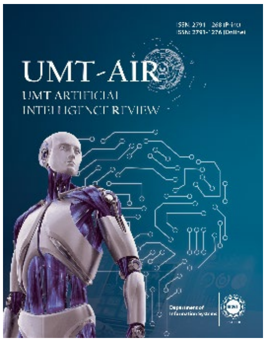


UMT Artificial Intelligence Review (UMT-AIR)
Volume 1 Issue 1, Spring 2021
ISSN(P): 2791-1276 ISSN(E): 2791-1268
Journal DOI: https://doi.org/10.32350/UMT-AIR
Issue DOI: https://doi.org/10.32350/UMT-AIR/0101
Homepage: https://journals.umt.edu.pk/index.php/UMT-AIR

Journal QR Code:

Article: **Self-Operating Stock Exchange – A Deep Reinforcement Learning Approach**

Author(s): Hammad Ghulam Mustafa

Affiliation: Bahria University, Lahore, Pakistan

Article QR:

Hammad Ghulam

Citation: G. M. Hammad, "Self-operating stock exchange – a deep reinforcement learning approach," *UMT Artificial Intelligence Review,* vol. 1, pp. 28–43, 2021. https://doi.org/10.32350/UMT-AIR/0101/02

A publication of the
Dr Hasan Murad School of Management
University of Management and Technology, Lahore, Pakistan

Self-Operating Stock Exchange: A Deep Reinforcement Learning Approach

Hammad Ghulam Mustafa[1]*

ABSTRACT: Stock trading approaches play an important role in equity. However, it is a difficult task to create a financially beneficial approach due to the complicated and ever-evolving nature of the stock market. In this study, we employed an Epsilon Greedy policy on our Deep Q-Learning (DQN) prototype that enable effective policy for the agent. This could optimize the predicted values of the total reward across any sequential steps. It also helps to maximize the state-action-value function by engaging with the environment q (s, a) to recommend when to buy, sell or hold. In this prototype, the state depends on routine principles of buying, selling or holding the existing data. The state alters as the buying and selling session alters. The prototype is able to grow rapidly with respect to the responses to market made by agents based on reward signals. It enhances the users' understanding about the holding and buying of stocks.

KEYWORDS: Deep Reinforcement Learning (DLR), Deep Q-Learning (DQL), Deep Q-Network (DQN), self-operating stock exchange, stock trading

I. Introduction

Stock trading has always been a controversial topic in the financial market due to the continuously changing nature of the stock trend. Now a days developing intelligent system to make strategies for stock trading is very challenging. With regards to the exponential increase in the application of artificial intelligence in the field of financial markets, particularly stock trading, reinforcement learning (RL) has been found to be most effective. Reinforcement learning is considered one of the most powerful AI technology used by financial traders ever since the program AlphaGo defeated the strongest human contemporary Go board game player Lee Sedol in 2016 [1]. Profitable trading strategies are of great importance not only for organizations but also for the individuals who wants to invest in it. The decisios related to

[1]Department of Computer Science, Bahria University , Lahore, Pakistan
*Corresponding Author: hammadgm2008@gmail.com

trading are easy for the successful traders who possess fundamental as well as technological expertise . However, it has been unappropriate for the normal investors due to lack of technological assistance, knowledge and trading experience [2]. In this study, we explored and optimized a reinforcement learning algorithm, that is, Deep Q-learning (DQN). DQN is already being effectively used in the stock trading system. Different studies produced optimal solutions. Therefore, we took 3 years of stock market data of three competitive companies, namely Apple Inc., UBER and Infosys Ltd, from the "Yahoo Finance" website. The data was collected through the Yahoo Finance library in order to get LiveData. The data consists of columns, such as "Open" which refers to the starting price, "High" which refers to the most expensive price, "low" which refers to the cheapest price, "close" which refers to the "ending price" of the day, "Volume" which refers to the number of shares traded, and "Adj Close" which refers to the close price adjusted for both dividends and splits. We have explored the DQN model through two different approaches; first, we updated the
discount factor, epochs and reward window and second, in the next strategy, we added two and three dense layers. We performed a series of experiments to find the best possible way to optimize the accumulative profit of the companies and then selected the most optimized solution. The findings of the analysis showed that DQN can make

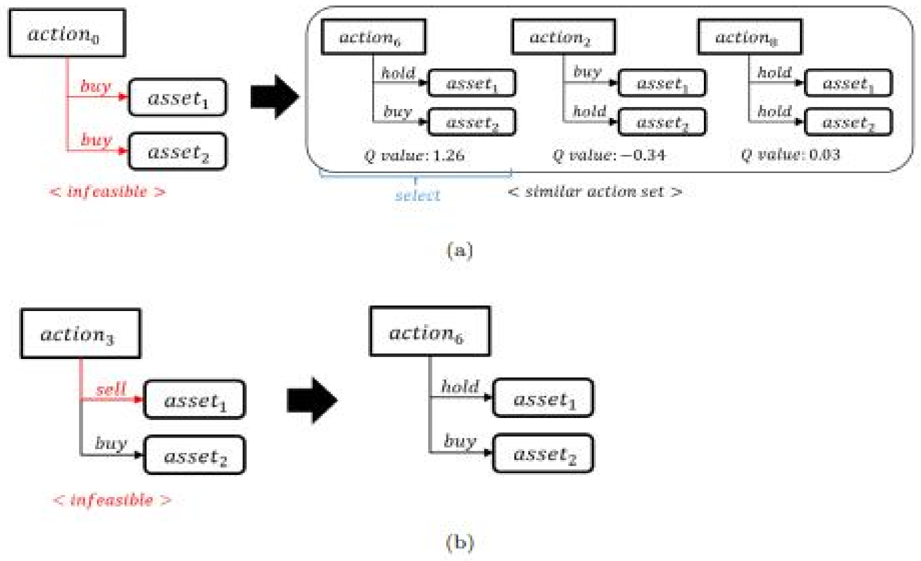


Fig. 1. Illustration of Map Functions

profitable patterns in the next strategy, we added two and three dense layers. We performed a series of experiments to find the best possible way to optimize the accumulative profits.

II. Related Work

The previous approaches were not contain exterior knowledge, rather, researcher only used past data of stock prices for the trading. Deep Q Learning was utilized in the study, which utilize the trading activities related to buying, holding, or selling In this wok, the dataset is based on US stock with the total of 504 samples for training and 218 samples for testing [3]. Intially, few of data preprocessing techniques are utilized that involves normalization and tensor packaging. Standardized training data was used as data tensor (categories) for the reinforcement learning (RL) module. Subsequently, the testing data was used to check the execution of the portfolio management system (PMS) and to produce stock weights for the testing phase. Whereas the other part of the study, is related to RL module which includes long and short RL models. The RL module becomes familiar with the portfolio weighting plan through communication between the situation and the agent to decide long and short stock management independently. The last module is equity market neutral (EMN) which consolidates long and short stock weights to forestall long and short

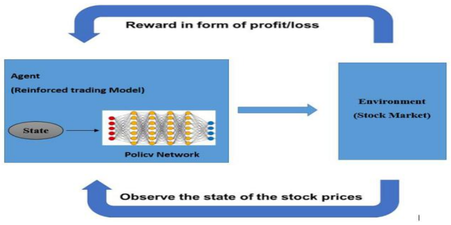

Reward in form of profit/loss
Agent
(Reinforced trading Model)
State
Policy Network
Environment
(Stock Market)
Observe the state of the stock prices

Fig. 2. Life Cycle of our Trading Model

equivalent stock from spending exchanging costs. Simultaneously, the EMN module builds up the last EMN stock weight [4]. This study utilised the Deep Q learning (DQL) approach for portfolio trading strategy. Due to action space having a few issues DQL agent may not determine a wise trading technique. The mapping function involves two planning rules, every one of which is needed for planning impracticable actions and is separated into two cases. Mainly, the measure of money is not enough to make a move which only includes purchasing resources. In this case, a comparative action set is inferred by holding instead of buying a

Visualization of our Trading Model

division of the resource gathering to be purchased in the first action. From that point, impracticable actions are planned to the most significant possible actions in the comparable action set. Let us suppose, due to the shortage of money, the system is unable to execute the act of buying both assets (1 and 2). In this case, mapping is done on a most feasible actions within the same action states. This involves the action of holding and buying of both the asset 1 and 2.Conversely, due to shortage of assets, if the system cannot execute the selling of the assets, then it would simply map to an original action when assets are not enough In such a case, it holds. This map function is shown in Figure 1[5].

Further, utilizing Neural Network to inexact state is not a smart approach since the preparation cycle is unappropriate. This issue is resolved by utilizing Deep Q Network (DQN) which executes experience replay to overcome the shakiness. Experience replay is an approach used to diminish the connection between A. preparation information by randomly clump examining N information focuses or preparing encounters. These calculation stores the most recent N information focuses on the memory

support. Further its randomly gets Batch size number which focuses each progression/emphasis to fit the model. It is computationally not plausible to assemble an ideal Q-table when the state space and activity space are huge. The computational issue of huge state-activity space is mitigated by utilizing a capacity approximator for state-activity compromise choice [6].

III. METHODOLOGY

We modelled our stock market trading process via Deep Q-learning (DQN) by using two different approaches. First, we updated the discount factor, epochs and reward window. Subsequently, in the next strategy, we added two and three dense layers. We applied this method because it is infinitely discrete when dealing with the continuous type of state space and action space. Furthermore, DQN works best for such type of large data where the system needs to perform a lot of iteration to get the values for states and actions based on the Q-table where q-learning fails to handle such complex samples [7]. DQN have a neural network-based structure where the computation of the Q-values is performed by taking states as input and actions as output for the given state [state, action] through the network.

Subsequently, we applied the Epsilon Greedy policy in our DQN model in order to get an optimal policy for the agent. This was done to maximize the expected value of the total reward over the successive steps starting from the current state. For examole to optimize the state-action-value function by interacting with the environment Q(s,a) to suggest when to buy, sell or hold. Here, the agent interacts with the environment in two ways: exploration and – regardless of the maximum future reward – exploitation. After considering all the actions, it takes the optimal to interact in the most optimal manner [8]. In this model, the state depends on the daily values of buy, sell or hold of 3 years of stock data. The state changes as the trading session (the day) change.

A. Parameters for Stock Trading Model

By keeping in mind the stochastic and interactive nature of the stock trading market, we implemented the DQN model considering the following parameters:

1. Here, Q (s, a) satisfies the Bellman equation when an action is performed in a state s in a given action a.
2. ε represents the greedy action selection. ε represents the greedy action selection.
3. γ max Q*(s',a')|s,a) represents the maximum expected accumulative reward. Hence, the above-defined function will provide the maximum reward at the end of the n number of training cycles/iterations.

$$Q^*(s,a) = \mathbb{E}_{s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

B. Setting up our stock trading environment

In order to develop the environment in which the agent can explore and exploit, we first installed the required following libraries:

1. NumPy
2. Pandas
3. Matplotlib
4. Seaborn
5. TensorFlow
6. Yahoo
7. Finance
8. Libraries

D Our, Strategies

In our project, we worked on 3 different stock market data, namely Uber, Infosys, and Apple, by applying the DQN model. The data was fetched through the Yahoo Finance website. We first performed some exploratory data analysis in order to make logics for our agent to explore and learn in order to maximize the reward. We worked on 2 different strategies. First, we changed the value of the discount factor (γ) with different window sizes. Subsequently, in the next strategy, we added two and three dense layers. Moreover, our framework is capable enough to work also on year by year and for number of years as well. The results will also upgrade whenever new data for the above-mentioned companies is available on Yahoo Finance. Furthermore we are using a model-free strategy, the agent is going to explore and exploit the environment (using Epsilon Greedy policy).

We have defined the agent class with simulations using different functions to make our agent learn. For example, "Get state" prompts the agent to analyze the state of the environment, "Action" function prompts the agent to randomly select its first action by exploring the environment, "Replay" function prompts it to store the past experience, "Buy_Sell" function prompts it to take decisions, and "Train" function prompts it to train our agents.

In addition to the functions mentioned above, we have defined different important functionalities during simulations, such as "batch size", since DQN uses experience replay to learn in small batches in order to avoid skewing the dataset distribution of different states, actions, rewards, and next states, "window_size" to cutout the time

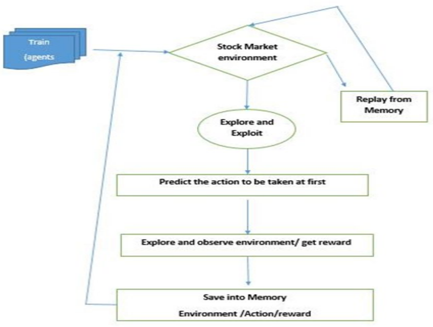

Train
{agents
Stock Market environment
Replay from Memory
Explore and Exploit
Predict the action to be taken at first
Explore and observe environment/ get reward
Save into Memory
Environment /Action/reward

Fig. 3.Working Flow of our

sequence of the data, and "deque" to handle the memory used by adding and removing elements from either end. Moreover, we have also defined a few static functions, such as "Epsilon Greedy", "Decay" and "Gamma".

C. Experiment and Simultaneous

In this study, we performed two iterations with different simulations. The purpose of these iterations was to analyze the impact of changing the parameters and dense layer of our DQN model so that agents can have a learning process and take better decisions to buy, sell or hold.

D. First Iteration

In the first iteration, we used a two-layered neural network to make decisions for the buy, hold, and sell call. Here "get_state" function and "Action" function takes up the value of the next state generated by the Neural network (action, q-value). Hence, the rewards are calculated by adding and subtracting the value that is generated by the execution of calls. Additionally, the action taken in the next state is determined by the action taken in the previous state, while accumulative rewards are saved in the variable named "Total Profit". During this iteration, we have changed three parameters: 'discount factor', which will be determined between immediate reward or the future reward, 'epochs', to train the agent by moving forward pass and back pass through the stock data, and 'window size', to train the agent for accurate decisions.

TABLE I. LENGTHS OF REWARD WINDOW CORRESPONDING TO DIFFERENT DISCOUNT FACTORS

Discount Factor (γ)	0.46	0.63	0.79
Epochs	100	200	300
Reward Window	10	20	30

If we take discount factor as 0.46, window size as 10, and epochs as 100, then the accumulative results by the three companies are as follows:

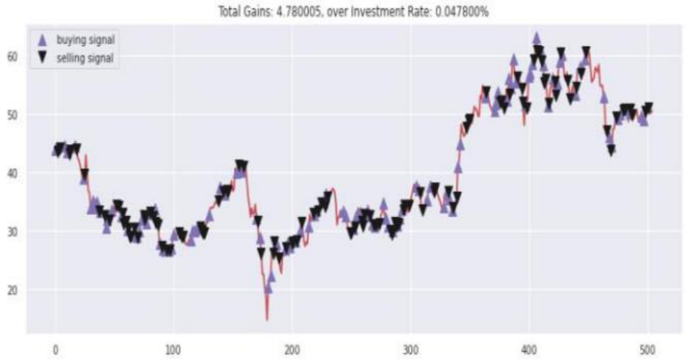


Fig. 4. Total Gains over investments by

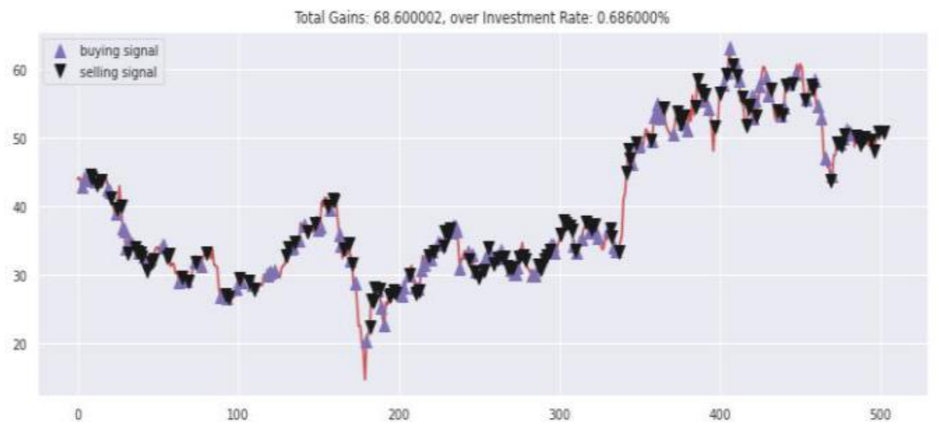


Fig. 5. Total Gains over investments by UBER, APPLE and INFO SYS

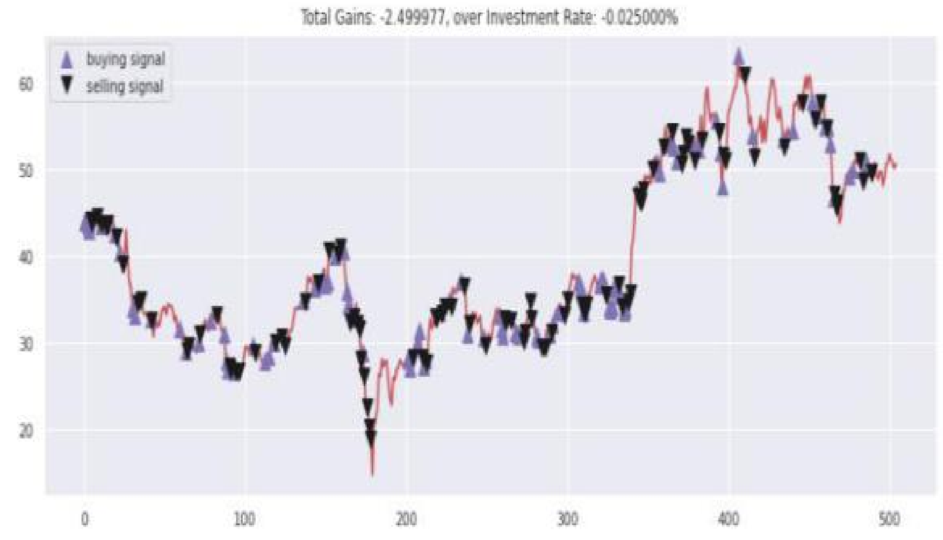


Fig. 6. Total Gains over investments by UBER, APPLE and INFO SYS Companies

UBER, APPLE and INFO SYS Companies If we take the discount factor as 0.63, window size as 20 and epochs as 200, then the accumulative results by the three companies are as follows:

By change/ if we take the discount factor as 0.79, window size as 20 and epochs as 300, then the accumulative results by the three companies are as follows:

E. Second Iteration

To overcome the above issue in the first iteration, we added a third dense layer in the network with the same values as shown:

From the above-given results in Figure 4, 5 and 6, it can be seen from the trends that when the discount factor is close to zero and the number of epochs are less than the model provide better results. However, we increase the discount factor and epochs size, model goes into overfitting on the training agent.

In the second iteration, we added additional dense, with the same discount factors, epochs, and window size as mentioned in Table 1

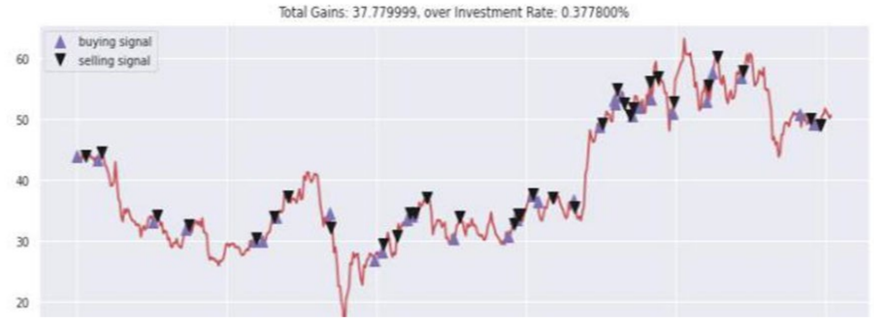


Fig. 7. Total Gains over investments by UBER, APPLE and INFO SYS companies

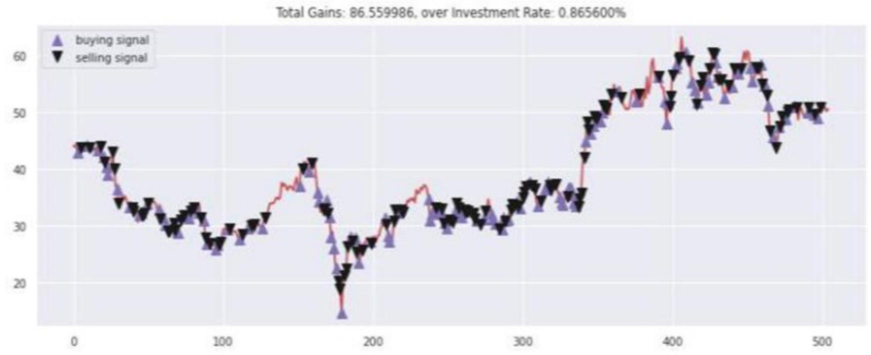


Fig. 8. Total Gains over investments by UBER, APPLE and INFO SYS companies

By taking the same values for parameters, that is, discount factor = 0.46, window size = 10 and epochs= 100, the accumulative results by the three companies are as follows

If we take the discount factor as 0.76, window size as 20, and epochs as 200, the accumulative results by the three companies are as follows:

Finally, if we take discount factor as 0.79, window size as 30, and epochs as 300, the accumulative results by the three companies are as follows:

From the above two iterations, we observed following points:

1. By increasing epochs, the model goes into overfitting.
2. As we increase the discount factor from 0.46 to 0.76, the chance of increasing future reward is increased, but action values may diverge as a result.
3. For reward window size, we observed that when the size increases from 10 to 30, the model's performance gets better.

Considering the above set of observations, we concluded that for our stock trading data, discount factor 0.76, window size 30, and epochs 200 is most appropriate.

IV. Results

In this project, we explored the performance of DQN at different parameters' values (Table 1) by adding dense layers. To analyze data from three different stocks, namely Apple Inc., UBER, and Infosys, the DQN with multi agents along with added dense layers out performed and gave profitable patterns for all mentioned companies. In order to optimize our DQN model, we performed multiple simulations by adding dense layers in the DQN model with different values of gamma in each iteration. The results of the study are given as follows:

By applying three dense layers with a discount factor of 0.67, our DQN model outperformed and gave realistic trend signalling for buying and selling. In the above-given Table 1, our model produced negative values for investment as well as for profit, but when we introduced another dense layer, the model optimized gradually and started producing better trends. This is because, our algorithm consists of three important components: Deep Q-Network to compute Q-value function, 'Epsilon Greedy' to optimize the accumulative reward function (total profit), and 'replay function' to remove the correlation. Vanilla DQN is found to be an efficient model in stock decisions. It is even better than its improved versions, such as Double DQN. Conversely, some studies showed that Deep Q-learning performed worse when there are drastic changes in the pattern of price. Resultantly, the trained and the test systems collapsed [9]. To overcome this issue, we performed simulations on 3 datasets and trained the agents to make better patterns. Moreover, we trained the agents to find per day rewards in order to analyze when companies need to invest in the stock, when they should buy when they should sell or hold. With this knowledge, agents are capable of learning the stock trend and take appropriate decisions. Figure 10 shows that the agent is capable of learning data each day by stating (a) with number of units sell with how much opening balance, (b) how much unit need buy or sell and how much to invest. All these decisions are taken by the agents through the rewards it gets during each state as shown in Figure 11. We also compared the proposed model with a baseline model. In the baseline models, trends of the graphs are given more focus to train the agents and the agent gets reward based on the trading trends [10]. However, on performing reversed opertation agents trained to learn the stock market data, perform 100 epochs to explore the environment, and finally make decisions based on the rewards it gets in order to make trend signals to buy, sell or hold.

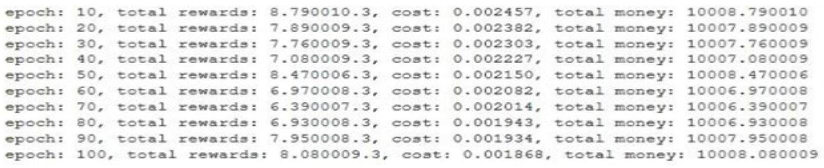
Figure 12 shows the performance of the Baseline model using DQN.

The model shows the trading trends on the National Association of Securities Dealers Automated Quotations (NASDAQ) index stock in terms of accumulative rewards. The results are then tested on Dow Jones Industrial Average (DJIA), NASDAQ, National Stock Exchange Fifty (NIFTY), and SENSEX data set as well.

Figure 13 shows the performance of our proposed model using three different datasets. It can be seen in Figure 4 that our model is capable enough to get the accumulative rewards individually for each company as well as the performance of overall model.

TABLE II.
AVERAGE ACCUMULATIVE PROFIT OVER DIFFERENT DQN LAYERS

Stock	Year	Discount Factor (γ)	Patterns(Buy/Sell) Based on Investment			
			Two Layers		Three Layers	
			Investment	Profit	Investment	Profit
UBER	2019 - 2021	0.46	-5,37000	8%	4,78000	12%
UBER	2019 - 2021	0.63	21,86998	21%	26,45999	26%
UBER	2019 - 2021	0.67	20,6588	20%	68,6000	36%
Apple Inc.	2019 - 2021	0.46	-2,49950	6%	47,86995	47%
Apple Inc.	2019 - 2021	0.63	26,51999	26%	53,90000	52%
Apple Inc.	2019 - 2021	0.67	1,58417	15%	86,55998	86%
Infosys Ltd	2019 - 2021	0.46	-1,23997	6%	28,68544	27%
Infosys Ltd	2019 – 2021	0.63	-5,49977	5%	37,77999	16%
Infosys Ltd	2019 - 2021	0.67	24,89541	15%	48,68547	48%



```
epoch: 10, total rewards: 8.790010.3, cost: 0.002457, total money: 10008.790010
epoch: 20, total rewards: 7.890009.3, cost: 0.002382, total money: 10007.890009
epoch: 30, total rewards: 7.760009.3, cost: 0.002303, total money: 10007.760009
epoch: 40, total rewards: 7.080009.3, cost: 0.002227, total money: 10007.080009
epoch: 50, total rewards: 8.470006.3, cost: 0.002150, total money: 10008.470006
epoch: 60, total rewards: 6.970008.3, cost: 0.002082, total money: 10006.970008
epoch: 70, total rewards: 6.390007.3, cost: 0.002014, total money: 10006.390007
epoch: 80, total rewards: 6.930008.3, cost: 0.001943, total money: 10006.930008
epoch: 90, total rewards: 7.950008.3, cost: 0.001934, total money: 10007.950008
epoch: 100, total rewards: 8.080009.3, cost: 0.001868, total money: 10008.080009
```

Fig. 10. Rewards over each iteration (epoch)

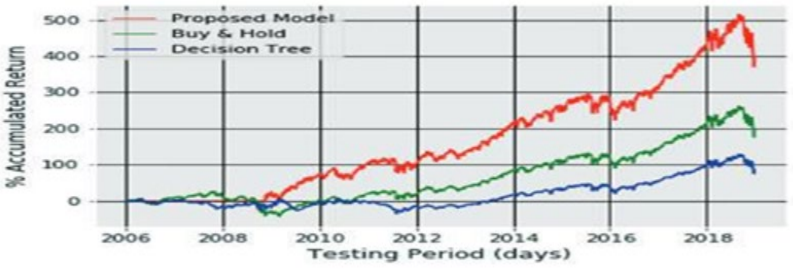
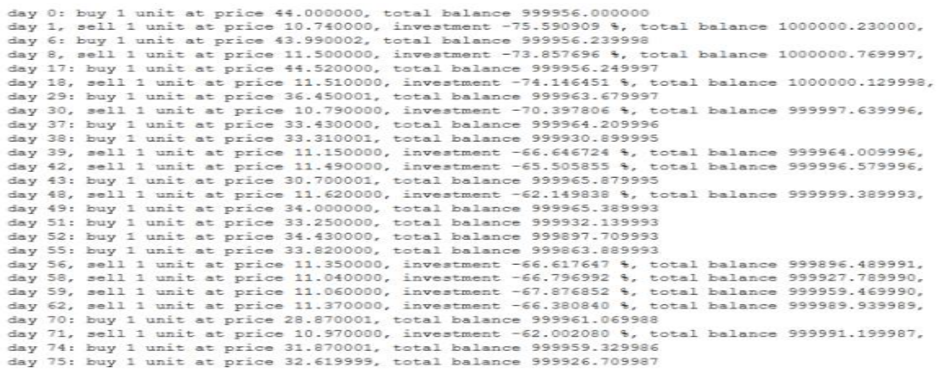


Fig. 11. Baseline model - Trend Following



```
day 0: buy 1 unit at price 44.000000, total balance 999956.000000
day 1, sell 1 unit at price 10.740000, investment -75.590909 %, total balance 1000000.230000,
day 6: buy 1 unit at price 43.990002, total balance 999956.239998
day 8, sell 1 unit at price 11.500000, investment -73.857696 %, total balance 1000000.769997,
day 17: buy 1 unit at price 44.520000, total balance 999956.249997
day 18, sell 1 unit at price 11.510000, investment -74.146451 %, total balance 1000000.129998,
day 29: buy 1 unit at price 36.450001, total balance 999963.679997
day 30, sell 1 unit at price 10.790000, investment -70.397806 %, total balance 999997.639996,
day 37: buy 1 unit at price 33.430000, total balance 999964.209996
day 38: buy 1 unit at price 33.310001, total balance 999930.899995
day 39, sell 1 unit at price 11.150000, investment -66.646724 %, total balance 999964.009996,
day 42, sell 1 unit at price 11.490000, investment -65.505855 %, total balance 999996.579996,
day 43: buy 1 unit at price 30.700001, total balance 999965.879995
day 48, sell 1 unit at price 11.620000, investment -62.149838 %, total balance 999999.389993,
day 49: buy 1 unit at price 34.000000, total balance 999965.389993
day 51: buy 1 unit at price 33.250000, total balance 999932.139993
day 52: buy 1 unit at price 34.430000, total balance 999897.709993
day 55: buy 1 unit at price 33.820000, total balance 999863.889993
day 56, sell 1 unit at price 11.350000, investment -66.617647 %, total balance 999896.489991,
day 58, sell 1 unit at price 11.040000, investment -66.796992 %, total balance 999927.789990,
day 59, sell 1 unit at price 11.060000, investment -67.876852 %, total balance 999959.469990,
day 62, sell 1 unit at price 11.370000, investment -66.380840 %, total balance 999989.939989,
day 70: buy 1 unit at price 28.870001, total balance 999961.069988
day 71, sell 1 unit at price 10.970000, investment -62.002080 %, total balance 999991.199987,
day 74: buy 1 unit at price 31.870001, total balance 999959.329986
day 75: buy 1 unit at price 32.619999, total balance 999926.709987
```

Fig. 12.

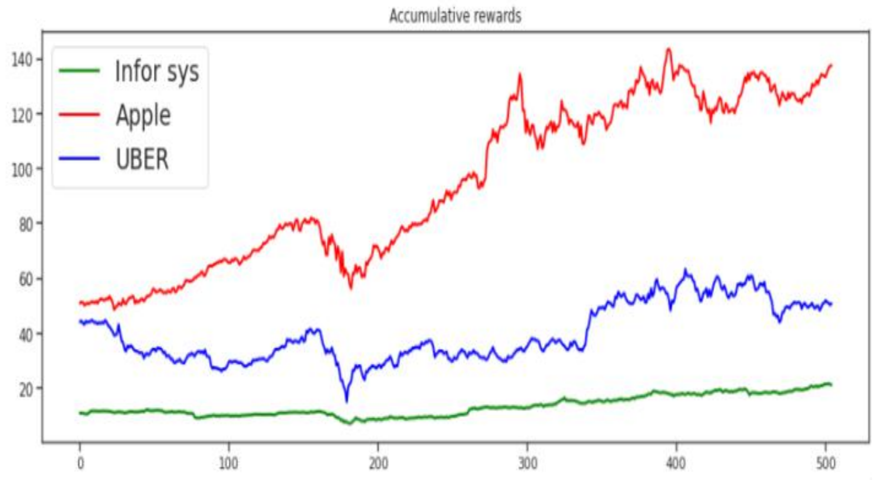


Fig. 13.

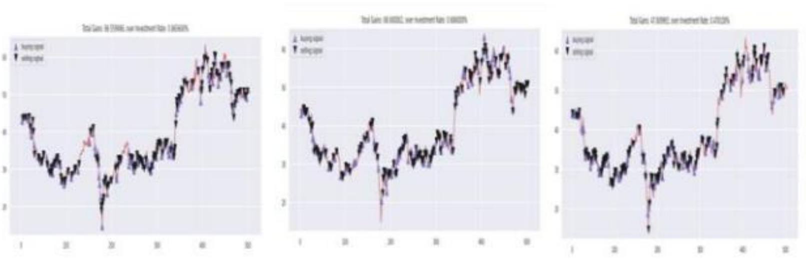


Fig. 14. Making Trendz

Discussion and Conclusion

We have explored the performance of our trading model by applying different strategies on three different datasets. Our DQN model with three layers outperformed in making trends to buy, sell, or hold signals, as shown in the figure below.

The model is capable of generating the trends based on reward signals given by the agents, which helps to understand when one should invest in the stocks and when one should hold. Other stock data can also be adapted to our model. This may help companies as well as individuals with limited information of the stock market and aid them in deciding when and how much they should invest in stocks. However, we believe that there is always room for improvement, which is why future researchers should apply other neural networks, such as RNN in the Deep Q-learning model. Since the trading data of RNN is sequential and continuous, applying QDN with RNN or any other neural network may give promising results. As the stock market is volatile and unpredictable for which risk control could be a significant factor. Therefore it would also like to explore other Reinforcement Learning methods as well.

References

1. T. L. Meng and M. Khushi, “Reinforcement learning in financial markets,” Data, vol. 4, no. 3, pp. 110, July. 2019. https://doi.org/10.3390/data4030110
2. H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, “Deep reinforcement learning for automated stock trading: An ensemble strategy,” in Proc. First ACM Int. Conf. AI in Finan., Oct. 2020, pp. 1-8. https://doi.org/10.1145/3383455.3422540
3. Q. V. Dang, “Reinforcement learning in stock trading,” in Int. Conf. Comp. Sci, Appl. Math. Applic., Dec. 2019, pp. 311–322. https://doi.org/10.1007/978-3-030-38364-0_28
4. M.-E. Wu, J.-H. Syu, J. C.-W. Lin, and J.-M. Ho, “Portfolio management system in equity market neutral using reinforcement learning,” Appl. Intell., vol. 51, no. 11, pp. 8119-8131, Mar. 2021. https://doi.org/10.1007/s10489-021-02262-0
5. H. Park, M. K. Sim, and D. G. Choi, “An intelligent financial

portfolio trading strategy using deep Q-learning," Expert Syst. Appl., vol. 158, pp. 113-573, Nov. 2020. https://doi.org/10.1016/j.eswa.2020.113573

6. J. Chakole and M. Kurhekar, "Trend following deep Q-Learning strategy for stock trading," Expert Syst., vol. 37, no. 4, pp. e12-514, Dec. 2020. https://doi.org/10.1111/exsy.12514
7. L. Chen and Q. Gao, "Application of deep reinforcement learning on automated stock trading," in 2019 IEEE 10th Int. Conf. Soft. Eng. Ser. Sci. (ICSESS), Beijing, China, Oct, 18-20, 2019, pp. 29-30. https://doi.org/10.1109/ICSESS47205.2019.9040728
8. J. B. Chakole, M. S. Kolhe, G. D. Mahapurush, A. Yadav, and M. P. Kurhekar, "A Q-learning agent for automated trading in equity stock markets," Expert Syst. Appl., vol. 163, pp. 113-761, Jan. 2021. https://doi.org/10.1016/j.eswa.2020.113761
9. K. Jakhar, "Reinforcement Learning: Cart-pole, Deep Q learning," Oct. 2019 [Online]. Available: https://medium.com/@karan_jakhar/100-days-of-code-day-4-6fbc672171e4
10. R. Purohit, "Simplified Reinforcement Learning: Q Learning," Nov. 2020 [Online]. Available: https://www.mygreatlearning.com/blog/simplified-reinforcement-learning-q-learning/