RL Policy Ensemble for Stock Trading

Ananya Kulshrestha
ananya_k@mit.edu

David Chaudhari
davidc03@mit.edu

Abstract

This paper explores the application of an ensemble of deep Reinforcement Learning models to optimize stock trading strategies. We develop a composite model integrating distinct algorithms—DQN, DDPG, A2C, and PPO—to leverage their unique strengths in a unified trading strategy. Unlike traditional approaches, our ensemble method employs a novel voting system based on thresholding the converted sum of the actions from each model, to adjust the influence of each model based on its performance and cumulative return. This strategy aims to outperform a simple intuitive baseline, and also address the challenge of navigating a highly volatile financial market with a robust, adaptable framework. The hope is that by combining the models, the strengths of the various models can be utilized to protect against this volatility.

1 Problem Motivation

The motivation behind this research is twofold: Firstly, the financial market's non-linear nature requires sophisticated models that can capture and exploit patterns not readily apparent to human traders or traditional algorithms. Secondly, the need for robustness in trading algorithms is paramount, as they must perform well under a variety of market conditions. This has led to the exploration of ensemble strategies, where multiple models are combined to enhance decision-making robustness and capitalize on the strengths of individual models while mitigating their weaknesses.

2 Methodology

This paper proposes an innovative ensemble strategy that integrates multiple deep Reinforcement Learning models, including Deep Q-Networks (DQN) [2, 4], Deep Deterministic Policy Gradient (DDPG), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO) [1]. See 1 for the flow of our method.

For the ensemble, we explored two approaches:

1. **Max Voting:** Actions are selected based on a majority vote among the agents.
2. **Converted Majority Thresholding:** Each agent converts its action into a standardized form (-1, 0, +1 for sell, hold, buy). Then the sum of the converted actions is taken, and returns the action based on the defined range it fits in. The chosen action is defined as follows:

$$\text{action} = \begin{cases} \text{buy} & \text{sum} > 1 \\ \text{hold} & -1 \leq \text{sum} \leq 1 \\ \text{sell} & \text{sum} < -1 \end{cases}$$

2.1 The Setup

We employ a custom-built stock trading simulator based on OpenAI Gym, using Apple stock (AAPL) throughout the experiments. The state space includes closing price information, represented by the

6.484 Computational Sensorimotor Learning Final Project (Spring 2022).

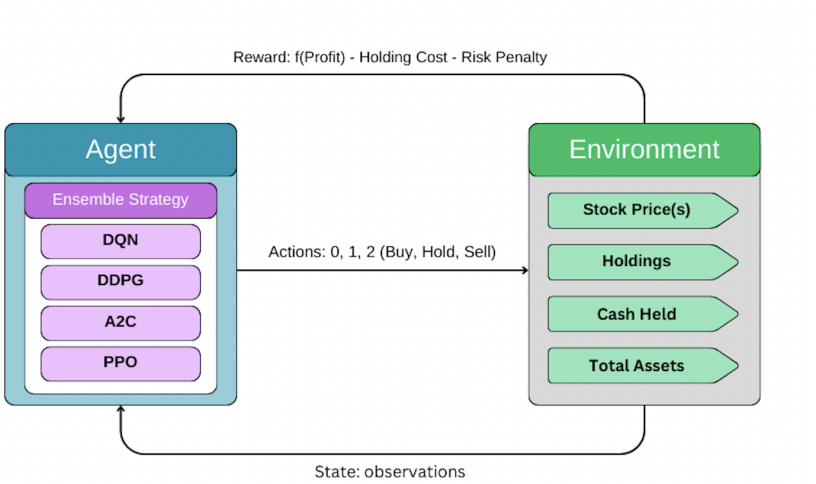

Figure 1: Methodology Diagram

current price and the next $n-1$ adjusted closing prices, where n is the window size. It also contains the number of holdings of the stock at the given time. The action space consists of buy, hold, and sell (0, 1, 2) actions for given stocks i.e. 3 action choices [3]. At a given step, you can only sell or buy up to one stock. Additionally, you can have negative holdings / negative cash as we begin with 0 holdings and \$0.

2.2 Data and Reward Design

We used Adjusted Closing Prices from Yahoo Finance as data for training and testing, to account for stock splits. However, in noticing that inflation was not factored in, we utilized CPIAUCNS data [5] to create a CPI Multiplier in terms of 2024 dollars and multiplied the Multiplier with the Adjusted Closing Prices to get new CPI Adjusted Closing Prices.

Reward was an aspect that was important to our experiments. Hence, we set up a generic reward structure to experiment with. It consists of:

- $+f(\text{profit})$
- $-\text{holding cost} = (\text{holding } \%) \cdot |\text{holdings}|$
- $-$risk penalty (calculated from total asset at a given step)

where profit is current price $-$ immediate future price for sell and immediate future price $-$ current price for buy. Our loss function for DQN was SmoothL1Loss while it was MSE for the other three agents.

2.3 Baseline Method

Initially, we were planning to use the DJIA index [1] as a baseline. However, due to our method only focusing on a singular stock, this index would not be a viable comparison. Hence our newly defined baseline uses a simpler strategy compared to the other agents. This simpler method checks the adjusted closing price of the past 4 days and the current day. If these prices over this period are decreasing, then it buys the stock. However, if the prices are increasing, then it sells the stock. The high level idea of using this strategy is that it is representative of a strategy that many traders use, relying on the momentum of the market over a period of time to buy or sell. After testing the baseline method on the stock data, we observed minimal gains throughout and that the net gain at the end of the testing window was $\sim$ \$0.

3 Experiments and Results

We conducted a series of experiments, changing the reward design and ensemble strategy. We trained our agents on data from January 1, 2010 to January 1, 2020 and testing them on data from January 1, 2021 to March 1, 2024. For training plots, we show the returns, while the testing plots show the total assets (cash + holdings · current price) on the given day.

Agent	γ (Gamma)	α (Learning Rate)	Batch Size	τ (Tau)	Memory Size	# of Episodes
DQN	0.99	0.005	32	0.005	2×10^5	100
PPO	0.99	0.002				25
A2C	0.99	0.001	2000			50
DDPG	0.98	0.005	64	0.005	2×10^5	100

Table 1: Hyperparameters per Agent

3.1 Initial Experiments

For our initial results, the parameters that we used are *holding cost* $= 0.0001 \cdot$ *initial price* $\cdot$ *# of holdings*, and *risk penalty* $=$ *total assets* $\cdot 10^6$. The $f(\text{profit})$ component was simply the profit itself. Running the experiments, we found that our calculations for the reward were scaled too high and the agents were not learning well.

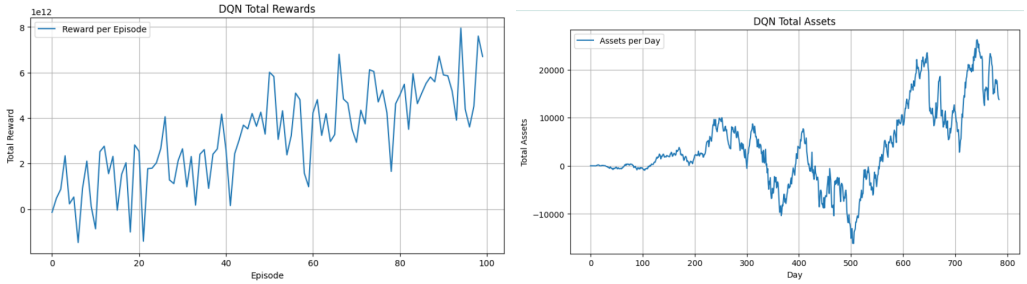

Figure 2: Training (left) and Testing (right) Results for DQN including Risk Penalty

Considering these results, we then created a second batch of experiments, where we kept the concept of PNL that we ran the initial experiments, however the *holding cost* $= 0.002 \cdot$ *initial price* $\cdot$ *# of holdings*. The risk penalty was also removed from the calculations. We then got the following results when Max Voting Ensemble was implemented.

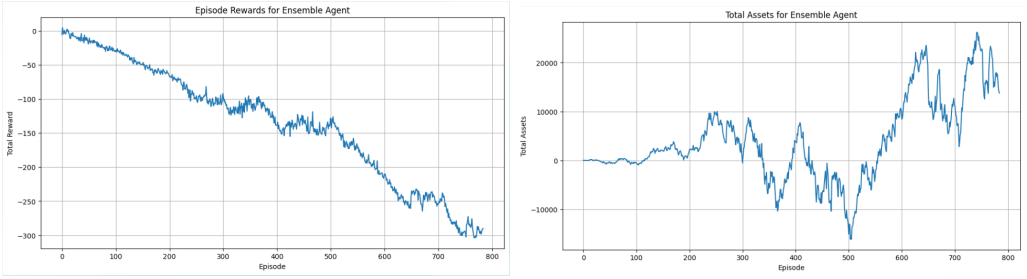

Figure 3: Training (left) and Testing (right) Results for Ensemble Without Risk Penalty

From our results, we then realized that using a Max Voting Ensemble wasn't the right approach, since it led to a abundance of buying, which led it to major losses at some points. Additionally, our reward design was lacking the risk penalty factor.

3.2 Change in Reward Design

From our initial experiments, we then changed our reward function to achieve better results. We then changed our paradigm to penalize losses more than we did previously. We did this by changing our reward function to $f(x) = \frac{x \cdot e^x + 0.01x}{e^x}$, in order to penalize negative total assets heavily. We

also changed the holding percentage to be 0.0025. We also changed the risk penalty, where we only considered the current price and introduced the concept of *drawdown* to make our agents more stable, which is calculated by $\frac{\text{peak total asset} - \text{total asset}}{\text{peak total asset}}$. We multiplied the drawdown by a risk $\%$ of 0.0075 to obtain the risk penalty term of our reward. This risk penalty avoided explosions, which better simulated the volatility of the stock market.

3.3 Final Results

After changing the ensemble to Converted Majority Thresholding and updating our reward design, we found that our ensemble method outperformed the baseline method.

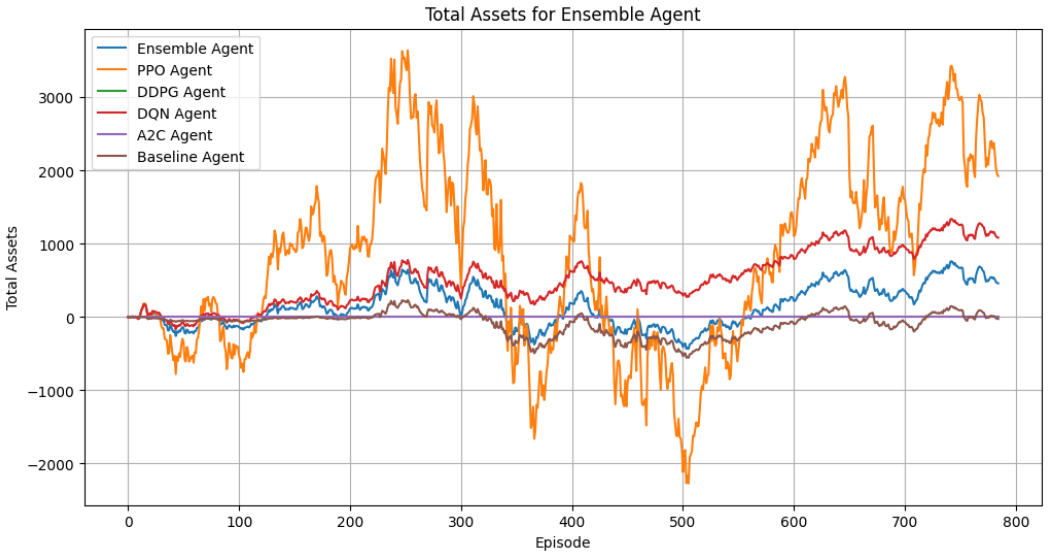

Figure 4: Testing Results for All Agents and Ensemble with Adjusted Parameters

4 Conclusions

In conclusion, as mentioned before, we noticed that the baseline is outperformed by the ensemble as well as DQN consistently. We notice that PPO is very volatile and not a reliable performer. This may be due to a lack of parameter tuning and the agent being presented in a complex environment.

We noted the importance of experimentation as well as how reward design and small parts of the methodology would lead to varied outcomes. No matter the experiment, A2C did not learn well and ended up holding throughout testing. This did indeed bring the ensemble down a bit, but the consistency of DQN led to the ensemble performing well. We chose a static ensemble of the trained agents to avoid having computationally intensive runs. Perhaps, with these results, future experiments could include training the weights for each agent's actions and deciding the optimal action from that.

The biggest challenge in the entire process was the volatility of the market [1]. As a result, our returns are also quite volatile at times. Our attempt at combating this was by including risk penalty [3]. However, it may have been useful to add a random turbulence factor [1], expand the action space, or perhaps include more information in the observation space in order to understand the stock trends. For simplicity, we also allowed negative cash and holdings in our experiments, which may not be possible in the real world. Representing the real stock environment is truly difficult, and we simulated it the best we could to obtain results.

5 Final Notes

5.1 Contributions

- Ananya wrote the DQN and DDPG agents, while David wrote A2C and PPO.
- Ananya preprocessed the data (from Yahoo Finance library) and adjusted it to inflation using CPIAUCNS [5].
- David wrote the baseline algorithm.
- Ananya set up a custom environment (based on OpenAI gym library).
- We made the two kinds of ensemble agents. David took care of Max Voting, while Ananya implemented further thresholding to create the converted majority thresholding mstrategy.
- Ananya ran preliminary tests and had to fix the agents and parameters (for the midterm report)
- David ran experiments with max voting and some changes in reward.
- Ananya ran experiments with new reward design and ensemble strategy
- David and Ananya made the presentation for the project and recorded it
- David and Ananya also wrote the final paper together

5.2 Other Resources

Other libraries/resources we used were matplotlib, pandas, numpy, torch, and random. We were also inspired by [1] in our work.

5.3 Acknowledgements

We would like to acknowledge 6.8200 Staff in helping us brainstorm through the project and a special appreciation for Professor Agarwal for teaching us the techniques we needed to implement this project.

References

[1] H. Yang, X. Liu, S. Zhong, A. Walid (2020) *Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy*. Retrieved from `https://openfin.engineering.columbia.edu/sites/default/files/content/publications/ensemble.pdf`

[2] S. Sarkar (2023). *Quantitative Trading using Deep Q Learning*. Retrieved from `https://arxiv.org/pdf/2304.06037.pdf`

[3] C. Huang (2018). *Financial Trading as a Game: A Deep Reinforcement Learning Approach*. Retrieved from `https://arxiv.org/pdf/1807.02787.pdf`

[4] L. K. Felizardo, F. C. L. Paiva, A. H. R. Costa, E. Del-Moral-Hernandez (2022). *Reinforcement Learning Applied to Trading Systems: A Survey*. Retrieved from `https://arxiv.org/pdf/2212.06064.pdf`

[5] *Consumer Price Index for All Urban Consumers: All Items in U.S. City.* Average Federal Reserve of St. Louis. Retrieved from `https://fred.stlouisfed.org/series/CPIAUCNS`