Portfolio Optimization Based on Deep Reinforcement Learning

Guoxin Sun[1, *]
[1]Computer Science and Engineering
University of California
San Diego, USA
*Corresponding author: g3sun@ucsd.edu

***Abstract*— Portfolio Optimization (PO) is a financial engineering technique that rebalances resources across multiple risky assets at a particular moment to optimize returns and minimize risks. This study reveals two methods (static and dynamic) for PO. The static method utilized Maximum Sharpe Ratio to determine allocation weights that optimize prior profits. The dynamic approach employs Deep Reinforcement Learning (DRL) to build a model that analyzes stock movement and recommends immediate trading actions (buy/sell/hold). The dynamic method utilizes Twin Delayed Deep Deterministic Policy Gradient (TD3) because it performs well in continuous action spaces without producing excessive biases. The purpose of this study is to evaluate the performance of TD3 on relatively small datasets. Experiments using criteria such as expected returns and Sharpe Ratio indicate that DRL models outperform baseline models. Hyper-parameter tuning has a beneficial effect overall.**

Keywords—Portfolio optimization; Reinforcement learning; Deep deterministic policy gradient

I. Introduction

We can address Portfolio optimization as a model that allocates a fixed number of resources between these risky assets to maximize the likelihood of returns [1]. This paper focuses on stock trading. The portfolio is an allocation of resources across owned stocks.

There are several Machine Learning techniques for stock trading. One approach is utilizing forecasting techniques to predict the stock's performance and building a heuristic-based model that uses the prediction to make decisions. Another strategy is to develop a model that analyzes stock performance and recommends buy/sell/hold decisions accordingly. The first approach assumes the optimal strategy based on past information can work in the future. While this method can serve as a guideline for future investment, it fails to give us an accurate forecast of future performance since the assumption may not be valid. The second technique will be an ideal application of DRL, as the collective outcome of the actions is only established after each trading epoch through learning, which corresponds to the trial-and-error character of DRL [2-4].

In the vanilla Deep Deterministic Policy Gradient (DDPG) approach [5], the agent optimizes its action policy by repeatedly attempting different actions to maximize the expected cumulative reward in the simulated environment. The agent in DDPG consists of two components: the actor, who determines its action based on its current state, and the critic, which evaluates the state and action pairings based on the estimated cumulative reward. However, the DDPG model suffers from overestimation (Q values of the critic network) from approximation errors of high dimensional functions in the neural network. Since actor-critic methods are bootstrapped [6], these errors accumulate during the training time. When these errors build up, the agent can fall into local optima or catastrophic forgetting. TD3 uses a pair of critic networks, delayed actor updates, and action noise regularization to address the issue above. This study aims to examine the performance of MPT and DRL models on a small dataset.

II. Methodology

A. *Asset Pre-selection*

This study selects stocks with diversification to minimize the risk factor. There are six stocks from different sectors: technology, energy, and food; they are AAPL, CVX, NSRGY, MSFT, PEP, and XOM. Figure 1 shows the correlation among the six stocks using a heat map.

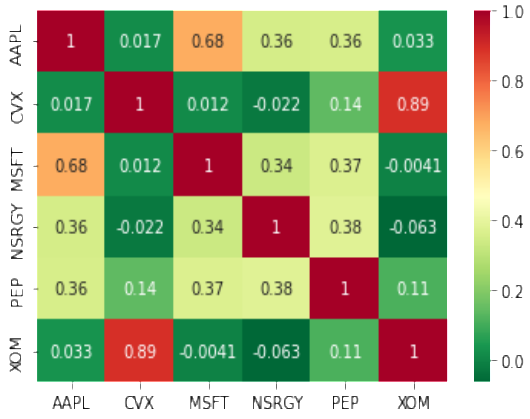

Figure 1. Heat Map of Different Assets.

B. *Data Collection*

The trading data come from Yahoo! Finance Python API. The data follows the format of Open, High, Low, and Close (OHLC) for a given trading period. Figure 2. shows the "Adj. Close" history of the data.

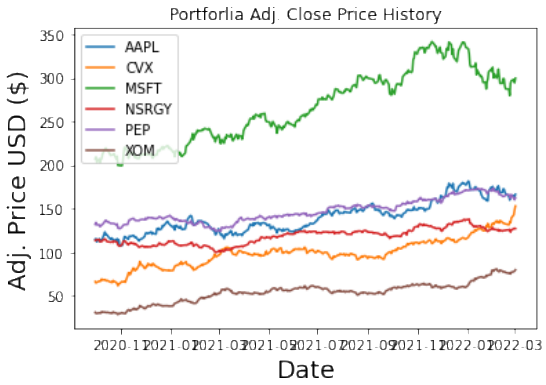

Figure 2. Adj. Close History of the Dataset.

C. *Modern Portfolio Theory (MPT)*

By changing the asset distributions, MPT seeks to optimize portfolio expected return under a specific level of risk or reduce the risk for a given expected return. Modeling a portfolio as a mixture of risky assets, MPT represents an asset's return as an elliptically distributed random variable, with risk as the standard deviation of return and a portfolio's return as the weighted sum of the returns of each asset. To harness the power of diversification, this study computes the mean-variance efficient frontier as a quadratic programming problem (or convex optimization problem), assuming that the constraints (quantifying risk with the asset return covariance matrix) are not too complicated.

First, this study selects a one-parameter family of efficient allocations among all possible combinations of assets; then, it performs a one-dimensional search, looking for the best among the efficient allocations numerically, with the help of the Sharpe Ratio, defined as

$$Sharpe\ Ratio = \frac{R_p - R_{rf}}{\sigma_p}. \quad (1)$$

R_p is the expected portfolio return, R_{rf} is the risk-free return rate, and portfolio volatility (standard deviation). Sharpe ratio measures the portfolio returns, adjusted by risk. The maximum Sharpe Ratio allocation represents the best portfolio under the above measurement.

PyPortfolioOpt [7] is an excellent open source tool to address the aforementioned convex optimization problem. This Python library contains numerous financial portfolio optimization techniques, including mean-variance optimization (MVO). MVO often results in many efficient assets getting negligible weights, which may not be ideal for diversification purposes. To get more non-negligible weights, the model in this paper uses a parameter γ to penalize small weights of objective functions. Then we compare the result from PyPortfolioOpt with the one from Monte Carlo Simulation, where there are different expected returns and expected volatility under various combinations of allocation weights.

D. *Portfolio Management Using Reinforcement Learning*

1) Reinforcement Learning Experiment Environment

a) Noise Generation Function

In Reinforcement Learning, the explore-exploit trade-off is fundamental. The model sometimes has to make bad choices to get optimal learning results. Actions could adopt some noises (typically uncorrelated mean-zero Gaussian noise) for more exploration during the training time.

b) State

A state at any time t is

$$S_t = \begin{pmatrix} c_t^0 & c_t^1 & \dots & c_t^m \\ c_{t-1}^0 & c_t^1 & \dots & c_t^m \\ \vdots & \vdots & \dots & \vdots \\ c_{t-d}^0 & c_{t-d}^1 & \dots & c_{t-d}^m \end{pmatrix}. \quad (2)$$

c_i^j is the close price for asset j at time i. d is the length of the state's window, which is the number of past time steps deemed relevant. The elements with superscripts 0 in the first column are hardcoded as one, representing cash data. This study denotes the z-normalized log return of asset j at the time i as

$$R_i^j = log(c_i^j) - log(c_{i-1}^j). \quad (3)$$

c) Action

The action in this problem is the weights of assets allocation in cash and m assets, with the sum of these weights as one.

$$A_t = w_t = (w_t^0, w_t^1, \dots, w_t^m), \sum_{i=0}^{m} w_t^i = 1. \quad (4)$$

d) Transaction Costs

Due to the price movement, the portfolio weight vector changes when buying or selling an asset. This study uses a transaction remainder factor μ_t to measure the value the portfolio shrinks due to reallocation.

$$\mu_t = c \sum_{j=1}^{m} |A_t'^j - A_t^j|. \quad (5)$$

where A' represents the new portfolio weightings and c is the transaction cost rate [8].

e) Reward

The profit at time timestamp T is

$$p_T = \prod_{t=1}^{T} c_t \cdot w_{t-1}. \quad (6)$$

Adding transaction costs gives us

$$p_T = \prod_{t=1}^{T} (1 - \mu_t) c_t \cdot w_{t-1}. \quad (7)$$

To avoid the sparsity of rewards, this study takes the logarithm of equation (7):

$$log(p_T) = \sum_{t=1}^{T} log(\mu_t c_t \cdot w_{t-1}). \quad (8)$$

Thus, we have $log(\mu_t c_t \cdot w_{t-1})$ reward for each timestamp.

2) Model Architecture

TD3 is the DRL model used for this study. Figure 3 depicts the overall structure of the model. In keeping with the characteristics of portfolio optimization, TD3 performs well with continuous action space, partial observability, and multidimensional data. TD3 addresses the overestimation bias

in DDPG by employing a pair of critic networks that uses the smallest value of the two when forming targets, using delayed updates on the actor-network compared to the value network to achieve a more stable and efficient training, and adding clipped noise to the selected action when calculating the targets in order to reduce the variance of target values when updating the critic. TD3 employs a replay buffer for efficient learning using small batches that reduce network update correlation and prevent forgetting. The predictor is based on work by Jiang et al. The model extracts patterns from numerical stock price histories using a Long Short-Term Memory (LSTM) network [9].

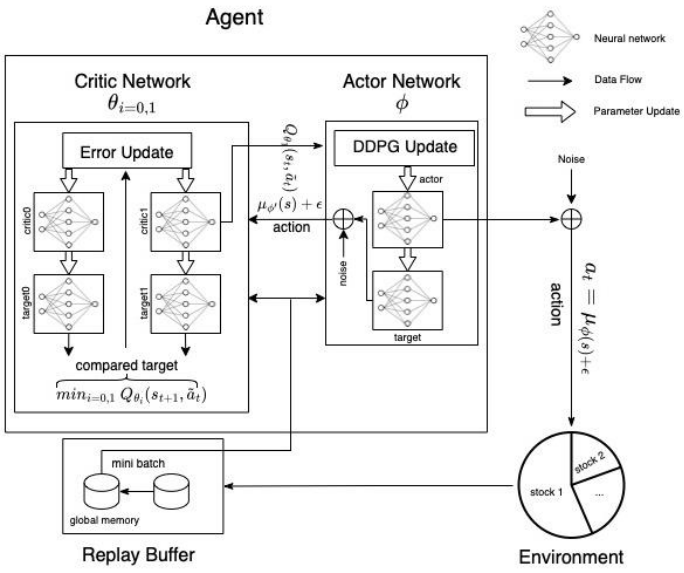

Figure 3. TD3 Architecture.

See Figure 4 and Table I for the information on actor and critic networks.

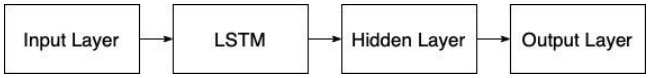

Figure 4. Network Architecture.

TABLE I. NETWORK DIMENSION.

Network	Dimension		
	Input Layer	***Hidden Layer***	***Output Layer***
Actor Network	$N \times (W + F)$	$N \times 32$	N
Critic Network	$N \times (W + F) \times 1$	$N \times 32$	1

III. EXPERIMENTS RESULTS AND DISCUSSION

A. *Experimental Setup*

The data for this study are collected from Yahoo Finance as described in Section II-B. See Table II for the statistics of the dataset for the DRL model.

TABLE II. DATASET STATISTICS.

Training Data		**Testing Data**	
Date Range	***Steps***	***Date Range***	***Steps***
2012/03/28 to 2020/09/29	2140	2020/09/29 to 2022/03/03	358

The date range for the MPT model is the same as the DRL model.

The training and testing split has a ratio of six to one. The model is trained across 400 epochs, each containing 1000 steps. Set a random starting point for each epoch's beginning. Uniformly sample a mini-batch of size 64 from the replay buffer. This study chooses these tunable hyper-parameters per past research to harness hardware resources at the computer architecture level [10].

The MPT models serve as baseline models. Then test DDPG and TD3 models with window-size 3, 7, and 14 (semi-weekly, weekly, and bi-weekly) under the same conditions.

B. *Results*

1)MPT

Figure 4, Figure 5, Table III, and Table IV present the corresponding experimental results.

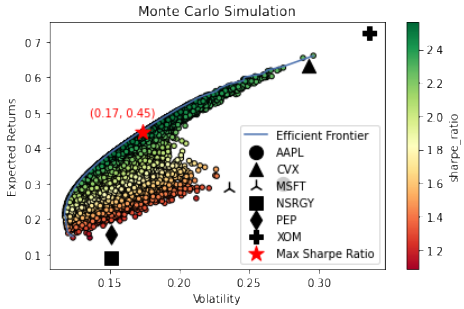

Figure 4. Monte Carlo Simulation.

TABLE III. MONTE CARLO SIMULATION RESULTS.

Performance			**Weights**					
Returns	***Volatility***	***Sharpe Ratio***	***APPL***	***CVX***	***MSFT***	**NSRGY**	***PEP***	***XOM***
44.59%	17.40%	256.25%	7.45%	22.5%	24.39%	7.68%	12.46%	25.45%

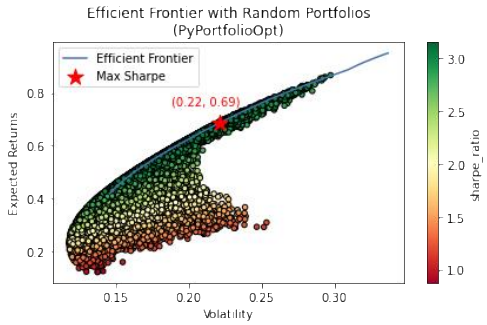

Figure 5. Efficient Frontier using PyPortfolioOpt.

TABLE IV. PYPORTFOLIOOPT RESULTS.

Performance			Weights					
Returns	*Volatility*	*Sharpe Ratio*	*APPL*	*CVX*	*MSFT*	NSRGY	*PEP*	*XOM*
68.32%	22.06%	247%	11.61%	31.19%	11.83%	2.70%	5.62%	37.07%

2) DRL

Figure 6 and Figure 7 show the training and testing results for the DRL model, respectively.

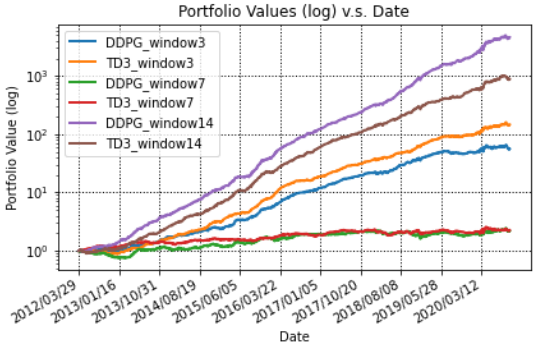

Figure 6. Training Results.

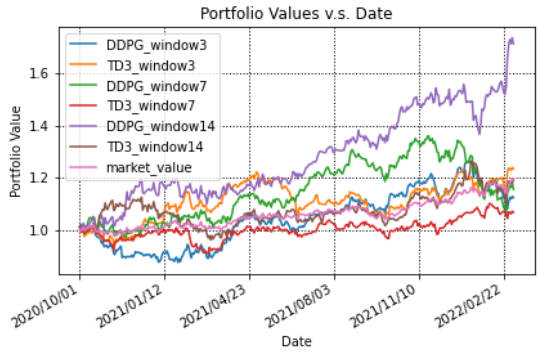

Figure 7. Testing Results.

Because DRL operates in continuous action space, using a pie chart (Figure 8) that illustrates the average allocation weights of assets in the dataset is a reasonable way to get insight into the allocation strategy.

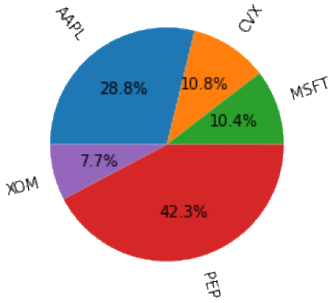

Figure 8. Average Allocation Weights.

3) Discussion

TD3 outperforms DDPG with window size three. Window size seven did not yield better results than window size three. However, the best model is DDPG with a window size of 14, which achieves 71.3% of portfolio value gain. Table III, IV, and Figure 8 show that the DRL model picked a different allocation weight than the MPT model. Note that the MPT model gets the optimal weights by looking over all the data, while the DRL model continuously changes the allocation weights through learning. Moreover, the MPT model has no transaction cost since the allocation is static for the whole time.

IV. CONCLUSION AND FUTURE WORK

This study utilized MPT and DRL models to optimize a portfolio. DRL models use DDPG and TD3 with varying observation window sizes. The DRL model's performance stands out among our models. There is a non-linear relationship between window size and performance. Note that this study uses a small set of assets to train the models, which can result in little training data.

In contrast to datasets such as SP500, the hyper-parameter adjustment for these datasets should have a finer granularity. One can also train a model using a much larger dataset and deploy the model onto a small dataset. Cloud computing services like Amazon Web Services (AWS) can alleviate the extended training time. In addition, the training and testing dataset split during the COVID-19 pandemic (March 2020). The unknown is the effect of COVID-19. A model that can address this unknown deviant phenomenon is worthy of further research. Additional research can investigate other opponent alternatives inside the architecture for the DRL model. As agent frameworks, Advantage Actor-Critical and Proximal Policy Optimization may be employed. This study employs a reward function that emphasizes the capital growth of a portfolio; a potential improved reward function can incorporate the Mean-Variance Theory for more volatility awareness.

REFERENCES

[1] H. Markowitz, "Portfolio Selection," The Journal of Finance, vol. 7, no. 1, pp. 77-91, 1952.

[2] Z. Jiang et al., "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem," 2017.

[3] J. Petterson, and A. Gibson, "Deep Learning-A Practitioner's Practice," 2017, ISBN 9781491914250.

[4] T. L. Meng and M. Khushi, "Reinforcement Learning in Financial Markets," Data, vol. 4, p.110, 2019.

[5] S. Hegde, V. Kumar, and A. Singh, "Risk aware portfolio construction using deep deterministic policy gradients," 2018.

[6] S. Fujimoto, H. Van, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," 35th International Conference on Machine Learning, ICML 2018, pp. 257-2601, 2018.

[7] R. Martin, "PyPortfolioOpt: portfolio optimization in Python," Journal of Open-Source Software, vol. 6, 3066, 2021.

[8] Z. Liang, H. Chen, J. Zhu, K. Jiang, and Y. Li, "Adversarial Deep Reinforcement Learning in Portfolio Management," 2018.

[9] S. Hochreiter, and S. Jürgen, "Long Short-Term Memory," Neural Computation, 9, 1997, pp. 1735-1780.