Long-Short Trading Strategy Using Deep Learning

•CURIC•

Author

Sameer Singh

Published by CityU Research and Investment Club

THE FINAL PAGE OF THIS REPORT CONTAINS A DETAILED DISCLAIMER

The content and opinions in this report are written by a university student from the CityU Student Research & Investment Club, and thus are for reference only. CityU Student Research & Investment Club is not responsible for any direct or indirect loss resulting from decisions made based on this report. The opinions in this report constitute the opinion of the CityU Student Research & Investment Club and do not constitute the opinion of the City University of Hong Kong nor any governing or student body or department under the University.

Table of Contents

1. Introduction

Artificial Intelligence (AI) is a term that sounds familiar to everyone in this day and age. The world is surrounded by some or the other application of AI and knowingly or unknowingly, most people make use of them in their everyday lives. The recommendations for videos, music and movies on YouTube, Spotify and Netflix, the 'explore' feature on Instagram, Siri on iPhone or Google Assistant on Android phones, the facial recognition software on smartphones – all these are features powered by AI that people come across regularly. More complex systems include self-driving cars, recommendation systems, question answering systems, robots, intelligent home appliances et cetera. 83% of the businesses say that AI is a strategic priority for their growth (Columbus, L., 2017)[1]. As of 2018, the field was valued at $21.46 billion and is likely to reach $190.61 billion by 2025 ("Artificial Intelligence Market by Offering", n.d.)[2], giving it a massive CAGR of 36.62%. It is not surprising to say that AI is one of the most widely discussed and researched topics today, with its capabilities being expanded to almost every field there is, and financial services is one such field. Asset management firms are exploring possibilities of using complex AI algorithms to make investment decisions and find new opportunities. Commercial banks use AI to check a person's credit score before approving a loan request. Individual investors use recommendation systems that recommend stocks based on their past investment decisions and market state. Whilst these are just a few examples of numerous applications currently being used, the possibilities are endless.

In this report, I attempt to make use of this rapidly growing field to design an algorithmic trading strategy that uses Deep Learning techniques. I construct a model to predict the *Open* and *Close* price for the next day based on the present day's *Open, Close, High, Low* prices and the *Volume* of shares traded. The strategy then checks if the stock price is supposed to increase (*Close* > *Open*) or decrease (*Open* > *Close*) based on the predictions made. This increase/decrease will indicate a

[1] Columbus, L. (2017, September 10). How Artificial Intelligence Is Revolutionizing Business In 2017. Forbes. Retrieved 2020, from https://www.forbes.com/sites/louiscolumbus/2017/09/10/how-artificial-intelligence-is-revolutionizing-business-in-2017/#5e981ba55463

[2] Artificial Intelligence Market by Offering (Hardware, Software, Services), Technology (Machine Learning, Natural Language Processing, Context-Aware Computing, Computer Vision), End-User Industry, and Geography – Global Forecast to 2025 (Rep.). (n.d.). doi:https://www.marketsandmarkets.com/Market-Reports/artificial-intelligence-market-74851580.html

long/short position on that stock. This model is a Neural Network that has a Long-Short-Term Memory architecture and is trained on the SPDR S&P 500 (NYSEARCA: SPY) data from 1995 until 2015. The SPDR S&P 500 is an ETF that tracks the S&P 500, hence giving an indication of the whole market position in general. The model will be backested on historical price data of various companies from 2010 until mid-2020.

In the following sections, I will start by discussing the theory and working of neural networks (Section 2). After this, in the main text of the report, I will demonstrate the development process of the strategy in detail using code snippets, and discuss the various libraries used and walk through each stage of the program development from scratch (Section 3 and 4). Moving on, I will define the trading strategy in detail, lay out the evaluation metrics and backtesting parameters before finally testing the strategy on Intel Corporation and Home Depot Inc., followed by a detailed discussion on the returns and the overall performance of the algorithm (Section 5). This section also includes test results of the strategy on 100 randomly selected S&P 500 stocks. I will finally conclude the report by discussing some improvements that can be made in order to make the algorithm more robust (Section 6). Even though it is advisable to go through each section in order to absorb the full essence of the report, readers can focus solely on particular sections as per their interests:

- Section 2 is an introduction to advanced AI concepts, suitable for people interested in the theory of Neural Networks.
- Section 3 and 4 are for those interested in programming, it will give a good idea of the general workflow of developing an AI system. This section will point the readers to resources that they can use to build their own programs.
- Section 5 is the Finance intensive section. All previous sections are a build up to this one. Suitable for those who are curious about the trading algorithm and its performance based on real market data.

The appendix at the end of the report is useful for interested readers as it contains articles and blogs that helped me understand the concepts used in this report. It also contains the source code to the algorithm, results of the backtesting as well as the predictions of prices made by the model.

The purpose of this report is to present the idea behind an algorithmic trading strategy. I also aim to demonstrate the development of the seemingly complex concept of neural networks and its possible application in finance.

2. Literature Review

Here I will discuss the base of my strategy: the neural network. This will help readers understand the technology behind the strategy.

I will start by explaining what Artificial Neural Networks (ANN) are, the motivation behind their development and their limitations. Following which, I will dive deeper into the ANNs by discussing Recurrent Neural Networks (RNN) and its Long-Short-Term Memory (LSTM) architecture.

2.1 Artificial Neural Network (ANN)

In laymen terms, the purpose of artificial intelligence is to make machines learn and make decisions like humans do. Humans do this using millions of interconnected cells in the nervous system called neurons. These neurons communicate with one another in unique ways and transmit information from the brain to various parts of our body. The brain is what it is due to the structural and functional properties of these interconnected neurons.

Since the nervous system is what helps humans become intelligent, researchers found it only natural to have a similar system for computers that would help them become more intelligent. Thus began the development of the Artificial Neural Networks for machines that would help achieve Artificial Intelligence. ANN is the building block of every complex AI system. It is nothing but a network of connected neurons, called nodes. Figure 1 visualizes an ANN with 2 hidden layers:

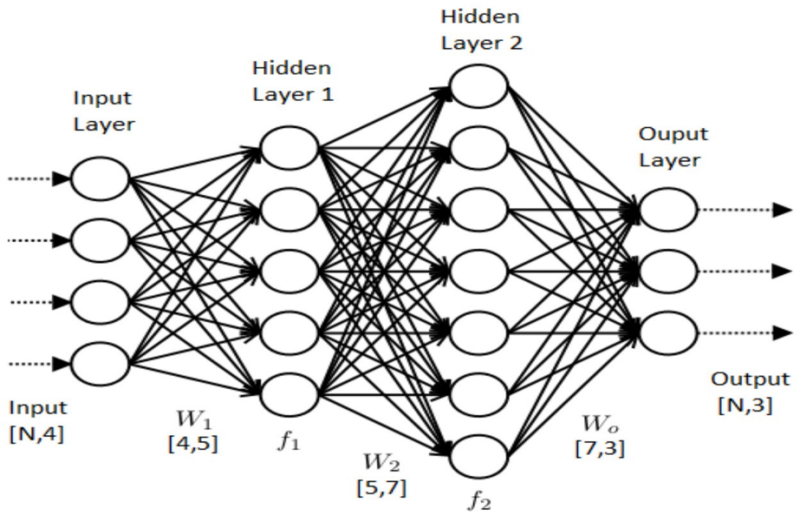


Figure 1: ANN with 2 hidden layers (Source)

As you can see from figure 1, a typical ANN consists of nodes (the circles) distributed across various layers. There are 3 layers in every network: **input, hidden and output;** responsible for accepting the inputs, processing the inputs and displaying the results respectively. The number of input and output nodes depend on the number of input and output parameters, while the number of nodes in the hidden layers depend on the complexity of the problem. Additionally, the number of hidden layers also depend on the complexity of the problem while there are always one input and one output layer in each ANN. The nodes in one layer are connected to every node in the subsequent layer, each of these connections have a specific weight attached to them which help in mapping the inputs to the output. The final values of these weights are iteratively adjusted through the training process. Thus, the main purpose of the network is to learn weights that would appropriately map the inputs to outputs.

The working of an ANN is beyond the scope of this report, interested readers may refer to resources mentioned in Appendix I.

2.2 Recurrent Neural Network (RNN)

A major drawback of ANN is its parameter independence. A predicted output at time t (Y_t) only depends on the input parameters (X_t) at that given time, the network does not consider the effect of any previous input. Many real-world tasks such as sentence prediction, pattern recognition, time series analysis et cetera, are optimized only by considering the effect of previous data. For instance,

in order to predict the stock price for Friday, based on Thursday's inputs, it would make sense to consider the price movements of Wednesday too. Traditional ANNs do not do this.

Recurrent Neural Networks were designed to tackle this problem. RNN is a special type of ANN which has a recurring connection to itself. This self-loop mechanism helps nodes consider the previous state of the network while calculating the current output. This gives the network a sense of time. Figure 2 shows a simplistic diagram of an RNN:

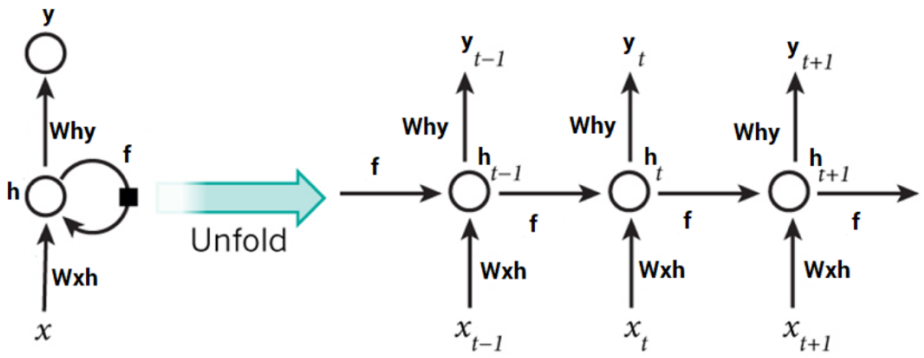


Figure 2: RNN (Source)

The network maintains a hidden state vector *H* which is passed from one node to the other. This is the "memory" of the network containing information about the previous output. This means that the output Y_t at time t is based on the input X_t as well as the hidden state vector H_t. This H_t is based upon H_{t-1} which in turn is based upon H_{t-2} which of course is based upon H_{t-3} and so on. This way, all the outputs partially take what has happened prior to the current time into account.
Again, the working of RNNs is beyond the scope of this report, interested readers may refer to the Appendix I.

2.3 Long-Short-Term Memory (LSTM)

As mentioned above, RNN gives neural network a memory which optimizes many real-world tasks. This works very well for tasks that do not deal with long sequence of data. For instance, RNN would work well when we are predicting the stock price for Friday using Thursday's inputs and considering price movements on Wednesday and Tuesday. However, estimates about price, earnings, sales et cetera, are typically made by considering performance of many years prior. The strategy presented in this report also trains the model on the market data for 20 years. When the

data that is to be processed is in such large quantities, RNN's memory falls short in its capabilities. The self-loop mechanism only lets the network have short-term memory due to a limitation termed as **vanishing gradient.** Now, theoretically when calculating the output Y_t at time t, the hidden state vector H_t should contain information about ALL of the past computations starting from t=0. However, as the hidden state vector is iteratively calculated at each epoch, the initial information starts losing its significance, thus making the state vector carry information only about the most recent iterations.

This is where LSTM comes into play. LSTM is a special type of RNN that enables the network to have long term memory. The core functionality of an LSTM network is the same as RNN: predict an output using the input provided and the hidden state, compute a new value of hidden state based on the output and pass it along to the next node. The difference lies in the node architecture. Figure 3 shows a single node of an RNN network and figure 4 shows the cell architecture of a single LSTM:

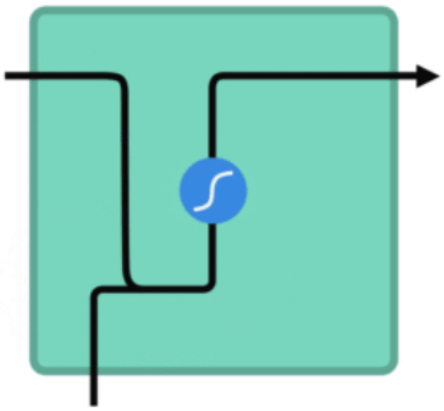


Figure 3: A single RNN cell

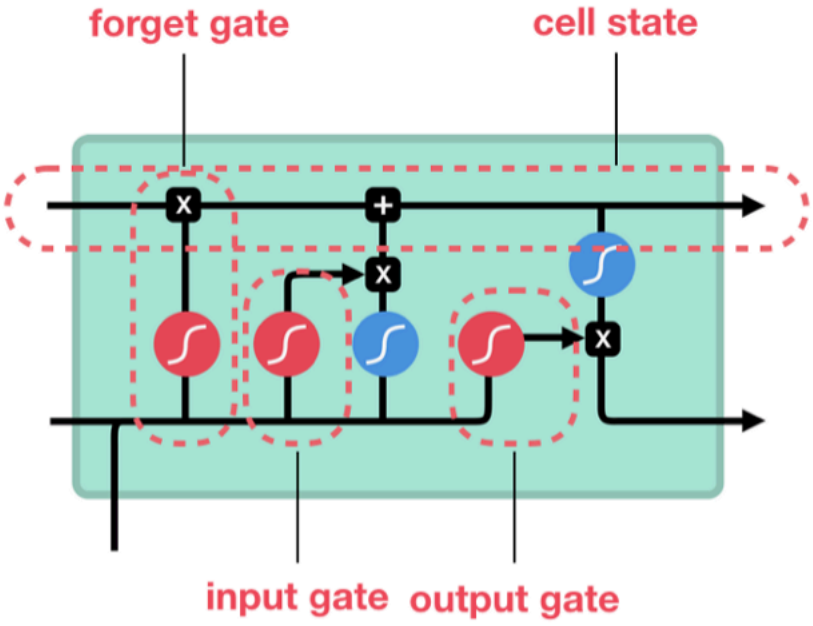


Figure 4: A single LSTM cell

As you can see from Figure 3 and 4, the cell of a LSTM network is much more complicated than that of the RNN network. The most important part of the former is the cell state that it maintains along with the hidden state vector. The cell state runs across one iteration to the next containing information about the past iterations. However, unlike the hidden cell state in RNN which

processes all information from each iteration using a single *tanh* function, the cell state is selective. It chooses to store the important information and forget any irrelevant ones, thus increasing the quality of information stored and the longevity of the memory. It does so by using 3 gates as shown in the figure: **forget, input and output** gate. It uses *sigmoid* and *tanh* functions, the hidden and cell states from previous cell, inputs and a series of addition and multiplication to compute the output, new cell state and a new hidden state.

Most state-of-the-art results in sequence prediction tasks such as time series analysis, natural language processing et cetera, make use of LSTM Networks.

For details of how LSTM works, please refer to the Appendix I.

2.4 Summary

This section described three types of Neural Networks that are used in real world applications: ANN, RNN and LSTM. ANN have no memory at all and one set of output does not depend on any previous output. RNN, on the other hand, maintains a hidden cell state using a self-looping mechanism, giving the network a memory. The output in each iteration depends on this hidden state and the input parameters. However, this memory is short lived due to the vanishing gradient. LSTM takes this one step further and maintains an additional cell state which selectively adds important information and forgets unimportant information, giving the network a much longer memory.

3. Neural Network Development

In this section, I will discuss the neural network model development, which is the driving force behind the trading strategy. I have used Python 3.7 as the programming language and Jupyter Notebook as the environment for development. I will include the screenshots of selected portions of the code, for the full source code, please check the GitHub link provided in Appendix IV.

3.1 Libraries

```
# Group 1
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
import math
import datetime
import random

# Group 2
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.callbacks import EarlyStopping
from keras import metrics

# Group 3
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score as r2

# Group 4
from bs4 import BeautifulSoup
import requests
```

Figure 5: Libraries

Figure 5 shows all the libraries needed to develop the program. As you can see, I have divided them into 4 groups.

Group 1

Group 1 contains all the general helper libraries: *pandas* for data processing and storage, *numpy* for scientific computation, *matplotlib* for data visualization, *yfinance* for retrieving stock price data from Yahoo finance, *math* for easily applying mathematical functions like log, datetime for retrieving today's and tomorrow's dates and *random* for selecting random companies from the S&P 500 to test the algorithm.

Group 2

Group 2 contains the libraries used for constructing the neural network. I use *Keras*, a high-level neural net library. It works as an API for the development of deep learning models. *Keras* can be thought of as a front-end library running on top of Tensorflow, Microsoft Cognitive Toolkit, R, Theano or PlaidML. In this project, it runs over Tensorflow. Even though it does not provide as

much flexibility in the details of the model, it makes the development much simpler as compared to using Tensorflow or any other deep learning library.

Group 3

Group 3 contains all the helper libraries for the deep learning model. I have imported various functions from scikit-learn, an open-source machine learning library. These functions will help process the inputs and outputs for the neural network and help evaluate the model performance.

Group 4

Finally, Group 4 contains the two libraries I use for web scraping. This is used only to extract the beta for various securities from Yahoo finance directly.

3.2 Workflow

Now that the various libraries have been discussed, the following paragraphs will discuss on how these are used in order to construct the model and evaluate its performance.

Since the aim is to build a model that can be implemented on stocks of various companies, I decided to train my model on market data. This will generalize the model and not inclined towards any one industry.

3.2.1 Data Preparation

The first requirement towards creating any neural network is to prepare the data. For this strategy, the dataset to be used is historical prices of an ETF tracking the S&P 500: The SPDR S&P 500 trust (NYSEARCA: SPY) with the timeframe set from January 1995 to January 2015. This timeframe will cover the .com bubble burst of ’02 and the ’08 financial crisis, providing enough variations for the model to learn.

This data can be downloaded using *yfinance* which returns the daily *High, Low, Open, Close* and *Adjusted Close* prices. Figure 6 shows the price movements of the data.

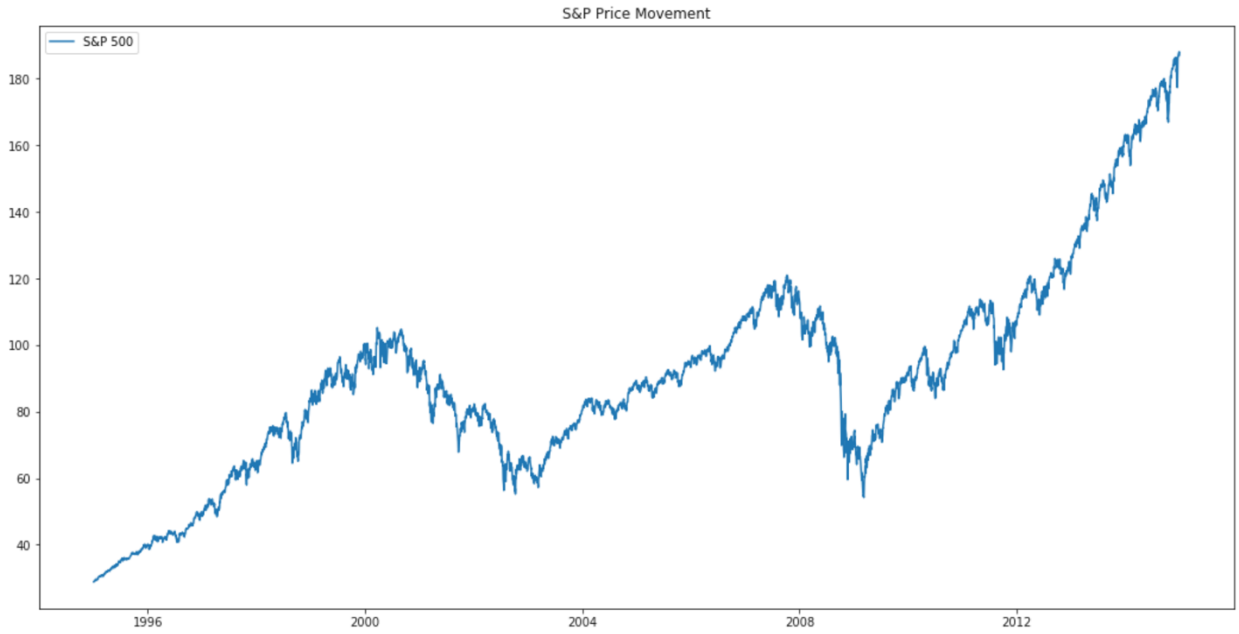


Figure 6: S&P 500 from 1995 to 2015

I have added two more columns: “Open next day” and “Close next day”. Figure 7 shows the raw data that will be used for the model.

Date	Open	High	Low	Close	Adj Close	Volume	open_next_day	close_next_day
1995-01-03	45.70	45.84	45.69	45.78	28.84	324300	45.98	46.00
1995-01-04	45.98	46.00	45.75	46.00	28.98	351800	46.03	46.00
1995-01-05	46.03	46.11	45.95	46.00	28.98	89800	46.09	46.05
1995-01-06	46.09	46.25	45.91	46.05	29.01	448400	46.03	46.09
1995-01-09	46.03	46.09	46.00	46.09	29.04	36800	46.20	46.14
					.....			
2014-12-23	208.17	208.23	207.40	207.75	187.16	122167900	208.02	207.77
2014-12-24	208.02	208.34	207.72	207.77	187.18	42963400	208.31	208.44
2014-12-26	208.31	208.85	208.25	208.44	187.78	57326700	208.22	208.72
2014-12-29	208.22	208.97	208.14	208.72	188.04	79643900	208.21	207.60
2014-12-30	208.21	208.37	207.51	207.60	187.03	73540800	207.99	205.54

Figure 7: Raw Data

The first 5 columns of this data will be used as the inputs and the last two columns will serve as the outputs. As seen, the integer value of volume is exponentially larger than all the other 4 input

parameters. It would therefore make sense to normalize these values to represent each parameter in the same range. This process is known as **feature scaling**, which is achieved using the MinMaxScalar class of the scikit-learn library (Group 3). An object of the class is created, and the input parameters are fit into it. This object then transforms every value to number between 0 and 1. The same is applied to the output parameters too. The same object can later be used to transform these scaled input and output values to their original values.

Figure 8 shows the scaled values for trading day 1995-01-06 *(see figure 7)*.

```
data['x'][3]
```

```
array([[0.00239838, 0.00165705, 0.00245318, 0.00135335, 0.00050389]])
```

Open Close High Low Volume

```
data['y'][3]
```

```
array([0.00030801, 0.0005531 ])
```

Open next day Close Next Day

Figure 8: Scaled Inputs and Outputs

- data[‘x’] corresponds to the inputs: an array of 5 elements corresponding to *Open, Close, High, Low* and *Volume* respectively.
- data[‘y’] corresponds to the outputs: an array of 2 elements corresponding to High next day and Low next day respectively.

These are one of 5035 such arrays used for training.

3.2.2 Constructing the Neural Network

Now that the pre-processing of the data is complete, the next step is to construct a neural network that we can feed the data into. The model used in this strategy is a simple 3-layer neural network: one input layer, one hidden layer and one output layer. The input and output layers will have 5 and 2 neurons respectively. The most crucial part of the network, the hidden layer will consist of 75 LSTM cells. The model would look something like this:

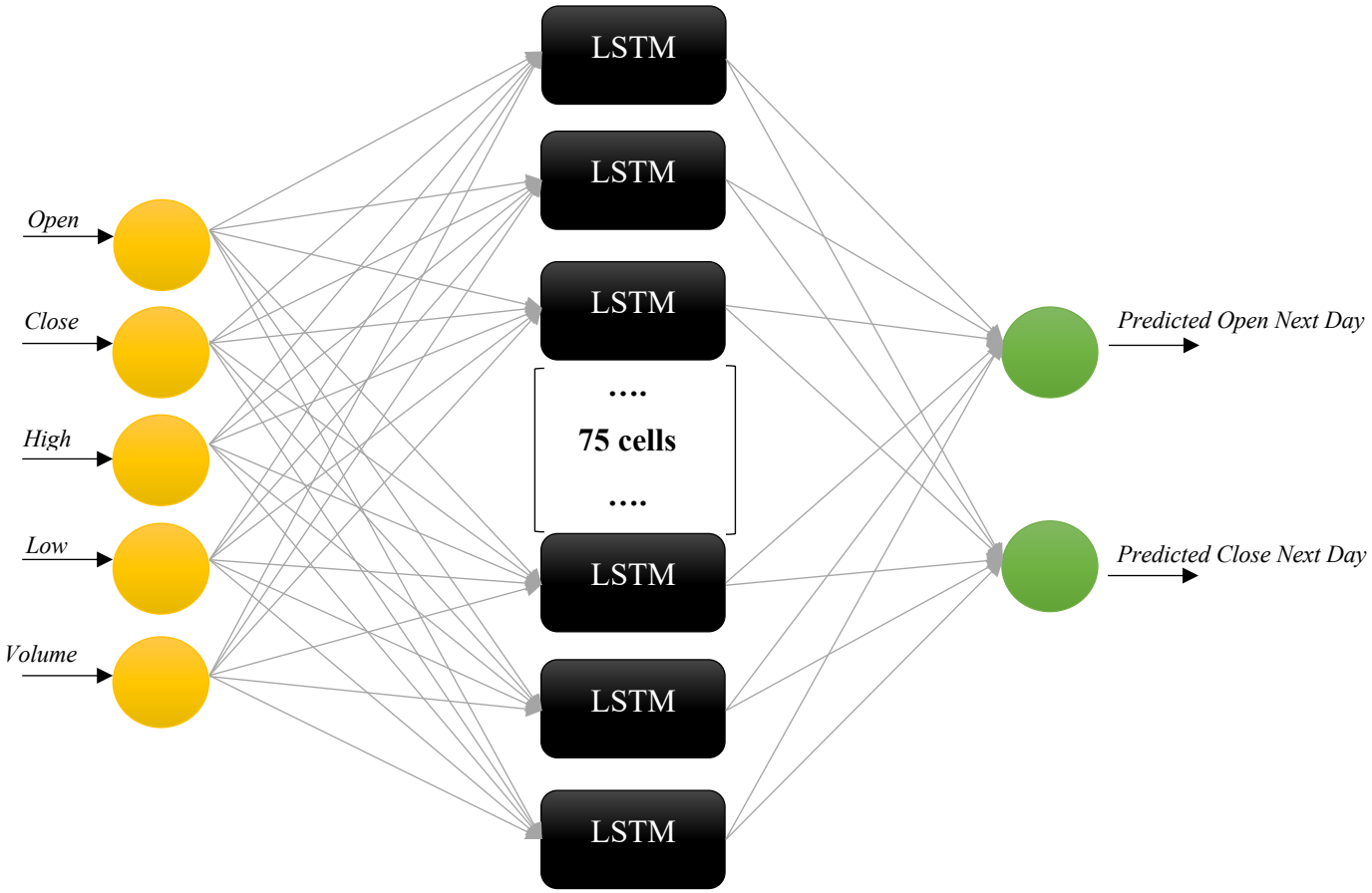


Figure 9: Neural Network Architecture

This model is created using *Keras* (Group 2). Figure 10 shows the Python code for the same:

```
model = Sequential()
model.add(LSTM(75, input_shape = (1,5), activation='tanh', recurrent_activation='sigmoid'))
model.add(Dense(2))

model.compile(loss='mse', optimizer = 'adam', metrics = [metrics.mae])

callback = EarlyStopping(monitor='loss', patience=3)
```

Figure 10: Keras Code for the model

The layers are defined in the second and third line. In the LSTM layer, the *sigmoid* function is used for the input, forget and output operations while the *tanh* function is used for calculating the hidden state. These functions play a crucial role in calculating the cell state of the network.

The compile function (line 5) establishes 3 important model parameters: **Loss function**: also called the 'cost function', measures how expensive a particular neural network is based on the set of weights used. As mentioned in section 2.1, the purpose of a neural network is to learn weights which would appropriately map inputs to outputs. Another way of saying this is: the purpose of a

neural network is to find weights which result in the lowest loss function. Here, the loss functions used is the **Mean Squared Error (MSE).** Formula for MSE is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Where $n = number\ of\ data\ points$

$y_i = Predicted\ Output\ of\ data\ point\ i$

$\tilde{y}_i = Actual\ Output\ of\ data\ point\ i$

The term in the parenthesis defines the deviation of a predicted output from the actual output at a particular datapoint. The average of sum of the squared of this deviation gives the Mean Squared Error. Squaring the deviation gives more weight to larger errors made by the model, making it an appropriate choice for evaluating the loss. It also measures the accuracy of the model, lesser the MSE more accurate the model is.

The second parameter is the **Optimizer.** The optimizer is an algorithm that defines how exactly the weights of the network are modified based on the loss function. It can be seen as the entity that does all the legwork to make the model as accurate as possible. The algorithm used in this network is called **Adam Optimizer.**

The final parameter defined is the **Metric.** A metric is a function that evaluates the performance of a neural network after each iteration. It can be seen as a supervisor that judges the optimizer's work. Similar to the loss function, a metric also measures the accuracy of the model. Lesser the better. The metric used here is the **Mean Absolute Error (MAE)** whose formula is given by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\, y_i - \tilde{y}_i \,|$$

The MAE is simply the average of the absolute value of the deviation of the model. It gives an idea about the total deviation of the predicted values from the actual outputs.

The architecture of the network along with the 3 parameters mentioned constitute the minimum requirements for any neural network regardless of the task. Additional modifications can be made at the discretion of the developer. Here, I have added a **callback function** *(last line Figure 10)* which monitors the loss function. Ideally, due to the work of the optimizer, the loss function should decrease in value after every iteration. However, due to various reasons, a situation can arise where

that value increases for a certain number of iterations indicating that the minimum has been reached. The callback function would terminate training at this point of minimum loss.

This concludes the construction of the LSTM Neural Network used in this strategy. The next step is training this network on the SPY data prepared in section 3.2.1.

3.2.3 Learning

The neural network is now ready to accept the data that we have initially prepared (5035 arrays of transformed input and output values) and learn how the price varies based on the previous day's price movements. There are two steps involved in the learning process: **training** and **testing. On one hand,** during the training phase, the network is provided access to the inputs (arrays of 5 elements) as well as the outputs (arrays of 2 elements) using which it can iteratively adjust its weights by minimizing the loss function.

On the other hand, the testing phase is the "exam". Here, the network is provided with a new set of inputs based on which it makes predictions using the weights learned during the training phase. These predictions are then compared to the corresponding true values in order to evaluate how well the model has learned.

The first step therefore, is to split the original data into train and test dataset. This is done using scikit-learn's train_test_split function (figure 11).

```
x_train, x_test, y_train, y_test = train_test_split(data['x'],data['y'],test_size = 0.25)
```

Figure 11: Splitting data into train and test

As shown in the figure, the test size is set as 0.25. This means that 75% of the data will be used by the network to learn appropriate weights and the rest will be used to evaluate its performance. The function randomly chooses 3776 (75% of 5035) rows from data['x'] and data['y'] and stores it into x_train and y_train respectively. The rest is stored in x_test and y_test. This type of learning where the model learns using both inputs and outputs is called **supervised learning.**

After splitting, the next step is to pass x_train and y_train to the model and run it over a number of iterations. Figure 12 shows the code to do this.

```
history = model['model'].fit(x_train, y_train, epochs=100, callbacks=[model['callback']])
```

Figure 12: passing data to model

We set the model to run a maximum of 100 iterations to learn the most optimal weights. Note that this is the maximum number of iterations, the training will be terminated if the loss function increases during the process due to the callback function defined in section 3.3.2. Here, the training stopped at epoch 23 (figure 13).

```
Epoch 1/100
3776/3776 [==============================] - 1s 143us/step - loss: 0.0479 - mean_absolute_error: 0.1470
Epoch 2/100
3776/3776 [==============================] - 0s 43us/step - loss: 0.0020 - mean_absolute_error: 0.0327
Epoch 3/100
3776/3776 [==============================] - 0s 44us/step - loss: 6.1552e-04 - mean_absolute_error: 0.0170
                                                    .....
Epoch 21/100
3776/3776 [==============================] - 0s 48us/step - loss: 6.0167e-05 - mean_absolute_error: 0.0054
Epoch 22/100
3776/3776 [==============================] - 0s 48us/step - loss: 5.9811e-05 - mean_absolute_error: 0.0054
Epoch 23/100
3776/3776 [==============================] - 0s 44us/step - loss: 5.9835e-05 - mean_absolute_error: 0.0054
Training finished
```

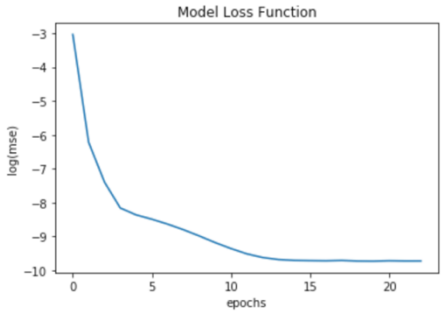


Figure 13: Model Training

As you can notice, the loss function has reduced from a value of 0.0479 in the first iteration to 0.000059 in the last iteration. The graph shows the log of the loss function at each iteration, a value that is consistently decreasing.

At this point, the network has learned how to predict the *Open* and *Close* prices for the next day to the best of its capabilities. Putting this to test, the "x_test" array is passed to the network and the outputs are compared with the true values. In order to evaluate the model's performance, I use 3 metrics:

1. Mean Absolute Error: *described previously*
2. Mean Squared Error: *described previously*
3. R^2 Score: Also called the **coefficient of determination**, is a common metric used to evaluate regression models. The value suggests the percentage of the variance in the output that the input can explain collectively.

The following table summarizes the performance for each output.

Metric	Open	Close
MAE	0.7102541699761716	1.1004447974583003
MSE	1.028312708498809	2.4780122319301032
R^2	0.9990199291650923	0.9976389885063128

With a MAE of 0.7 for *Open* and 1.1 for *Close* and MSE of 1.02 for *Open* and 2.4 for *Close*, the predictions for *Open* prices are slightly better than for the *Close* prices. However, in general the error values are low enough for the model to be labelled a success. The R^2 value of 0.99 suggests than 99% of the variation in *Open* and *Close* prices for the next can be explained by the *Open, Close, High, Low* prices and *Volume* of shares traded of the current day.

The model has now successfully learned how to predict the *Open* and *Close* prices for any day using the 5 parameters from the previous day and is ready to be applied to stocks of different companies.

4. Testing on Specific Companies

In this section, I will test the model's predictions on specific companies by predicting the *Open* and *Close* prices from 1st January 2010 to 30th May 2020. This timeframe will include the 2020 market crash, putting the model to a more rigorous test.

As demonstrated in the previous section, the model learned weights based on the SPY data and produced pleasingly accurate predictions on the testing data. It will be interesting to explore how it will perform on a company's stock.

For this, I will pick two securities from different industries. The first one is a very prestigious Wall Street darling from the Information Technology industry: Intel Corporation (NASDAQ: INTC). The second is another market giant from the Consumer Discretionary industry: Home Depot (NYSE: HD).

The stock price data will of course be downloaded using *yfinance* from 1st Jan 2010 until 30th May 2020; this will serve as our testing data. All the data pre-processing steps mentioned in section

2.3.1 will be applied to this data: creating 'open_next_day' and 'close_next_day' columns, feature scaling the inputs and storing them in an array. However, this time, the output data will never be passed to the network and will only be used to compute the model's predictions.

4.1 Model Performance on Intel Corporation [INTC]

After running the model on this data, the same 3 metrics are used for its evaluation and are summarized in the table below:

Metric	Open	Close
MAE	0.29984347439212805	0.4434707765699388
MSE	0.2366591654251627	0.5132274590953694
R^2	0.9982822916931641	0.9962831711994932

The network's performance is better on INTC than it was on SPY. Both MSE and MAE are lower for Open and Close while R^2, as expected is the same. Figure 14 visualizes the performance for the past 100 days (for the plot of the whole timeframe, refer to Appendix II):

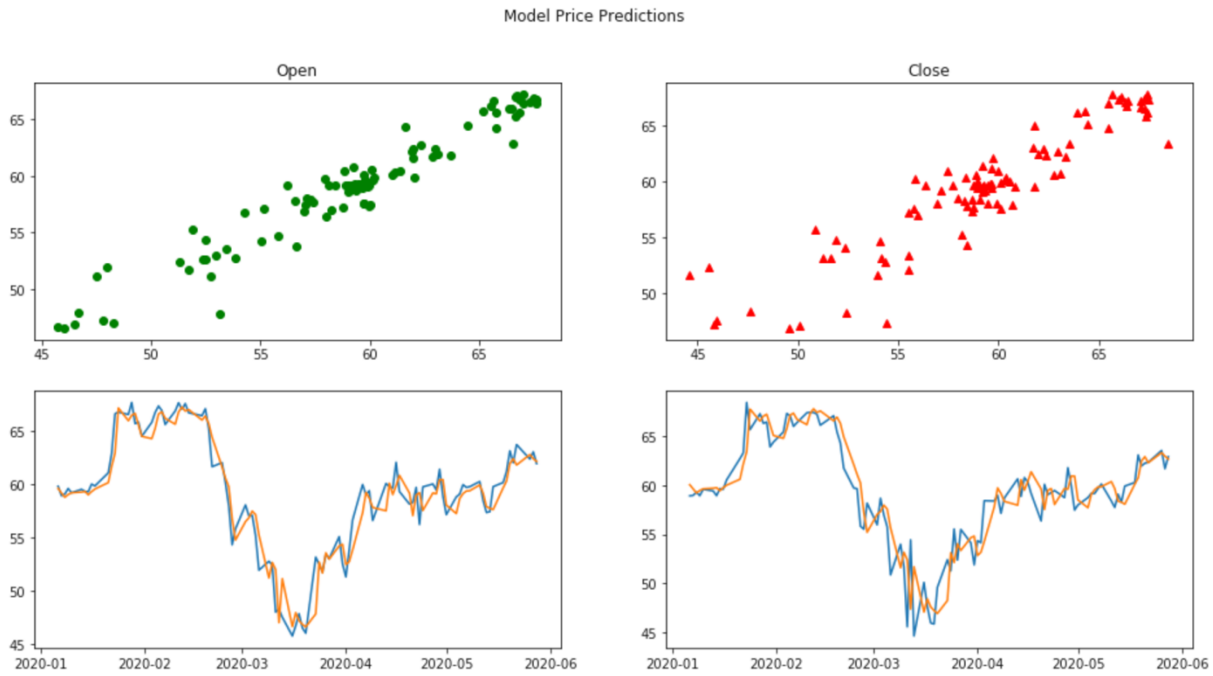


Figure 14: Model Performance on INTC

The first row shows a scatter plot with true values on the x-axis and predicted values on the y-axis. As you can see, even during the 2020 market crash, the scatter plot displays a somewhat linear relationship. Joining forces with the low evaluation metrics, this further stresses on the accuracy of the model.

The second row plots the predicted (orange) and the true (blue) values for INTC for the past 100 days. It is clear that the predicted values follow the same general direction of the true values pointing to satisfying performance of the model.

4.2 Model Performance on Home Depot [HD]

The following table summarizes the model's performance on HD.

Metric	Open	Close
MAE	1.0686788586780012	1.3628216446989043
MSE	3.1279949879025626	5.049304219578166
R^2	0.9992003693175584	0.9987107190506682

Here we see that the model has performed worse than it did on the SPY data. The MAE and MSE for both Open and Close are higher than the SPY dataset while R^2, as expected is the same. Despite performing worse, the model still predicts the prices with satisfactory results as depicted by figure 15:

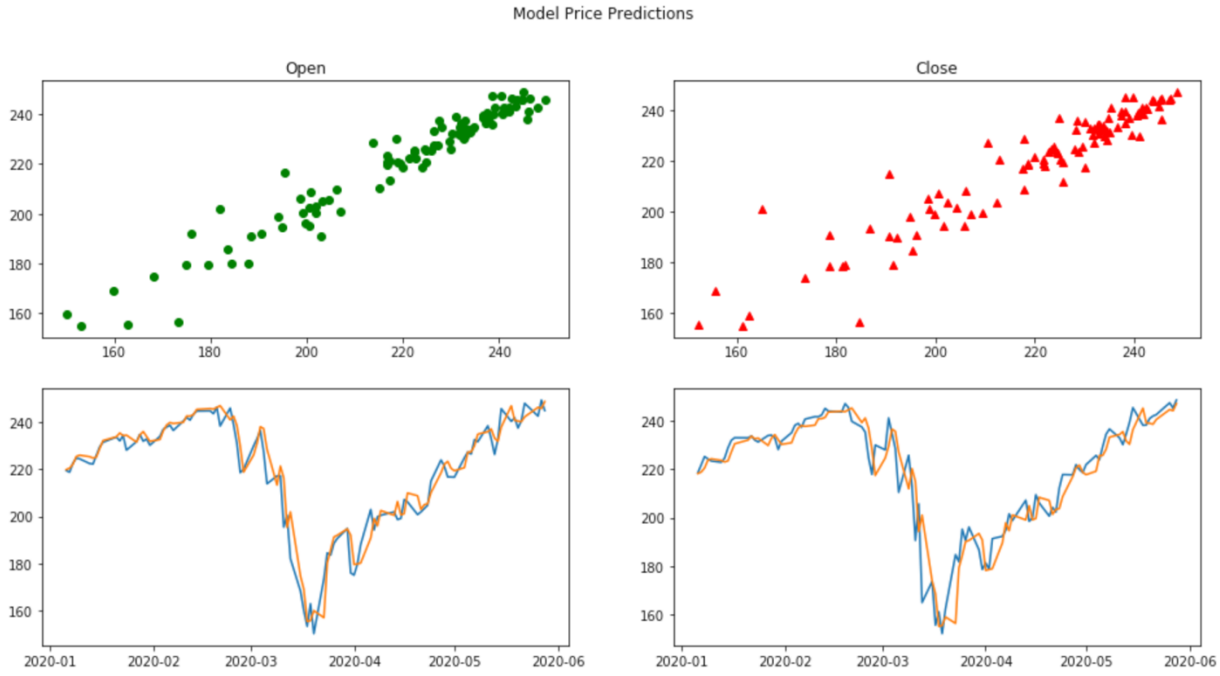


Figure 15: Model Performance on HD

Similar to INTC, the scatter plot again shows a more or less linear relationship between the true and predicted values and the two follow the same general direction in the plot, hence giving a positive indication of the model predictions.

If examined closely, the scatter plot is less scattered than that of INTC while the line plot seems to be more closely related to the true values. This is in contradiction to the lower error values. A possible explanation for this is: since the plots are visualizing 100 days prior to May 30th, 2020, it covers the coronavirus market crash. The contradicting graphs suggest that Home Depot was slightly less affected than Intel Corporation due to it, hence the predictions are more accurate in these days.

In conclusion, we see that weights learned by the model on SPY data from 1995 to 2015 works well when directly applied to stocks of different companies (in different industries) on a completely different timeframe. Having proved that, it is ready to be tested on a trading strategy.

5. Trading Strategy

Now that the model has learned appropriate weights, and it has been established that these weights can make reasonable predictions on different companies, it is finally time to use it on a trading strategy. In this section, I will first be defining a long-short day trading strategy built around the neural network's ability to predict open and close prices for any particular day. I will then move on to testing this strategy on historical price data of Intel Corp. and Home Depot Inc. and discuss the performance in detail. I will also present the result of the algorithm on 100 randomly selected companies from the S&P 500. This process of testing a trading strategy on past data is called **Backtesting.** All sections prior to this have been a buildup leading to the actual strategy defined in the coming sections.

5.1 Strategy Algorithm

As mentioned above, the strategy is of a *long-short day trading* type. Breaking this down, day trading indicates that the algorithm will indulge in the buying and selling of stocks on a daily basis i.e. it will not buy and hold as is done by long-term value investors. *Long short* indicates that the algorithm will indulge in or attempt to both buy low first and then sell high i.e. going long when the price appears to increase on a particular day as well as selling high and then buying low i.e. short selling on days when the price appears to decline. This makes it a very aggressive trading strategy. Moving on to the details, how does the algorithm decide when to go long or sell short? This is where the neural network comes into play. The idea is that once the market closes on a particular day (4pm EST for US), the neural network will predict the *Open* and *Close* prices for the next trading day based on the current day's *Open*, *Close, High, Low* prices and *Volume* of shares traded. Now these two values will indicate if the stock price will increase or decrease the next day. If the predicted *Open* price is higher than the predicted *Close* price, then according to the model, the stock price will be decreasing, giving a short signal. On the other hand, if the predicted *Close* price is higher than the predicted *Open* price, the model gives a long signal.

Needless to say, the strategy described is not for the faint-hearted in that it contains a considerable degree of risk. Now the risk of unlimited loss associated with short selling is already contained as there is no day in which the shares will not be bought back, even if the actual price is increased.

In order to further control the risk, only 100 shares will be sold short during those days regardless of the amount of capital in hand. On days when a long signal is placed, the number of shares bought will depend on the amount of capital at hand (initial capital + net gain). This will result in more and more money invested as time passes if the strategy seems to be successful and less money invested if the strategy seems to be generating losses. No shares will be bought on margin.
The following flow diagram summarizes the strategy:

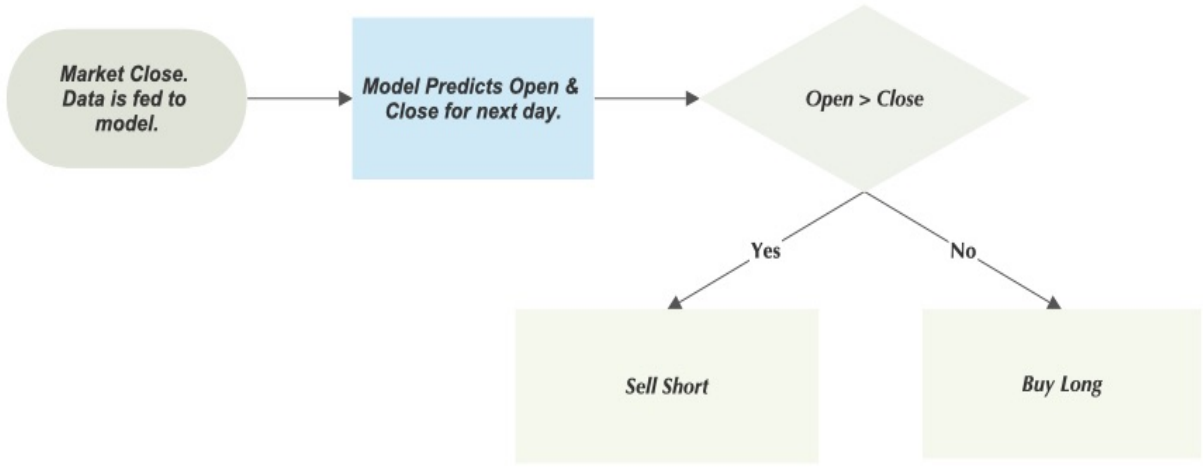


Figure 16: Trading Algorithm

5.2 Backtesting

Now that the strategy is defined, it is time to explore the kind of returns it will produce when applied to real companies. I will backtest it on the two companies mentioned in the previous section: **Intel Corporation** and **Home Depot,** followed by a detailed discussion of the returns produced versus the market and the dollar value of the daily profit/loss. I will conclude my discussion on this topic by testing the program on 100 randomly selected companies from the S&P 500 Index.
Note that the calculation of returns will ignore transaction costs, income taxes as well as dividend returns.

5.2.1 Evaluation Metrics

Before discussing these results, there are a few metrics that are worth mentioning in order to better understand the discussion. These metrics will be used to evaluate the model's performance.

1. **Total Return:** Total Return is the percentage gain or loss of the initial capital incurred during the testing timeframe.
2. **Annualized Return:** This is a common metric used to measure how well an investment performs on an annual basis. It is the geometric average of the amount of money earned. It is calculated as:

$$Annualized\ Return = (1 + Total\ Return)^{1/TimeFrame} - 1$$

3. **Standard Deviation:** The standard deviation is a universally accepted measure of the risk of an investment. It measures the extent to which the daily returns deviate from the mean. Higher the deviation, higher the risk.

4. **Sharpe Ratio:** Sharpe Ratio is a ratio that helps understand the return of a portfolio compared to its risk. It is a ratio of the return premium of the investment and its standard deviation. It is calculated as:

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p}$$

$Where,$

$R_p = Annualized\ Return$

$R_f = Risk\ Free\ Rate$

$\sigma_p = Standard\ Deviation$

5.2.2 Testing Parameters

The timeframe used will be 1st January, 2010 to 30th May, 2020, or roughly 10.5 years (10 years and 5 months). In order to calculate the real rate of return, the average annual inflation rate for this

period will be taken as 1.7% ("Current US Inflation Rates", 2020)[3]. The risk-free rate used in the calculation of the **Sharpe Ratio** is equal to the 10 Year US Treasury Yield of 0.94%. Note that this value is alarmingly low compared to historical standards due to the 2020 market crash. The benchmark for the performance will be the S&P 500 index; with an annualized return of 12.22% in the given timeframe. The average annualized return for the same over the past 90 years is 9.8% (Santoli, M., 2017)[4]. The initial capital will be set at $100,000.

5.2.3 Case I - Trading Performance on Inter Corporation [INTC]

After running the algorithm on INTC price data, we get the following statistics:

```
Beta: 0.91

------------------- EVALUATION METRICS -------------------

Total Return: 384.56%
Annualized Return: 16.22%
Std. Deviation: 1.3%
Sharpe Ratio: 11.77%
Total Profit: 384555.48
Final Value of Capital: 484555.48

------------------- TRADING STATISTICS -------------------

Total days long: 2544
Total days short: 2
Number of days with loss: 1196
```

Figure 17: Trading Statistics for INTC

Before beginning the analysis on returns, an observation is worth mentioning it interesting to note that out of 2546 trading days, 1196 i.e. about 46% of the days resulted in a loss. This high percent of loss further stresses upon the extreme risk associated with the algorithm. It also shines a light on the neural network's limitation; a loss means that the network predicted that price will drop on a particular day but in reality, it increased or vice versa.

[3] Current US Inflation Rates: 2009-2020. (2020, June 10). Retrieved June 17, 2020, from https://www.usinflationcalculator.com/inflation/current-inflation-rates/

[4] Santoli, M. (2017, June 18). INVESTING The S&P 500 has already met its average return for a full year, but don't expect it to stay here. CNBC. Retrieved 2020, from https://www.cnbc.com/2017/06/18/the-sp-500-has-already-met-its-average-return-for-a-full-year.html

Percentage return analysis

Moving forward to the main discussion. After 10.5 years, the initial $100,000 dollars invested will turn into $484,555.48 giving a total return of 384.56% or a 16.22% annualized return. This return is extremely impressive for such a long period of time. Compared to the benchmark return of 12.22%, the algorithm beats the market by a good 4%. After adjusting for inflation, the real rate of return is 14.52%. The well-known risk-return trade-off hold true here as the standard deviation (SD), representing the volatility of this return has a value of 1.3%, which is slightly higher than the market SD of 1.08%.

Figure 18 plots the annual returns produced by the algorithm in comparison with the market:

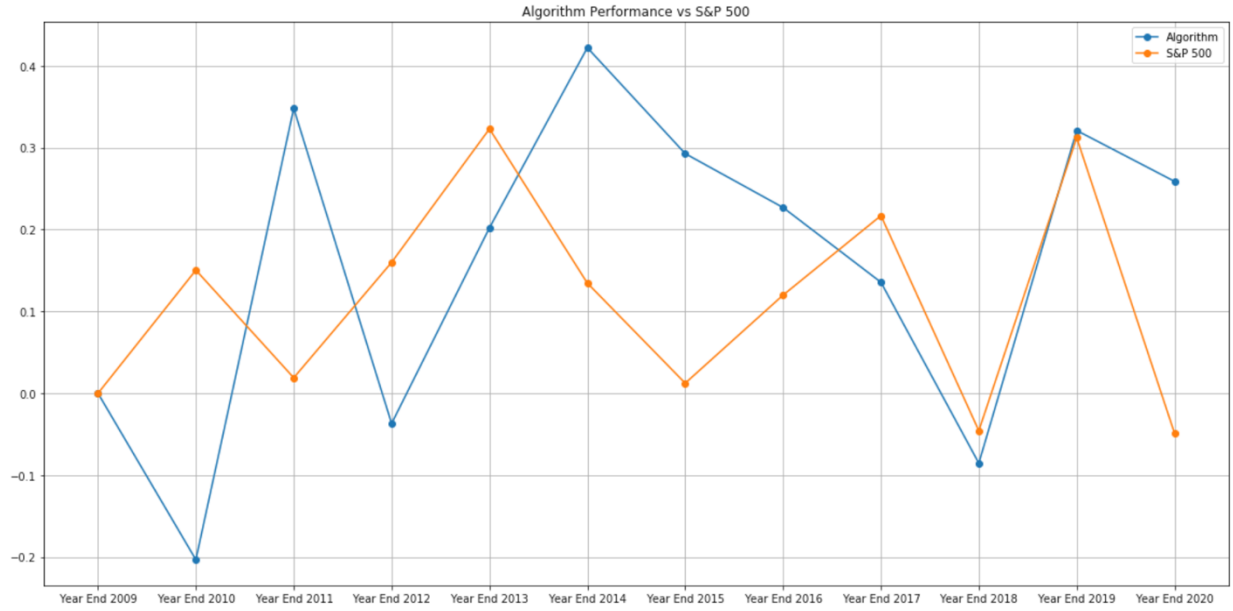


Figure 18: Algorithm vs S&P

This figure provides for an interesting analysis. In the first year, 2010, the algorithm suffered a loss while the S&P gained about 15%. However, the algorithm bounced back tremendously in the following year with a return of over 30% while the S&P fell. It was then a repeat of 2010 in 2012 as the algorithm produced a negative return contrary to what S&P did. The following five years were pleasing for the algorithm as it recorded healthy profits and beat the market in 3 of them. 2018 was grim with both recording slight losses before making healthy recoveries in 2019. So far in 2020, S&P took a major hit due to the coronavirus crash while the algorithm surprisingly has recorded a healthy return of over 20%.

In conclusion, the algorithm beat the market in 5 years from 2010 to 2019, while also beating it currently during 2020. However, when it did get beaten by the market, the difference in returns was smaller compared to its better years, resulting in an overall larger return. There does not seem to be a direct correlation between the two returns. There were 4 years when the market and the algorithm returns moved in opposite directions out of which 3 were consecutive years (2010,11,12). This goes against the nature of INTC, which has a beta of 0.91 indicating that the stock moves very closely with the market in general.

Dollar value analysis

In order to further investigate the performance of the algorithm, I will analyze the metric that most people understand better: dollar profit or loss. As per the design of the algorithm, two trades take place every single day resulting in a profit or a loss. Figure 19 shows two graphs which will be the base of this discussion:

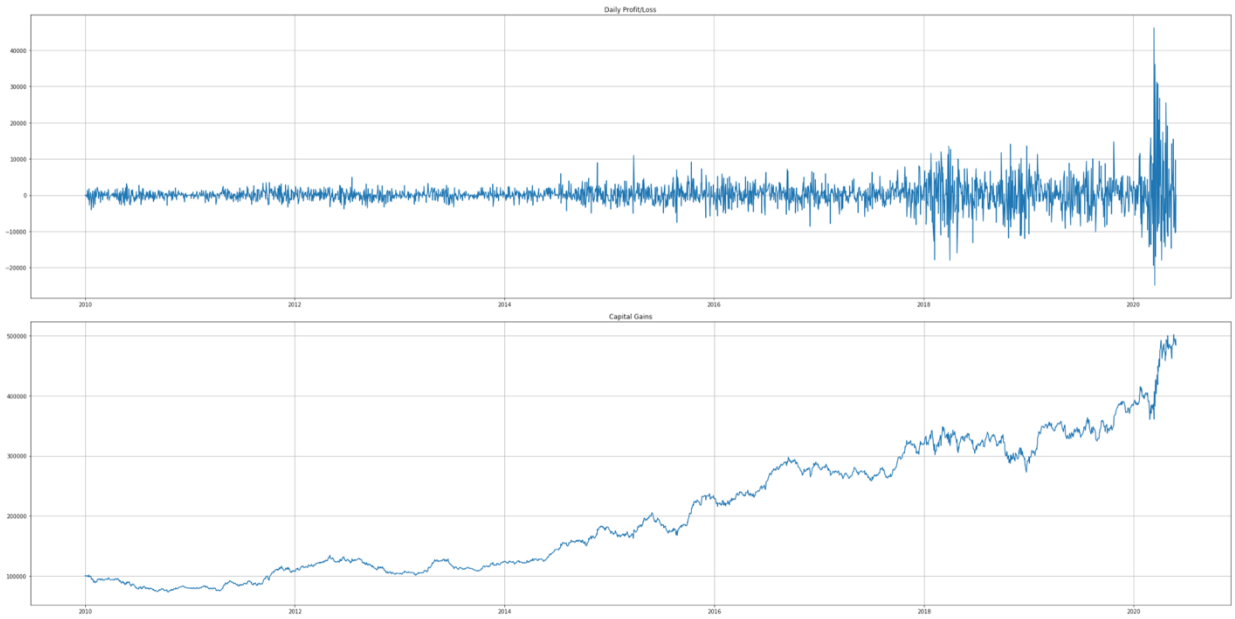


Figure 19: INTC Dollar Profit/Loss

The graph on the top plots daily profit or loss and the bottom graph shows how our initial $100,000 moved during the trading period.

As clearly shown, the capital value increases gradually to reach a value of about $484k by the end of the trading period. Meanwhile, the driver behind this growth: the daily gains swing drastically between positive and negative values. This swinging intensifies towards the end of the trading

period with higher frequencies being noted at the start of 2019. A possible explanation for this is: according to the algorithm, all accumulated profits are reinvested during the days when a long signal is generated, hence resulting in larger gains (or losses if the model produced a wrong signal). These frequencies reach their maximum values in 2020, interestingly coinciding with the coronavirus market crash; during which, the algorithm produced a profit of $46,164 on 13th March and a loss of $24,858 merely 3 days later on 16th March. These were the largest single day profit and loss respectively during the 10.5 years. This was expected since the algorithm was trading regularly with larger amounts of money at times when the financial community took a step back and moved away from the market.

5.2.4 Case II - Trading Performance on Home Depot [HD]

```
Beta: 0.97

------------------- EVALUATION METRICS -------------------

Total Return: 203.19%
Annualized Return: 11.14%
Std. Deviation: 0.83%
Sharpe Ratio: 12.27%
Total Profit: 203189.69
Final Value of Capital: 303189.69

------------------- TRADING STATISTICS -------------------

Total days long: 1208
Total days short: 1393
Number of days with loss: 1282
```

Figure 20: Algorithm Performance on HD

In this case, the model produced a short signal for more days than a long. This is in stark contrast with the previous case which only had 2 days of short selling. However it should be noted that out of 2611 trading days, 1282 days or 49% of the days resulted in a loss. This number raises possibilities of further development needs in the neural network.

Percentage return analysis

The algorithm when applied to NYSE: HD for 10.5 years turned the initial $100,000 to $303,189.69, a total profit of $203,189.69 or 203.19%. This gives a respectable annualized return of 11.14%. While this return is quite high for such a long period of time and would please many investors, it still fails to beat the market by 1.08%. Needless to say, the volatility of these returns (risk) is lower than the market's volatility indicated by the value of SD of 0.83%.

Figure 21 plots the annual returns produced by the algorithm as compared to S&P 500.

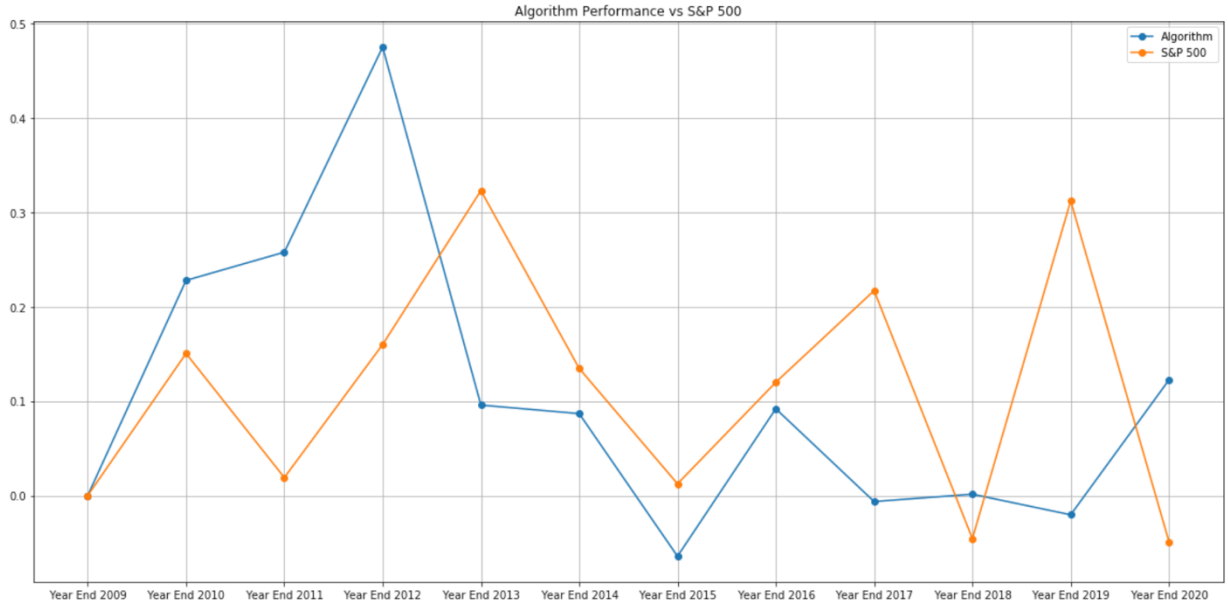


Figure 21: Algorithm vs S&P 500

The first three years (2010-12) were remarkable for the algorithm as it produced astonishingly large returns with its peak almost hitting the 50% mark in 2012. However, this explosive start did not last long as it was followed by 3 consecutive years of decreasing returns with the algorithm producing a loss in 2015, a year when the S&P also produced an underwhelming result. It made a decent recovery in 2016, returning almost 10%. However, this recovery did not last, as the returns came crashing back to produce a loss in 2 of the next 3 years (produced near 0% return in the third). So far in 2020 during the market crash, the algorithm has performed extremely well and returned more than 10%.

In conclusion, the pleasing annualized return of 11.14% is largely due to explosive first 3 years. After this, the algorithm did not return over 10% in any subsequent years (except so far in 2020). The algorithm beat the market 4 times in 10 years from 2010 to 2019, 3 of which were the inital first three years. Fourth was when the S&P incurred a loss and the algorithm produced a return of nearly 0%. Lastly, similar to the previous case, there does not seem to be a direct correlation between the two returns. They moved in the same direction in 5 out of 10 years. This observation again is not in line with the value of the stock's beta: 0.97.

Dollar value profit/loss analysis

Figure 22 plots the daily profit/loss and the value of the initial capital over the trading period:

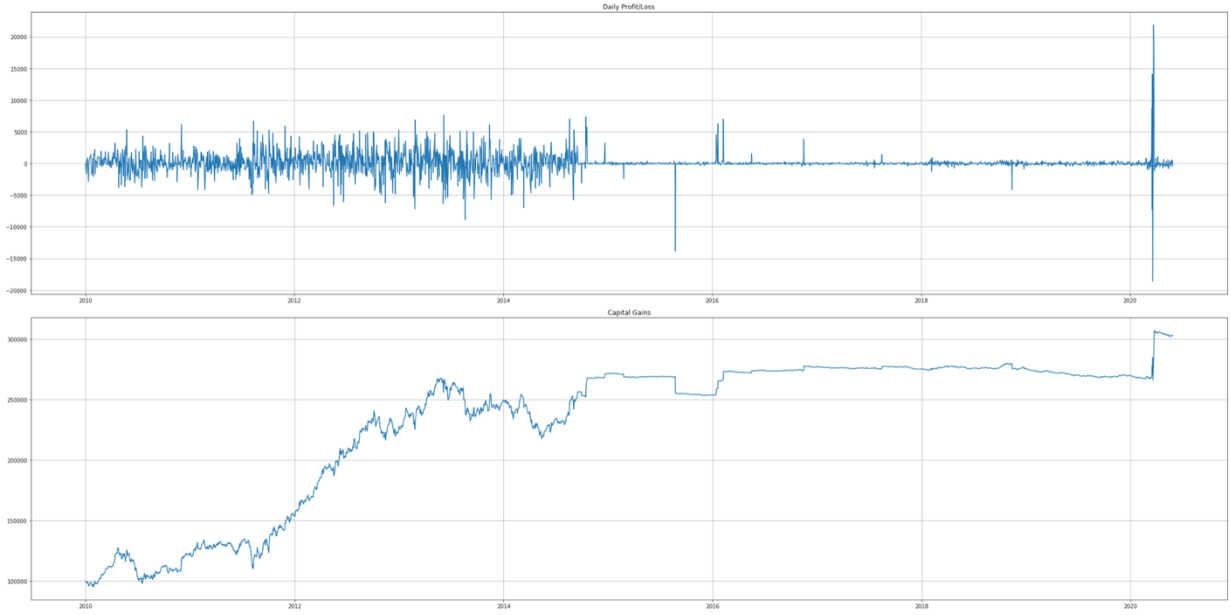


Figure 22: HD Dollar Profit/Loss

The dollar gains charts are rather awkward in this scenario. In the early years: 2010 until 2014, there was a considerable amount of swinging between loss and profit on daily basis, which also resulted in a steady increase of the initial capital. However, after 2014 the swinging subsided drastically until 2020 except some big profits and losses here and there. This is in contrast with the case of INTC, where the swinging gradually increased with time. A possible explanation for this can be the difference between the process of short and long selling as per the algorithm. As mentioned previously, a maximum of 100 shares are shorted on a particular day regardless of the capital at hand. This limits the amount of profit or loss incurred on that day. On the other hand, when a long signal is generated, all of the capital at hand is used to buy shares at the *Open*, hence resulting in large profit/loss. In the case of INTC, there were merely 2 days in 10 years when the shares were sold short therefore the large profits and losses were recorded each day. Meanwhile in this case, there are 1393 days with a short signal. It is reasonable to assume that almost all of these came after 2014, hence producing an almost flat profit/loss plot in Figure 22. This reflects the defensive approach to short selling the algorithm takes.

5.2.5 Testing Performance on Other Companies

As demonstrated, the algorithm beats the market in case I and gets beaten in case II. In order to have a more concrete picture of the strategy's performance, I have backtested it on 100 randomly selected stocks from the S&P 500 Index. Note that for companies that went public after 2010, the timeframe for the backtest is from their IPO date until 30th May 2020.

Please refer to Appendix III to get a list of these 100 stocks and their evaluation metrics.

The following chart summarizes the returns:

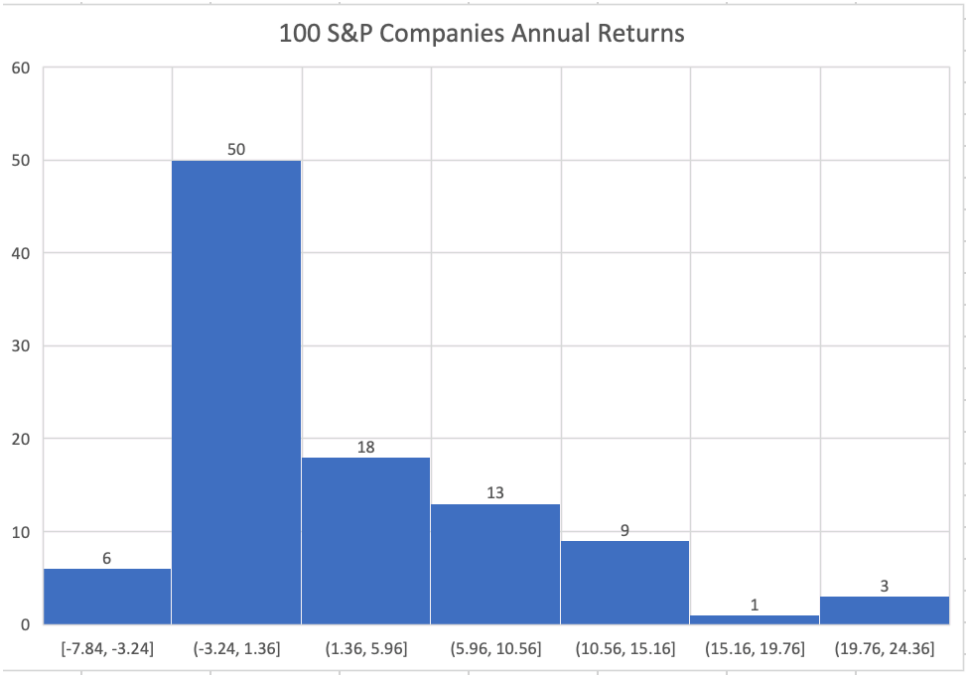


Figure 23: 100 Company Backtest Returns

Let's first look at the outstanding performers of this test. Out of 100, there were a total of 11 stocks that beat the market return of 12.22%. The top 3 performers were Fortune Brands Home & Security Inc. (NYSE: FBHS) with an enormous annualized return of 23.33%, Domino's Pizza Inc. (NYSE: DPZ) returning 21.16% and Mettler Toledo (NYSE: MTD) returning 20.88% annually. These were also the only companies returning over 20% annualized return. The other market beaters included: WW Grainger (16.31%), NextEra Energy (14.69%), AON (14.37%), S&P Global Inc. (14.01%), Illinois Tool Works (12.98%), Intuitive Surgical Inc. (12.9%), Tractor Supply Company (12.73%) and The Clorox Company (12.33%). Apart from these 11, there were 3 more companies: F5 Networks (9.88%), Duke Realty (10.79%) and Crown Castle International (12.17%) that returned better than the market's average annualized return over the past 90 years of 9.8%.

While these returns are amazing for such a long period of time and should please even the most discerning of investors, these constitute only 14% of the testing companies. A more complete picture shows that 43 of these 100 companies incurred a net loss over the 10.5 years. Amongst the other 57 companies, 14 returned less than the average inflation rate of 1.7% , thus resulting in a negative real rate of return – essentially, only 43 companies out of 100 returned a real profit.

There is a silver lining however: the losses were largely contained since the largest negative return has a value of -6.13% for SL Green Realty. Additionally, 27 out of the 43 companies that produced a loss produced a meagre loss of less than 1%.

6. Conclusion & Further Developments

In the real world, large investment firms use highly complex algorithms to make investment decisions. The strategy presented in this report is one of the most basic use of AI techniques for investment purposes. It can be used as a starting point for an algorithm to be used in the real world, using it directly would most likely result in a disappointing result.

Having said that, the report does give a complete picture of how AI can be applied to the field of finance. Sections 3,4 and 5 cover 3/4th of the steps in the algorithmic trading pipeline: Strategy Identification, Program Development and Backtesting. The 4th step: Implementation is not in the scope of this report. The results presented section 5.2 show some signs of promise indicated by enormous annual returns when backtested on companies such as Fortune Brands Home & Security, Domino's Pizza, Mettler Toledo and Intel Corporation. However, with more than 50% of the companies producing meagre returns, there is large room for improvement before it can be considered a successful algorithm.

6.1 Further Developments

Selecting Companies

There are 12 companies in this report which beat the market when the algorithm was applied to their stock. At this stage, there has not been a correlation established between returns and other

metrics of a particular stock. A deeper research can be conducted to find a relation between the returns and metrics such as the Beta, MSE, MAE, PE Ratio, Market Cap et cetera. This can then be used as a screen to filter out companies with the highest potential to produce whopping returns.

Neural Network

In the program development stage, the neural network designed is a very simple 3-layer network. This architecture works for the purpose of this report, however, moving forward the network can be made more robust. Inputs can be modified to include factors that affect a stock price such as the interest rates, risk free rate, public sentiment et cetera. Extra layers can be added to add more complexity and the nodes can be further modified.

Backtesting

As seen in section 5.2, all the backtesting was done programmatically on the Jupyter Notebook, with around 9 functions being written for the whole process. This legwork can be avoided by using an established algo-trading platform which provides the means to backtest a custom strategy. The initial plan was to use Quantopian for the development, however they do not support python's *Keras* library which was used to develop the neural network so that idea was not adopted. Moving forward, other platforms can be explored and used which support all the libraries required (figure 5).

APPENDIX

I. Neural Networks

Artificial Neural Networks (ANN)

https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

https://towardsdatascience.com/introduction-to-artificial-neural-networks-ann-1aea15775ef9

Recurrent Neural Networks (RNN)

https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

Long-Short-Term Memory (LSTM)

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

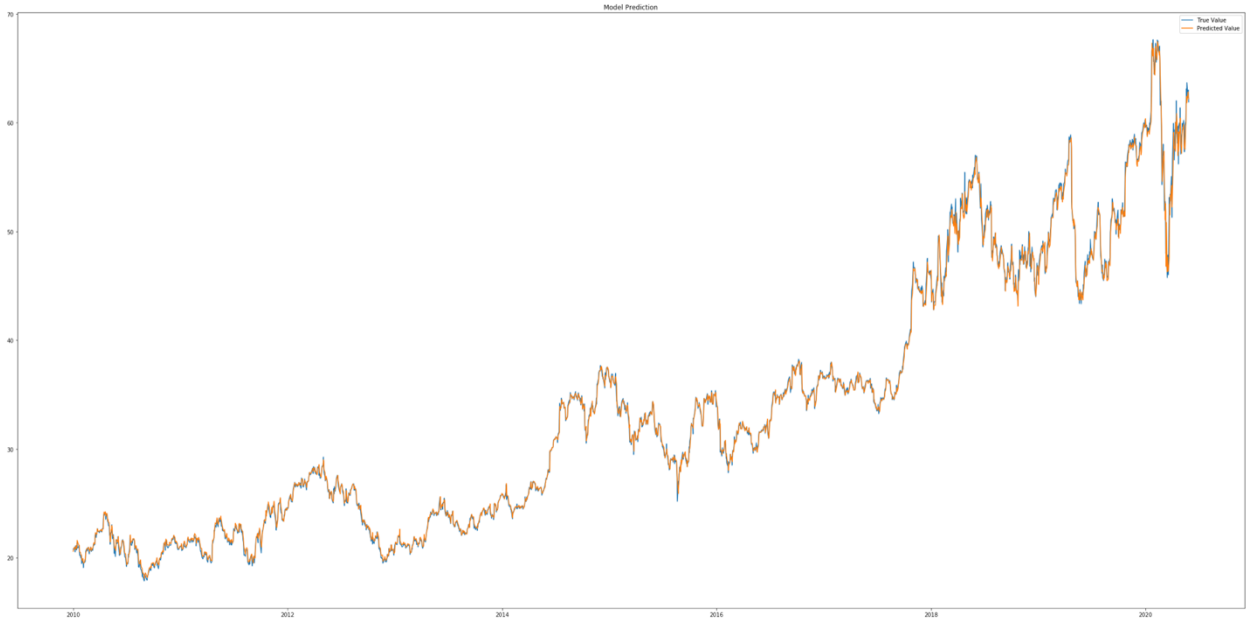
https://brohrer.github.io/how_rnns_lstm_work.html

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

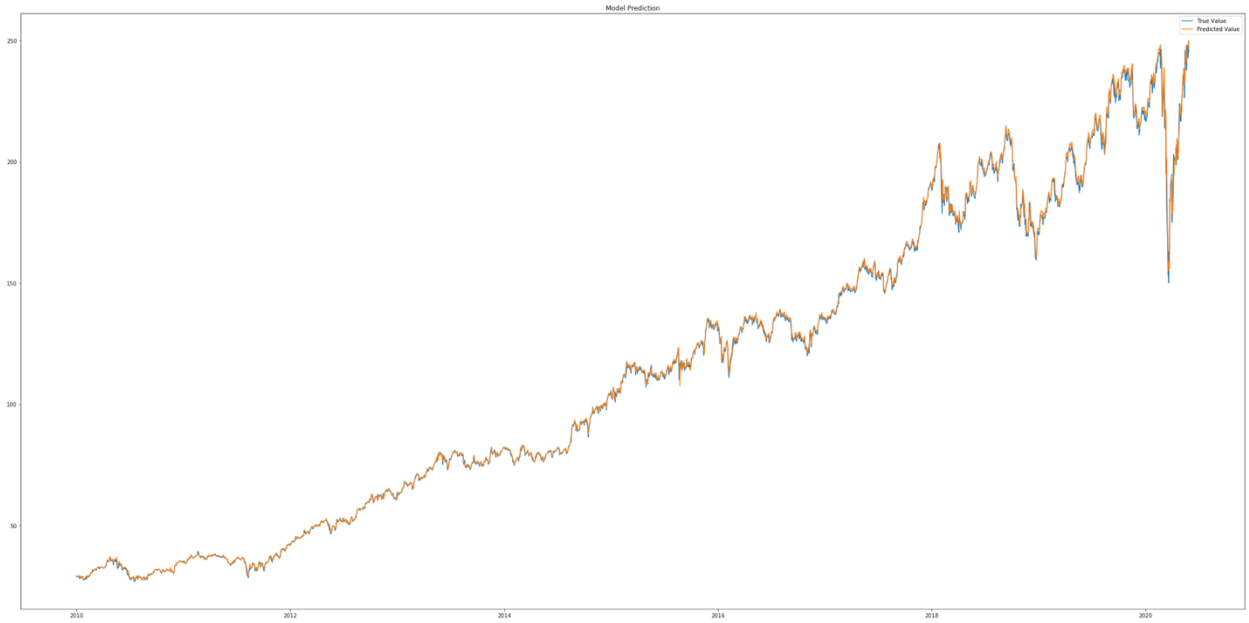
https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

II. Model Performance

Model Prediction on INTC



Model Prediction on HD



III. Backtesting Result

Download CSV file containing the Backtesting result. The data is in the format: Ticker, Beta, Risk, Sharpe Ratio, Return, Profit.

Download : Backtest_100_Stocks.csv

IV. Source Code

Github Link: https://github.com/SameerSinghDudi/LSTM-Stock_Market_Prediction-Quant2020

For those who don't use Github, download zip file: LSTM Algo Trading.zip

I strongly suggest using the Jupyter Notebook (.ipynb) for your use as the strategy was developed using it. For those who are not familiar with it, there are 8 python files (.py) containing the same functions.

Reference List

Artificial Intelligence Market by Offering (Hardware, Software, Services), Technology (Machine Learning, Natural Language Processing, Context-Aware Computing, Computer Vision), End-User Industry, and Geography – Global Forecast to 2025 (Rep.). (n.d.). doi:https://www.marketsandmarkets.com/Market-Reports/artificial-intelligence-market-74851580.html

Columbus, L. (2017, September 10). How Artificial Intelligence Is Revolutionizing Business In 2017. Forbes. Retrieved 2020, from https://www.forbes.com/sites/louiscolumbus/2017/09/10/how-artificial-intelligence-isrevolutionizing-business-in-2017/#5e981ba55463

Current US Inflation Rates: 2009-2020. (2020, June 10). Retrieved June 17, 2020, from https://www.usinflationcalculator.com/inflation/current-inflation-rates/

Santoli, M. (2017, June 18). INVESTING The S&P 500 has already met its average return for a full year, but don't expect it to stay here. CNBC. Retrieved 2020, from https://www.cnbc.com/2017/06/18/the-sp-500-has-already-met-its-average-return-for-a-full-year.html

DISCLAIMER

This report is produced by university student members of CityU Student Research & Investment Club (the Club). All material presented in this report, unless otherwise specified, is under copyright of the Club. None of the material, nor its content, nor any copy of it, may be altered in any way without the prior express written permission and approval of the Club. All trademarks, service marks, and logos used in this report are trademarks or service marks of the Club. The information, tools and materials presented in this report are for information purposes only and should not be used or considered as an offer or a solicitation of an offer to sell or buy or subscribe to securities or other financial instruments. The Club has not taken any measures to ensure that the opinions in the report are suitable for any particular investor. This report does not constitute any form of legal, investment, taxation, or accounting advice, nor does this report constitute a personal recommendation to you. Information and opinions presented in this report have been obtained from or derived from sources which the Club believes to be reliable and appropriate but the Club makes no representation as to their accuracy or completeness. The Club accepts no liability for loss arising from the use of the material presented in this report. Due attention should be given to the fact that this report is written by university students. This report is not to be relied upon in substitution for the exercise of independent judgement. The Club may have issued in the past, and may issue in the future, other communications and reports which are inconsistent with, and reach different conclusions from, the information presented in this report. Such communications and reports represent the different assumptions, views, and analytical methods of the analysts who prepared them. The Club is not under an obligation to ensure that such communications and reports are brought to the attention to any recipient of this report. This report, and all other publications by the Club do not constitute the opinion of the City University of Hong Kong, nor any governing or student body or department under the University aside from the Club itself. This report may provide the addresses of, or contain hyperlinks to, websites. Except to the extent to which the report refers to website material of the Club, the Club has not reviewed any such website and takes no responsibility for the content contained therein. Such addresses or hyperlinks (including addresses or hyperlinks to the Club's own website material) is provided solely for your own convenience and information and the content of any such website does not in any way form part of this Report. Accessing such website or following such link through this report shall be at your own risk.