COMPARING TRADING STRATEGIES PERFORMANCES UNDER VARYING CONDITIONS

John Chieng Xiang Hao

Abstract

Trading stocks is a fundamental aspect of financial management, yet it is fraught with challenges due to the volatile and unpredictable nature of the economic environment. Time and energy investments are required for conducting thorough analysis, developing strategies, and making informed decisions. In light of these challenges, leveraging autonomous agents to assist in the analysis and modeling of historical data for generating trading strategies holds immense practicality and efficiency.

Reinforcement learning has emerged as a powerful tool in various domains, particularly in implementing time-series-based decision-making strategies. In this study, a comprehensive pipeline was constructed to replicate the conditions of the stock trading environment. Within this framework, different deep reinforcement learning techniques, including Deep Q-learning, Deep SARSA, and Policy Gradient, were employed to instruct an agent in executing automated stock trading operations.

The performance of these agents was evaluated under both favorable (prior to 2021) and unfavorable (2021-2022) circumstances, utilizing historical data from prominent stocks such as Google, Apple, Tesla, Meta, Microsoft, and IBM. These equities, widely followed in the market, serve as reliable indicators of market trends and fluctuations.

The results demonstrate that prior to 2021, all three reinforcement methods consistently yielded favorable profit returns, with Policy Gradient exhibiting the highest performance. However, following 2021, marked by adverse economic conditions potentially influenced by external factors such as the COVID-19 pandemic, Deep Q Learning emerged as the superior performer.

This study provides valuable insights into the efficacy of reinforcement learning techniques in autonomous stock trading strategies. The findings underscore the importance of adaptability and algorithm selection in navigating the complexities of stock market dynamics, offering practical implications for traders and researchers alike.

1 Introduction

Stock trading stands as a cornerstone of financial management, offering avenues for profit generation amidst the ever-changing landscape of economic conditions (Fabozzi & Peterson (2003) [7]. However, the complexities inherent in

market dynamics demand extensive analysis, strategic formulation, and timely decision-making, often requiring significant investments of time and energy. In response to these challenges, I embarked on a journey to explore the integration of autonomous agents trained with reinforcement learning algorithms as a promising approach to streamline and enhance stock trading processes.

This study seeks to address the pivotal question: *What is the optimal algorithm for autonomous trading agents in stock market environments?* To tackle this inquiry, I delved into the realm of reinforcement learning, leveraging its capabilities to train autonomous agents to navigate the complexities of stock trading. Specifically, I employed a Markov Decision Process (MDP) framework to simulate trading interactions, enabling agents to learn and adapt their strategies based on historical market data.

Within this framework, I evaluated the performance of three prominent reinforcement learning algorithms: Deep Q learning (Fan et al., 2020)[21], Deep SARSA (Zha et al., 2016;[27] Andrecut & Ali, 2001[5]; Alfakih et al., 2020)[4], and Policy Gradient (Sutton et al., 1999;[23] Peters & Schaal, 2006)[20]. Each algorithm underwent rigorous testing under varying market conditions, ranging from favorable periods predating 2021 to the challenging economic landscape post-2021, potentially influenced by external factors such as the COVID-19 pandemic (Aggarwal et al., 2021[2]; Hashim JH et al., 2021[10]; Ngwakwe et al., 2020[19]).

My exploration uncovered intriguing insights into the efficacy of these algorithms in autonomous stock trading. I found that while all algorithms demonstrated promising results in favorable market conditions, the optimal performer varied depending on the economic context. Pre-2021, the Policy Gradient algorithm emerged as the frontrunner, showcasing superior profit returns. However, post-2021, characterized by pessimistic economic conditions, the Deep Q Learning algorithm exhibited heightened effectiveness in navigating market volatility.

By elucidating the performance nuances of different reinforcement learning algorithms in autonomous stock trading, this study contributes to advancing our understanding of effective trading strategies amidst evolving market dynamics. Moreover, it underscores the importance of adaptability and algorithm selection in optimizing stock trading outcomes.

In the subsequent sections, I delve deeper into the methodology employed, empirical findings, and implications derived from my analysis, offering valuable insights for practitioners and researchers alike in the realm of autonomous stock trading.

2 Literature Review

Financial investment and management are advantageous for the market and the general public. Peterson & Fabozzi (2003)[7]. A prevalent method of financial administration and investment, stock trading relies on the provision of publicly accessible data that can be scrutinised and assessed. Numerous exemplary works, including those by Aguirre et al. (2020)[3], Adrian (2011)[1], Maitah et al. (2016)[17], and Gurrib (2018)[9], employ a single number to summarise historical patterns and illustrate the prevailing trend. These techniques facilitate market inspection and serve as a valuable aid in formulating trading strategies.

Reinforcement learning has demonstrated resilience across several tasks. Qiang and Zhongli (2011)[22] and Naeem et al. (2020)[18]. The process of stock trading can be represented as a Markov Decision Process (MDP) according to Puterman (1990).[21] This MDP incorporates time series data from the environment and involves making decisions to interact with the environment. Well-known reinforcement learning techniques such as Deep Q-Learning and Deep SARSA Watkins & Dayan (1992)[24] and Corazza & Sangalli (2015)[6] are being utilised to determine the optimal policies in Markov Decision Processes (MDP). Nevertheless, the work of stock trading is a significant challenge for the basic form of these training methods. The primary factor is that the data exists in a continuous space, and converting the data into discrete values could result in a substantial number of parameters due to the extensive range of stock prices. Another issue is that the discretization method may lead to a lack of generalisation. Methods for deep reinforcement learning. Fan et al. (2020)[8], Alfakih et al. (2020)[4], Zhao et al. (2016), Andrecut & Ali (2001)[5], Sutton et al. (1999)[23], and Peters & Schaal (2006)[20] have demonstrated the ability to teach the agent to do the task in increasingly intricate contexts. Li [14] and Zhang et al. (2019)[26] included a deep neural network into the Markov Decision Process (MDP) framework and utilised a gradient-based method to optimise the parameters. Yang et al. (2020)[25] and Liu et al. (2021[16]; 2022[15]) have utilised a deep reinforcement learning technique to tackle the stock trading task. Their work offers valuable instructions on constructing the simulation pipeline and training environment for the agent.Thanks to their work, I can effectively replicate the complete environment and training process to perform further study and apply the technique to different scenarios.

In recent times, the stock market has been significantly impacted by the epidemic, resulting in decreased stability. The publications by Aggarwal et al. (2021)[2], Hashim JH et al.,2021[10], and Ngwakwe et al.(2020)[19] are referenced. Individuals with average expertise, such as ourselves, must dedicate additional time and effort to thoroughly analyse the market and exercise caution while making decisions. The stock market is perceived as unfavourable during the pandemic and favourable during a period of relative stability. This project involves training the agent using deep reinforcement learning methods and assessing its performance in both favourable and unfavourable scenarios. Additional information and in-depth analysis will be provided in the subsequent parts.

3 Experiments

3.1 Environment

For the experiment environment,I followed Yang et al. (2020)[25] to configure the MDP environment for this stock trading task. The programming language used is Python, and each element of this MDP is presented in detail in successive subsections.

3.1.1 Input Data

The input for this application comprises time-series data representing the price of a solitary share of stock over a one-day interval. Additional information regarding the intervals between training and assessment dates will be presented in the Result 4 section.

3.1.2 Action

The action space of the trading agent is whether to buy, hold, or sell the stock in each time step. I set a constraint here where the agent can trade at most k stocks each time, either buy or sell. With $k > 0 \in Z$, the actions A is defined quantitatively as

$$A = -k, -k+1, ..., -1, 0, 1, ..., k-1, k$$

where negative and positive values represent sales and purchases, respectively, and 0 means holding.

3.1.3 State

In this application, the task of trading stocks is essentially modelled as a Markov Decision Process (MDP). The state s at time t was initially defined as

$$s_t = [p_t, b_t, h_t]$$

where p_t is the current price of a single share of stock, b_t is the current balance available in the port-folio, and h_t is the current number of shares hold in the portfolio. However Yang et al. (2020)[25] proposed the use of market index as features which is the Moving Average Convergence Diver-gence (MACD) Aguirre et al. (2020)[3], Relative Strength Index (RSI) Adrian (2011)[1], Commodity Channel Index (CCI) Maitah et al.(2016)[17], Average Directional Index (ADX) Gurrib (2018)[9].These indices are hand made features where each of them summarize the trend in the past 2 or 3 weeks that are informative for decision making. Thus, the final version of the state is represented as

$$s_t = [p_t, b_t, h_t, MACD, RSI, CCI, ADX]$$

The transition from state t to $t+1$ is deterministic. The price at $s_t + 1$ is fixed in the training set, balances and number of shares are calculated based on the action at, and those market indices are recalculated based on the past data till $t+1$.

3.1.4 Reward

The reward set is the changing in the total asset value of the portfolio. More specifically, the reward R given the states at time t and t + 1 with action at is defined as

$$R(s_t, a_t, s_t + 1) = (b_t + 1 + p_t + 1 \cdot h_t + 1) - (b_t + p_t \cdot h_t) - c_t$$

where ct is the small fees applied on each trading event. This reward can be seen as the intermediate gain or loss of a portfolio after a trading event.

3.2 Assumption and Constraints

I followed Yang et al. (2020)[25] to construct the environment with the following assumptions:

- Orders can be executed quickly at the close price.
- The stock market is assumed to not be affected by the reinforcement trading agent.
- Because the trading fees vary across platforms and services, for simplicity, the fee for each trade is assummed as $c_t = 0.1\% \cdot p_t \cdot (|h_t - h_t - 1|)$, where $|h_t - h_t - 1|$ represents the number of shares that are bought or sold.

There are two constraints in the environment:

- When conduct buying action, the total price cannot make the balance below the certain tolerance value (e.g. \$0, \$-100, etc).
- When conducting a sale action, the number of shares being sold cannot exceed the number of shares that remain in the portfolio.

3.3 Deep Q-Learning

The first algorithm I applied to train the agent is Q-Learning. Because I have the state declared as $s_t \in \mathbb{R}^7$ where $|S| = \infty$ and I have action with $|A| = 2k+1$, a table of $q(s, a)$ that contain all the combinations cannot be constructed. By decretizing the state, the table can be formed, but the space of $\mathbb{R}^7$ would make the table too large, hence still not feasible for applying the original Q-Learning algorithm. To address this situation,a neural network is trained to approximate the q function. The procedure of training the agent with deep Q-Learning is shown in algorithm 1, with α as learning rate, γ as discount factor, π as policy, and q as the neural network that take state s as input and output the q values for each action (p.s. $q(s, a)$ means feed s to q, and extract the value at corresponds to the index of a in the output).

Algorithm 1 Deep Q-Learning, Input: α, γ, π, q

Initialize π as uniformly distributed over actions for all states
for $episode = 1, 2, \ldots$ **do**
    Generate a trajectory from current policy π
    $\hat{q}(s, a) = COPY(q(s, a))$
    for $epoch = 1, 2, \ldots$ **do**
        for Each s_t, a_t, R_t **do**
            $q_{new}(s_t, a_t) = q(s_t, a_t) + \alpha(R_t + \gamma \max_{a'} \hat{q}(s_{t+1}, a') - q(s_t, a_t))$
            Loss = HuberLoss$(q(s_t, a_t), q_{new}(s_t, a_t))$
            Conduct Backpropagation from Loss
            Update weights of q with Adam Optimizer
        end for
    end for
    Update π with ϵ-greedy strategy
end for

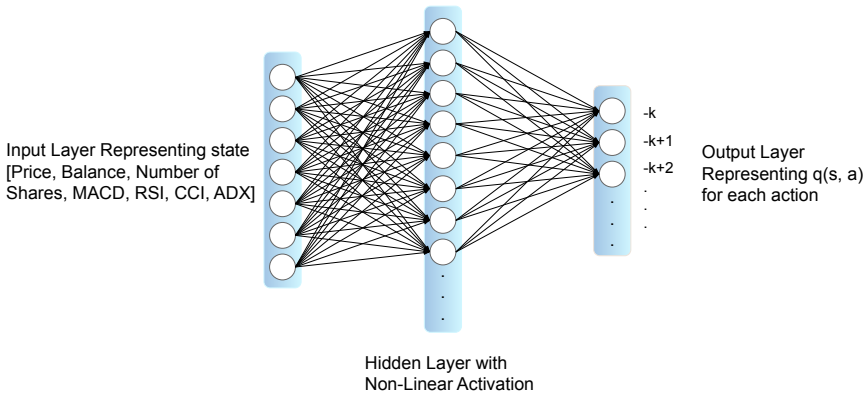

Figure 1: Deep Q Network

More details of the algorithms are described below. First, the policy for generating the trajectory in each episode follows the ϵ-greedy policies. The estimates optimal action is $a^* = argmax_{a' \in A} q(s, a')$. Then the actions in each step are sampled according to the following probabilities:

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = a^* \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases}$$

Secondly, the loss function chosen as the objective function for q neural

networks is Huber Loss [12, 11] that have been shown to be robust for modeling regression tasks, because it kept the strength of both Mean Absolute Error (MAE) and Mean Squared Error (MSE). And the optimizer used to update the weights of the neural network is Adam [13].

Finally, the neural network constructed for estimating the q function is a three layers multi-layer perceptrons, with ReLU as activation function to create non-linearity. The workflow of this deep neural network is shown in Figure 1.

3.4 Deep SARSA

Subsequently, the Deep SARSA algorithm was implemented, functioning similarly to Deep Q-Learning.Initially,the algorithm was evaluated using the same neural network architecture as Deep Q-Learning before attempting alternative architectures in an effort to identify the model with the highest performance.In the end, it was evident that the optimal performance could be generated using the identical structure. The difference between Deep Q-Learning and Deep SARSA is that instead of selecting update $q_{new}(s_t, a_t) = q(s_t, a_t) + \alpha(R_t + \gamma \max_{a'} \hat{q}(s_{t+1}, a') - q(s_t, a_t))$ and then compute the Huber loss and backpropagate to update the gradient,the next state and corresponding action is found from the simulation of the episode and then the update rule is: $q_{new}(s_t, a_t) = q(s_t, a_t) + \alpha(R_t + \gamma \hat{q}(s_{t+1}, a_{t+1}) - q(s_t, a_t))$. The Deep SARSA algorithm has the same hyperparameters as Deep Q-learning, but has different values after the hypertune process. In addition, the ϵ-policies update rule has been kept in the Deep SARSA model.

3.5 Policy Gradient Method

Furthermore, the third algorithm that was implemented is the Policy Gradient Method. Instead of learning the parameters of the q function, the policy gradient method trained the parameters for the policy itself. To train the policy, the neural network model was updated so that the last output will be a probability distribution over all action through the softmax function. The procedure of training the policy gradient model is shown in Algorithm 2. There are several parameters that are used as input in the algorithm, such as α, which is the step size in updating the gradient, then γ is used as the discount rate when calculating the return, and the policy π.

Notably, when calculating the loss using the policy gradient algorithm, the negative sign is used because this is an optimising procedure and not a minimising one. By performing a multiplication operation with -1, it is possible to modify the gradient in the desired direction for training the model parameters. Furthermore, in order to achieve the most effective model, various sets of hyperparameters is evaluated, such as the learning rate, gamma, epsilon, episodes, and epochs, in an effort to optimise the hyperparameters. It is noteworthy that the hyperparameter configuration remains constant across all three models; the sole distinction lies in the optimizer selected, which can be either the policy gradient method or deep learning algorithms.

Algorithm 2 Policy Gradient, Input: α, γ, π

Initialize π as uniformly distributed over actions for all states
for *episode* $= 1, 2, \ldots$ **do**
  Generate a trajectory from current policy π
  for *epoch* $= 1, 2, \ldots$ **do**
    for Each step(t) in the episode **do**
      $G_t = \sum_{K=t}^{T} \gamma^{k-t} R_k$
    end for
    Loss $= -\frac{1}{t} \sum_{t}^{T} ln(G_t \pi(a_t|s_t, \theta)$
    Conduct Backpropagation from Loss
    Update weights of π with SGD Optimizer
  end for
end for

4 Results

4.1 Evaluation Schema

The reinforcement learning methods in two scenarios is evaluated in two scenarios, where one is conducted before the beginning of 2021, another is conducted after 2021. Because, starting in 2021, many of the stocks undergo relatively very different trends than they had before 2021. The training and testing range is shown in Table 1 where the dates are presented with order: month, day, year.

Scenarios	Train	Test
Before 2021	01-08-2013 to 01-02-2019	01-03-2019 to 12-30-2020
After 2021	01-08-2013 to 12-08-2020	12-09-2020 to 12-01-2022

Table 1: Overview of data sets.

The stocks chosen for this project are Google, Apple, Tesla, Meta, Microsoft, and IBM as these stocks are among the popular ones, and the changes in trends are representative in terms of having good and bad situations. The good situation occurs when the market is relatively stable and the prices are increasing overall. The bad situation occurs when the prices oscillate with patterns very differently from the patterns of past data. The data used are shown in Figure 2.

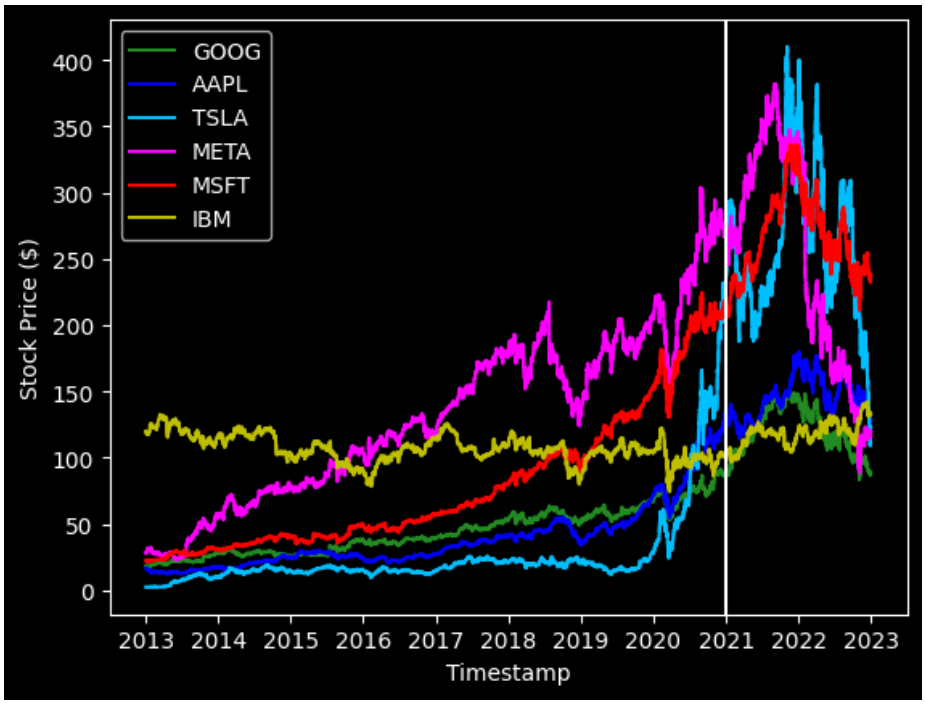

Figure 2: Prices of the selected stocks. The split line separate the good and bad situations.

The initial balance for each stock is \$1000 (i.e. \$6000 in total for the investment), and each trading event (buy or sell) contains at most 5 numbers of shares. There is a tolerance that the balance can go to negative to $\$ - 100$ for each stock. For hyperparameters, the setting is fine tuned until an exponential decrease in loss of estimation by neural networks is observed. The figure shows that the loss decreases during the training process and finally converges. The final setting is shown in Table 2 and applies to the three models.

As shown in the previous algorithm, the optimizer used for deep Q-Learning and deep SARSA is ADAM, and the optimizer that was eventually selected for policy gradient is SGD.

Furthermore, the training and testing is ran for a total of 20 epochs and the average performance is recorded and shown in the following subsections.

4.2 Before 2021 - Good Situation

Table 3,4 and 5 presents the profit along with the annual return rate on the test set before the beginning of the year 2021.

Hyperparameter	Value
Episode	30
Discount Factor γ	0.6
Learning Rate α	0.7
Initial ϵ policy	0.8
Minimum ϵ policy	0.2
ϵ Decay	linearly Decay with factor of 0.9
Epochs for Updating NN	10
Learning Rate for NN	1e-5

Table 2: Setting of hyperparameters. The notations stay consistent as described in previous sections. NN stands for Neural Network

Deep Q-Learning	GOOG	AAPL	TSLA	META	MSFT	IBM	Total
Profit	\$238.97	**\$1654.77**	**\$8475.20**	**\$500.10**	\$496.03	**\$75.75**	\$11340.82

Table 3: Deep Q Returned Profits on the test set before 2021.

Deep SARSA	GOOG	AAPL	TSLA	META	MSFT	IBM	Total
Profit	\$32.80	\$485.14	\$3172.59	\$63.24	\$138.95	\$30.88	\$3923.6

Table 4: Deep Sarsa Returned Profits on the test set before 2021.

Policy Gradient	GOOG	AAPL	TSLA	META	MSFT	IBM	Total
Profit	**\$309.47**	\$1125.49	\$8254.33	\$121.79	**\$638.45**	\$61.02	**\$11510.55**

Table 5: Policy Gradient Returned Profits on the test set before 2021.

In the initial phase of our analysis, encompassing data preceding 2021, we trained all three models—Deep Q Learning, Deep SARSA, and Policy Gradient—to make decisions aimed at securing profits across the selected stocks. Overall, our experimentation revealed that each method exhibited competitive performance, consistently maintaining a positive return on average. However, the Policy Gradient Method emerged as the standout performer in terms of profit generation.

This superior performance of the Policy Gradient Method can be attributed to its unique approach of directly optimizing policy to maximize expected cumulative rewards. In an optimistic market environment characterized by discernible and consistent trends, this method excels at exploiting these trends efficiently. Unlike value-based methods such as Deep Q Learning and SARSA, which focus on estimating the value of each action, the Policy Gradient Method directly learns the optimal policy, enabling it to leverage market trends more effectively.

4.3 Starting 2021 to 2022 - Bad Situation

Table 6,7 and 8 present the profit along with annual return rate on the test set after the beginning of the year 2021.

Deep Q-Learning	GOOG	AAPL	TSLA	META	MSFT	IBM	Total
Profit	$223.98	**$322.80**	**$979.93**	$-253.44	$114.43	**$224.79**	**$1612.49**

Table 6: Deep Q Returned Profits on the test set after 2021.

Deep SARSA	GOOG	AAPL	TSLA	META	MSFT	IBM	Total
Profit	$11.55	$157.01	$789.13	**$-73.01**	$24.19	$78.65	$987.52

Table 7: Deep Sarsa Returned Profits on the test set after 2021.

Policy Gradient	GOOG	AAPL	TSLA	META	MSFT	IBM	Total
Profit	**$233.41**	$257.44	$900.11	$-302.46	**$154.08**	$129.26	$1371.84

Table 8: Policy Gradient Returned Profits on the test set after 2021.

When we transitioned to testing the models using data from after 2021, a noticeable decline in performance was observed across all models for each stock. In some instances, the models even incurred losses by the end of the training period. This decline in performance can be attributed not only to the structure and hyperparameters of the neural network but also to the nature of the input data.

The distribution of the test set data from 2021 to 2022 significantly differed from that of the data preceding 2021. Consequently, the neural networks embedded in the agents lacked generalization on trends they hadn't encountered during training. This lack of exposure to diverse market conditions rendered the models less adept at adapting to the evolving dynamics of the market, resulting in degraded performance.

In this unfavorable scenario, while the stock trading agents still managed to achieve a positive return overall, instances of negative returns were more prevalent compared to the performance observed in more favorable market conditions. As discussed, the shift in data distribution was the primary driver behind this degradation in performance.

Introducing data from past financial crises into the training set could potentially mitigate this decline by exposing the models to a wider range of market scenarios. This would also provide an opportunity to assess the robustness of the methods under adverse conditions.

In my investigation, the Deep Q Learning Method demonstrated the most robust performance among the three models, achieving a profit of 1612.49. This

resilience can be attributed to the nature of Deep Q Learning, which directly approximates the value function. By estimating the value of each action in a given state, Deep Q Learning is better equipped to navigate through uncertain and volatile market conditions, compared to Policy Gradient Methods and SARSA, which rely on optimizing policy or estimating state-action values, respectively.

5 Conclusion

In this project, we explored the effectiveness of three different deep reinforcement learning methods—Deep Q Learning, Deep SARSA, and Policy Gradient—in training autonomous stock trading models. Our evaluation was conducted under two contrasting market scenarios: one characterized by optimism and the other by pessimism.

Under optimistic market conditions, we observed that the Policy Gradient method consistently outperformed the other two methods. However, in the face of pessimistic market conditions, the Deep Q Learning method emerged as the superior performer.

These findings highlight the importance of selecting the appropriate reinforcement learning method based on the prevailing market conditions. While Policy Gradient excelled in exploiting opportunities during optimistic periods, Deep Q Learning demonstrated resilience and adaptability in navigating turbulent market environments.

Moving forward, there are opportunities for further exploration and refinement of these methods. One avenue for future research involves introducing "noisy" or pessimistic stock data into the training process to assess how the performance of each method is impacted. This would provide insights into the robustness and generalizability of the trained models in real-world scenarios.

In conclusion, our study underscores the significance of understanding the strengths and limitations of different deep reinforcement learning methods in the context of autonomous stock trading. By leveraging these insights, practitioners can make informed decisions to optimize trading strategies and adapt to dynamic market conditions effectively.

References

[1] ran-Moroan Adrian. The relative strength index revisited. *African Journal of Business Management*, 5(14):5855–5862, 2011.

[2] Shobhit Aggarwal, Samarpan Nawn, and Amish Dugar. What caused global stock market meltdown during the covid pandemic–lockdown stringency or investor panic? *Finance research letters*, 38:101827, 2021.

[3] Alberto Antonio Agudelo Aguirre, Ricardo Alfredo Rojas Medina, and Néstor Darío Duque Méndez. Machine learning applied in the stock market through the moving average convergence divergence (macd) indicator. *Investment Management & Financial Innovations*, 17(4):44, 2020.

[4] Taha Alfakih, Mohammad Mehedi Hassan, Abdu Gumaei, Claudio Savaglio, and Giancarlo Fortino. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa. *IEEE Access*, 8:54074–54084, 2020.

[5] M Andrecut and MK Ali. Deep-sarsa: A reinforcement learning algorithm for autonomous navigation. *International Journal of Modern Physics C*, 12(10):1513–1523, 2001.

[6] Marco Corazza and Andrea Sangalli. Q-learning and sarsa: a comparison between two intelligent stochastic control approaches for financial trading. *University Ca'Foscari of Venice, Dept. of Economics Research Paper Series No*, 15, 2015.

[7] Frank J Fabozzi and Pamela P Peterson. *Financial management and analysis*, volume 132. John Wiley & Sons, 2003.

[8] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pages 486–489. PMLR, 2020.

[9] Ikhlaas Gurrib. Performance of the average directional index as a market timing tool for the most actively traded usd based currency pairs. *Banks and Bank Systems*, 13(3):58–70, 2018.

[10] Hashim JH, Adman MA, Hashim Z, Mohd Radi MF, and Kwan SC. Covid-19 epidemic in malaysia: Epidemic progression, challenges, and response. *Frontiers in Public Health*, 2021.

[11] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[12] Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101, 1964.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[15] Chunli Liu, Carmine Ventre, and Maria Polukarov. Synthetic data augmentation for deep reinforcement learning in financial trading. In *Proceedings of the Third ACM International Conference on AI in Finance*, pages 343–351, 2022.

[16] Xiao-Yang Liu, Hongyang Yang, Jiechao Gao, and Christina Dan Wang. Finrl: Deep reinforcement learning framework to automate trading in quantitative finance. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9, 2021.

[17] Mansoor Maitah, Petr Prochazka, Michal Cermak, and Karel Šrédl. Commodity channel index: Evaluation of trading rule of agricultural commodities. *International Journal of Economics and Financial Issues*, 6(1):176–178, 2016.

[18] Muddasar Naeem, Syed Tahir Hussain Rizvi, and Antonio Coronato. A gentle introduction to reinforcement learning and its application in different fields. *IEEE access*, 8:209320–209344, 2020.

[19] Collins C Ngwakwe et al. Effect of covid-19 pandemic on global stock market values: a differential analysis. *Acta Universitatis Danubius. Œconomica*, 16(2):255–269, 2020.

[20] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225. IEEE, 2006.

[21] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[22] Wang Qiang and Zhan Zhongli. Reinforcement learning model, algorithms and its application. In *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, pages 1143–1146. IEEE, 2011.

[23] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[24] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[25] Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. Deep reinforcement learning for automated stock trading: An ensemble strategy. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

[26] Zidong Zhang, Dongxia Zhang, and Robert C Qiu. Deep reinforcement learning for power system applications: An overview. *CSEE Journal of Power and Energy Systems*, 6(1):213–225, 2019.

[27] Dongbin Zhao, Haitao Wang, Kun Shao, and Yuanheng Zhu. Deep reinforcement learning with experience replay based on sarsa. In *2016 IEEE symposium series on computational intelligence (SSCI)*, pages 1–6. IEEE, 2016.