Project Report: Reproducibility and Analysis of "Practical Deep Reinforcement Learning Approach for Stock Trading" and "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy"

Dixin Li[1*] **Siyuan Chen**[1*] **Weiyue Cai**[1*]
[1] McGill University
{dixin.li,siyuan.chen8,weiyue.cai}@mail.mcgill.ca

Abstract

In this project, we aim to explore the application of various deep reinforcement learning (DRL) algorithms in optimizing stock trading strategies, inspired by the methodologies described in "Practical Deep Reinforcement Learning Approach for Stock Trading" (1) and "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy" (2). Our primary goal is to enhance our practical skills in applying DRL algorithms rather than conducting deep theoretical research. To this end, we will leverage the Stable Baselines 3 (3) for implementation, without developing the algorithms from scratch.

1 Introduction

In our project, we aim to reproduce and extend the findings of two papers (1; 2) that explore the application of reinforcement learning algorithms to develop adaptive trading strategies for the stock market. The first article focuses on the Deep Deterministic Policy Gradient (DDPG) algorithm, while the second introduces an ensemble strategy that integrates three algorithms: Proximal Policy Optimization (PPO), Advantage Actor Critic (A2C), and DDPG. These methodologies are applied to trade 29 stocks (see Appendix section 6.1) from the Dow Jones index, leveraging the same dataset and parameter settings as those specified in the authors' original implementations [1]. In an effort to broaden the scope of this reproducibility study and explore the potential for enhanced trading strategies, we also experiment with two additional reinforcement learning algorithms: Twin Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC). Our objective is to extend the original findings and to compare these algorithms in the dynamic and complex environment of stock trading.

2 Related works

Recent applications of deep reinforcement learning in stock trading use either discrete or continuous state and action spaces. These applications employ one of the following learning methods: critic-only, actor-only, or actor-critic approaches (4; 2). The critic-only learning approach, which is widely used, tackles problems with discrete action spaces by employing methods such as Deep Q-learning (DQN) and its enhancements (2). For instance, the trading environment randomly selects a start date to sample the stock price time series of a single ticker, simulating a trading period. In each state, the agent receives historical returns for multiple time lags and various technical indicators. The agent has three possible actions to choose from: buy, hold, or short (12). The actor-only approach involves

[1] https://github.com/AI4Finance-Foundation/FinRL

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

the agent directly learning the optimal policy itself. In this method, the neural network is trained to learn the policy, which is a probability distribution representing the strategy for a given state (2). Our project builds on research that employs actor-critic approaches for trading with a large stock portfolio. The concept involves updating the actor network, which represents the policy, and the critic network, which represents the value function, simultaneously. The critic network estimates the value function, and the actor network updates the policy probability distribution, guided by the critic's assessments using policy gradients. Over time, the actor becomes more adept at selecting advantageous actions, while the critic improves its accuracy in assessing these actions (1; 2).

3 Methodology

3.1 Trading environment and Dataset

In our experimental setup, the foundation for training a trading agent is the construction of a trading environment, facilitated by code provided by the authors. Given that the environment is constructed using the OpenAI Gym, the step function usually contains the core logic of the environment. It receives an action as input, calculates the resulting state of the environment upon applying that action, and returns a 4-tuple consisting of (observation, reward, done, info). In essence, the agent's goal is to learn a policy, which is a strategy for choosing actions based on states, that maximizes its total reward over time. From an implementation standpoint, the model's predict function takes an observation (a state) as input and outputs an action determined by the policy. Subsequently, the environment's step function is called with this action, and it returns the subsequent state. In our approach to modeling the trading of multiple stocks within a continuous action space, we incorporate a comprehensive state space representation to facilitate the decision-making process of our trading agent.

Action Space: The action space for trading a portfolio of 29 stocks is represented by a 29-dimensional vector. Each element of this vector corresponds to a value in $\{-k, ..., -1, 0, 1, ..., k\}$ where k and $-k$ presents the number of shares we can buy and sell. To accommodate the requirements of reinforcement learning algorithms like A2C and PPO, which apply policies based on a Gaussian distribution, this action space is normalized to the range [-1, 1]. The action value, which falls within the normalized range of [-1, 1] , is scaled by h_{max} (a parameter defined in the trading environment) to calculate the exact number of shares to transact. (2)

State Space: Our portfolio consists of 29 distinct stocks, with the state space captured by a 291-dimensional vector divided into seven key segments. This multidimensional vector is structured as follows: 1. **Portfolio Value**: The first element of the vector represents the total value of the portfolio in the previous state. 2. **Stock Close Prices**: The subsequent 29 elements of the vector correspond to the closing prices of each of the 29 stocks from the previous trading session. 3. **Stock Shares**: The next 29 elements denote the quantity of shares the agent holds for each of the 29 stocks. 4. **Technical Indicators**: The remaining 232 elements are devoted to technical indicators [2], with eight indicators per stock for all 29 stocks, totaling 232 values. These indicators, detailed further in the Appendix section 6.2, provide a deep dive into the market's technical analysis. (2)

Dataset: Our study utilizes data spanning from January 1st, 2009, to September 30th, 2015, for training purposes, while the period from October 1st, 2015, to January 1st, 2021, serves as the timeframe for executing trades.

3.2 Model descriptions

3.2.1 Advantage Actor Critic (A2C)

A2C (7), standing for Advantage Actor-Critic, is an enhancement of the Actor-Critic method aiming to reduce variance and enhance learning stability. A2C introduces the Advantage function $A(s, a)$, which measures the relative benefit of taking a particular action over the average action. The policy gradient, incorporating the advantage function, is represented by:

$$\nabla_\theta J_\theta(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)\right] (2) \quad (1)$$

[2] `https://pypi.org/project/stockstats/`

The Advantage function is defined as the difference between the Q-value for a state-action pair and the V-value for the state represents the relative value of taking action a in state s compared to the average action value at that state.

3.2.2 Deep Deterministic Policy Gradient (DDPG)

DDPG (8), standing for Deep Deterministic Policy Gradient, is an advanced reinforcement learning algorithm designed for environments with continuous action spaces, where it elegantly circumvents the computational difficulties associated with traditional Q-learning.

- The **parameterized actor function** $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action. The actor is updated using policy gradient methods designed for continuous action spaces.
- The **critic function** $Q(s, a|\theta^Q)$ is used to approximate the value of state-action pairs, similar to Q-learning but tailored for continuous domains.

The policy gradient for the actor is computed as:

$$\begin{aligned} \nabla_{\theta_\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)} \right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t} \right] (8) \end{aligned} \tag{2}$$

3.2.3 Proximal Policy Optimization (PPO)

PPO (9) is designed to regulate the updates of the policy gradient, guaranteeing that updates do not significantly diverge from the preceding policy (2). The clipped surrogate objective function of PPO (9):

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{3}$$

$r_t(\theta)$ is the policy ratio, that is, the ratio of the new policy $\pi_\theta(a_t|s_t)$ to the old policy $\pi_{\theta_{\text{old}}}(a_t|s_t)$: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. $\hat{A}_t$ is the advantage estimate at time t, used to assess the relative benefit of taking action a_t in state s_t compared to the average. $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ limits the range of variation of the policy ratio $r_t(\theta)$, ensuring that the policy update steps are not too large. The update of policy parameters involves deriving the objective function $L^{\text{CLIP}}(\theta)$ with respect to the policy parameters θ.

3.2.4 Ensemble Strategy

The authors implement an ensemble approach that dynamically selects the top-performing agent from among PPO, A2C, and DDPG, based on their Sharpe ratio performance. The ensemble process is described as follows: Firstly, we employ a training approach where our agents are retrained over a period of n months. Secondly, following the training phase, we evaluate each of the three agents over a subsequent validation period to pick the best performing agent with the highest Sharpe ratio. Lastly, the agent demonstrating the highest performance is then used to predict and trade for the trading period for each iteration (2).

Due to space constraints, further details on Twin Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC) can be found in the sections 6.4.2 and 6.4.3 of Appendix.

4 Results

In the configuration of experimental settings for the hyperparameters of DRL agents and the parameters of the trading strategy, we drew inspiration from the hyperparameters exemplified by the authors within the FinRL library [3] [4]. We employed varying turbulence thresholds and risk indicators to execute experiments utilizing two trading strategies.

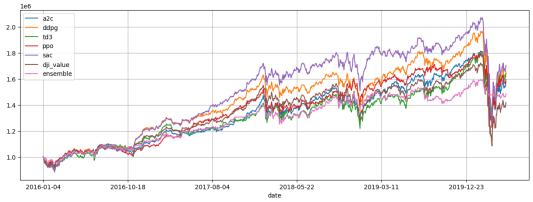
[3] https://github.com/AI4Finance-Foundation/FinRL/blob/master/examples/Stock_NeurIPS2018_SB3.ipynb

[4] https://github.com/AI4Finance-Foundation/FinRL/blob/master/examples/FinRL_Ensemble_StockTrading_ICAIF_2020.ipynb

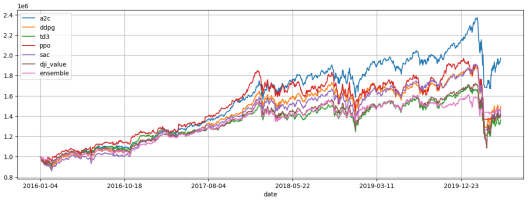
Strategy 1: To ensure uniform environments for both methods, we adopt the default settings. This includes using the default risk indicator, which is turbulence by default, and a default turbulence threshold of "None" for both training and trading environments.

Strategy 2: We use the same settings outlined in the notebook supplied by the authors. For both the single-agent and ensemble methods, we maintain default value for turbulence thresholds and risk indicators for training purposes. In the single-agent strategy, we employ "VIX" [5] as the risk indicator and set the turbulence threshold at 70 for trading. Conversely, for the ensemble strategy, we keep the default value for both turbulence thresholds and risk indicators.

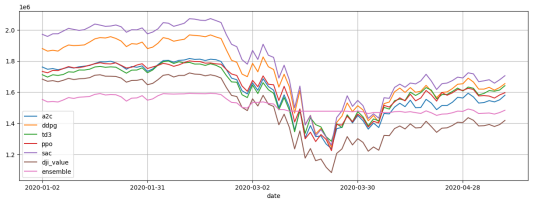
The figures labeled "Strategy 1" and "Strategy 2" illustrate the changes in portfolio value from January 4, 2016, to August 5, 2020, beginning with an initial amount of $100,000 for different algorithms and the benchmark, Dow Jones Industrial Average (DJI). These results, generated using an adapted implementation, reveal that the SAC and A2C strategies yield the highest final portfolio values. However, the "Strategy 1 stock crash" figure highlights the ensemble strategy as the most resilient during the stock market downturn. Additionally, we replicated the experiments using notebooks provided by the authors. These results confirmed that A2C offers the best performance in terms of final portfolio value, while the ensemble strategy consistently demonstrates robustness during market crashes, as indicated by the brown line in the first quarter of 2020.



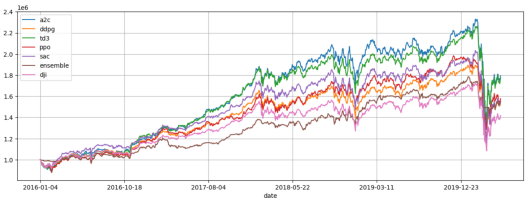
(a) Strategy 1



(b) Strategy 2



(a) Strategy 1 stock crash



(b) Strategy 1 using author's notebooks

5 Discussions

In the previous section of our study, we ran into a few challenges with our experiments that are worth discussing. Firstly, the results exhibited non-deterministic behavior even with exactly the same setting detailed in Appendix 6.5.1. This variability is a characteristic feature of reinforcement learning experiments due to their stochastic nature. Typically, to mitigate such randomness, multiple runs (e.g., 10) are conducted, and the average of these runs is used to plot the final results. Unfortunately, this method was not applied in our project due to the real-time nature of daily stock trading, for which the authors did not provide a methodology to achieve consistently stable results. Secondly, we observed that the outcomes heavily depend on the choice of training data and the specifics of the trading strategy implemented, as further discussed in Appendix 6.5.2. This dependency highlights the importance of selecting appropriate and representative datasets and trading strategies to ensure the generalizability of the findings in practical applications. Due to space constraints, please see Section 6.5.3 in the Appendix for the conclusion.

[5]The CBOE Volatility Index (VIX) is a real-time measure reflecting the anticipated volatility of the S&P 500 Index (SPX) over the near term. It calculates this by evaluating the prices of short-term SPX index options, thereby providing a 30-day forecast of volatility.

6 Appendix

6.1 29 stocks from the Dow Jones index

Apple, Amgen, American Express Company, Boeing Company, Caterpillar, Salesforce, Cisco Systems, Chevron Corporation, Walt Disney Compan, Goldman Sachs Group, Home Depot, Honeywell International, International Business Machines, Intel Corporation, Johnson & Johnson, JPMorgan Chase & Co, Coca-Cola Company, McDonald's Corporation,3M Company, Merck & Co., Microsoft, NIKE, Procter & Gamble, The Travelers Companies, UnitedHealth Group Incorporated, Visa, Verizon Communications, Walgreens Boots Alliance, Walmart.

6.2 8 Technical Indicators

1. **Moving average convergence/divergence (macd)**: a tool that shows the difference between two exponential moving averages (EMAs) of a stock's price to help identify trend direction and momentum. [6].
2. **Bollinger Bands (boll_ub, boll_lb)**: a set of three lines plotted on a price chart that measure the volatility of an asset. The upper and lower bands help identify potential overbought (when price is too high) and oversold (when price is too low) levels, indicating possible reversal points in the market. [7].
3. **Relative Strength Index (rsi_30)**: "RSI measures the speed and magnitude of a security's recent price changes to evaluate overvalued or undervalued conditions in the price of that security. " [8]
4. **Commodity Channel Index (cci_30)**: "a technical indicator that measures the difference between the current price and the historical average price. When the CCI is above zero, it indicates the price is above the historic average. Conversely, when the CCI is below zero, the price is below the historic average. " [9]
5. **Directional Index (dx_30)**: "indicates whether an asset is trending by comparing highs and lows over time." [10]
6. **Simple Moving Average (close_30_sma, close_60_sma)**: calculates the average of a selected range of prices, usually closing prices, by the number of periods in that range. [11]

6.3 Training and trading process

In the case of the trading strategy that employs a single agent, this training dataset is treated as in-sample data, enabling the agents to learn and subsequently simulate trading actions throughout the designated trading period. For the ensemble strategy, we introduce two parameters to guide the trading process: the rebalance window and the validation window, both set to a default duration of 63 days. The trading activities commence from October 1st, 2015, plus the duration of the validation window, marking the initial start date for trading. In each iteration, the agent is trained on data from January 1st, 2009, up to the current iteration's trading start date, denoted as S, minus the validation and rebalance windows. For validation purposes, to determine the most effective method among PPO, A2C, and DDPG, the agent utilizes data from S minus the validation window and rebalance window up to S minus the rebalance window. Following this selection process, the optimal method is then applied to conduct trades from S minus the rebalance window to S, with the outcomes of these trades being recorded. (2)

6.4 DRL

6.4.1 PPO

The core of the PPO algorithm is to update policy parameters by optimizing a specific objective function, enabling the agent to achieve higher returns in the environment. There are two main variants of the PPO algorithm: PPO-Clip and PPO-Penalty, which use different methods to constrain the extent of policy updates to ensure the stability of the learning process. In practical applications, the PPO

[6] https://www.investopedia.com/terms/m/macd.asp
[7] https://openreview.net/pdf/bc65b9f49bc68ebc7aea07f696b803a08ba406c5.pdf
[8] https://www.investopedia.com/terms/r/rsi.asp
[9] https://www.investopedia.com/terms/r/rsi.asp
[10] https://www.investopedia.com/terms/d/dmi.asp
[11] https://www.investopedia.com/terms/s/sma.asp

algorithm typically uses the PPO-Clip variant because it simplifies the adjustment of hyperparameters and has shown good performance in multiple tasks. PPO-Clip employs a technique called "clipping" to limit the magnitude of policy updates. Given a small positive number ϵ (for example, 0.1 or 0.2), The clipped surrogate objective function of PPO (9):

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{4}$$

where:

- $r_t(\theta)$ is the policy ratio, that is, the ratio of the new policy $\pi_\theta(a_t|s_t)$ to the old policy $\pi_{\theta_{\text{old}}}(a_t|s_t)$: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
- $\hat{A}_t$ is the advantage estimate at time t, used to assess the relative benefit of taking action a_t in state s_t compared to the average.
- $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ limits the range of variation of the policy ratio $r_t(\theta)$, ensuring that the policy update steps are not too large.

PPO-Penalty, due to its need for adjusting more hyperparameters (including β and the target KL divergence), may not be as intuitive and easy to use in practical applications as PPO-Clip. However, PPO-Penalty provides a more flexible framework for controlling the granularity of policy updates, which may bring additional advantages for certain specific tasks and environments. PPO-Penalty constrains policy updates by adding a penalty term for policy variation. Its objective function includes a squared term regarding policy variation:

$$L^{\text{PENALTY}}(\theta) = \hat{\mathbb{E}}_t \left[r_t(\theta)\hat{A}_t - \beta \cdot KL[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right]$$

where:

- $KL[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]$ represents the KL divergence (Kullback-Leibler divergence) between the old policy $\pi_{\theta_{\text{old}}}$ and the new policy π_θ, used to measure the change before and after the policy update.
- β is a positive scaling factor used to adjust the weight of the KL divergence term.

6.4.2 TD3

The TD3 algorithm (10) maintains two Critic networks, Q_{θ_1} and Q_{θ_2}, along with their target networks Q'_{θ_1} and Q'_{θ_2}. The update of the Critic network is achieved by minimizing the following loss function:

$$L(\theta_i) = \frac{1}{N} \sum_j \left[Q_{\theta_i}(s_j, a_j) - y_j \right]^2, \quad i = 1, 2 \tag{5}$$

where y_j is the target Q-value, calculated as

$$y_j = r_j + \gamma \min_{i=1,2} Q'_{\theta_i}(s_{j+1}, \tilde{a}) \tag{6}$$

$\tilde{a} = \pi'_\phi(s_{j+1}) + \epsilon$, and ϵ is the noise added to the target policy action for target policy smoothing, usually limited within a certain range (e.g., $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)$), γ is the discount factor. The update of the Actor network is accomplished by maximizing the expected return estimated by the Critic network(6). In TD3, the Actor network uses only one Critic network (usually the smaller of the two) to estimate this expected return, with the policy gradient formula being:

$$\nabla_\phi J(\phi) = \frac{1}{N} \sum_j \nabla_a Q_{\theta_1}(s, a)|_{s=s_j, a=\pi_\phi(s_j)} \nabla_\phi \pi_\phi(s)|_{s_j} \tag{7}$$

Here, J refers to the objective function that the actor network aims to maximize.

$$\nabla_a Q_{\theta_1}(s, a)|_{s=s_j, a=\pi_\phi(s_j)} \tag{8}$$

represents taking the action $a = \pi_\phi(s)$ generated by the Actor network and inputting it into the Critic network Q_{θ_1}, and calculating the gradient of the output Q-value. Then, this gradient is used to update the parameters ϕ of the Actor network, thereby enabling the generated actions to achieve a higher expected return.

6.4.3 SAC

The Soft Actor-Critic (SAC) (11) is a deep reinforcement learning algorithm for continuous action spaces that improves exploration efficiency by maximizing an objective function that includes an entropy term. The goal is to maximize both the expected return and the entropy of the policy. Below are some key formulas in the SAC algorithm:

The objective of the value function $V_\psi(s)$ is to minimize the following loss function:

$$L(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi} \left[Q_\theta(s_t, a_t) - \log \pi(a_t|s_t) \right] \right)^2 \right] \tag{9}$$

where $\mathcal{D}$ represents the experience replay buffer, π is the policy function, and $Q_\theta(s_t, a_t)$ is the action value function estimated by the Critic network.

The soft Q-function $Q_\theta(s, a)$ is updated by minimizing the following target loss:

$$L(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - (r_t + \gamma \mathbb{E}_{s_{t+1} \sim \mathcal{E}} \left[V_{\psi'}(s_{t+1}) \right]) \right)^2 \right] \tag{10}$$

The policy network is updated by maximizing the expected soft Q-value minus the log probability of the policy, aiming to balance the return and entropy to encourage exploration:

$$\pi^* = \arg\max_\pi \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi} \left[Q_\theta(s_t, a_t) - \alpha \log \pi(a_t|s_t) \right] \tag{11}$$

where α is the entropy regularization coefficient, controlling the importance of policy entropy in the objective function.

The SAC algorithm adaptively adjusts the entropy regularization coefficient α to meet a specific entropy target, by minimizing the following loss function:

$$L(\alpha) = \mathbb{E}_{a_t \sim \pi} \left[-\alpha \log \pi(a_t|s_t) - \alpha \overline{\mathcal{H}} \right] \tag{12}$$

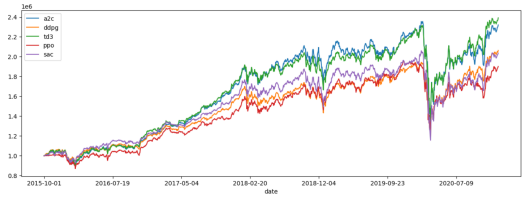
$\overline{\mathcal{H}}$ is the desired policy entropy.

Through these formulas, the SAC algorithm not only efficiently learns policies in continuous action spaces but also promotes better exploration through entropy regularization, enabling efficient learning in complex environments.

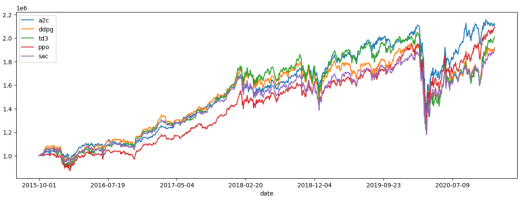
6.5 Results

6.5.1 Non-deterministic outcomes

The figures below represent individual runs for both single agent and ensemble strategies, utilizing data from January 1, 2009, to September 30, 2015. These strategies were then applied in trading scenarios from October 1, 2015, through May 8, 2020. This time frame allows us to analyze the effectiveness of each strategy over different market conditions and periods.

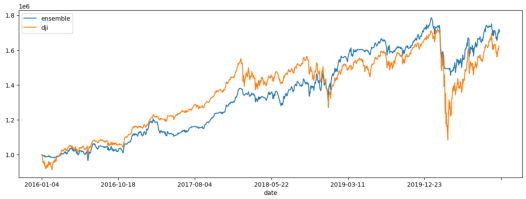


(a) first run: training from 2009-1-1 to 2015-9-30, trading from 2015-10-1 to 2020-5-8



(b) second run: training from 2009-1-1 to 2015-9-30, trading from 2015-10-1 to 2020-5-8

Figure 1: 6.5.1 Single DLR strategy



(a) first run: training from 2009-1-1 to 2015-9-30, trading from 2015-10-1 to 2020-5-8

(b) second run: training from 2009-1-1 to 2015-9-30, trading from 2015-10-1 to 2020-5-8

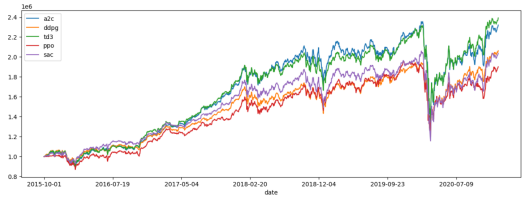
Figure 2: 6.5.1 Ensemble strategy

6.5.2 Impact of Training Data on Results

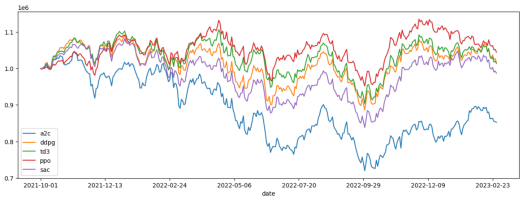
The figures below illustrate the performance of both single agent and ensemble strategies across two distinct periods:

First Period: Utilizing historical data from January 1, 2009, to September 30, 2015, these strategies were subsequently applied in live trading scenarios from October 1, 2015, to May 8, 2020.

Second Period: Drawing on more recent data, from January 1, 2010, to October 1, 2021, the strategies were then tested in trading scenarios running from October 1, 2021, to March 1, 2023.



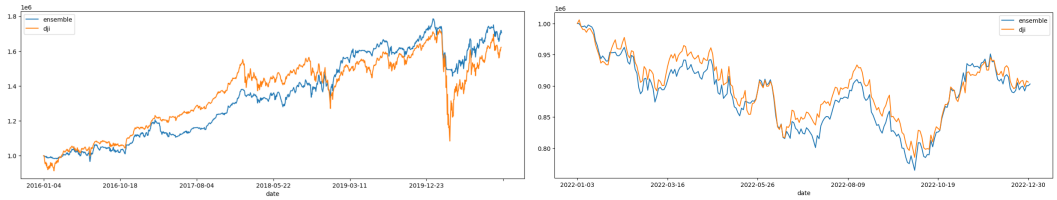
(a) training from 2009-1-1 to 2015-9-30, trading from 2015-10-1 to 2020-5-8



(b) training from 2010-01-01 to 2021-10-01, trading from 2021-10-01 to 2023-03-01

Figure 3: 6.5.2 Single DLR strategy

From the graphs, it's evident that the trading results are significantly influenced by the choice of training data. Given that the trading strategy remains constant, it proves challenging to adapt



(a) training from 2009-1-1 to 2015-9-30, trading from 2015-10-1 to 2020-5-8

(b) training from 2010-01-01 to 2021-10-01, trading from 2021-10-01 to 2023-03-01

Figure 4: 6.5.2 Ensemble strategy

it to varying historical stock time series effectively. This limitation highlights a potential area for improvement: developing a more flexible approach that allows for adapting different trading strategies to suit diverse market scenarios. By addressing this, we could enhance the strategy's responsiveness and potentially improve overall trading performance.

6.5.3 Conclusion

In conclusion, when applying DRL to stock trading, the design of the state space and action space is crucial, possibly even more so than the choice of algorithm itself. Currently, our approach simply concatenates various inputs such as the closing stock values, the number of shares owned by the portfolio, and various technical indicators to form the state. The actions, represented by a normalized vector, are scaled by a parameter to determine the exact number of shares to trade. This simplistic model may not sufficiently capture the complexities of a real-world trading environment. Enhancing the sophistication of the state and action representations could significantly improve the system's effectiveness and its ability to thrive in dynamic market conditions.

References

[1] Xiao-Yang Liu & Zhuoran Xiong & Shan Zhong & Hongyang Yang & Anwar Walid, 2018. "Practical Deep Reinforcement Learning Approach for Stock Trading" Papers 1811.07522, arXiv.org, revised Jul 2022. NeurIPS Workshop on Challenges and Opportunities for AI in Financial Services: the Impact of Fairness, Explainability, Accuracy, and Privacy, 2018. `https://arxiv.org/abs/1811.07522`

[2] Hongyang Yang & Xiao-Yang Liu & Shan Zhong & Anwar Walid, 2020. "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy" ICAIF '20: Proceedings of the First ACM International Conference on AI in Finance, October 2020. Article No.: 31 Pages 1–8. `https://doi.org/10.1145/3383455.3422540`

[3] Stable-Baselines3, `https://stable-baselines3.readthedocs.io/en/master/`.

[4] Thomas G. Fischer. Reinforcement learning in financial markets - a survey. FAU Discussion Papers in Economics 12/2018. Friedrich-Alexander University Erlangen-Nuremberg, Institute for Economics.

[5] Isabeau Prémont-Schwarz. *RL: Policy Gradient – Actor-Critic Algos*. 2024, p23-24.

[6] Bowei Sun, Minggang Song, Ang Li, Nan Zou, Pengfei Pan, Xi Lu, Quan Yang, Hengrui Zhang, Xiangyu Kong. *Multi-objective solution of optimal power flow based on TD3 deep reinforcement learning algorithm*, Sustainable Energy, Grids and Networks, Volume 34, June 2023, 101054. [Online]. Available: `https://www.sciencedirect.com/science/article/abs/pii/S2352467723000620`.

[7] Volodymyr Mnih, Adrià Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. The 33rd International Conference on Machine Learning (02 2016).

[8] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. International Conference on Learning Representations (ICLR) 2016 (09 2015).

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. arXiv:1707.06347 (07 2017).

[10] Scott Fujimoto, Herke van Hoof, David Meger. Addressing function approximation error in actor-critic methods. In: arXiv preprint arXiv:1802.09477 (2018).

[11] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. ICML 2018: 1856-1865.

[12] Jansen, Stefan. Machine Learning for Algorithmic Trading: Predictive Models to Extract Signals from Market and Alternative Data for Systematic Trading Strategies with Python.