Study and Improvement on a Reinforcement Learning Framework for the Financial Portfolio Management Problem

Jun Zhai
junzhai0516@uchicago.edu
University of Chicago

Abstract

Financial portfolio management is the decision making process of continuously reallocating money into a number of different financial assets. The goal of the management process is to maximize portfolio profits while retaining the risk. In this paper, a novel ensemble strategy is proposed that employs and integrates two reinforcement learning policy gradient methods: Proximal Policy Optimization (PPO) and Twin Delayed Deep Deterministic Policy Gradient (TD3), thereby robustly adjusting to different market situations. The performance of the trading agent is evaluated and compared with three benchmarks: Uniform Buy-And-Hold, Random Buy-And-Sell, and Minimum-Variance. The proposed ensemble strategy is able to robustly select the best underlying agent and shown to outperform the benchmarks in terms of cumulative return, annualized return and Sharpe ratio.

Keywords: Financial portfolio management; Reinforcement learning; Ensemble strategy; Policy Gradient method; Proximal Policy Optimization; Twin Delayed Deep Deterministic Policy Gradient

1 Introduction

Financial portfolio management is the decision making process of continuously reallocating money into a number of different financial assets. The goal of the management process is to maximize portfolio profits while retaining the risk. Designing a good investment and trading strategy is one of the major focus of institutional investors such as trading firms, hedge funds, investment banks, and wealth management companies. Due to the competitive and proprietary nature of the portfolio management industry, companies with long-term profit-making strategies are more likely to attract investors and capitals into their funds, and thus boosting their business and increasing revenues.

Traditionally, investment strategies are prior-constructed with little machine learning elements. Portfolio managers usually analyze available historical data using some statistical models and transform the asset allocation problem into an optimization problem that can be solved analytically or numerically. The most well-known one is the Markowitz mean-variance portfolio theory [1]. However, it is challenging for analysts to consider all relevant factors in a complex and dynamic financial market. With recent advance of using machine learning and reinforcement learning algorithms in many complex real-world applications, especially in the high-frequency and algorithmic trading areas, being able to master reinforcement learning algorithms and utilize advanced technology infrastructure to train those models become new challenges to portfolio managers.

This paper studies and applies two policy gradient reinforcement learning algorithms to solve the portfolio management problem. Those two algorithms are Proximal Policy Optimization (PPO) [2] and Twin Delayed Deep Deterministic Policy Gradient (TD3) [3]. A novel ensemble strategy is proposed to combine them together and finds the optimal model dynamically within the same
reinforcement learning framework. Model performances are compared with three benchmark models including Uniform Buy-And-Hold, Random Buy-And-Sell, and Minimum-Variance.

The paper is organized as follows. Section 2 briefly introduces the generic reinforcement learning framework and two policy gradient methods PPO and TD3. Section 3 summarizes two papers that are most relevant to this work. They solve the same problem using similar reinforcement learning algorithms. In Section 4 the portfolio management problem and models are presented in the formal mathematical language based on the reinforcement learning framework. This section includes model assumptions, description of customized reinforcement learning components, the novel ensemble strategy, performance measures and benchmark models for model evaluation. Section 5 presents the data processing steps, experiments setup and results analysis. Section 6 concludes this paper and proposes suggestions for future work. Section ?? describes the effort on this project.

2 Background

2.1 Reinforcement Learning and Policy Gradient Methods

Reinforcement learning considers the paradigm of an agent interacting with its environment with the aim of learning reward-maximizing behavior. At each timestep $t$, with a given state $s_t \in S$, the agent selects an action $a_t \in A$ with respect to its policy $\pi_\theta(a|s_t)$, receiving a reward $R(s_t, a_t, s_{t+1})$ and transiting to the new state of the environment $s_{t+1}$ according to the transition probability $P(s_{t+1}|s_t, a_t)$. The cumulative return is defined as the sum of discounted rewards $G_t = \sum_{i=t}^{\infty} \gamma^{i-t}R(s_i, a_i, s_{i+1})$, where $\gamma \in [0, 1]$ is a discount factor determining the trade-off between short-term and long-term rewards. Note that the return depends on the actions chosen, and therefore on the policy $\pi_\theta(a|s_t)$. Policy gradient methods directly parameterize a policy $\pi_\theta(a|s_t)$ and update parameters $\theta$ so as to maximize the expected cumulative return, i.e. $\arg\max_{\pi_\theta} E_{\pi_\theta}[G_t]$.

2.2 Proximal Policy Optimization (PPO)

PPO combines ideas from Deterministic Advantage Actor Critic (A2C) and Trust Region Policy Optimization (TRPO). The main idea is that after each update step, the new policy should not be too far from the old one. For that, PPO uses a clipped surrogate objective to avoid too large update. PPO has been tested on a set of benchmark tasks and proved to produce awesome results with much greater simplicity. PPO is good for trading because it is stable, fast, and simple to implement and tune.

2.3 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is a direct successor of Delayed Deep Deterministic Policy Gradient (DDPG) and improves it using three major tricks to prevent the overestimation of the value function: clipped double Q-Learning, delayed policy update and target policy smoothing. This approach mimics the idea of SARSA update and enforces that similar actions should have similar values. TD3 is effective at handling continuous action space and improves DDPG, and so it is appropriate for trading.

3 Related Work

The use of reinforcement learning to manage portfolios is not a novel concept. However, beyond any similarities in training methods, each paper adopts a different approach when constructing the algorithm.
Jiang et al. (2017) proposed a reinforcement learning framework using a deterministic policy gradient algorithm specifically for solving the portfolio management problem. The core of the framework is the Ensemble of Identical Independent Evaluators (EIIE) topology. An IIE is a neural network whose job is to inspect the history of an asset and evaluate its potential growth for the immediate future. The EIIE is trained in an Online Stochastic Batch Learning scheme (OSBL), which is compatible with both pre-trade training and online training during back-tests or online trading. Three different species of IIEs are tested in their work, a Convolutional Neural Network (CNN) [8] [9] [10], a basic Recurrent Neural Network (RNN) [11], and a Long-Short Term Memory (LSTM) [12].

Yang et al. (2020) proposed a novel ensemble strategy that combines three actor-critic algorithms and finds the optimal trading strategy in a complex and dynamic stock market. The three algorithms are Proximal Policy Optimization (PPO) [2], Advantage Actor Critic (A2C) [4], and Deep Deterministic Policy Gradient (DDPG) [6]. Further, they use an ensemble strategy to automatically select the best performing agent among PPO, A2C, and DDPG to trade based on the Sharpe ratio every three months.

4 Methods

4.1 Assumptions

In this work, strategies are back-tested by pretending to be back in time at a point in the market history, not knowing any future market information, and do paper trading from then onward. The following assumptions are imposed:

(1) Zero slippage
   The liquidity of all market assets is high enough and each trade can be executed immediately without incurring any bid-ask spread charges.

(2) Zero market impact
   The trade made by the trading agent is so insignificant that has no influence on the market. In other words, all prices are given as immutable input data and should not be affected by the agent’s actions.

(3) No short selling
   The trading agent is not allowed to do short selling, i.e. borrow a security and sell it, plan to buy it back later. Under this assumption, the account balance cannot go negative. The worst scenario is losing all invested fund.

(4) Market order
   The trading agent can only place market orders that are executed immediately at the latest close price. Market order is a commonly used order type for most investors.

(5) Transaction cost
   The transaction costs are incurred for each trade. There are many types of transaction costs such as exchange fees, execution fees, and SEC fees. Different brokers also charge different commission fees. Since fee calculation is not the focus of the algorithm, a fixed transaction cost is assumed to be 0.1% of the value of each trade (either buy or sell).

(6) Loss aversion
   The prospect theory states that people evaluate a prospect or a lottery based on gains or
losses from the reference (reference point dependence). When valuing losses and gains, losses loom larger than the same amount of gains (loss aversion) [14]. The sensitivity to gains and losses exhibits a diminishing trend (diminishing sensitivity, see Figure 1).

Figure 1: Value Function of Prospect Theory

(7) Closed-end fund
The trading account mimics the closed-end fund setup and raises a fixed amount of capital at the initial public offering (IPO) and then trades shares using all available capitals. Investors are not allowed to deposit additional cash or withdraw any cash from the fund after trading starts.

(8) Infinite investment time horizon
There is no predefined fixed time horizon that the trading process will terminate. Therefore, it is an ongoing process without a terminal state.

In a real-world trading environment, if the trading volume of the assets in a market is high enough, the first two assumptions are near to reality. The third assumption is commonly seen in practice since many firms or investors are not allowed to do short selling due to its complicated mechanism and unlimited downside loss. The loss aversion assumption quantifies the psychological feeling of how people react to gains and losses, and it has become a frontier topic in behavioral finance. Many empirical experiments support prospect theory. Above all, those assumptions are reasonable to make.

4.2 Mathematical Formalism
This section provides a detailed mathematical setting of the portfolio management problem based on the reinforcement learning framework described in Section 2.1.
In the problem of portfolio management, the agent is the software portfolio manager performing trading actions in the environment of a financial market. This environment consists of all available assets in the markets and the expectations of all market participants towards them. The goal of the algorithmic agent is to generate a time-sequence of portfolio weights in order to maximize the cumulative portfolio value, taking transaction cost into account.

4.2.1 State Space

State space should contain all information from the financial market that is relevant for the agent to make trading decisions. Let the market consist of \(m+1\) assets with one special asset called cash. In this work, the state space \(S\) consists of the following features.

- **Asset prices**
  Open, high, low, and close prices at time \(t\) for each asset \(i\) are observable from the market, denoted as \(p^{open}_{i,t}, p^{high}_{i,t}, p^{low}_{i,t}, p^{close}_{i,t} \in \mathbb{R}_+\). Based on Efficient Market Theory, current asset price reflects all information available in the market and future expectations towards this asset. Investors do not need to look at data other than asset prices to make trading decisions. Therefore, current asset prices are the most important state features. However, there are still many traders who believe in technical analysis and look for patterns in the historical prices. Prices from recent history in some cases are still useful. Let \(p^{close}_{i,t} - p^{close}_{i,t-f_j}\) be the close price for asset \(i\) at time \(t-f_j\), where \(f_j \in F\) is a period of time. Let the size of set \(F\) be \(k\). For each asset \(i\), a set of past close prices \(p^{close}_{i,t-f_1}, p^{close}_{i,t-f_2}, \ldots, p^{close}_{i,t-f_k}\) \(\in \mathbb{R}_+^k\) are included in the state space. In sum, all asset prices observed by the agent in state \(s_t\) can be written as the following vector

\[
p_t = [p^{open}_{1,t}, p^{high}_{1,t}, p^{low}_{1,t}, p^{close}_{1,t}, p^{close}_{1,t-f_1}, \ldots, p^{open}_{m,t}, p^{high}_{m,t}, p^{low}_{m,t}, p^{close}_{m,t} - f_k] \in \mathbb{R}_{+}^{(4+k)m}
\]

- **Cash**
  Available cash in the account at time \(t\) is \(c_t \in \mathbb{R}_{0+}\). \(c_0\) is the initial investment amount.

- **Asset shares**
  Number of shares owned of asset \(i\) at time \(t\) is \(h_{i,t} \in \mathbb{N}_{0+}\), where \(h_{i,0} = 0, \forall 1 \leq i \leq m\). Based on assumption (3) in Section 4.1 negative shares are not allowed. Denote shares owned of all assets at time \(t\) as a vector \(h_t = [h_{1,t}, \ldots, h_{m,t}] \in \mathbb{N}_{0+}^m\).

- **Volatility**
  Volatility is a statistical measurement of asset prices movement. The higher the volatility means the possibility of larger gain or loss. Exponentially Weighted Moving Average model (EWMA) assumes price returns on assets have serial correlations and gives more weight to the latest returns than the old ones. JP Morgan uses EWMA model for Value-at-Risk calculation and it is proved to be an effective measure of asset volatility [15]. Therefore, EWMA for asset \(i\) at time \(t\), denoted as \(\sigma_{i,t} \in \mathbb{R}_+\), is included in the state space to represent asset volatility. The equation to calculate EWMA at time \(t\) is:

\[
\sigma^2_{i,t} = \lambda \sigma^2_{i,t-1} + (1-\lambda)r^2_{i,t}
\]

where \(\lambda\) is an exponential factor, \(r_{i,t} = \ln \left( \frac{p^{close}_{i,t}}{p^{close}_{i,t-1}} \right)\) represents logarithmic return of asset \(i\) at time \(t\), and \(\sigma^2_{i,0} = r^2_{i,0}\). Denote EWMA for all assets at time \(t\) as a vector \(\sigma_t = [\sigma_{1,t}, \ldots, \sigma_{m,t}] \in \mathbb{R}_+^m\).
• Account balance

The total account balance of the portfolio at time \( t \), denoted as \( b_t \), satisfies the following equation:

\[
b_t = c_t + \sum_{i=1}^{m} p_{i,t}^{\text{close}} h_{i,t}
\]

Based on assumption (3) in Section 4.1, \( b_t \in \mathbb{R}_+ \) cannot go negative.

• Time and period

Timing is an important factor that impacts whether or not a trade can make profit. For example, trading volumes tend to be higher near the open and close time of the trading session than in the middle of the session. Therefore, prices are more volatile and more opportunities are available during those periods. There are also some who believe that certain days offer systematically better returns than others. For example, people believe that the first day of the week is the best. It is called the Monday Effect. Further, some assets prices show seasonality effect. For example, natural gas and crude oil prices tend to be higher during winter time when their demands are higher than those during summer time. Based on these empirical evidences of the timing effect, the following features are included in the state space. Assume current time is at \( t \):

\[
\begin{align*}
&\text{day of week } d_{t}^{\text{week day}} \in \{\text{Monday, Tuesday, Wednesday, Thursday, Friday}\}, \\
&\text{day of month } d_{t}^{\text{day}} \in \{1,2,\ldots,31\}, \text{31 is the maximum possible days in a month}, \\
&\text{month of year } d_{t}^{\text{month}} \in \{1,2,\ldots,12\}.
\end{align*}
\]

Denote those three features at time \( t \) as a vector \( d_t = [d_t^{\text{week day}}, d_t^{\text{day}}, d_t^{\text{month}}] \).

In sum, state \( s_t = [p_t, c_t, h_t, \sigma_t, b_t, d_t] \in \mathcal{S} \).

4.2.2 Action Space

For a single asset, the action space \( \mathcal{A} \) is defined as an integer set: \( \{-h_{\text{max}}, \ldots, -1, 0, 1, \ldots, h_{\text{max}}\} \), where \( h_{\text{max}} \) sets the limit on the maximum number of shares that the agent can buy or sell. Let \( a_t = [a_{1,t}, \ldots, a_{m,t}] \) be all actions to take at time \( t \), where \( a_{i,t} \in \mathcal{A} \) is the action to take for asset \( i \) at time \( t \).

• If \( a_{i,t} > 0 \), the agent buys \( a_{i,t} \) shares of asset \( i \).
• If \( a_{i,t} < 0 \), the agent sells \( |a_{i,t}| \) shares of asset \( i \).
• If \( a_{i,t} = 0 \), the agent does nothing to asset \( i \).

Based on assumption (3) in Section 4.1, borrowing or short selling is not allowed. The number of shares that the agent can buy for asset \( i \) is capped by the cash currently available in the account, and the number of shares that the agent can sell is capped by the number of shares of asset \( i \) currently available in the account. The modified action of the agent taking all constraints into account is as follows:

• If \( a_{i,t} > 0 \), the agent buys \( \min\{\frac{c_t}{p_{i,t}^{\text{close}}}, a_{i,t}, h_{\text{max}}\} \) shares of asset \( i \).
• If \( a_{i,t} < 0 \), the agent sells \( \min\{|a_{i,t}|, h_{i,t}, h_{\text{max}}\} \) shares of asset \( i \).
If $a_{i,t} = 0$, the agent does nothing to asset $i$.

Denote the index sets of assets that are bought, held, and sold at time $t$ as $\mathcal{H}_b, \mathcal{H}_h, \mathcal{H}_s$, where $\mathcal{H}_b \cup \mathcal{H}_h \cup \mathcal{H}_s = \{1, \ldots, m\}$ and $\mathcal{H}_b \cap \mathcal{H}_h = \emptyset, \mathcal{H}_b \cap \mathcal{H}_s = \emptyset, \mathcal{H}_s \cap \mathcal{H}_h = \emptyset$. The following relations between $h_{i,t+1}$ and $h_{i,t}$ hold:

$$
h_{i,t+1} = \begin{cases} 
  h_{i,t} + \min\{\frac{c_{t}^{\text{close}}}{p_{i,t}^{\text{close}}}, a_{i,t}, h_{\text{max}}\} & i \in \mathcal{H}_b \\
  h_{i,t} & i \in \mathcal{H}_h \\
  h_{i,t} - \min\{|a_{i,t}|, h_{i,t}, h_{\text{max}}\} & i \in \mathcal{H}_s
\end{cases}$$

(3)

To simplify the notation going forward, $a_{i,t}$ means the modified shares to buy or sell at time $t$ for asset $i$ after taking all constraints into account. Therefore, the following relation between $h_{i,t}$ and $h_{i,t+1}$ holds:

$$
h_{i,t+1} = h_{i,t} + a_{i,t}
$$

(4)

From above, the total size of the action space for $m$ assets is $(2h_{\text{max}} + 1)^m$. Since the action space is exponentially large, convergence could be slow and difficult to reach. Following advice from [19], the action space for a single asset is normalized to lie in a continuous interval $A' = [-1, 1]$. Algorithms work on action space $A'$ for each asset and the agent has a function $f : A' \rightarrow A$ to map the continuous interval back to integers. The function chosen is $f(x) = \text{round}(x \cdot h_{\text{max}})$, where $\text{round}(\cdot)$ is to round a real number to its nearest integer.

### 4.2.3 Rewards

The reward of the problem is defined as the change of the account balance $b_t$ after action $a_t$ is taken at state $s_t$ and arriving at a new state $s_{t+1}$:

$$
R(s_t, a_t, s_{t+1}) = b_{t+1} - b_t
$$

(5)

Denote the percentage of transaction cost described in Section 4.1 assumption (5) as $q$. The total cash paid to buy assets $i \in \mathcal{H}_b$ at time $t$ is

$$
c_t^b = \sum_{i \in \mathcal{H}_b} a_{i,t} p_{i,t}^{\text{close}} (1 + q)
$$

(6)

The total cash received from selling assets $i \in \mathcal{H}_s$ at time $t$ is

$$
c_t^s = \sum_{i \in \mathcal{H}_s} |a_{i,t}| p_{i,t}^{\text{close}} (1 - q)
$$

(7)

Therefore, the following relation holds between $c_t$ and $c_{t+1}$:

$$
c_{t+1} = c_t - c_t^b + c_t^s = c_t - \sum_{i \in \mathcal{H}_b} a_{i,t} p_{i,t}^{\text{close}} (1 + q) + \sum_{i \in \mathcal{H}_s} |a_{i,t}| p_{i,t}^{\text{close}} (1 - q)
$$

(8)
To further decompose the reward function equation (5), plug equations (4) and (8) into (5):

\[
R(s_t, a_t, s_{t+1}) = \left( c_{t+1} + \sum_{i=1}^{m} p_{i,t+1}^{\text{close}} h_{i,t+1} \right) - \left( c_t + \sum_{i=1}^{m} p_{i,t}^{\text{close}} h_{i,t} \right)
\]

\[
= (c_{t+1} - c_t) + \left( \sum_{i=1}^{m} p_{i,t+1}^{\text{close}} h_{i,t+1} - \sum_{i=1}^{m} p_{i,t}^{\text{close}} h_{i,t} \right)
\]

\[
= (c_{t+1} - c_t) + \left( \sum_{i=1}^{m} p_{i,t+1}^{\text{close}} (h_{i,t} + a_{i,t}) - \sum_{i=1}^{m} p_{i,t}^{\text{close}} h_{i,t} \right)
\]

\[
= (c_{t+1} - c_t) + \left( \sum_{i=1}^{m} p_{i,t+1}^{\text{close}} - \sum_{i=1}^{m} p_{i,t}^{\text{close}} \right)\]

\[
= (\sum_{i \in \mathcal{H}_s} |a_{i,t}| p_{i,t}^{\text{close}} (1 - q) - \sum_{i \in \mathcal{H}_b} a_{i,t} p_{i,t}^{\text{close}} (1 + q)) + \sum_{i=1}^{m} p_{i,t+1}^{\text{close}} a_{i,t}
\]

(9)

The last equality in equation (9) shows that the reward consists of three components: cash balance change from taking trading actions including transaction cost, gain/loss due to price change without taking any trading actions, and gain/loss from taking trading actions. The second component is simply a Buy-And-Hold strategy. The task of the agent is to take actions that optimizes the other two components.

To further take assumption (6) in Section 4.1 into account, a value function of prospect theory is applied to the reward equation (9) to transform the reward in dollar term into the utility that people obtain from the investment process. The value function suggested by [14] has the following form:

\[
v(x) = \begin{cases} 
   x^\alpha & x \geq 0 \\
   -\eta(-x)^\beta & x < 0 
\end{cases}
\]

(10)

where \(\eta\) is the coefficient of the loss aversion. Therefore, the reward function of this problem is defined as

\[
v(R(s_t, a_t, s_{t+1}))
\]

(11)

4.2.4 Goal

The goal of the problem is to design a trading strategy \(\pi_\theta(a|s_t)\) that maximizes the expected discounted cumulative value of reward, i.e.

\[
\arg \max_{\pi_\theta} \mathbb{E}_{\pi_\theta} \left[ \sum_{t=0}^{\infty} \gamma^t v(R(s_t, a_t, s_{t+1})) \right]
\]

(12)

Since the portfolio management is an ongoing process without a terminal state, to guarantee convergence it is required that \(\gamma \in [0, 1)\). The policy is parameterized by \(\theta\) and can be solved by policy gradient methods.

4.3 Ensemble Strategy

Ensemble methods, such as Bagging and Random Forests, have been shown to be effective in combining predictions from several single classifiers, leading to better accuracy and robustness than using a single classifier. In reinforcement learning, ensemble methods have been used for
combining function approximators to store the value function [16], and this can be an efficient way for improving an reinforcement learning algorithm. In contrast to previous research, this paper proposes a novel ensemble strategy that combines different reinforcement learning algorithms that learn separate value functions and policies into one integrated reinforcement learning framework. The following specifies the ensemble strategy and applies it to the financial portfolio management problem.

The idea of the ensemble strategy is that the ensembler agent treats the underlying agents as part of the augmented environment and observes their actions as state features in addition to all existing features in the state space. The action of the ensembler is to select one of the underlying agents and uses it to take actions on its behalf at that timestep. The goal of the ensembler is to select the best underlying agent given the current state that maximizes the same value function as the original problem. The ensembler-environment interaction is shown in Figure 2. This figure shows the training and testing step of the ensembler when the underlying agents have been trained and their policies are fixed. For the underlying agents, they will observe the state from the environment, query their own policies and send their actions to the ensembler as candidates.

$$\text{Figure 2: The Ensembler-Environment Interaction}$$

Mathematically, the ensemble strategy can be formalized as follows. Let the number of underlying agents be $n$. Denote the common state space observed by each underlying agent as $S$ and the state observed by them at time $t$ as $s_t$. Denote the action space of each underlying agent $i$ as $A^{(i)}$ and the action taken by agent $i$ at time $t$ as $a^{(i)}_t$. In the following ensemble strategy, we assume that the action spaces for each underlying agent are the same, i.e. $A^{(1)} = \ldots = A^{(n)}$.

### 4.3.1 State Space

The state space observed by the ensembler agent is defined as $S^* = S \otimes (\otimes_{i=1}^n A^{(i)})$ where $\otimes$ is the cartesian product of sets, and the state observed by the ensembler agent at time $t$ is $s^*_t =$...
\[ s_t, a_t^{(1)}, \ldots, a_t^{(n)} \]. Specifically, let the action taken by the underlying trading agent \( i \) at time \( t \) be \( a_t^{(i)} = [a_{1,t}^{(i)}, \ldots, a_{m,t}^{(i)}] \) where \( a_{j,t}^{(i)} \) is the number of shares to buy, hold or sell for asset \( j \) by agent \( i \) at time \( t \). Based on the state space \( S \) defined in Section 4.2.1, the augmented state space \( S^* \) and state at time \( t \) observed by the ensembler agent is

\[ s_t^* = [p_t, c_t, h_t, \sigma_t, b_t, d_t, a_t^{(1)}, \ldots, a_t^{(n)}] \in S^* \]

### 4.3.2 Action Space

Denote \( D \) as the index set of underlying agents, i.e. \( D = \{1, 2, \ldots, n\} \). Denote the underlying agent index that the ensembler chooses at time \( t \) as \( u_t \in D \). The action space \( A^* \) of the ensembler is \( D \). The action taken by the ensembler at time \( t \) is \( a_t^* = u_t \). Specifically, in the portfolio management problem, the action of the ensembler agent is to choose one underlying agent and uses it as the trading agent to trade at time \( t \).

### 4.3.3 Rewards

The reward has the same form as the original one but plugs in state and action of the ensembler agent instead, i.e. \( R(s_t^*, a_t^{(u_t)}, s_{t+1}^*) \). The reward function of the portfolio management problem is defined in equation (11).

### 4.3.4 Goal

The goal of the ensembler agent is the same as all its underlying agents. The goal of the portfolio management problem is defined in equation (12). This ensures that the policy learned by the ensembler agent is the solution to the same problem as its underlying agents. It is known that the optimal value function to the Bellman equation is unique but the optimal policies can be many. Therefore, the ensembler and all its underlying agents possibly learn different policies reaching the same goal.

### 4.3.5 Ensemble Algorithm

In this work, two underlying agents are chosen. One uses PPO algorithm and the other uses TD3 algorithm. The algorithm that the ensembler uses is PPO. Those choices can be investigated further in the future.

Algorithm 1 describes the ensemble algorithm used in the back-testing experiments. During each step of the experiment, the dataset is split into three disjoint periods, \( [0, t), [t, t + \Delta t), [t + \Delta t, t + 2\Delta t) \). This mimics the training set, validation set and test set concepts used in machine learning algorithms. We use a growing dataset from 0 to \( t \) to train underlying agents. This guarantees that all underlying agents have sufficient data to train and reach convergence as they see more and more data and market environments. The ensembler is trained using the next \( \Delta t \) window of data, i.e. \( [t, t + \Delta t) \). This choice makes sure that underlying agents have not seen the dataset before and the ensembler can validate their generalization power. Based on the trading performance of all underlying agents during this period, the ensembler is trained to pick the best model under those market scenarios. Further, since the ensembler picks the agent based on recent data, it is more likely to choose the one that works well under near future market scenarios. At the last stage of the step, the ensembler is tested using data in the next \( \Delta t \) window, i.e. \( [t + \Delta t, t + 2\Delta t) \). Figure 3 shows the dataset splitting for one experiment step.
Algorithm 1 Ensemble Algorithm

1: procedure Back-test(data[0,...,T], Δt, t_{start})
   \( data \): dataset. \( Δt \): rebalance window. \( t_{start} \): initial training set size.
2: \( t = t_{start} \) \( \triangleright \) Initialize timestep.
3: ppo_agent = PPO() \( \triangleright \) Initialize two agents and one ensembler using chosen RL models.
4: td3_agent = TD3()
5: ensembler = PPO()
6: results = [] \( \triangleright \) Hold back-testing results.
7: while \( t + 2Δt < T \) do
8: \( \text{training\_set = data}[0,...,t] \) \( \triangleright \) Training set for underlying agents.
9: ppo_agent.learn(training_set) \( \triangleright \) Train PPO agent.
10: td3_agent.learn(training_set) \( \triangleright \) Train TD3 agent.
11: validation_set = data[t,t + Δt] \( \triangleright \) “validation” set = training set for ensembler.
12: ensembler.learn(validation_set, ppo_agent, td3_agent) \( \triangleright \) Train ensembler.
13: test_set = data[t + Δt, t + 2Δt] \( \triangleright \) Test set for ensembler.
14: result = ensembler.predict(test_set) \( \triangleright \) Test ensembler.
15: results.append(result) \( \triangleright \) Save result.
16: \( t = t + Δt \) \( \triangleright \) Step to next step.
17: end while
18: return results \( \triangleright \) Return back-testing results.
19: end procedure

4.4 Benchmarks

To evaluate the performance of reinforcement learning trading strategies proposed in this work, three baseline trading models are selected as benchmarks.

- Uniform Buy-And-Hold (UBAH)
  At time 0, the agent distributes all cash uniformly among all \( m \) assets in the market and the number of shares are held constant without any rebalancing, i.e.
  \[
  h_{i,t} = \frac{c_0}{m p_{t,0}^{\text{close}}}, \forall t \geq 0, 1 \leq i \leq m
  \]
  (13)

  This strategy would be most useful when the returns across all assets are purely random and investors have no views on which one they prefer. It also represents the raw performance of all assets.

- Random Buy-And-Sell (RBAS)
  At time \( t \), the agent selects an action for asset \( i \) by generating an uniform random integer \( a_{i,t} \in \mathcal{A} \). This strategy can be used to verify whether the learner has learned anything meaningful or it is acting randomly and just getting rewards by chance.
• Minimum-Variance (MV)
According to Markowitz mean-variance portfolio theory [1], there is a portfolio that minimizes variance of portfolio returns on the efficient frontier. It is a conservative strategy that aims to preserve the portfolio value in the long term. The optimal weights \( w^*_t \in \mathbb{R}^m \) at time \( t \) are the solution to the following optimization problem:

\[
\arg \min_{w} w^T \Sigma_t w \\
\text{s.t. } w^T e = 1
\]

(14)

where \( \Sigma_t \in \mathbb{R}^{m \times m} \) is a covariance matrix of returns of \( m \) assets up to time \( t \), \( e = [1, \ldots, 1] \in \mathbb{R}^m \). The constraint requires that the sum of percentage weights must be 1. The optimal number of shares the agent chooses to hold at time \( t \) for asset \( i \) is

\[
h_{i,t} = \frac{w^*_i b_t}{p_{close}}
\]

(15)

This strategy can be used to compare the stability and conservativeness of portfolios.

4.5 Performance Measures

Different metrics are used to measure the performance of strategies at any given time \( T \).

• Cumulative return
This is the most direct measurement of how successful portfolio management is over a timespan \([0, T]\). It is calculated as follows:

\[
r_c = \frac{b_T - b_0}{b_0}
\]

(16)

• Annualized return
Denote the daily return of the portfolio as \( r_t = \frac{b_t - b_{t-1}}{b_{t-1}} \). The sample average of daily return is:

\[
\bar{r} = \frac{1}{T} \sum_{t=1}^{T} r_t
\]

(17)

The annualized return is defined as follows:

\[
r_p = 252 \bar{r}
\]

(18)

This assumes that the daily returns are independently identically distributed and each year has about 252 business days.

• Annualized volatility
The annualized volatility measures the standard deviation of portfolio returns. It is defined as follows:

\[
\sigma_p = \sqrt{\frac{252}{T-1} \sum_{t=1}^{T} (r_t - \bar{r})^2}
\]

(19)

The same assumption is made as the annualized return.
Sharpe ratio
Sharpe ratio (SR) is used to take risk into account. The ratio is a measurement of risk adjusted mean return. It is defined as follows:

\[ SR = \frac{r_p - r_f}{\sigma_p} \]  

(20)

where \( r_f \) is the risk-free rate.

Max drawdown
Max drawdown (MDD) is the maximum percentage of loss during the trading period, i.e. the biggest percentage of loss from a peak to a trough. It is defined as follows:

\[ MDD = \min_{0 \leq t_1 < t_2 \leq T} \frac{b_{t_2} - b_{t_1}}{b_{t_1}} \]  

(21)

5 Experiments
5.1 Dataset
Historical open, high, low, and close prices are available at histdata.com [18]. Each downloaded file contains one year of data for one asset. The format of the raw data file is as follows.

YYYY.MM.DD,HH:MM,Open,High,Low,Close,Volume

Based on the website notice, they are not able to collect and provide meaningful historical volume data in the file and only insert dummy values. Therefore, “Volume” data are not used at all. Data downloaded for all experiments are at 1 minute time frame ranging from 2011-01-02 to 2020-10-23. Since data across assets may not be all available at a given timestep, the following data cleaning process is used to align data. It uses a MapReduce framework to remove data at a timestep if not all four prices for all selected assets are available at that timestep. The cleaned dataset contain timesteps that have all four prices available for all selected assets. This process is the same as performing a join operation on timestep for all assets prices files.

5.1.1 MapReduce for Data Cleaning
The size of one ten-year data file for one asset is about 200MB. If the program extends to a very large set of assets or is going to tested on much longer period, it is difficult to perform the data cleaning step on a single local machine. Therefore, a MapReduce framework is used to process data to achieve program scalability. The following specifies the MapReduce algorithm.

The Map Function:
For each line of the file described above, emit

\[ \text{key: (YYYY.MM.DD,HH:MM), value: (symbol,open,high,low,close)} \]

where “symbol” in the value tuple is the asset ticker symbol read from input file name.

The Reduce Function:
For each key (YYYY.MM.DD,HH:MM), if the size of list of its associated values equals to the pre-selected number of assets for the experiments, for each of the value in the list, emit

\[ \text{key: 1, value: (YYYYMMDDHHMM,symbol,open,high,low,close)} \]
Otherwise, do nothing. *key* is a dummy value and YYYYMMDDHHMM in the value tuple is an integer that concatenates date and time digits. Special characters such as “,” and “:” in the date and time strings are removed as they are not useful and integer is easier to process than date format in the program. This also reduces output file size and saves disk space.

The data cleaning MapReduce process ran on AWS EMR cluster with two m4.large nodes. It took 8 minutes to process ten-year data files for four assets. The cleaned total number of timesteps in the output file is about 2,500,000. The amount of data points is sufficient for reinforcement learning training and testing.

5.2 Hyper-parameters

This section describes the model hyper-parameters in Section 4.2 that are chosen to perform experiments.

- **Asset pre-selection**
  Four high-volumed assets are pre-selected for the portfolio. Large volume implies better market liquidity of an asset. In turn it means that the market condition is closer to assumptions (1) and (2) set in Section 4.1. They cover different classes of assets and are commonly traded by investors.

  - **S&P 500 Index**
    A stock market index that measures the stock performance of 500 large companies listed on stock exchanges in the United States. It is one of the most commonly followed equity indices.

  - **Nasdaq Composite**
    A stock market index that includes almost all stocks listed on the Nasdaq stock market. Along with the S&P 500 Index, it is one of the most-followed stock market indices in the United States.

  - **Euro-Dollar Pair**
    This foreign exchange pair is popular with traders because its constituents represent the two largest and most influential economies in the world.

  - **West Texas Intermediate (WTI)**
    A crude oil product that serves as one of the main global oil benchmarks.

The number of assets in the portfolio is \( m = 4 \). This number is chosen by experience and can be adjusted in future experiments.

- **Previous close prices**
  Previous close prices from 1 minute ago, 10 minutes ago, 30 minutes ago, 1 hour ago, and 1 day ago from current time \( t \) are included in the state \( s_t \), i.e. the period set \( \mathcal{F} = \{1, 10, 30, 60, 1500\} \). Those choices are based on popular bar charts that traders usually use for technical analysis.

- **Initial account balance**
  To avoid issues when asset prices are too large to allow an action to achieve its desired result, all portfolios and benchmarks start with $1,000,000 initial cash, i.e. \( c_0 = b_0 = 1000000 \).

- **EWMA exponential factor**
  It is determined in [15] that the EWMA exponential factor \( \lambda \) for volatility forecasting is about 0.94. This paper follows that suggestion.
• Maximum number of shares
  The maximum number of shares \( h_{\text{max}} \) that the agent can buy or sell is set to 1,000,000 which is the same number as the initial cash.

• Value function of prospect theory
  Kahneman and Tversky (1992) reported that the median of both \( \alpha \) and \( \beta \) was 0.88 and the median of \( \eta \) was 2.25. This paper follows their suggestion.

• Discount factor
  The discount factor \( \gamma \) is chosen to be 0.99 for all experiments which is the default value set by stable-baselines package \[20\] \[21\].

• PPO and TD3 algorithms parameters
  All parameters required by PPO and TD3 use the default ones set by stable-baselines package \[20\] \[21\]. Those default values are chosen by authors of their original papers and are optimal in most of their experiments. Since the purpose of this paper is not to fine-tune any model parameters, default values are used.

• Rebalance window
  The rebalance window \( \Delta t \) mentioned in Algorithm 1 is chosen to be 3 months which follows the same choice as the training and validation rolling window in \[13\]. For benchmark strategies Random Buy-And-Sell and Minimum-Variance, their shares are rebalanced every 3 months to match the ensembler strategy.

• Initial training set size
  The initial training set size \( t_{\text{start}} \) is set to 6 years. It means that all underlying agents start training with data from 2011-01-01 to 2015-12-31. The training set size keeps growing with 3 months more data for each step in the while loop. The experiment ends at 2020-10-08.

• Risk-free rate
  The risk-free rate \( r_f \) in the Sharpe ratio calculation is set to 0. Given current close-to-zero interest rate scenario, this setting is reasonable.

5.3 Program Modules

The program for experiments is implemented in python. It has six major modules:

• Configuration
  This module sets all model hyper-parameters described in Section 5.2 and other experiment parameters such as input/output file name and path.

• Data pre-processing
  This module implements the MapReduce data cleaning process described in Section 5.1.1. It uses mrjob library and can run on AWS EMR clusters.

• Models and environment
  This module includes code for benchmark models and reinforcement learning models implementations. For reinforcement learning models, one major package used is the stable-baselines library \[22\]. It is a set of improved implementations of reinforcement learning algorithms based on OpenAI Baselines. This toolset is a fork of OpenAI Baselines, with a major structural refactoring, and code cleanups. It provides implementation of PPO and TD3 models and
allows user to quickly experiment, debug, and tweak models, which is critical to experiment different ideas. The customized reinforcement learning components described in Section 4.2 are implemented as a customized environment class in this module. It can be used by the two reinforcement learning models directly.

- Performance evaluation
  This module includes tools to calculate performance measures described in Section 4.5. It also includes functions to generate plots presented in this paper.

- Utility
  All utility functions are included in this module, including logging, data loading, and file parsing.

- Back-testing scripts
  There are some back-testing related scripts included in the codebase. The “main” function that runs a reinforcement learning back-testing experiment is implemented in a python script “backtesting.py”. The one for benchmarks experiment is implemented in a python script “backtesting_baselines.py”. Two notebook files are provided, one for Google Colab, and one for AWS SageMaker. Those notebook files include platform specific setup before calling those back-testing python scripts.

The detailed code documentation is available on the Github 1 “main” branch README file as well as in the source code comments.

5.4 Results and Discussion

From model performance summary in Table 1, we observe that PPO and Ensemble models have the best performance in terms of cumulative return, annualized return and Sharpe ratio. The reason why PPO and Ensemble models have the same performance is that the ensembler always selects the PPO underlying agent to trade on its behalf across all back-testing periods. The ensembler learned that PPO is a better agent to use than the TD3 agent based on all training data and past performances it sees from both underlying agents. This model selection is stable and does not change during the testing periods. From the performance summary in Table 1, the account balance comparison across all back-testing periods in Figure 4 and Sharpe ratio comparison across all back-testing periods in Figure 5, we can confirm that PPO beats TD3 most of the time and is a better model among those two based on those performance measures. This confirms that the ensembler works as expected and is able to robustly select the best agent based on the reinforcement learning settings mentioned in Section 4.3 without any human intervention about model selection or knowing any of those model performance measures. The only signal that the ensembler receives is the reward defined in Section 4.3.3. This experiment shows the power of reinforcement learning that it uses a generic and simple framework to learn a good strategy under complex and dynamic environment. Meanwhile, we observe that the Minimum-Variance strategy has the best performance in terms of annualized volatility and max drawdown. This observation makes sense as the Minimum-Variance strategy is designed to be conservative and aims to minimize the portfolio volatility. Since the volatility in this strategy is optimized to be minimum, the portfolio return is not large based on the risk-return trade off. In contrast, the definition of reinforcement learning reward function in this problem only considers account value change and not penalizes volatility. Therefore, the agent is not able to control portfolio volatility. That is the reason why we observe a relatively large

1 https://github.com/zhaijunjay/mpcs53112.git
annualized volatility and drawdown for reinforcement learning algorithms, e.g. PPO and TD3, compared to other models.

Table 1: Model Performance Summary

<table>
<thead>
<tr>
<th></th>
<th>UBAH</th>
<th>RBAS</th>
<th>MV</th>
<th>PPO</th>
<th>TD3</th>
<th>Ensemble</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2016-01-01 to 2020-10-08)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cumulative Return</td>
<td>57.86%</td>
<td>45.65%</td>
<td>17.70%</td>
<td>95.53%</td>
<td>76.18%</td>
<td>95.53%</td>
</tr>
<tr>
<td>Annualized Return</td>
<td>9.17%</td>
<td>8.21%</td>
<td>2.99%</td>
<td>13.68%</td>
<td>12.11%</td>
<td>13.68%</td>
</tr>
<tr>
<td>Annualized Volatility</td>
<td>16.53%</td>
<td>18.78%</td>
<td>6.47%</td>
<td>21.11%</td>
<td>22.11%</td>
<td>21.11%</td>
</tr>
<tr>
<td>Sharpe Ratio</td>
<td>0.56</td>
<td>0.44</td>
<td>0.46</td>
<td>0.65</td>
<td>0.55</td>
<td>0.65</td>
</tr>
<tr>
<td>Max Drawdown</td>
<td>-9.35%</td>
<td>-11.57%</td>
<td>-3.35%</td>
<td>-9.33%</td>
<td>-7.37%</td>
<td>-9.33%</td>
</tr>
</tbody>
</table>

Figure 4: Account Balances of Ensemble Model and TD3
It is often difficult to interpret the strategy that the agent learns since the algorithm behind the scene is complicated and sometimes uses deep neural network. However, we are still interested to see what kind of trading strategy that each agent uses and try to interpret it if possible. The top two subplots in Figure 6 show the account value timeseries and the portfolio constituents of the TD3 and Ensemble models at each timestep during the back-testing periods. The color represents the corresponding four assets at the bottom subplots. From those subplots, we can see that the agent holds a single asset in the portfolio at any time without distributing money across different assets. Therefore, it is a Buy-And-Hold strategy till it decides to switch to a different asset. That explains why it does not take any volatility into consideration like Minimum-Variance strategy does. The portfolio exposes to risk from a single asset and therefore is a very risky strategy. Further, we can see that two models learn different strategies and hold different assets most of the time. That is the reason why the performance between Ensemble (PPO) and TD3 models are different. PPO is able to pick the asset that has better growth potential than the one TD3 chooses. That potential can be realized into larger actual gains most of the time compared to the one TD3 chooses. It can be seen from the Ensemble subplot that after most switches, there is often an upward trend of the account balance which confirms that the switch is a good decision. Further, the agent often switches asset when there is a downward trend of the account balance which mimics a stop-loss strategy. Therefore, the trading strategy that the reinforcement learning agent learned can be summarized as follows.

- In an upward trend, buy and hold a single asset that has the best growth potential.
- In a downward trend, sell all shares if there is any in the portfolio and buy another asset that has the best growth potential.
6 Future Work

There are some more work that can be done in the future. First, effort on feature engineering is needed. One of the features is text based, such as financial market news, social media posts, and ESG features, thus natural language processing can be implemented to extract and provide useful information to the agent. Some other features for the state space can be explored such as adding advanced transaction cost and liquidity model to relax the assumptions made in Section 4.1. More assets can be included in the portfolio and tested on longer period. Further, it is interested to use Sharpe ratio directly as the reward function or adding volatility as a penalty term in the hope that portfolio risk can be controlled. In that case, the agent needs to observe a lot more historical data, and the state space will increase substantially. A good approximation algorithm for Sharpe ratio is required. Given the large parameter sets for reinforcement learning, achieving the optimal parameters is difficult and time consuming. However, there is still possible room to make improvement by further tuning the model parameters. Last but not least, the proposed ensemble strategy can be tested on other reinforcement learning problems and compare its performance with existing algorithms.

Acknowledgement

This work is supported by Professor Amitabh Chaudhary, Department of Computer Science, The University of Chicago.
References


