NATIONAL INSTITUTE OF TECHNOLOGY, HAMIRPUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Stock Prices Prediction and Decision Making

MAJOR PROJECT- I (CSD-419)

Under the guidance of :
Dr. Basant Subba

Submitted By:
Simpy Kumari (16MI544)
Renuka Jangid (16MI533)
Rajnish Aryan (16MI542)
Aman Gupta (16MI556)

ABSTRACT

Financial markets are highly volatile and generate huge amounts of data daily. Investment is a commitment of money or other resources to obtain benefits in the future. A correct prediction of stocks can lead to huge profits for the seller and the broker. Frequently, it is brought out that prediction is chaotic rather than random, which means it can be predicted by carefully analyzing the history of respective stock market. Machine learning is an efficient way to represent such processes. It predicts a market value close to the tangible value, thereby increasing the accuracy. Introduction of machine learning to the area of stock prediction has appealed to many researches because of its efficient and accurate measurements.

Apart from correct prediction of the future stock prices, the decision that has to be taken is another major component of the total profit generated at the end. Since a trader acts like a reinforcement agent that is, buying and selling (that is, action) particular stock changes the state of the trader by generating profit or loss, that is, reward, we can apply Reinforcement Learning for the action that a trader should be taking at a given time step.

Our objective is to develop an AI to predict the stock prices and accordingly decide on buying, selling or holding stock. The AI algorithm should be flexible to consider various trading environmental factors like stock price changes, latency, gaining protection from future price movements.

CONTENTS

1. INTRODUCTION

Financial markets are highly volatile and generate huge amounts of data daily. Investment is a commitment of money or other resources to obtain benefits in the future. Stock is one type of securities. It is the most popular financial market instrument and its value changes quickly. It can be defined as a sign of capital participation by a person or an enterprise in a company or a limited liability company. The stock market provides opportunities for brokers and companies to make investments on neutral ground. Stock prices are predicted to determine the future value of companies' stock or other financial instruments that are marketed on financial exchanges. However, the stock market is characterized by nonlinearities, discontinuities, and high-frequency multi-polynomial components because it interacts with many factors such as political events, general economic conditions, and traders' expectations. Therefore, making precise predictions of stock values are challenging.

A correct prediction of stocks can lead to huge profits for the seller and the broker. Frequently, it is brought out that prediction is chaotic rather than random, which means it can be predicted by carefully analyzing the history of respective stock market. Machine learning is an efficient way to represent such processes. It predicts a market value close to the tangible value, thereby increasing the accuracy. Introduction of machine learning to the area of stock prediction has appealed to many researches because of its efficient and accurate measurements.

Thus, the prediction can be handled by 'Supervised learning approach using LSTM'. The LSTM or Long Short-Term Memory neural networks allow us to learn the context required to make predictions in time series forecasting problems, rather than having this context pre-specified and fixed. LSTM thus can look at the history of a sequence of trading data and predict what the future elements of the sequence are going to be.

Apart from correct prediction of the future stock prices, the decision that has to be taken is another major component of the total profit generated at the end. An emerging area for applying Reinforcement Learning is the stock market trading, where a trader acts like a reinforcement agent since buying and selling (that is, action) particular stock changes the state of the trader by generating profit or loss, that is, reward. To develop a TradeBot, we have to model stock prices correctly, so as a stock buyer one can reasonably decide when to buy stocks and when to sell them to make a profit, which involves developing a Time-Series model.

Reinforcement learning is a branch of ML which involves taking suitable action to maximize reward in a particular situation. RL differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. A forecast predicts future events whereas an RL agent optimizes future outcomes.

2. RELATED WORK

1. Neha Bhardwaj et al.[1], 2019, this paper presented comparison of machine learning aided algorithms to evaluate the stock prices in the future to analyze market behaviour.
    - Random Forest (Ensemble Learning)- required more no. of trees to predict accurately that makes model slow.
    - Logistic Regression (non-Ensemble Learning)- the limitation of selecting right Features to Fit.
2. H. Gunduz, Z. Cataltepe and Y. Yaslan [2] predicted stock prices using deep neural network techniques.
3. Liu, G. Liao and Y. Ding [3] 2018 conducted similar work and designed a model for applying LSTM to stock prediction with lots of scope for improvements to prediction accuracy.

Table I. Literature Review in a tabular form

Sr. No.	**Research Paper**	**Advantages**	**Disadvantages**
1.	Neha Bhardwaj et al.[1], 2019, this paper presented comparison of machine learning aided algorithms to evaluate the stock prices in the future to analyze market behaviour.	Random Forest - Pretty Flexible and easy to train. Logistic Regression - Provides very accurate results.	Random Forest - Require more no of trees to predict accurately that makes model slow. Logistic Regression - Selecting right Features to Fit.
2.	H. Gunduz, Z. Cataltepe and Y. Yaslan [2] predicted stock prices using deep neural network techniques.	It is accurate and fast as long as the market assumptions stay stationary	Market assumptions change quickly over time so models can quickly go from good to useless.
3.	Liu, G. Liao and Y. Ding [3] 2018 conducted similar work and designed a model for applying LSTM to stock prediction with lots of scope for improvements to prediction accuracy.	The experimental results show model can play a better forecasting effect, even though the accuracy is not very high, only about 72% for the short period of data.	If the stack layer is higher, the computational resources consumed by the model will increase.

3. PROPOSED METHODOLOGY

3.1 Prediction :

LSTM Neural Network :

LSTM is the advanced version of Recurrent-Neural Networks (RNN) where the information belonging to previous state persists. These are different from RNNs as they involve long term dependencies and RNNs works on finding the relationship between the recent and the current information. This indicates that the interval of information is relatively smaller than that to LSTM. The main purpose behind using this model in stock market prediction is that the predictions depends on large amounts of data and are generally dependent on the long term history of the market [4]. So LSTM regulates error by giving an aid to the RNNs through retaining information for older stages making the prediction more accurate [3]. Thus proving itself as much more reliable compared to other methods.
Since stock market involves processing of huge data, the gradients with respect to the weight matrix may become very small and may degrade the learning rate.[5].This corresponds to the problem of Vanishing Gradient. LSTM prevents this from happening. The LSTM consists of a remembering cell, input gate, output gate and a forget gate. The cell remembers the value for long term propagation and the gates regulate them.

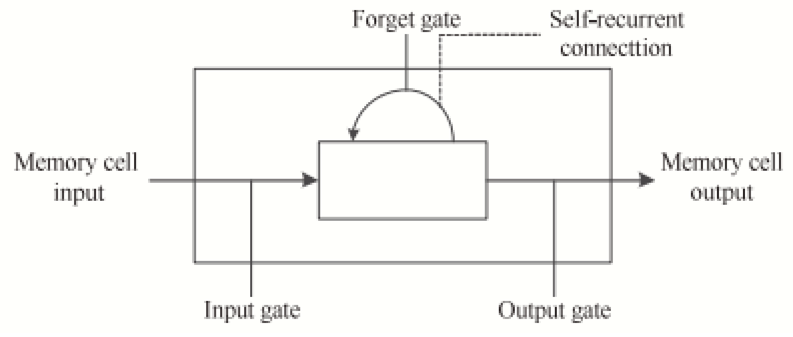


Fig. 1 Illustration of an LSTM memory cell

- Cell state - This represents the internal memory of the cell which stores both short term memory and long-term memories
- Input gate - Decides how much information from current input flows to the cell state
- Forget gate - Decides how much information from the current input and the previous cell state flows into the current cell state
- Output gate - Decides how much information from the current cell state flows into the hidden state, so that if needed LSTM can only pick the long-term memories or short-term memories and long-term memories

3.2 Decision Making :

Reinforcement Learning (RL):

It is a framework where an agent is trained to behave properly in an environment by performing actions and adapting to the results. It is different from other Machine Learning systems, such as Deep Learning, in the way learning happens:
It is an interactive process, as the agent actions actively changes its environment.

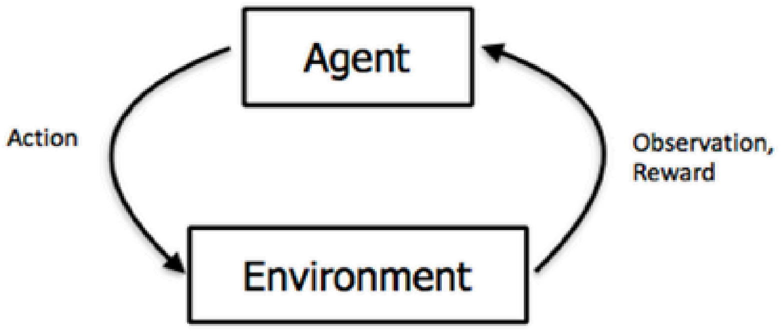


Fig 2. Reinforcement Learning diagram

During this iterative process, the agent performs actions over the environment and observes the immediate result; this feedback is used to improve the following action taken and the process starts again.

Q learning RL algorithm:
We have used Deep Q learning RL algorithm to train the TradeBot. The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. In general, Q learning involves the following flow:

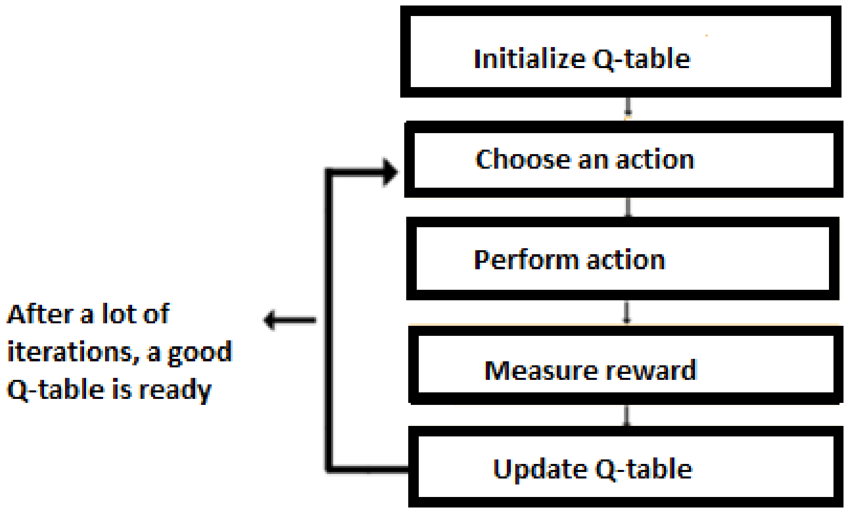


Fig 3. Q-Learning General Flow

Bellman Equation:

$$Q_{s,a} \leftarrow r + \gamma \max{}_{a' \in A} Q_{s'a'} \tag{1}$$

where, s - state

a - action

$Q_{s,a}$ - Q value for action a taken in present state s.

r - Reward for going into next state

γ – Discount factor

A – Action set

This is called the Bellman optimality equation. It states that, for any state-action pair (s,a) at time t, the expected return from starting in state s, selecting action a and following the optimal policy thereafter (AKA the Q-value of this pair) is going to be the expected reward we get from taking action a in state s, which is r, plus the maximum expected discounted return that can be achieved from any possible next state-action pair (s′,a').

To Update the Q table-

1. Start with an empty table, mapping states to values of actions.

2. By interacting with the environment, obtain the tuple s, a, r, s' (state, action, reward, new state). In this step, we need to decide which action to take, and there is no single proper way to make this decision. We have taken one of the approaches. There has to be a balance between exploration vs exploitation for any RL Bot.

3. Update the Q(s, a) value using the Bellman approximation described above.

4. Repeat from step 2 for all the states

More specifically, to tell the agent what action to take under which circumstances we learn a policy, which is represented by a so-called Quality Table with all possible states and actions. Each time the agent selects an action a_t, observes a reward r_t, and enters a new state s_{t+1} , the Q-Table is updated in the following way:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha).Q(s_t, a_t) + a.(r_t + \gamma.\max_a Q(s_{t+1}, a)) \tag{2}$$

where $Q(s_t, a_t)$ is old value, α is learning rate, r_t is reward, γ is discount factor, $\max_a Q(s_{t+1}, a)$ is estimate of optimal future value and $(r_t + \gamma.\max_a Q(s_{t+1}, a))$ is the learned value.

The discount factor γ determines the importance of future rewards. If it is 0 our agent will only learn to consider current rewards, while a γ of 1 will make it strive for a long-term high reward.

4. IMPLEMENTATION

4.1 Prediction:

A. Data Source

The experiment data comes from the stock data interface of Yahoo! Finance platform. In this platform , we can get stock historical transaction data and obtain the k line data by setting related parameters.

Features of dataset:

Opening price- It is the price at which a security first trades upon the opening of an exchange on a trading day.

High- Today's high is the highest price at which a stock traded during the course of the trading day.

Low- Today's low is the lowest price at which a stock trades over the course of a trading day.

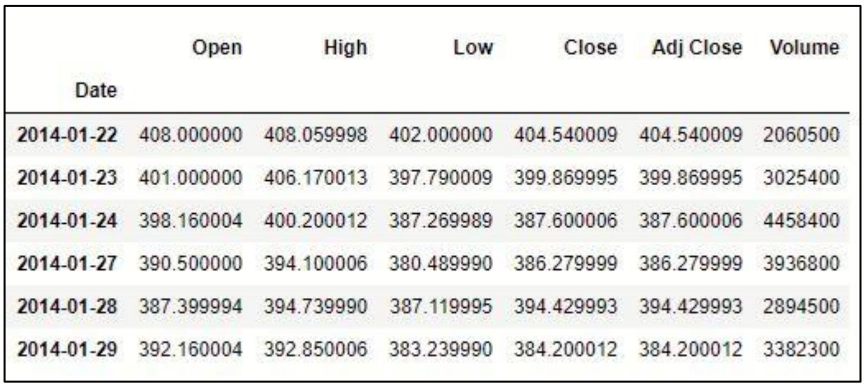
Closing price- It generally refers to the last price at which a stock trades during a regular trading session.

Adjusted closing price- It amends a stock's closing price to accurately reflect that stock's value after accounting for any corporate actions. It is considered to be the true price of that stock and is often used when examining historical returns or performing a detailed analysis of historical returns.

Volume- Volume is the number of shares or contracts traded in a security or an entire market during a given period of time.

In this experiment, we select data from Yahoo Finance over 10 years from 2001-01-02 to 2010-12-31.
The dataset shape is (2515, 6).
For example, some Index data is as shown in "TABLE II" .

Table II. STOCK DATA SAMPLE



	Open	High	Low	Close	Adj Close	Volume
Date						
2014-01-22	408.000000	408.059998	402.000000	404.540009	404.540009	2060500
2014-01-23	401.000000	406.170013	397.790009	399.869995	399.869995	3025400
2014-01-24	398.160004	400.200012	387.269989	387.600006	387.600006	4458400
2014-01-27	390.500000	394.100006	380.489990	386.279999	386.279999	3936800
2014-01-28	387.399994	394.739990	387.119995	394.429993	394.429993	2894500
2014-01-29	392.160004	392.850006	383.239990	384.200012	384.200012	3382300

B. Data Pre-Processing

Firstly, calculate the stock's MA , EMA index by the closing price data.
MA : Moving Average. C_i is the closing data on a certain day.

$$MA = \sum_{i=1}^{n} X_i / n \tag{3}$$

EMA : Exponential Moving Average. X is a variable, N is a certain day, Y ' is EMA of last cycle.

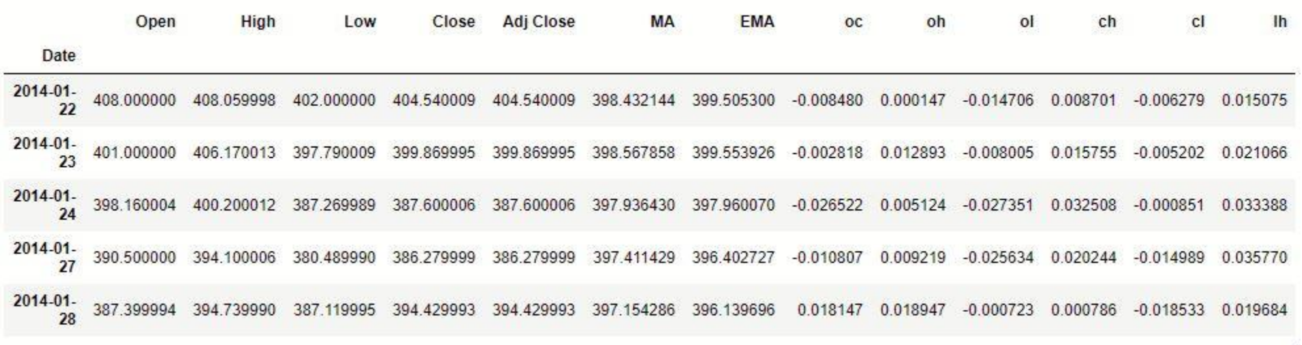
$$EMA(X, N) = Y = (2 * X + (N-1) * Y')/(N+1) \tag{4}$$

Secondly, preprocess the correlation index of stock data according to the following methods.

$$\begin{aligned} oc &= (close - open)/open \\ ol &= (low - open)/open \\ oh &= (high - open)/open \\ ch &= (high - close)/close \\ cl &= (low - close)/close \\ lh &= (high - low)/low \end{aligned} \tag{5}$$

After adding all these new features, the new dataset is :

Table III. STOCK DATA SAMPLE AFTER ADDING NEW FEATURES



Date	Open	High	Low	Close	Adj Close	MA	EMA	oc	oh	ol	ch	cl	lh
2014-01-22	408.000000	408.059998	402.000000	404.540009	404.540009	398.432144	399.505300	-0.008480	0.000147	-0.014706	0.008701	-0.006279	0.015075
2014-01-23	401.000000	406.170013	397.790009	399.869995	399.869995	398.567858	399.553926	-0.002818	0.012893	-0.008005	0.015755	-0.005202	0.021066
2014-01-24	398.160004	400.200012	387.269989	387.600006	387.600006	397.936430	397.960070	-0.026522	0.005124	-0.027351	0.032508	-0.000851	0.033388
2014-01-27	390.500000	394.100006	380.489990	386.279999	386.279999	397.411429	396.402727	-0.010807	0.009219	-0.025634	0.020244	-0.014989	0.035770
2014-01-28	387.399994	394.739990	387.119995	394.429993	394.429993	397.154286	396.139696	0.018147	0.018947	-0.000723	0.000786	-0.018533	0.019684

4.2 Decision Making:-

Producing and updating a Q-table can become ineffective in big state space environments. For instance, imagine an environment with 10,000 states and 1,000 actions per state. This would create a table of 10 million cells. It is pretty clear that we can't infer the Q-value of new states from already explored states. This presents two problems:

- First, the amount of memory required to save and update that table would increase as the number of states increases
- Second, the amount of time required to explore each state to create the required Q-table would be unrealistic

We are using a deep neural network to estimate the Q-values for each state-action pair in a given environment, and in turn, the network will approximate the optimal Q-function.
Suppose we have some arbitrary deep neural network that accepts states from a given environment as input. For each given state input, the network outputs estimated Q-values for each action that can be taken from that state. The objective of this network is to approximate the optimal Q-function, and remember that the optimal Q-function will satisfy the Bellman equation:

$$q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(s',a')]$$

With this in mind, the loss from the network is calculated by comparing the outputted Q-values to the target Q-values from the right hand side of the Bellman equation, and as with any network, the objective here is to minimize this loss.

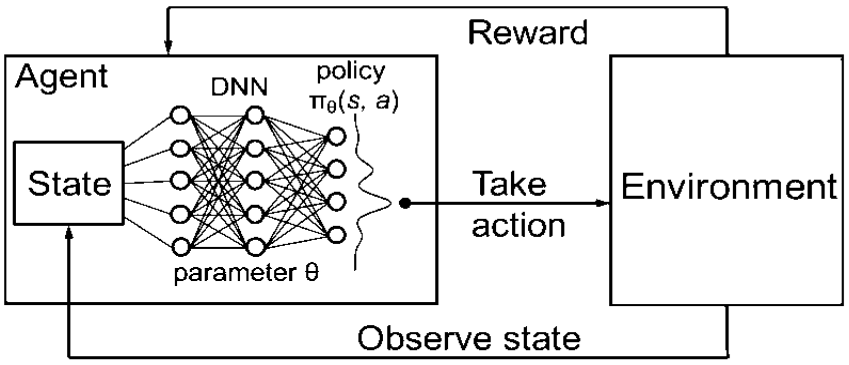


Fig 4. Deep Reinforcement Learning Diagram

Experience Replay and Replay Memory

With deep Q-networks, we often utilize this technique called *experience replay* during training. With experience replay, we store the agent's experiences at each time step in a data set called the *replay memory*. We represent the agent's experience at time t as e_t.

At time t, the agent's experience e_t is defined as this tuple:

$$e_t=(s_t,a_t,r_{t+1},s_{t+1})$$

This tuple contains the state of the environment s_t, the action at taken from state s_t, the reward r_{t+1} given to the agent at time t+1 as a result of the previous state-action pair (s_t,a_t) and the next state of the environment s_{t+1}. This tuple indeed gives us a summary of the agent’s *experience* at time t.

All of the agent's experiences at each time step over all episodes played by the agent are stored in the *replay memory*. In practice, we usually see the replay memory set to some finite size limit, N, and therefore, it will only store the last N experiences. This replay memory data set is what we randomly sample from to train the network. The act of gaining experience and sampling from the replay memory that stores these experiences is called experience replay. A key reason for using replay memory is to break the correlation between consecutive samples. If the network learned only from consecutive samples of experience as they occurred sequentially in the environment, the samples would be highly correlated and would therefore lead to inefficient learning. Taking random samples from replay memory breaks this correlation.

Data: We are using closing price as input-

```
[1283.27002,
 1347.560059,
 1333.339966,
 1298.349976,
 1295.859985,
 1300.800049,
 1313.27002,
 1326.819946,
 1318.550049,
 1326.650024]
```

Time step: window of 10 days is used as 1 time step

Setting up

Before training starts, we first initialize the replay memory data set D to capacity N. So, the replay memory D will hold N total experiences. Next, we initialize the network with random weights.

Gaining experience

Now, for each time step t within the episode, we either explore the environment and select a random action, or we exploit the environment and select the greedy action for the given state that gives the highest Q-value.

The policy network

After storing an experience in replay memory, we then sample a random batch of experiences from replay memory.

The input state data then forward propagates through the network, using the same forward propagation technique. The model then outputs an estimated Q-value for each possible action from the given input state.

The loss is then calculated. We do this by comparing the Q-value output from the network for the action in the experience tuple we sampled and the corresponding optimal Q-value, or *target Q-value*, for the same action.
The target Q-value is calculated using the expression from the right hand side of the Bellman equation. The loss is calculated by subtracting the Q-value for a given state-action pair from the optimal Q-value for the same state-action pair.

$$q_*(s,a) - q(s,a) = loss$$

$$E[R_{t+1} + \gamma \max_{a'} q_*(s',a')] - E[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}] = loss$$

Calculating the max term

When we are calculating the optimal Q-value for any given state-action pair, notice from the equation for calculating loss that we used above, we have this term here that we must compute:

$$\max_{a'} q_*(s',a')$$

Here, s′ and a′ are the state and action that occur in the following time step.

In order to find this max term , we pass s′ to the policy network, which will output the Q-values for each state-action pair using s′ as the state and each of the possible next actions as a′. Given this, we can obtain the max Q-value over all possible actions taken from s′, giving us $\max_{a'} q_*(s',a')$.

Once we find the value of this max term, we can then calculate this term for the original state input passed to the policy network.

$$E[R_{t+1} + \gamma \max_{a'} q_*(s',a')]$$

this term enables us to compute the loss between the Q-value given by the policy network for the state-action pair from our original experience tuple and the target optimal Q-value for this same state-action pair.

Training the policy network

So after we're able to calculate the optimal Q-value for our state-action pair, we can calculate the loss from our policy network between the optimal Q-value and the Q-value that was output from the network for this state-action pair.

Gradient descent is then performed to update the weights in the network in attempts to minimize the loss. In this case, minimizing the loss means that we're aiming to make the policy network output Q-values for each state-action pair that approximate the target Q-values given by the Bellman equation.

Up to this point, it was all for one single time step. We then move on to the next time step in the episode and do this process again and again time after time until we reach the end of the episode. At that point, we start a new episode, and do that over and over again until we reach the max number of episodes we've set. We'll keep repeating this process until we've sufficiently minimized the loss.

5. EXPERIMENT RESULTS

5.1 Prediction:-

In order to calculate the error, RMSE (root mean square error) technique is used which produced :

Training error: 38.82

Testing error: 147.38

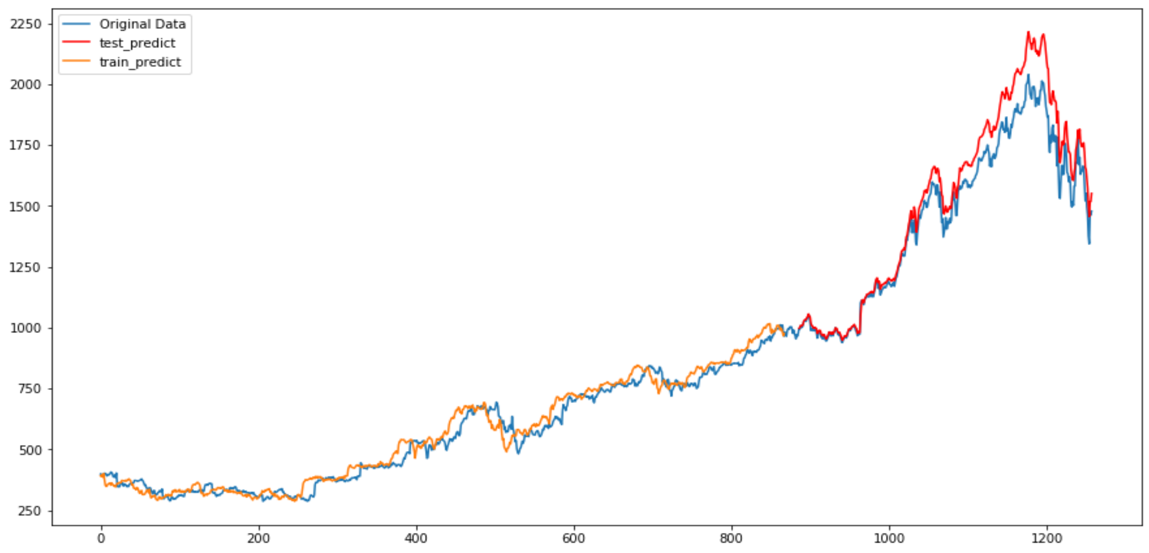


Fig 5. Prediction over 5 year data

5.2 Decision making:-

In accordance with the policy learned over the training period by the model, it takes such actions which contribute to optimal profit. In order to evaluate the model's performance we have limited the amount of money (asset) that the agent will be having initially. For instance in this project the initial money that the agent is provided with is 10000$.

When we test it over 9 years data without any intervention in parameters, it generates profit of 8%.

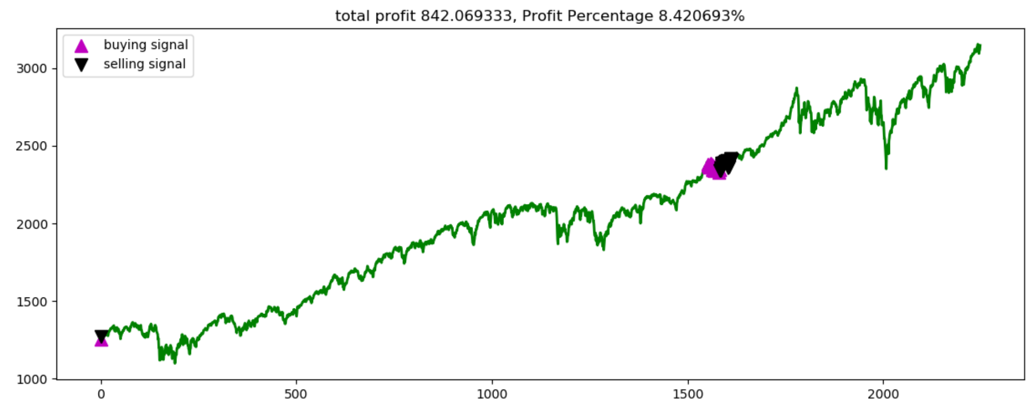


Fig 6. Result over 10 year data

If we change a few parameters such as rewarding the agent a huge loss (i.e. -500) for more than 20 consecutive sit_counts (i.e. if the agent doesn't take buy or sell as its action for an extended period of time) or imposing buy over more than 40 consecutive sell_counts (number of sell action) with no stock in its inventory then the graph came out to be more scattered and generated profit of approximately 32%.

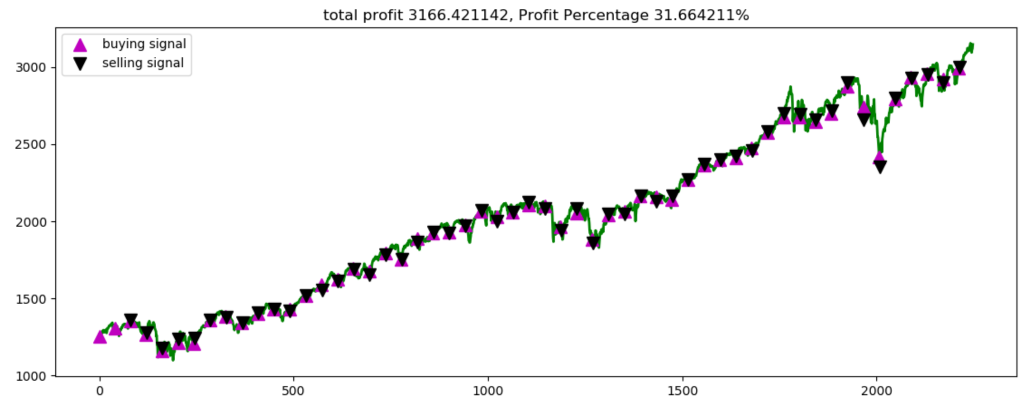


Fig 7. Result over 10 year data (changed parameters)

When we test it over 1 year data, we can distinctly see the actions taken and the profit generated. This has been implemented without giving huge negative reward.

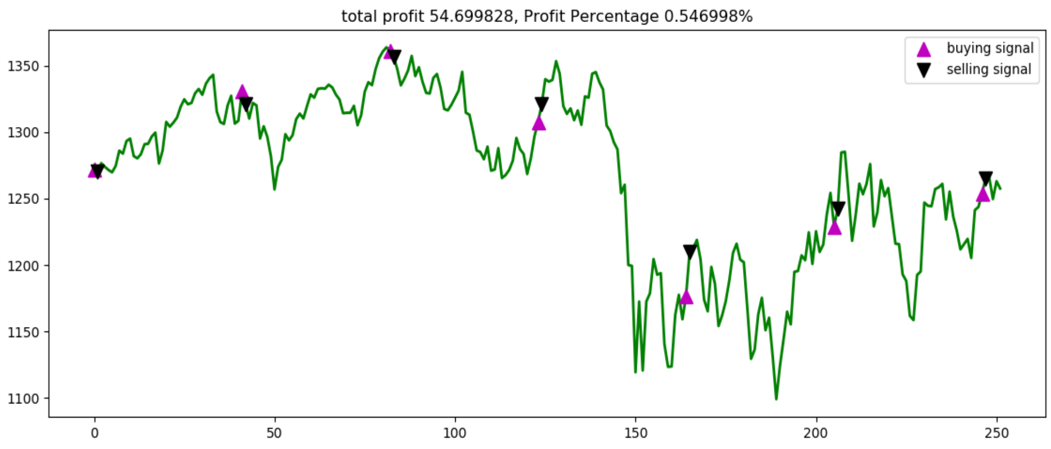


Fig 8. Result over 1 year data

If we change a few parameters such as rewarding the agent a huge loss (i.e. -500) for more than 20 consecutive sit_counts (i.e. if the agent doesn't take buy or sell as its action for an extended period of time) or more than 40 consecutive sell_counts (number of sell action) with no stock in its inventory then although the graph came out to be more scattered but it suffered from negative profit which is not the desirable output that any trader looks forward to.

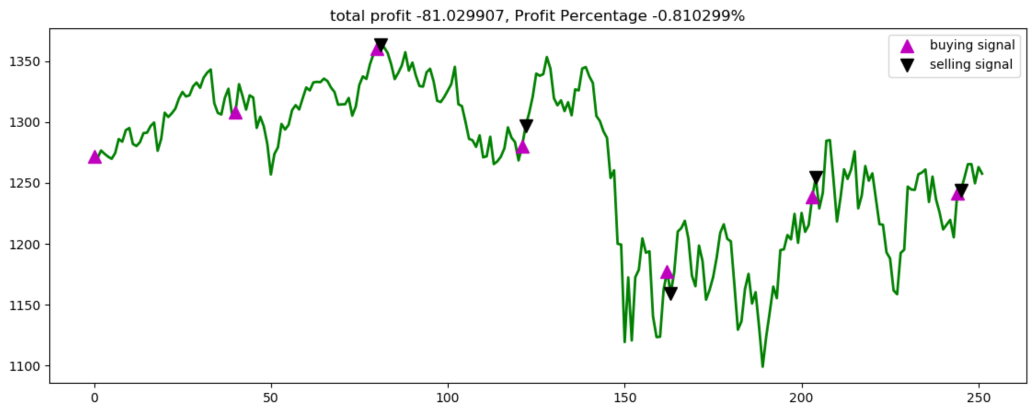


Fig 9. Result over 1 year data (changed parameters)

As a result of the changes made to the parameters, we conclude that:

1. Agent is producing optimal profit as per its policy when we set the variable sell_count to 40.
2. This decision making model on stock prices is more efficient for long term as compared to the short term data. This conclusion is based on the profit percentage it's generating.

REFERENCES

1. Neha Bhardwaj, MD Akil Ansari, “Prediction of Stock Market using Machine Learning Algorithms”,International Research Journal of Engineering and Technology (IRJET), Volume: 06, Issue: 05, p-ISSN: 2395-0072, May 2019.
2. H. Gunduz, Z. Cataltepe and Y. Yaslan, "Stock market direction prediction using deep neural networks," 2017 25th Signal Processing and Communications Applications Conference (SIU), Antalya, 2017, pp. 1-4.
3. S. Liu, G. Liao and Y. Ding, "Stock transaction prediction modelling and analysis based on LSTM," 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), Wuhan, 2018, pp. 2787-2790.
4. K. V. Sujatha and S. M. Sundaram, "Stock index prediction using regression and neural network models under non normal conditions," INTERACT-2010, Chennai, 2010, pp. 59-63.
5. T. Gao, Y. Chai and Y. Liu, "Applying long short term memory neural networks for predicting stock closing price," 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, 2017, pp. 575-578.