in window_data:
        normalised_window = [((float(p) / float(window[0])) - 1) for p in window]
        normalised_data.append(normalised_window)
    return normalised_data

```

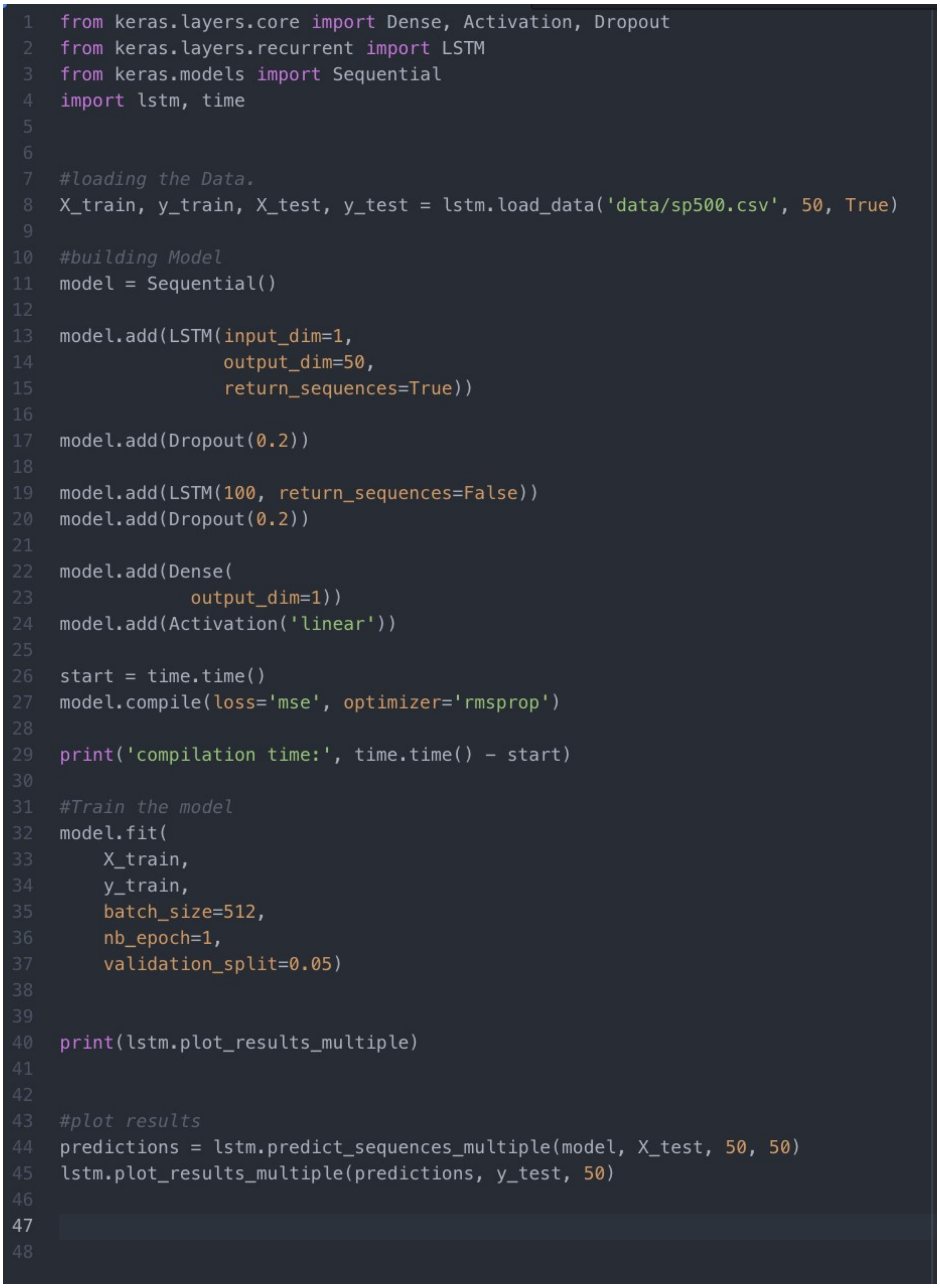


```python

def build_model(layers):
    model = Sequential()

    model.add(LSTM(
        input_shape=(layers[1], layers[0]),
        output_dim=layers[1],
        return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(
        output_dim=layers[3]))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop")
    print("> Compilation Time : ", time.time() - start)
    return model
```



```python

def predict_point_by_point(model, data):
    #Predict each timestep given the last sequence of true data, in effect only predicting 1 step ahead each time
    predicted = model.predict(data)
    predicted = np.reshape(predicted, (predicted.size,))
    return predicted

def predict_sequence_full(model, data, window_size):
    #Shift the window by 1 new prediction each time, re-run predictions on new window
    curr_frame = data[0]
    predicted = []
    for i in range(len(data)):
        predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
        curr_frame = curr_frame[1:]
        curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1], axis=0)
    return predicted

def predict_sequences_multiple(model, data, window_size, prediction_len):
    #Predict sequence of 50 steps before shifting prediction run forward by 50 steps
    prediction_seqs = []
    for i in range(int(len(data)/prediction_len)):
        curr_frame = data[i*prediction_len]
        predicted = []
        for j in range(prediction_len):
            predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
            curr_frame = curr_frame[1:]
            curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1], axis=0)
        prediction_seqs.append(predicted)
    return prediction_seqs

def plot_results_multiple(predicted_data, true_data, prediction_len):
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    #Pad the list of predictions to shift it in the graph to it's correct start
    for i, data in enumerate(predicted_data):
        padding = [None for p in range(i * prediction_len)]
        plt.plot(padding + data, label='Prediction')
        plt.legend()
    plt.show()
```

Running the model:



```python
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import lstm, time


#loading the Data.
X_train, y_train, X_test, y_test = lstm.load_data('data/sp500.csv', 50, True)

#building Model
model = Sequential()

model.add(LSTM(input_dim=1,
               output_dim=50,
               return_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(100, return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(
            output_dim=1))
model.add(Activation('linear'))

start = time.time()
model.compile(loss='mse', optimizer='rmsprop')

print('compilation time:', time.time() - start)

#Train the model
model.fit(
    X_train,
    y_train,
    batch_size=512,
    nb_epoch=1,
    validation_split=0.05)


print(lstm.plot_results_multiple)


#plot results
predictions = lstm.predict_sequences_multiple(model, X_test, 50, 50)
lstm.plot_results_multiple(predictions, y_test, 50)
```