Exploration and Agent Improvement under Near-Real Market Environments Simulation Using Deep Reinforcement Learning

Yichen Li
EECS
University of Michigan
Ann Arbor, US
liyichen@umich.edu

Yilin Li
EECS
University of Michigan
Ann Arbor, US
yilinliz@umich.edu

Jiaming Zeng
EECS
University of Michigan
Ann Arbor, US
zjiaming@umich.edu

I. Problem Description

We aim to solve the stock trading problem in extreme conditions. We will simulate the market trading environments in the real world and apply several popular deep reinforcement learning (DRL) algorithms to trade multiple stocks. Finally, we will analyze various factors that affect the trading environments and evaluate the profitability of agents with different algorithms in extreme trading environments.

A. Motivation

Deep reinforcement learning has shown great possibilities in simulating real-world market environments with well-designed agents competing with each other [2], [3]. However, due to the high complexity and dynamic nature of real-world markets, current models trained using historical data might not be able to faithfully simulate real-world trading scenarios. We shall focus on the adversarial behavior among market investors to better reflect the situation that the users may encounter in the real-world market and better help them understand or simulate an optimal option they may take for different situations.

B. Challenges

Stock trading is an extremely challenging task due to many factors. Firstly, real-world markets are very complex and dynamic. The market trading environments are very hard to simulate. Although we can fully observe the trading market and directly obtain the opening and closing prices of each stock, it is not easy to obtain the characteristics behind the prices. We don't know what causes the increases or decreases in the prices. Besides, the state space is very large, we can buy or sell any shares of stocks. The size of the data we need to analyze is huge too. In addition, the model we created can easily get overfitted. There is so much a simulation-to-reality gap that degrades the performance of DRL strategies in the real market.

C. Contribution

In this paper, we will do an investigation into some of the common scenarios and extreme situations that may occur in the real-world market. Then we will perform simplified simulations for these situations to see how different agents would develop their behaviors and strategies. After that, we will improve some of the logic or models that agents may use to achieve a better result under specific circumstances. Finally, we will give our conclusion regarding which agent is a better choice under specific circumstances.

D. Task Allocations

- **Yichen Li:** environment analysis, experiment implementation
- **Yilin Li:** baseline method implementation and testing
- **Jiaming Zeng:** ensemble methods implementation

II. Related Work

A. Existing Research

To address our interest, we investigate some of the existing data-driven reinforcement learning (RL) methods and related tools that can help with the implementation.

First of all, to acquire a fundamental understanding, and for a general reference, the paper Deep Reinforcement Learning in Quantitative Algorithmic Trading [1] contains a basic introduction to reinforcement learning usage in the quantitative domain. Also, a popular repository (OpenAI Gym) [5] provides standardized

environments for a collection of benchmark problems that expose a common interface, which is widely supported by many languages and libraries. Three trading environments, TradingEnv, ForexEnv, and StocksEnv, are included to support the Stock markets. And as a really up-to-day package, FinRL [6] provides a full pipeline for financial reinforcement learning which help perform the environment build up and trading. It contains multiple market environments, i.e., stock trading, portfolio allocation, with two extra data sources, i.e., Yahoo Finance and WRDS. And at last, the repository ElegentRL [7] contains agent implementation sample that is the simulation of the real-world stock trading market environment, which is also the main reference of our current work.

B. Our New Discovery and Analysis

Although ElegentRL [7] does implement some of the basic reinforcement learning methods and some common environments that appear constantly in the real-world situation. However, from what we have experienced in our own investigation, we find out that "normal scenarios" are not the main concern of the investors. Instead, "extreme scenarios", like bad-performing markets, and some emergency economic measures published by governments or exchanges are what matters to investors. In our environment exploration, we find out that the basic DRL models have bad performances toward those environments. To explore this condition, we

- build up a poor-performing environment based on the S&P 500 dataset
- analyze the performance of current DRL models on these extreme environments
- come up with two new methods, the best ensemble method and the rounding ensemble method, to implement all of the models, make the best simulation, and acquire the best accumulative return on the extreme environments we built

III. Dataset and Data Preprocessing

A. Dataset Overview

There are many different stock markets in the world. Among those, the 500 stocks included in the Standard and Poor's 500 index (S&P 500) are very representative and have a large amount of data. So, in this paper, we choose a dataset of S&P 500 accessed from Yahoo Finance [10]. We set the period to be from 2009-01-01 to 2022-03-31. The size of the dataset is over 1 million. Examples of data are as shown in Table I.

B. Data Preprocessing

After we have accessed all data required, we need to deal with missing data in the data cleaning part and add more features in the feature engineering part to convert the data into a model-ready state. The whole data layer is as shown in Figure 1.

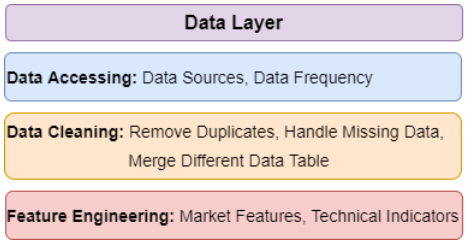

Fig. 1. Data layer overview

1) Data Cleaning: The cleaning processes of missing data are usually different for various time frequencies. In the low-frequency case, we directly delete the rows with missing values, reflecting suspension in simulated trading environments. While in the high-frequency case, we fill the open, high, low, and close columns with the last valid value of close price and the volume column with 0, which is a standard method in practice.

2) Feature Engineering: To better understand the characteristics of the data and facilitate subsequent training, we add additional features including statistical features (technical indicators, fundamental indicators), sentiment features (headlines, lexicon-based news, sentiment), and embedding features (transformer-based embeddings). Currently, we have successfully added several common technical indicators including Moving Average Convergence Divergence (MACD) [14], Relative Strength Index (RSI) [15], and Commodity Channel Index (CCI) [16].

- **Moving Average Convergence/Divergence:**

$$\begin{aligned} &\text{EMA}_i = p_i \times \frac{2}{n+1} + \text{EMA}_{i-1} \times (1 - \frac{2}{n+1}) \\ &\text{MACD} = \text{EMA}_{12} - \text{EMA}_{26} \\ &n = \text{number of days} \\ &p_i = \text{the value of the stock} \end{aligned} \tag{1}$$

The MACD is a momentum indicator calculated from the Exponential Moving Average (EMA).

	date	open	high	low	close	volume	tic	day
0	2009-04-01	10.815451	11.537911	10.743920	10.393170	5530768.0	A	2
1	2009-04-01	2.460000	2.700000	2.440000	2.451128	4184300.0	AAL	2
2	2009-04-01	40.520000	42.240002	40.400002	38.975742	1973000.0	AAP	2
3	2009-04-01	3.717500	3.892857	3.710357	3.308904	589372000.0	AAPL	2
4	2009-04-01	16.180000	16.420000	15.900000	13.337540	3438000.0	ABC	2

TABLE I
S&P 500 DATA EXAMPLE

- **Relative Strength Index:**

$$\text{RSI}_{\text{step one}} = 100 - [\frac{100}{1 + \frac{\text{Average gain}}{\text{Average loss}}}]$$
$$\text{RSI}_{\text{step two}} = 100 - [\frac{100}{1 + \frac{(\text{Previous Average Gain} \times 13) + \text{Current gain}}{(\text{Previous Average Loss} \times 13) + \text{Current loss}}}] \quad (2)$$

The RSI is a momentum indicator and it compares a security's strength on days when prices go up to its strength on days when prices go down.

- **Commodity Channel Index:**

$$\text{CCI} = \frac{\text{TP} - \text{MA}}{0.015 \times \text{MD}}$$
$$\text{TP} = \sum_{i=1}^{P}((\text{High} + \text{Low} + \text{Close})/3)$$
$$\text{P} = \text{Number of periods} \quad (3)$$
$$\text{MA} = (\sum_{i=1}^{P} \text{TP})/P$$
$$\text{MD} = (\sum_{i=1}^{P} |\text{TP} - \text{MA}|)/p$$

The CCI quantifies the difference between the current price of a stock and its average price.

IV. Problem Formulation

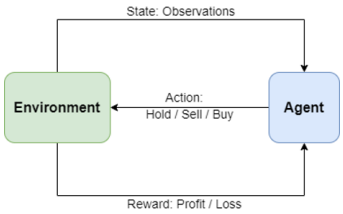

Fig. 2. Formulation of the Stock Trading Problem

Assuming full observability, we model a trading task as a Markov Decision Process (MDP) with five tuples (S, A, P, r, γ).

- **State** $s \in S$: Remaining balance, stock prices, and stock shares.
- **Action** $a \in A$**:** For each share of stocks, choose allowed actions including selling, buying, or holding.
- **Reward** $r(s, a, s')$**:** The asset value change of taking action a at state s and arriving at new state s'.
- **Transition Model** $P(s, a, s')$**:** The transition probability of an unknown environment.
- **Discounting Factor** γ**:** $\gamma \in (0, 1]$.
- **Policy** $\pi(s)$**:** The trading strategy at state s, which is a probability distribution over actions at state s.
- **Q-function** $Q(s, a)$**:** The expected return of taking action a at state s following policy π.
- **Goal:** Maximize the discounted cumulative return R.

V. Methodology

A. Environment

1) Simulation System Framework: There are three main "layers" or components of this system, the Data Layer, Environment Layer, and the Agent Layer. The Data Layer serves for the data cleaning and preprocessing while the Environment Layer serves for receving the actions, changing environments' states and returning the rewards. Finally, there can be multiple agents given, and each can work on the training, testing, trading independently. Their actions can be given to the Environment Layer and receive further states and rewards. The architecture is shown in the figure 3.

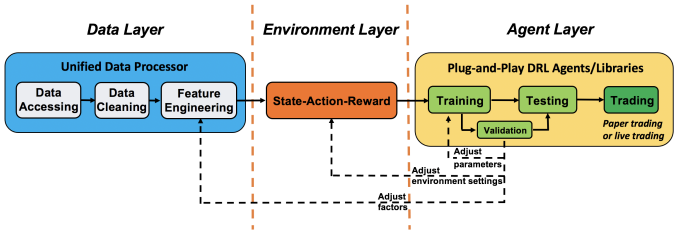

Fig. 3. An image of Financial Real-Market Simulation Pipeline, from [2].

Our inputs to our environment on any given date is a pandas data frame of open, close, high, low, technical indicators related to each of our referenced stocks. Given actions produced by our actor-network from the agent layer, we take them to update the shares of stocks we hold and further update our cash balance. The output of the simulation is a vector of rewards that the agent process from the given environment. We will use this

vector to calculate both cumulative reward and daily reward in order to compare the agents with the performance of S&P 500 funds to see how the agents perform. The cumulative reward plot is used to see the cumulative reward that each investor can acquire during a period of time. The typical chart of a cumulative reward is shown in Figure 4.

Fig. 4. A typical chart of cumulative reward generated from the simulation

2) Environment Introduction: We define our environment as S&P 500 stocks and use their close prices as the prices for each day. The environment in our system defines the states of a market, it is consisted by two parts. The first part of the environment including the close prices and indicators like turbulence/fluctuation, the Relative Strength Index (RSI), and Moving Average Convergence Divergence (MACD), etc. which are introduced above. Those indicators are either calculated from the prices of the chosen stocks or set to demonstrate information about them. These indicators simulate the "reference" that an investor refers to when making an investment in a real-world market. Given this information, we are able to compute the technical indicators explained further in the Data section.

The second part of the environment is the parameters of the market rules, like Reward-Scale, Buy-Cost, Sell-Cost, etc. which can be set by changing the parameters of the environment directly. These values simulate the rules set by market managers. So the environment combines these two factors to simulate a more comprehensive market scenario. The environment is complex enough because of the

B. Methods

1) Baseline Methods: We use the three most popular and best-performed methods, Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Deep Deterministic Policy Gradient (DDPG) as our baseline models.

Advantage Actor-Critic

A2C is a policy gradient algorithm and it is part of the on-policy family. To get a deeper understanding of A2C, we must first understand Actor-Critic Approach. The actor-critic algorithm consists of the actor and the critic working together. The actor chooses an action at each time step for deciding the steps to take. While in contrast, the critic evaluates the quality (Q-value) of a given input state. We use the critic network to learn which states are better, and we also use the actor to utilize the evaluation to teach the agent how to make actions to seek out good input states and avoid bad states.

The learning objective of the Actor Critic Methods could be described by finding the value proportional to the gradient described below.

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \sum_{s\in S} d^\pi(s) \sum_{a\in A} Q^\pi(s,a)\pi_\theta(a|s) \\ &\propto \sum_{s\in S} d^\pi(s) \sum_{a\in A} Q^\pi(s,a)\nabla_\theta \pi_\theta(a|s)\end{aligned} \tag{4}$$

Then in the Advantage Actor-Critic Algorithm, the Advantage Function calculates the agent's TD Error as its prediction error. Temporal Difference Error is given by the function:

$$\begin{aligned}\text{TD}_{error} &= \text{TD}_{target} - V(s) \\ \text{TD}_{target} &= R + \gamma * V(S^{'}), \text{ when not last step} \\ &= R, \text{ when last step}\end{aligned} \tag{5}$$

In the Advantage Actor-Critic algorithm, the Advantage is equal to the TD Error shown above. The formal definition of the Advantage (prediction error of the A2C algorithm) an be found below,

$$R_t = \sum_{k=0}^{\infty} \gamma^k * r_{t+k} \tag{6}$$

Note that the advantage function may not always be the same as the TD Error function. The Advantage function tells us if a state is performing good or not. If an action is performing better than we expected (the advantage is larger than 0), we would like to improve the actor to take more actions like that. If an action is worse than expected (the advantage is less than 0), we would discourage the actor to take the opposite of that action. If an action performs exactly as expected (the advantage equals 0), that means the actor doesn't learn anything from this advantage and action. As the agent explores its environment, the critic network is attempting to drive the advantage function to 0 so no more actions can be further learned.

Instead of learning the Q value, A2C tries to learn the advantage function $A(s,a)$. The advantage function is defined as

$$A(s,a) = Q(s,a) - V(s) = r + \gamma V(s) - V(s) \tag{7}$$

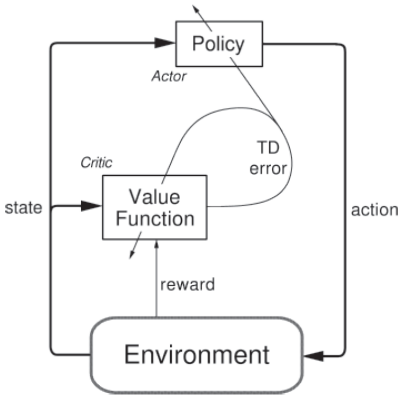

Fig. 5. A2C model description

While this is efficient an efficient method compared with pure Q-learning, it does waste a lot of training time and transitions before a global update on the actor-critic networks. As a result, it performs worse than either Deep Deterministic Policy Gradient (DDPG) or Proximal Policy Optimization (PPO). Thus, we just briefly mention this method as one of our baseline choices.

Proximal Policy Optimization

Proximal Policy Optimization, or PPO, is a policy gradient method for reinforcement learning. The motivation is to have an algorithm with data efficiency and reliable performance of TRPO while using only first-order optimization.

Let $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, so $r(\theta_{old}) = 1$. TRPO maximizes "surrogate" objectives:

$$L^{\text{CPI}}(\theta) = \mathbf{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\mathbf{A}_t] = \mathbf{E}_t[r_t(\theta)\mathbf{A}_t] \quad (8)$$

Where CPI refers to a conservative policy iteration, $\mathbf{A}_t$ is denoted by an advantage function.

$$\mathbf{A}_t(s,a) = \mathbf{Q}_t(s,a) - V(s) \quad (9)$$

The advantage function tells us if the action we are taking is any good - positive values will reinforce the behaviors, while negative ones will be discarded. However, our rewards are usually unknown, so we estimate the advantage function using estimated rewards from our Q network. While this does give us a limit on the size of our policy updates, TRPO is very complex, and PPO aims to approximate the objective function with a first-order approximation.

Without a constraint, maximization of L^{CPI} would lead to an excessively large policy update. Thus, the PPO modifies the objective, to penalize changes to the policy that move $r_t(\theta)$ away from 1:

$$J^{\text{CLIP}}(\theta) = \mathbf{E}[min(r_t(\theta)\mathbf{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\mathbf{A}_t)] \quad (10)$$

Where ϵ is a hyper-parameter measuring the scale of the "interval" that the clip can acquire. This forces PPO to stay within a specific interval for our policy updates but prevents us from getting very large policy rewards. PPO has been proven to work well on several stock trading applications, including where it outperforms every other benchmark. We use it to see if it can beat the benchmark algorithms on our own dataset.

Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is an algorithm that concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function and uses the Q-function to learn the policy.

This approach is closely connected to Q-learning, and is motivated the same way: if you know the optimal action-value function $Q^*(s,a)$, then in any given state, the optimal action $a^*(s)$ can be found by solving

$$a^*(s) = \underset{a}{\text{argmax}}\, Q^*(s,a) \quad (11)$$

DDPG interleaves learning an approximator to $Q^*(s,a)$ with learning an approximator to $a^*(s)$, and it does so in a way that is specifically adapted for environments with continuous action spaces. But what does it mean that DDPG is adapted specifically for environments with continuous action spaces? It relates to how we compute the max over actions in $\max_a Q^*(s,a)$.

Compared to on-policy reinforcement learning algorithms (such as A2C and PPO), DDPG performs just as well without needing to resolve to on-policy learning. We utilize it as another baseline to see if off-policy algorithms can perform as well as on-policy algorithms. In addition, it serves as another baseline in our survey of reinforcement learning algorithms.

2) Ensemble Method 1 (Best Ensemble Model): Before we introduce the ensemble methods, we first introduce the parameters we use for training the three models' baselines. We set the A2C steps to 5, the learning rate to 0.0005 and setting the entropy coefficient for the loss calculation to 0.01. We then set the PPO training parameters with the same entropy coefficient of 0.01, the number of steps taken to 2048, and the learning rate to 0.00025. Finally, we set the learning rate of the DDPG

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ_1 Q-function parameters ϕ, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta, \phi_{\text{targ}} \leftarrow \phi$

repeat

Observe state s and select action $a = \text{elip}\left(\mu s(s) + c, a_{\text{Low}}, a_{\text{High}}\right)$, where $\epsilon \sim \mathcal{N}$

Execute a in the environment

Observe next state s', reward $r_{,\text{ and}}$ done signal d to indicate whether s' is terminal

Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

If s is terminal, reset the environment state.

if it's time to update **then**

for however many updates do **do**

Randomly sample a butch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

Compare targets:

$$y\left(r, s', d\right) = r + \gamma(1-d) Q_{\text{thars}}\ \left(s', \mu_{\text{otwa}}\ \left(s'\right)\right)$$

Update the Q-function by one step of gradient descent using

$$\nabla_{\phi} \frac{1}{|B|} \sum_{\langle ta, r, A, A\rangle \in B} \left(Q_{\phi}(s, a) - u\left(r, s', d\right)\right)^2$$

Update policy by one step of gradient ascent using

$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\theta}\left(s_s \mu_{\theta}(s)\right)$$

Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}}\ + (1-\rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}}\ + (1-\rho)\theta$$

end for

end if

until convergence =0

algorithm to 0.0005.For each algorithm, we use a total of 10000 training timestamps.

With all the parameters fixed, we could start training our first ensemble model named Best Ensemble Model. To further introduce this ensemble method, we first have to divide the datasets into multiple phases. In the current setting, we would use a total training and validation dataset of 12 years as an example. In the first phase, the training datasets have one year and the validation has the rest 11 years. In the meantime, we would predict the first one-twelfth of the trading datasets in the first window. Note that we recalculate the turbulence threshold in each of the windows based on the training datasets. This is because we don't want our model to select high-risk stocks, and we avoid such a decision by selling the stocks which have the highest percentile of turbulence. We then train all three basic models (A2C, PPO, DDPG) with the training datasets and evaluate their performance based on the validation dataset we have allocated in the first phase. Their performance is further measured by the Sharpe Ratio, which is a mathematical expression of the insight that excess returns over a period of time may signify more volatility and risk. [17] We select the best-performing models among the three models we just trained and use that trained model to predict the one-twelfth of the testing dataset. With this method, we ensure that we are using the best-performing model to perform the trading. Then, in the second phase, we would do one more round of splitting between the training and validation datasets. Following the 12-year trading example, we would select the first two years as the training dataset and the rest as the validation datasets. The selected model would be used to trade the second one-twelfth of the trading dataset. Also, the results from the first one-twelfth trading dataset including the shares we hold (also meaning the state space) would be further used to trade the second phase. It ensures that the trading is continuous even we are using different models to improve their performances.

Algorithm 2 Best Ensemble Model

Input: first phase training data, validation data, testing data, all the model parameters

repeat

Train the A2C based on the given training data

Train the PPO based on the given training data

Train the DDPG based on the given training data

Evaluate the three baseline models with the validation data and their Sharpe Ratio

Select the model with the best performance (Highest Sharpe Ratio) for trading

Trading for the given testing data period

until Remaining Phase =0

3) Ensemble Methods 2 (Rounding Ensemble Model): After evaluating on the effectiveness of the first model, we further come up with the idea of improving the model to become more robust by taking a rounding of the actions of the three models. We first use the same method to divide the training and validation datasets. We also follow the steps we have introduced in the first ensemble method to complete the training. However, in this method set, we decide to utilize the fact that we have a total of three training models. Instead of selecting the best-performing model, we try to use all three methods

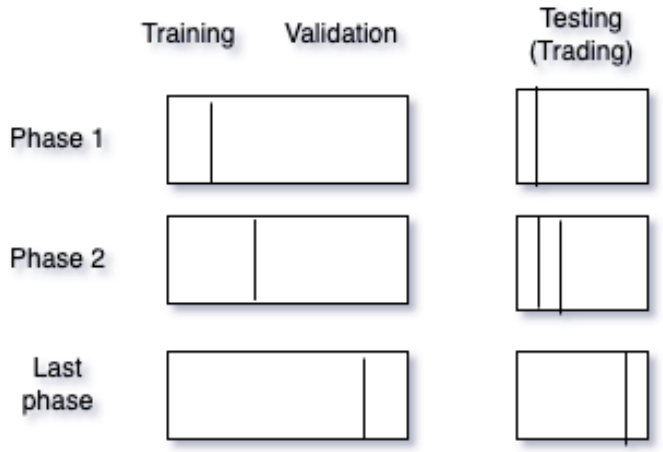

Fig. 6. Best Ensemble Model Training and Validation Data Split

to predict one-twelfth of the trading dataset. We then add up the actions that the three trained basic models have predicted, and round their mean into an integer representing the stocks we are going to buy or sell. We perform this because we notice that all of three basic models have their best performing period. If we take an action mean of the three models, we would receive a more stable model for not making too extreme decisions.

Algorithm 3 Rounding Ensemble Model

Input: first phase training data, validation data
repeat
Train the A2C based on the given training data
Train the PPO based on the given training data
Train the DDPG based on the given training data
Trading all the three baseline models
Getting the average of the actions produced by the three models and trading with the given actions

until Remaining Phase =0

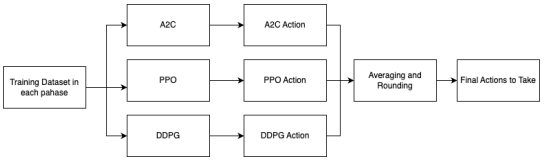

Fig. 7. Best Ensemble Model Training and Validation Data Split

VI. Experiment and Result

A. Dataset selection and spliting

We utilize the datasets from S&P 500, and further select one hundred poorest performance stocks among all of the 500 stocks to build a bad performance funding pool with the lowest MACD. This is because MACD is an indicator to show whether the given stock is bullish or bearish. Choosing the stock with the lowest MACD would give us the poorest-performing markets. We then split the datasets into training, validation, and testing exactly as what we have introduced in the Ensemble Methods section by dynamically allocating the dates to split the training and validation datasets. The training dataset and validation dataset is consists of dates from 2009-01-01 to 2020-06-30. We then have a total of 12 phases over these 12 years. In the first window period, the training dataset consists of dates from 2009-01-01 to 2009-12-31. And the validation dataset is from 2010-01-01 to 2020-06-30. The second window is from 2009-01-01 to 2011-01-01, and the validation is from 2011-01-01 to 2020-03-31. Each of the rest of the phases have one more year for the training period and one less year for the validation. Finally, the testing(trading) dataset is from 2020-07-01 to 2022-04-01. The first phase would only trade from 2021-07-01 to 2021-08-31. Each next phase would trading around next two months until the ending.

B. Evaluation metrics

We use the following metrics to measure the trading performance:

- **Cumulative return** $R = \frac{v-v_0}{v_0}$, v is the final portfolio value, v_0 is the original capital.
- **Annualized volatility** $\sigma_a = \sqrt{\frac{\sum_{i=1}^{n}(r_i-\bar{r})^2}{n-1}}$, r_i is the annualized return in year i, $\bar{r}$ is the average annualized return, and n is the number of years.
- **Sharpe ratio** $S_T = \frac{mean(R_t)-r_f}{std(R_t)}$, $R_t = \frac{v_t-v_{t-1}}{v_{t-1}}$, r_f is the risk-free rate, and $t = 1, ..., T$.
- **Max. drawdown** The maximal percentage loss in portfolio value.

C. Model and Performance

We first report the baseline results of the three models which consist of A2C, PPO, and DDPG independently. DDPG has received the best results among all the three basic models by receiving an accumulative return of 24.9% over the period of 2020-07-01 to 2022-04-30. A2C are performing a little bit worse than the DDPG, giving an accumulative return of 18.6%. However, PPO seems to perform the worst and giving a negative return of 13.5%. We can see that in this type of bearish markets environment, the model is achieving negative returns. This tell us the necessity to find some better replacement for the baseline models. In contrast, with our first implemented model, Best Ensemble Model, we have found that it achieves higher cumulative returns over the trading period with a carefully calculated turbulence

threshold. The turbulence threshold is chosen by the 99 percentile of all the stocks turbulence given over the training period. All of the stocks which have higher turbulence than the threshold would become out of the choice and would be sold. After using exactly same parameters as the three baseline models described in the Method Section, we have successfully achieved the most robust and best performance of the three baseline models. We find that the best ensemble methods returning an accumulative return of 29.9%, which is clearly higher than the baseline models. We also tried to implement our rounding methods over the same trading period with the same turbulence threshold selection methods. It seems that the rounding is giving us an accumulative return of 27.9%. Even though the algorithm doesn't work better than the Best Ensemble Model, it still achieves a relative good results compare to the baseline models. We found that sometimes more models would achieve more robustness than single model predictions. Using the best model selection methods would help us achieve a more robust model when we facing more extreme environments (poor performing stocks). Using the round ensemble methods would also help us through building more stable models. This definitely can help trading become more stable when facing more extreme environments and hoping to get stable returns.

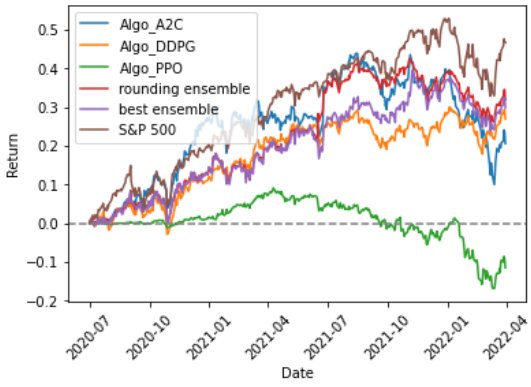

Fig. 8. Best Ensemble Model and Rounding Ensemble Model

VII. Future Work

As our development goes on, we find out that there are multiple points that are available for further optimization. For example, we have our averaging method hard-coded to be 1/3 of each method's contribution, but to be optimal, this parameter needs to be set in the model and be regarded as a variable that needs to be trained by given data. It also seems that the PPO are not trading effectively in this type of environments and can be marked as less important. Although the thought and the result proved its efficiency, it is still a kinda "tricky" method. Besides, although the ensemble method can maintain a fairly good performance under the worst circumstance, it is still costly when training and developing. The ensemble method takes more than 3 times of each baseline training because of the algorithm structure. It still awaits us to find out a more efficient way of training. And, we develop this method in a bad-performing environment, we are also looking forward to having a normal environment included in the "comfort zone" of the method. We did our exploration of normal cases, the ensemble method performs well but still costs a long time to train, which is relatively uneconomical. Developing new methods or algorithms is still the most efficient way to strengthen the reinforcement learning application in the financial environment. We will keep trying and learning about these topics in the future.

VIII. References

References

[1] Tidor-Vlad Pricope. Deep Reinforcement Learning in Quantitative Algorithmic Trading: A Review, 2021; arXiv:2106.00123.

[2] Liu, X. Y., Rui, J., Gao, J., Yang, L., Yang, H., Wang, Z., ... Guo, J. (2021). FinRL-Meta: A Universe of Near-Real Market Environments for Data-Driven Deep Reinforcement Learning in Quantitative Finance. arXiv preprint arXiv:2112.06753.

[3] Zhang, Z., Zohren, S., Roberts, S. (2020). Deep reinforcement learning for trading. The Journal of Financial Data Science, 2(2), 25-40.

[4] Marco Raberto, Silvano Cincotti, Sergio M Focardi, and Michele Marchesi. Agent-based simulation of a financial market. Physica A: Statistical Mechanics and its Applications, 299(1-2):319–327, 2001.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang and Wojciech Zaremba. OpenAI Gym, 2016; arXiv:1606.01540.

[6] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao and Christina Dan Wang. FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance, 2020; arXiv:2011.09607.

[7] Liu, Xiao-Yang and Li, Zechu and Wang, Zhaoran and Zheng, Jiahao. ElegantRL: Massively Parallel Framework for Cloud-native Deep Reinforcement Learning. Github, 2021.

[8] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[9] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236

[10] Yahoo. https://finance.yahoo.com/

[11] Kabbani, T., Duman, E. (2022). Deep reinforcement learning approach for trading automation in the stock market.

[12] W. F. Sharpe, "The sharpe ratio," The Journal of Portfolio Management, vol. 21, no. 1, pp. 49–58, 1994.

[13] Understanding actor critic methods and a2c — by Chris Yoon — towards ... (no date). Available at: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f (Accessed: November 18, 2022).

[14] Aguirre, Alberto Antonio Agudelo, 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 (2020): 44.

[15] Adrian, ran-Moroan. "The relative strength index revisited." African Journal of Business Management 5.14 (2011): 5855-5862.

[16] Maitah, Mansoor, et al. ”Commodity channel index: Evaluation of trading rule of agricultural commodities.” International Journal of Economics and Financial Issues 6.1 (2016): 176-178.
[17] Fernando, J. (2022, November 4). Sharpe ratio formula and definition with examples. Investopedia. Retrieved December 10, 2022, from https://www.investopedia.com/terms/s/sharperatio.asp