ASEN 5264 Final Project: Reinforcement Learning the Stock Market for Portfolio Allocation

A Comparison of Off-The-Shelf Deep RL Algorithms for Maximizing Capital Gains

1st Ryan Block
Ann and H.J. Smead Aerospace Engineering Sciences
Unversity of Colorado at Boulder
Boulder, Colorado
ryan.block@colorado.edu

2nd Cole MacPherson
Ann and H.J. Smead Aerospace Engineering Sciences
Unversity of Colorado at Boulder
Boulder, Colorado
cole.macpherson@colorado.edu

***Abstract*—This paper presents a comparative study of the effectiveness of deep reinforcement learning algorithms for portfolio allocation in the stock market. The objective of this study is to investigate the performance of 5 off-the-shelf deep reinforcement learning algorithms: Advantage Actor Critic (A2C), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Twin-Delayed Deep Deterministic policy gradient (TD3), and Soft Actor Critic (SAC), on an OpenAI style gym environment that includes real-world financial data. The data-set consists of stock prices of the 30 constituents within the Dow Jones Industrial Average spanning over a period of 10 years of training (2000 - 2010) and 11 years of trading (2011-2022). Using Feature Engineering to format Yahoo financial data as am MDP, we are able to apply DRL algorithms to learn portfolio allocation that is able to outperform the Dow Jones Industrial Average after 5 and 10 years of trading. We evaluate the algorithms' performance on the basis of key metrics such as cumulative returns, percent improvement over DJIA, and training time. Our experimental results indicate that each of the DRL algorithms is successful at different points, and that different training runs impact the quality of policies produced, often in a way that is difficult to explain or understand. These findings suggest that while all of the algorithms are able to perform portfolio allocation better than the DJIA, with potential for practical implementation in the financial industry, PPO is the most promising deep reinforcement learning algorithm for portfolio allocation in the stock market.**

I. Introduction

As the stock market continues to be both a complex and volatile area, predicting stock movement and properly allocating a portfolio remains a challenge for investors. While many methods and tools have been developed to analyze market trends and patterns, traditional approaches are often limited by their ability to adapt to rapidly changing market conditions and truly leverage all available data. To address this issue, this project seeks to utilize reinforcement learning to preform portfolio allocation by learning from past market data and providing feedback quickly. By utilizing the power of reinforcement learning, the project aims to pick stocks to maximize money made. To do this, we will compare the performance of off the shelf reinforcement learning algorithms in generating policies that buy and sell from a set of 30 stocks.

To preform reinforcement learning on the stock market, we have to first set up the problem as an Markov Decision Process (MDP), and subsequently solve the MDP with the goal of maximizing cumulative reward. In this work we use FinRL, a financial focused reinforcement learning library in Python, and OpenAI's Stable Baselines 3 deep reinforcement learning algorithms to approach this problem.

In particular, we will compare the performance of Advantage Actor Critic (A2C), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Twin-Delayed Deep Deterministic policy gradient (TD3), and Soft Actor Critic (SAC). These particular algorithms were chosen because they were already implemented within FinRL- future work should build on the previous implementations and introduce additional algorithms to the environment.

II. Background and Related Work

Since the advent of machine learning, researchers, businesses, and investors have been attempting to utilize it on the stock market. In researching for this project, we decided to focus on research that specifically utilizes a Markov framework and reinforcement learning.

Several efforts have been made to capture the stock market in an MDP formulation [3] [6] [11]. The MDP structure has been shown to aid in decision making for both single stocks and stock portfolios. The Partially Observable MDP framework has also been used [5], however we only consider the fully observable MDP for this project.

Reinforcement learning has been used to predict stock prices [7], model stock market trends [13], and trade stocks - portfolio allocation. We are interested in portfolio allocation for this project.

Portfolio allocation, also known as asset allocation, is the process of dividing an investment portfolio among different asset classes, such as stocks, bonds, real estate, commodities, and cash, based on the investor's risk tolerance, investment objectives, and time horizon. In this work, we will focus on allocating a portfolio of 30 stocks, with no other classes of assets. Extensive work on the bond market (with the added risk of bonds that can default) is shown in reference [11].

There are some projects that have attempted to use genetic algorithms with these MDP formulations to pick stocks [1].

Deep RL algorithms were even utilized in automated trading, allowed to learn through trial and error, and were shown to be capable of achieving Sharpe ratios of 2.68, which is much better than a typical human-allocated portfolio [5]. On a smaller scale, our project attempts to automate trading on real world data to allocate a portfolio.

This project heavily leverages the Liu, Xiong, Zhong, Yang, and Walidpaper's excellent paper "Practical Deep Reinforcement Learning Approach for Stock Trading" [9]. This paper, written by the team behind FinRL, gives an outstanding framework for beginners to get started in Deep Reinforcement Learning for stock market applications. Truly, it would not have been possible to attempt a problem this complex without this paper and the corresponding libraries.

III. Problem Formulation

Turning the stock market into an Markov Decision Process is a challenging task. The MDP formulation used for this paper is based on reference [9]. Using this MDP, we then can treat this as a maximization problem on overall profit. The MDP is built as follows:

A. State

The state space, at first glance, can be unintuitive. This is intentional, however, because we want to capture as much of the "hidden data" as possible for our deep reinforcement learning. The state is constructed as:

$$s = \{M, MACD, RSI, CCI, ADX\} \tag{1}$$

Where M is the covariance matrix, $MACD$ is Moving Average Convergence Divergence (effectively the momentum of the stock's price), RSI is the Relative Strength Index (an indicator whether a stock is overbought or oversold), CCI is Commodity Channel Index (measures the difference between the current price and historical average price), and ADX is the Average Direction Index (shows short term the direction a stock is trending).

B. Action

The action of this MDP is a reallocation of the portfolio. There is a weight corresponding to each stock between [0,1] that is reassigned at every step. Softmax is used to normalize all actions to sum to 1.

C. Transitions

Transitions are deterministic. If the recommended action is to buy, hold, or sell a stock, the order is guaranteed to succeed.

D. Reward

The reward for this MDP is simply the value of the portfolio, the sum of the value of every stock allocated at the weight of each stock:

$$r(s, a, s^{'}) = porfolio\_value \tag{2}$$

IV. Solution Approach

A. Data Sources

Stock Market data was accessed using the Yahoo Finance API. Initial trial runs of this project were attempted on the S&P 500, but 500 companies worth of data made data downloading and preprocessing time consuming and unwieldy. In addition, the size of the state space ballooned with increasing numbers of stocks, which made it impractical for iteration on our student level laptop hardware.

For this reason the Dow Jones Industrial Average 30 was used as our primary source of stocks. This index tracks the top 30 stocks in the United States, and we could compare our portfolio allocation of our stocks to the Dow 30 as a direct comparison of performance. With 30 stocks selected, the state space structure consisted of 34 rows and 30 columns at each time step, which was more manageable for our computing power.

B. Data Preprocessing

We first preprocess the raw stock market data by checking for missing data and applying feature engineering techniques to convert the data points into states. To achieve this, we use the FeatureEngineer class in Python, which takes in several parameters such as common stock market technical indicators.

In particular, we include two trend-following technical indicators, MACD and RSI, which are commonly used in practical trading to capture trends and momentum in stock prices. Additionally, we add a turbulence index to measure extreme fluctuations in asset prices, which helps to control risk in worst-case scenarios such as financial crises.

After preprocessing the data, we create a combination of all possible dates and tickers and merge it with the processed data to obtain a full data-set with all possible combinations of dates and tickers. We then sort the data by date and ticker, fill any missing values with 0, and select only the columns relevant for our portfolio allocation model, namely the date, ticker, and closing price.

This preprocessed and cleaned data-set is then used as input to train our deep reinforcement learning algorithms for portfolio allocation. Overall, this implementation ensures that the data is properly prepared and cleaned to maximize the effectiveness of our DRL algorithms in making optimal portfolio decisions.

C. Deep Reinforcement Learning Algorithms

We compare the effectiveness of 5 different reinforcement learning algorithms in this paper. None of these algorithms were implemented by the authors of this project, all were sourced from OpenAI's Stable Baselines3. A table containing key claims from [9] is shown below, contrasting the differences between our chosen algorithms: While these algorithms have different strengths and weaknesses, they are more similar than different, and can be considered "from the same family". Future work characterizing the performance more varieties of RL algorithms on portfolio allocation is needed.

TABLE I
DRL ALGORITHMS AND THEIR FEATURES, IMPROVEMENTS, AND ADVANTAGES [9]

Algorithm	Features and Improvements	Advantages
A2C	Advantage function, parallel gradients updating	Stable, cost-effective, faster and works better with large batch sizes
DDPG	Being deep Q-learning for continuous action spaces	Better at handling high-dimensional continuous action space
PPO	Clipped surrogate objective function	Improve stability, less variance, simple to implement
TD3	Clipped double Q-learning, delayed policy update, target policy smoothing	Improve DDPG performance
SAC	Entropy regularization, exploration-exploration trade-off	Improve stability

1) A2C: Advantage Actor Critic is an extension of the more basic Actor-Critic algorithm, which combines the benefits of both value-based and policy-based methods [10]. In A2C, the actor and critic are both represented as deep neural networks, which are trained simultaneously. The actor network is responsible for selecting actions, while the critic network evaluates the value of the current state. The term ”advantage” refers to the difference between the predicted value of the current state and the expected return from that state, which is used to adjust the actor’s policy.

A2C also has the advantage of being an on-policy algorithm, meaning that it updates the policy based on the most recent experience. This can be beneficial in situations where the environment is highly dynamic or changes frequently- such as the stock market.

2) DDPG: Deep Deterministic Policy Gradient (DDPG) is a powerful algorithm for solving continuous control tasks in reinforcement learning [8]. The DDPG algorithm is an extension of the Deep Q-Network (DQN) algorithm to continuous action spaces.

DDPG combines the advantages of policy-based and value-based methods by using a deterministic policy function to directly output the action given the state, and by maintaining a separate critic network that learns the Q-value function. The critic network is trained using a variant of the Q-learning algorithm, where the target Q-value is obtained by bootstrapping from the target actor network. The actor network is trained by maximizing the expected return using the policy gradient theorem. This allows the agent to learn both the optimal policy and the optimal Q-value function simultaneously.

To address the problem of exploration in continuous action spaces, DDPG employs an exploration policy that adds a Gaussian noise to the actions selected by the actor network. This noise is annealed over time to gradually decrease the exploration rate as the agent learns to exploit the environment.

DDPG has been shown to achieve state-of-the-art performance on a variety of continuous control tasks. In addition, DDPG is robust to hyper-parameter settings, making it easy to implement and tune.

3) PPO: Proximal Policy Optimization (PPO) is a state-of-the-art reinforcement learning algorithm for solving complex tasks with continuous action spaces [12]. The PPO algorithm is an extension of the Trust Region Policy Optimization (TRPO) algorithm that improves sample efficiency and stability.

PPO uses an actor-critic architecture, where the actor network outputs the policy and the critic network estimates the state-value function. The actor and critic networks are updated using gradient descent with a clipped surrogate objective, which prevents large policy updates that could lead to instability. This objective also includes a penalty term that discourages the policy from deviating too far from the previous policy.

The key innovation of PPO is the use of a clipped surrogate objective to improve sample efficiency and stability. The clipped surrogate objective limits the change in the policy between successive iterations, which prevents the agent from making large policy changes that could lead to catastrophic outcomes. This objective also reduces the variance of the gradient updates and improves convergence of the policy and value networks.

PPO also uses a mini-batch of experiences collected from the environment to update the policy and value networks, which improves sample efficiency and allows for more frequent updates.

PPO has been shown to achieve state-of-the-art performance on a variety of RL tasks, including Atari games and robotic manipulation. In addition, PPO is easy to implement and can be parallelized efficiently for faster training.

4) TD3: Twin Delayed Deep Deterministic (TD3) is a powerful reinforcement learning (RL) algorithm for solving continuous control tasks [2]. The TD3 algorithm is an extension of the Deep Deterministic Policy Gradient (DDPG) algorithm that improves stability and sample efficiency.

TD3 uses an actor-critic architecture, where the actor network outputs the policy and the critic network estimates the state-action value function. The key innovation of TD3 is the use of twin critics that estimate the state-action value function independently. This reduces the overestimation bias that can occur in single critic methods, improving the stability and performance of the algorithm.

TD3 also employs a target network that is periodically updated with the parameters of the main network. This improves the stability of the training by reducing the variance of the target value estimates.

To further improve stability and sample efficiency, TD3 uses two tricks during the policy update. First, it uses a clipped double-Q approach to compute the critic target values, which reduces the overestimation bias even further. Second, it delays the policy update relative to the critic update, which allows for better convergence and prevents the policy from chasing noisy updates from the critic.

TD3 has been shown to achieve state-of-the-art performance on a variety of continuous control tasks, including robotic manipulation and locomotion. In addition, TD3 is easy to implement and can be parallelized efficiently for faster training.

5) SAC: Soft Actor-Critic (SAC) is a powerful reinforcement learning algorithm for solving continuous control tasks [4].The SAC algorithm is an extension of the Deep Deterministic Policy Gradient (DDPG) algorithm that uses a soft value function and entropy regularization to improve performance and stability.

SAC uses an actor-critic architecture, where the actor network outputs the policy and the critic network estimates the state-value function. The critic network is trained using the Bellman equation, and the actor network is trained using the policy gradient theorem. However, unlike DDPG, SAC uses a soft value function, which maximizes both the expected return and the entropy of the policy. This encourages the agent to explore more and improves the robustness of the policy.

SAC also uses entropy regularization to further encourage exploration and improve stability. The entropy regularization term is added to the objective function to encourage the policy to be more stochastic and explore more widely. This regularization term is weighted by a temperature parameter, which controls the trade-off between exploration and exploitation.

To address the problem of exploration in continuous action spaces, SAC employs an exploration policy that adds a Gaussian noise to the actions selected by the actor network. This noise is annealed over time to gradually decrease the exploration rate as the agent learns to exploit the environment.

SAC has been shown to achieve state-of-the-art performance on a variety of continuous control tasks. In addition, SAC is robust to hyper-parameter settings, making it easy to implement and tune.

V. Results

The following results were based on a run of all five algorithms. The algorithms were trained on stock market data from January 1st 2000 to January 1st 2011, and then were allowed to trade from January 1st 2011 until 2022.

A. Reward Performance

The best way to characterize the success of these algorithms in terms of portfolio allocation is to simply look at the cumulative value of the portfolio over time. This is a sum of the value of each of the individual stocks, multiplied by the number of shares owned in the portfolio. The results after one year are shown below:

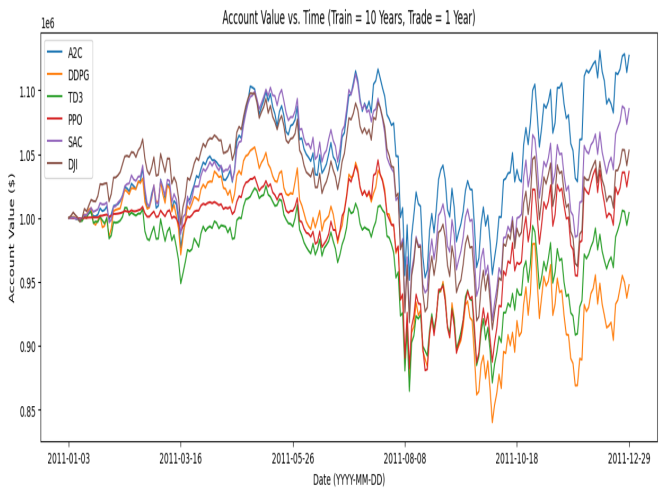

Fig. 1. Cumulative Portfolio Value vs Time, 1 years

In Fig.[1] above, the brown line represents the Dow Jones Industrial Average. This is an index that tracks the Dow 30, and is the baseline for this experiment. If our algorithms build a portfolio higher than the DJIA, we consider that a success, but if it is lower, then our algorithm is under-performing. At one year, the only policies performing better than the DJIA are the policies provided by A2C and SAC. This can be easier seen in Fig.[2]:

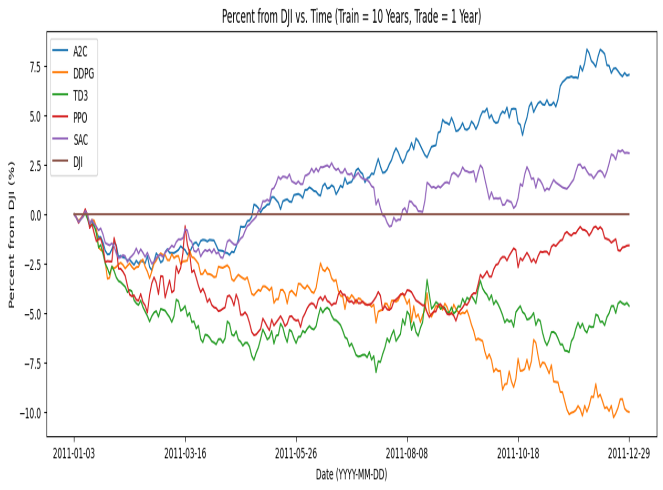

Fig. 2. Percent Difference between DJIA and DRL Algorithms, 1 year

Here, the brown line is the DJIA, normalized to 0, and the plot shows the percent difference between the DJIA and the DRL portfolios. After one year, A2C is 7.5% greater than the index, and SAC is 3% greater, but PPO, TD3, and DDPG are all much lower, including a dismal 10% less from DDPG.

Over 5 years, the policies all begin to outperform the DJIA. The following plot shows account value vs. time from 2011 to 2015:

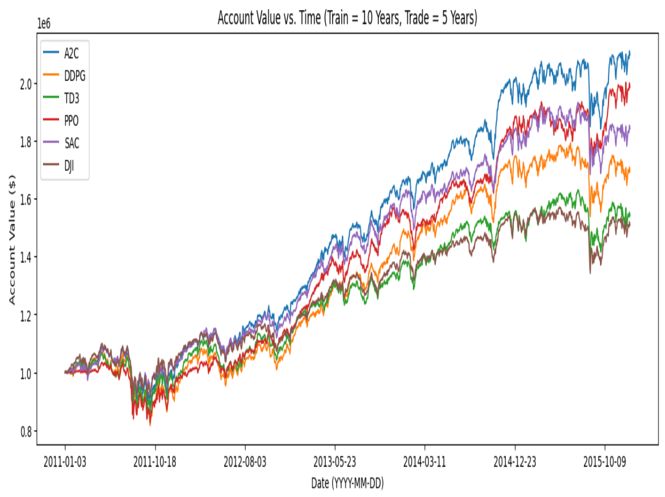

Fig. 3. Cumulative Portfolio Value vs Time, 5 years

Here, we can see that after 3 years, the policies produced by DRL have all surpassed the DJIA. Again, plotting on a normalized graph, this becomes even clearer:

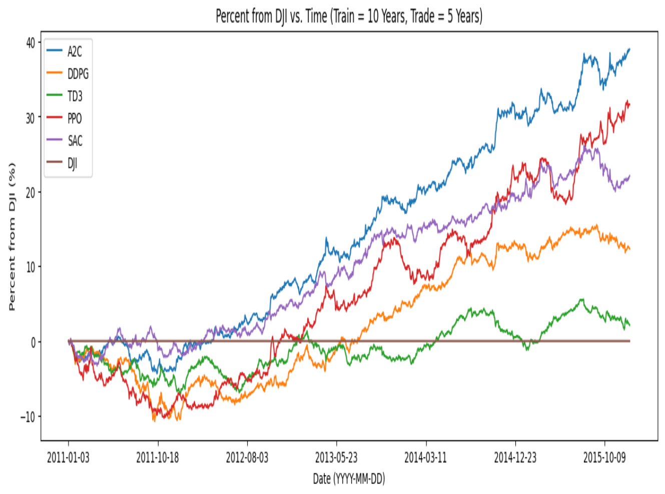

Fig. 4. Percent Difference between DJIA and DRL Algorithms, 5 years

In Fig.[4], we can see that the best performing algorithm, A2C, is a staggering 40% more valuable than the DJIA. The investor who trades using A2C would make significantly more money than the investor who is simply invested in the DJIA index, which is a price weighted average of the Dow 30 stocks. This trend continues into the 10 years of trading time frame:

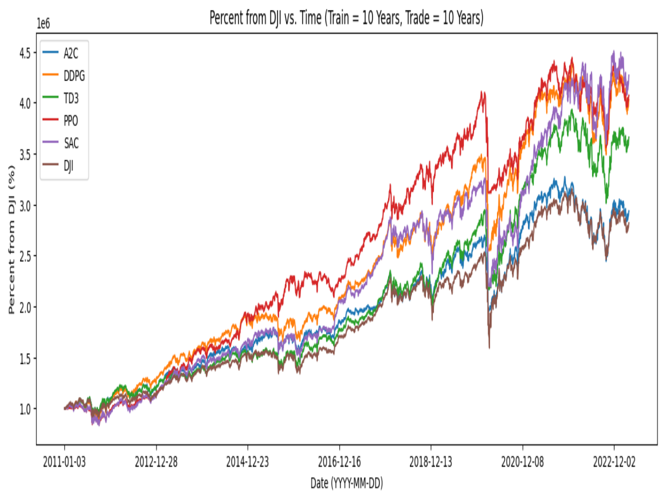

Fig. 5. Cumulative Portfolio Value vs Time, 10 years

After 10 years of trading, the highest performing DRL algorithm policy has amassed a cumulative portfolio value of 4.5 million USD from an initial investment of 1 million USD. The DJIA index only has a value less than 3 million USD. The following percentage plot shows the percent deviation from the DJIA reach as high as 90% during the COVID-19 induced stock market crash, and settling in at around 50% for the best performing DRL policies:

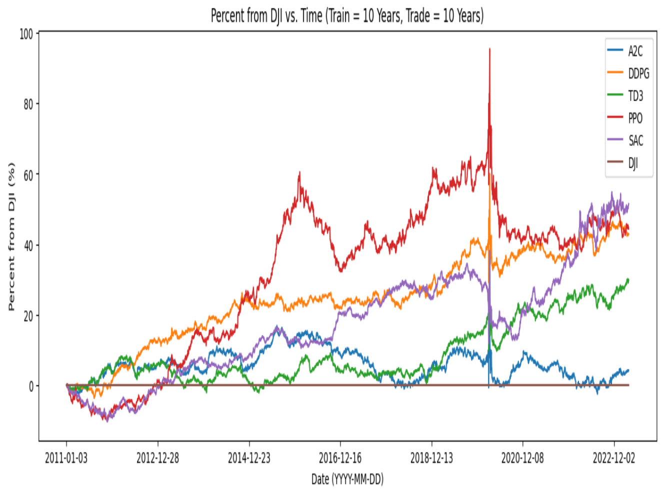

Fig. 6. Percent Difference between DJIA and DRL Algorithms, 10 years

An interesting and concerning feature of these results, however, is the deviation in performance strength across algorithms. You may notice that while A2c generated the strongest performing policy in Figures [1], [2], [3], and [4], but in figures [5] and [6], DDPG, SAC, and PPO produce the best performing policies. What is even more surprising, is that A2C, the champion of the first 5 years, now produces the worst policy, only slightly better than DJIA after 10 years of trading. The reason for this divergence is because these sets of figures were generated on two separate training runs. In the first run, A2C trained the best policy, and it appears that SAC trained the best policy on the second run. The question then arises, why were these policies so different if there were trained on the same data, with the same hyper-parameters, on the same hardware? This is one of the first issues with this approach, which will be discussed later in this section.

B. DRL Algorithms Training Time Comparison

When focusing on the practical side of Deep Reinforcement Learning, as we chose to do for this project, the time of training matters. For this reason, we decided to benchmark the DRL Algorithms Training time in the table below:

TABLE II
DRL Algorithm Training Times

	Training Time		
Algorithm	Long	Medium	Short
A2C	˜11 min	˜5 min	˜4 min
DDPG	˜42 min	˜36 min	˜34 min
PPO	˜11 min	˜4 min	˜3 min
TD3	˜40 min	˜35 min	˜33 min
SAC	˜42 min	˜36 min	˜34 min

The fastest algorithm tested in this project is the popular PPO, followed closely by A2C. The algorithm that took the longest was a tie between DDPG and SAC. TD3 is slightly faster, but in the same ballpark as DDPG and SAC. This makes a lot of sense, given that SAC and TD3 are both very similar to DDPG. Given the fastest run time, and consistency in performance between runs, PPO is a strong favorite for our group. Further, training times were gathered from 3 training time lengths, 10 years (long), 5 year (medium), and 1 years (short). It is seen that longer training periods lead to longer training times for the algorithms. Interestingly, there is not a large time increase from the short ot medium length training periods but, there is a larger increase when using the long training period. This may potentially signify that it would be very time/computationally intensive to train for very long durations of time.

C. Issues With this Approach

As is the case with lots of Artificial Intelligence, Machine Learning, and Reinforcement Learning projects, we have several issues with our current approach, namely repeatability, explainability, and scalability.

1) Repeatability: As we show above, different training runs produced vastly different policies, even when training on the same data-set with the same hyper-parameters. If we had more time and greater computing resources, we would like to train thousands of times, and compare the average policy return over many different learned policies for each algorithm, but unfortunately that is out of scope for this project.

2) Explainability: A challenge that exists at all levels of reinforcement learning is explainability. When the algorithms performed well, we did not know why, exactly, and when they performed poorly, we were equally lost for an explanation. The black box nature of the internal neural networks make it nearly impossible for us to parse what the algorithms are doing to learn the stock market, and that lends little to no practical action items for an investor trying to improve portfolio allocation strategy. Simply put, unless you can use these algorithms to directly automate trading for you, you are not gaining much insight on market dynamics that you can explain at the human level. Another challenge arises on the topic of risk, Since you cannot explain how the policy works, or how it was made, there is more risk in giving a DRL policy control of a portfolio compared to letting a human manage it. The stock market is already inherently risky, which can dissuade people from investing even if it is in their best interest. Adding the additional risk of an unpredictable autonomous portfolio manager may be too much risk for an individual or company to tolerate.

3) Scalability: We were slightly discouraged by only being able to use 30 stocks for this project- originally we attempted to learn the entire S&P 500, which was just too much for our computers to handle.

These methods are powerful, and offline training on large state spaces of thousands of stocks is certainly possible, but the training time required is simply not reasonable for an individual investor, which raises questions about fairness and equity.

An interesting future project would be to examine the necessity of all the indicators in the state space. Are they all necessary to predict the stock's viability? How much information can be removed while maintaining similar levels of performance? Here, we were looking at stock data on a day to day basis, but with enough speed improvements, this could be used as a day trading algorithm that could make many more trades in a smaller amount of time, extracting more value from the individual stock variance at a much higher rate.

VI. Conclusion

In conclusion, this paper presents a comparative study of 5 off-the-shelf deep reinforcement learning algorithms for portfolio allocation in the stock market. By using an OpenAI style gym environment and Yahoo financial data formatted as an MDP, the algorithms were able to outperform the Dow Jones Industrial Average after 5 and 10 years of trading. Key metrics such as cumulative returns, percent improvement over DJIA, and training time were used to evaluate the algorithms' performance. The experimental results suggest that each of the DRL algorithms is successful at varying levels, and that different training runs impact the quality of policies produced. However, PPO is identified as the most promising deep reinforcement learning algorithm for portfolio allocation in the stock market, with potential for practical implementation in the financial industry.

While many issues exist with Deep Reinforcement Learning, there is plenty of power and promise in the application of an MDP framework to the Stock Market, and value can be extracted from policies generated using DRL algorithms.

VII. Contributions and Release

Ryan and Cole worked in tandem on all aspects of this project. The authors grant permission for this report to be posted publicly.

Acknowledgment and Remarks

This project would not have been possible without the extensive tutorials and excellent documentation in the FinRL ecosystem.

Additionally, the authors would like to extend a heartfelt thank you to both Professor Zachary Sunberg and TA Jackson Wagner for an excellent semester of ASEN 5264 at the University of Colorado, Boulder.

References

[1] Incorporating markov decision process on genetic algorithms to formulate trading strategies for stock markets. *Applied Soft Computing*, 52:1143–1153, 2017.

[2] Nicholas Baard and Terence L. van Zyl. Twin-delayed deep deterministic policy gradient algorithm for portfolio selection. In *2022 IEEE Symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr)*, pages 1–8, 2022.

[3] Souradeep Chakraborty. Capturing financial markets to apply deep reinforcement learning, 2019.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[5] Taylan Kabbani and Ekrem Duman. Deep reinforcement learning approach for trading automation in the stock market. *IEEE Access*, 10:93564–93574, 2022.

[6] Kavitha Koppula, Babushri Srinivas Kedukodi, and Syam Prasad Kuncham. Markov frameworks and stock market decision making. *Soft Computing*, 24(21):16413–16424, Nov 2020.

[7] Jae Won Lee. Stock price prediction using reinforcement learning. In *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No.01TH8570)*, volume 1, pages 690–695 vol.1, 2001.

[8] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[9] Xiao-Yang Liu, Zhuoran Xiong, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading, 2022.

[10] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[11] IKER PEREZ, DAVID HODGE, and HUILING LE. Markov decision process algorithms for wealth allocation problems with defaultable bonds. *Advances in Applied Probability*, 48(2):392–405, 2016.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[13] Deneshkumar Venugopal, Senthamarai Kannan Kaliyaperumal, and Sonai Muthu Niraikulathan. Stock market trend prediction using hidden markov model. In Abdo Abou Jaoude, editor, *Forecasting in Mathematics*, chapter 5. IntechOpen, Rijeka, 2020.