Portfolio Optimization: Deep Reinforcement Learning vs. Classical Methods

Tsoy Roman
Department of Industrial Engineering
UNIST, South Korea

Nguyen Minh Duc
Department of Computer Science and Engineering
UNIST, South Korea

Almas Dossay
Department of Industrial Engineering
UNIST, South Korea

Kim Artem
Department of Industrial Engineering
UNIST, South Korea

Abstract

In this paper, we performed portfolio optimization using 2 different approaches. First, we used classical methods for portfolio optimization, namely mean-variance optimization, efficient mean-semivariance optimization, Black-Litterman allocation, and hierarchy. For the second approach, we used deep reinforcement learning, and compared optimization methods based on expected returns, volatility, and Sharpe ratio. While advanced classical methods such as hierarchical risk parity show good performance based on expected returns and volatility, the reinforcement learning method applied outperforms classical methods based on the Sharpe ratio.

1 Introduction

Portfolio optimization is an important task for private investors as well as investor companies to construct a portfolio that usually will maximize the return while minimizing the risk. The complexity of a dynamic financial market is very unpredictable and the solution might lie in a highly computational approach of machine learning to optimize a portfolio to a full extent. However, the traditional methods based on mean-variance optimization have been extensively used for decades and have shown a good performance for investors.

In this project, we focus on the comparison and evaluation of the performance of both the traditional mean-variance optimization methods and the deep reinforcement learning approach for portfolio optimization. The conventional approach of mean-variance optimization relies on statistical methods and efficient frontier theory, while DRL, leverages neural networks and sequential decision-making to learn optimal strategies.

Problem and Motivation The foundation of the market is millions of interrelated decisions of humans that predetermine the movement of the price of assets and have a lot of hidden factors. Classical methods might struggle to capture the intricate relationships between assets and adapt swiftly to changing market conditions. Deep reinforcement learning has the ability to learn from sequential data and optimize decision-making through interactions with the environment. The motivation behind this comparison is to explore if DRL can outperform traditional methods in constructing robust portfolios under the dynamic nature of financial markets.

Dataset For this comparative study, we used the historical financial data of the Dow Jones Industrial Average(DJIA) index which includes 30 stocks of the most prominent American companies. For both methods the trading period was from 2021.01.01 to 2023.11.15 and for the training of DRL we used datasets with different starting points to consider the performance of DRL to an input of just new data and old with new data.

2 Related Work

Classic methods of optimization Classical methods of portfolio optimization such as mean-variance optimization are still in use, as they lie in the foundation of many advanced portfolio optimization methods. However, it has some limitations, particularly, assumptions and sensitivity to input data. Thus, in finance, the mean-variance optimization method is extended and modified to improve performance. Black-Litterman allocation[7] is an example of the improved version of mean-variance optimization, as it combines the subjective views of an investor with equilibrium market returns. Nowadays, for practitioners, the best approach is to combine traditional algorithms for portfolio optimization with alternative approaches based on their investment goals, market conditions, and risk tolerance. As, based on investment objectives, it can be a risk-averse or risk-seeking approach.

Deep Reinforcement Learning for Portfolio Optimization Deep reinforcement learning has huge potential for successful trading and portfolio optimization activities as a financial market simulator. However, it might not work well in the real world due to the high complexity and dynamic nature of the financial market. The historical data contains a lot of noise and it can affect the prediction of the DRL agents degrading the performance of the portfolio. For that reason, the real-world financial market should be examined by multiple and diverse DRL agents, which increases demand for a universe of market environments and imposes a challenge on simulation speed. The FinRL-Meta[4] is an example of the universe of market environments for data-driven financial reinforcement learning which separates financial data processing from the design pipeline of DRL-based strategy, provides open source data engineering tools for financial big data and provides hundreds of market environments for various trading tasks, and enables multiprocessing simulation and training by exploiting thousands of GPU cores. Despite the difficulties, numerous attempts have been made to tackle this problem. Liu et al. [5] proposed the use of the Actor-Critic Network to optimize the trading strategy. Yang et al. [8] introduced a novel Ensemble Strategy for automated stock trading. Briola et al. [1] introduce the first end-to-end Deep Reinforcement Learning framework for high-frequency trading.

3 Methodology

3.1 Classical Methods

Mean-Variance Optimization Mean-variance optimization focuses on constructing portfolios that balance expected returns and risk. Mean refers to the expected returns, calculated as the weighted average of individual asset returns. Variance assesses the risk, measured by the dispersion or variability of asset returns around their mean. So this optimization aims to plot these portfolios on an efficient frontier, identifying those that offer the highest return for a given level of risk or the lowest risk for a specified return, thereby optimizing the risk-return trade-off. This method is implemented by the principle of diversification, implying that a well-balanced portfolio can mitigate risk more effectively than the selection of individual assets based on returns alone.

Mean-Semivariance Optimization Efficient semi-variance optimization in stock portfolio management is an approach that specifically targets downside risk, distinctively focusing on the lower half of the return distribution. This method measures the semi-variance, which quantifies the variability of stock returns falling beneath a defined threshold, such as the portfolio's mean return. The primary advantage of this approach lies in its emphasis on minimizing potential losses, aligning with the risk profiles of more conservative investors who prioritize downside protection over total return volatility. Efficient semi-variance optimization is particularly applicable in the context of stock markets, where return distributions can exhibit asymmetry or significant skewness. By employing this strategy, we can create portfolios that are more stable even in negative market movements, thereby enhancing the stability and downside risk management of their investment portfolios.

Black-Litterman Allocation Black-Litterman allocation[7] approach is based on computing a weighted average of the prior estimates of return and particular views held by investors for each of the assets. The main advantage of this approach is that the weighting is dictated by the confidence levels given by the investor for each view he sets. So, it allows for each investor to create his own personalized strategy for portfolio optimization and it helps to overcome the problem of input-sensitivity. As the end goal, getting Black-Litterman posterior returns results in much more stable portfolios than using mean-historical return.

Hierarchical Risk Parity Hierarchical Risk parity[2] is a modern portfolio optimization method that takes into account the hierarchical structure of asset classes and their correlations. Its main aim is to create a well-diversified portfolio by allocating weights to asset classes based on their risk contributions. A risk parity investment strategy is aimed at allocating weights to assets based on their risk contributions rather than their market values. The final goal of this strategy would be to make each asset contribute an equal amount of risk. While hierarchical risk parity[2] considers the hierarchical structure of asset classes. Meaning that 2 assets from the same class would be more correlated with each other rather than 2 assets from different classes. The end goal would be by considering the hierarchical structure of asset classes, to create a more robust and diversified portfolio, where it allocates weights to asset classes based on their risk contributions at different levels of the hierarchy.

3.2 Deep Reinforcement Learning

3.2.1 Problem Formulation

Stochastic Optimal Control Due to the stochastic and interactive nature of the stock trading environment, we decided to model the problem as a Stochastic Optimal Control problem. In particular, the problem settings can be modeled as a Markov Decision Process. Considering a stock trading environment where a trading agent wants to maximize his wealth by actively optimizing his portfolio consisting of the market's stocks. We assume a small commission fee of \$0.001 for both buy and sell actions. Furthermore, the agent cannot buy stock shares if that action results in a negative account balance and he starts the trading period with no shares of any stocks. Formally, we define our trading environment as follows:

- State $S = \left\{ [b, \mathbf{h}, \mathbf{c}, \mathbf{i}]^\intercal \;\middle|\; b \in \mathbb{R}_+, \mathbf{h} \in \mathbb{N}^d, \mathbf{c} \in \mathbb{R}_+^d, \mathbf{i} \in \mathbb{R}^{kd} \right\}$: a set that includes the remaining balance b, the number of holdings of each stock $\mathbf{h}$, the closing price of stocks $\mathbf{c}$, and the technical indicators of each stock $\mathbf{i}$, where $d \in \mathbb{N}$ and $k \in \mathbb{N}$ denotes the number of stocks in the market and the number of technical indicators of each stock respectively. Note that $[b, h, c, i]^\intercal \in \mathbb{R}^p$, and $p = 1 + (2 + k)d$. Let $s_t \in S$ be the state of the trading environment at time t.
- Control (action) $A = \left\{ \mathbf{a} \;\middle|\; \mathbf{a} \in \mathbb{Z}^d \right\}$: a set of actions over all d stocks.
- Reward function $R(t, s_t, \pi_t)$: the change in portfolio value when action π_t is taken over state s_t at time t. The portfolio value is the total equities in all holding stocks and the balance, i.e., $\mathbf{c}^\intercal \mathbf{h} + b$. The change can be modeled mathematically as $V(s_{t+1}) - V(s_t)$, where $V(s) = s_c^\intercal s_h + b$ is the portfolio value of state s.
- Action-Value function $Q_\pi(s_t, a_t) = \mathbb{E}\left[R(t, s_t, a_t) \middle| s_t, a_t \right]$: the expected reward obtained by taking action a_t over the state s_t.
- Policy $\pi(s)$: the trading strategy given state s. This can be modeled as a probability distribution over the action a at state s.

The dynamic of the trading environment In each state, there are three possible action types that can be taken:

- Sell: For each stock i, there are $\mathbf{q}[i] \in [1, \mathbf{h}[i]]$ shares can be sold from the current portfolio. Note that $\mathbf{q}[i]$ is an integer. In this case, the holdings become $\mathbf{h}_{t+1}[i] = \mathbf{h}_t[i] - \mathbf{q}[i]$, and the balance would be $b_{t+1} = b_t + \mathbf{c}_t[i]\mathbf{q}[i]$.
- Hold: For each stock i, $\mathbf{q}[i] = 0$, hence, no change in the holdings and balance.
- Buy: For each stock i, there are $\mathbf{q}[i]$ shares can be bought to the current portfolio. Note that $\mathbf{q}[i]$ should be an integer. In this case, $\mathbf{h}_{t+1} = \mathbf{h}_t + \mathbf{q}[i]$, and the balance would

be $b_{t+1} = b_t - \mathbf{c}_t[i]\mathbf{q}[i]$. For consistency, we can denote $\mathbf{q}[i] := -\mathbf{q}[i]$ and the balance transition would be the same as selling:

$$\begin{cases} \mathbf{h}_{t+1} = \mathbf{h}_t - \mathbf{q} \\ b_{t+1} = b_t + \mathbf{c}_t{}^\intercal \mathbf{q} \end{cases} .$$

Trading Goal We aim to maximize the expected portfolio value $V(s_t^\pi)$ by optimally optimize the portfolio (finding the best π_t) over N periods:

$$\max_\pi \mathbb{E}\left[V(s_N^\pi)\middle|\mathcal{F}_0\right].$$

We define the value function as follows:

$$H(t,s) = \max_{\pi_t,\pi_{t+1},\dots,\pi_N} \mathbb{E}\left[V(s_N^\pi)\middle|\mathcal{F}_t\right].$$

This can be rewritten as

$$H(t,x) = \max_{\pi_t,\pi_{t+1},\dots,\pi_N} \mathbb{E}\left[V(s_t) + \sum_{j=t}^{N-1} R(j,s_j,\pi_j) \middle| s_t = x\right]. \tag{1}$$

From the Dynamic Programming Principle,

$$H(t,x) = \max_{\pi_t}\left\{V(x) + \mathbb{E}\left[R(t,x,\pi_t)\middle| s_t = x\right] + \mathbb{E}\left[H(t+1, s_{t+1}^{\pi_t}) \middle| s_t = x\right]\right\}. \tag{2}$$

Our objective is to find $H(0,s_0)$ and its corresponding maximizer $\pi^* = \{\pi_t^* \mid t = 0,\dots,N-1\}$, i.e., the maximum account balance of the agent after taking the optimal portfolio optimizations over N periods.

3.2.2 Actor Critic Methods

One possible solution to the aforementioned optimization problem is to model the dynamic of the stocks' closing prices mathematically and solve it using Dynamic Programming. However, it is difficult to derive an accurate mathematical model for the movement of the stocks. Therefore, one could utilize the Reinforcement Learning technique to automatically learn the value function and the optimal policy. Among the family of Reinforcement Learning algorithms, Actor-Critic methods [3] are adopted in many algorithms thanks to their performance and their ability to reduce variance in Reinforcement Learning. An actor-critic algorithm consists of two parts: an *Actor* (policy network) controlling how the agent behaves, and a *Critic* (value network) measuring how good the taken action is. In this paper, we employ an Actor-Critic algorithm to maximize the investment return.

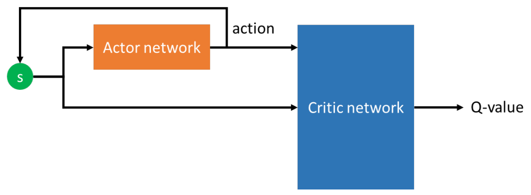


Figure 1: Actor-Critic Architecture

Actor-Critic Architecture Figure 1 summarizes the overall architecture of the Actor-Critic method. The Actor-Critic algorithm consists of two 2 neural networks: a policy network (the Actor) $\pi_\theta(s)$ parameterized by θ, and a value network (the Critic) $\hat{q}_w(s,a)$ parameterized by w. For each episode, the Actor obtains the current state from the environment and produces an action a_t, then the Critic takes both the current state and the action to produce a value "criticizing" the taken action. The action a_t is applied to the environment to transit to the next state s_{t+1} and to obtain the reward $r_t = R(t,s_t,a_t)$. The Actor-Network's parameter (θ) is optimized via the equation

$$\Delta\theta = \alpha\nabla_\theta\left(\log \pi_\theta(s_t)\right)\hat{q}_w(s_t,a_t). \tag{3}$$

After being optimized, the Actor receives the next state s_{t+1} to produce the next action a_{t+1}, which will then be used for updating the Critic-Network via the equation

$$\Delta w = \beta\left[R(t+1,s_{t+1},a_{t+1}) + \gamma\hat{q}_w(s_{t+1},a_{t+1}) - \hat{q}_w(s_t,a_t)\right]\nabla_w\hat{q}_w(s_t,a_t). \tag{4}$$

The Actor and Critic Networks Conventional policy and value networks are constructed from Multilayer Perceptron (MLP) thanks to its universal approximation property. However, in this paper, we investigate another possible construction of the two networks with convolution operators. Deep Convolution Neural Network (Deep CNN) has achieved state-of-the-art performance in multiple Computer Vision tasks thanks to its superior feature extraction from the data. In our networks, we employ multiple 1D Convolution Networks to extract different hidden features from the environment's state. Please refer to Figure 2 for an overview of the network.

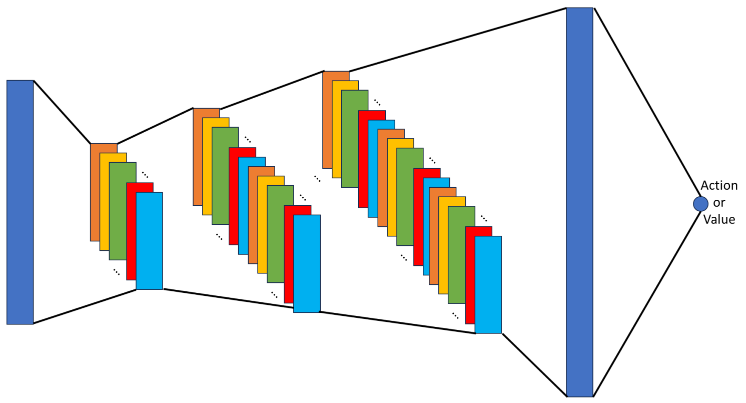


Figure 2: Network Architecture

4 Implementation

Mean-Variance Optimization Our portfolio optimization analysis employed the mean-variance optimization framework augmented with L2 regularization to construct a well-diversified portfolio across several sectors. Leveraging historical data and the CAPM model for expected returns, we calculated a Ledoit-Wolf shrinkage covariance matrix to underpin our allocation strategy. The resulting portfolio, as illustrated in the pie chart below, optimally balances risk and return, aiming for a target volatility of 25 percent. The diversified allocation across 30 stocks, with sector constraints duly applied, stands as a testament to the model's robustness, aiming to maximize returns within the specified risk parameters.

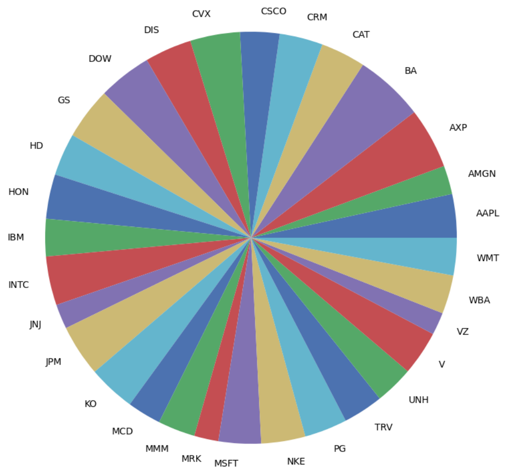


Figure 3: Prior views

Mean-Semivariance Optimization In the implementation of efficient semi-variance optimization, we prioritized minimizing downside risk by employing a semi-covariance matrix, which better captures the portfolio's susceptibility to negative returns compared to traditional covariance. By setting a return target of 11 percent(below the maximum possible return) the EfficientSemivariance class from PyPortfolioOpt was utilized to fine-tune our portfolio, adhering to our risk constraint while aiming for optimal performance. The heatmap illustrates the semi-covariance among assets:

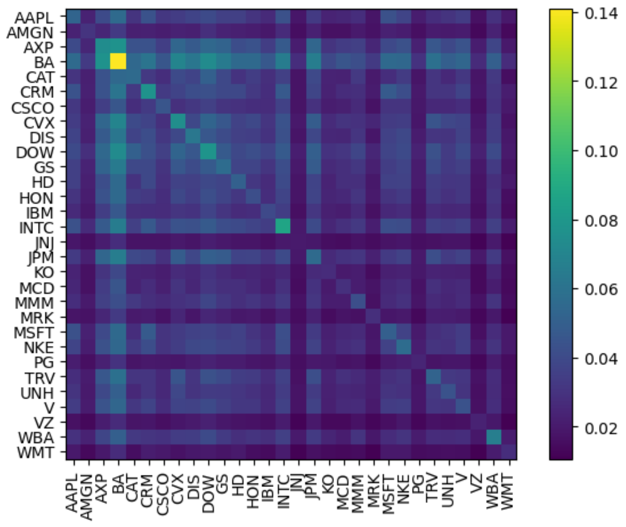


Figure 4: Prior views

Guiding our allocation to mitigate underperformance during market downturns, ultimately resulting in a portfolio with a lower Sortino ratio and higher semi-variance when using the heuristic approach, suggesting a more conservative risk profile.

Black-Litterman Allocation For Black-Litterman Allocation, the first step is constructing prior returns based on the covariance matrix, and we got the following prior allocation of stocks:

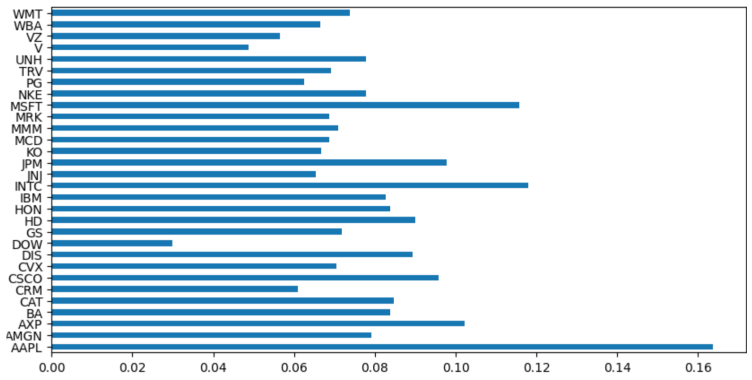


Figure 5: Prior views

By providing views and confidence scores for each of the stocks, they were combined together with prior estimates, to get posterior estimates. Next, all 3: prior estimates, views of investors(us), and posterior estimates were visualized:

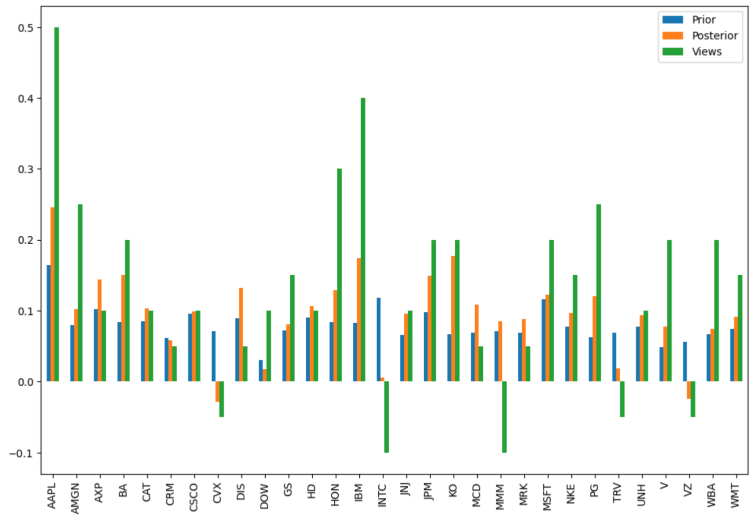


Figure 6: Posterior views

Hierarchical Risk Parity As, the first step, in this paper, the pairwise correlation matrix of asset returns was calculated. Based on this correlation matrix, hierarchical clustering with a single linkage method is done to distribute assets in the hierarchy. The following dendrogram was obtained:

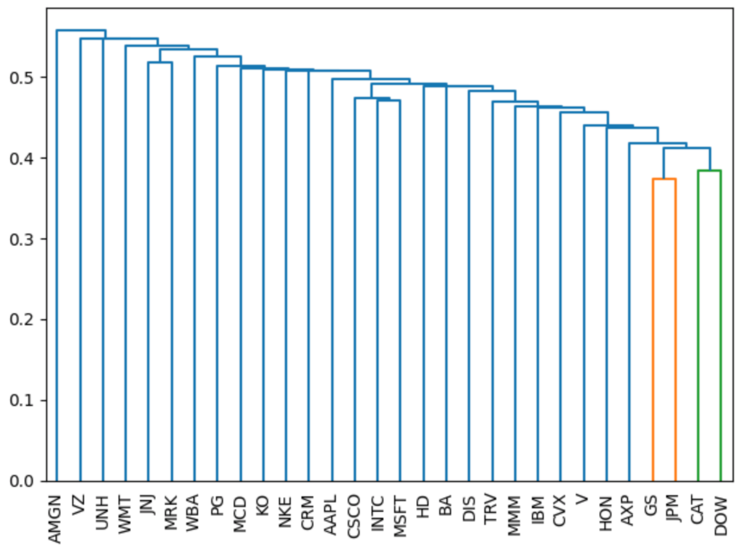


Figure 7: Dendogram

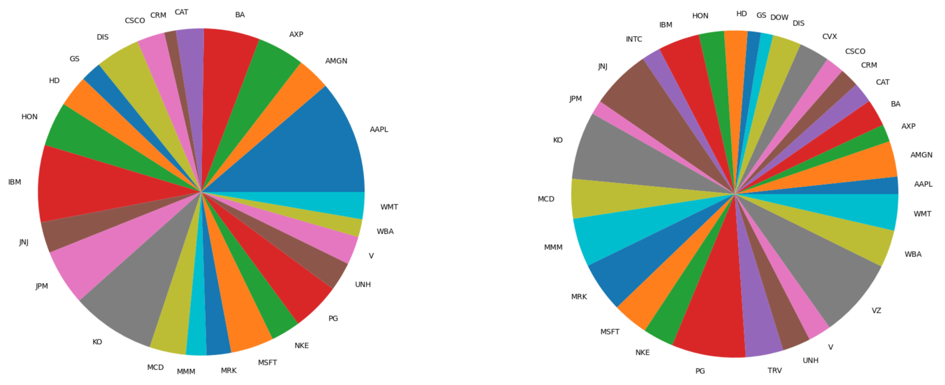
Based on the obtained clusters, weights were allocated based on the risk contributions of each asset class in 3 steps. 1st step is to calculate the inverse variance for each asset. Next, the risk contributions of each asset are calculated. And finally, weights are allocated based on risk contributions at different levels of the hierarchy.

Deep Reinforcement Learning Following the work of Liu et al. [6], we utilized the open-source library FinRL to implement the environment and the Actor-Critic algorithm. As for the technical indicators, we utilized well-known attributes: Moving Average Convergence Divergence, Bollinger Bands, Relative Strength Index, Commodity Channel Index, Directional Movement Index, and Close Smooth Moving Average. Figure 10 shows our implementation of the trading environment with `StockTradingEnv` package from FinRL. As for the network architecture, we inherit the

`ActorCriticPolicy` class from `stable_baselines3` and implement our custom CNN model, please refer to Figure 11 for detailed implementation.

5 Evaluation and Results

Finally, for each of the methods, we obtained a posterior allocation of weights as follows:



(a) Black-Litterman Allocation

(b) Hierarchical Risk parity

Figure 8: Posterior weights allocation

For evaluation, we split the data into 2 parts: In-sample data from 2010 to 2021 for training and optimizing portfolio weights and out-sample data from 2021 to 2023 for benchmarking the five methods. Running all of the aforementioned methods, their investment returns were obtained:

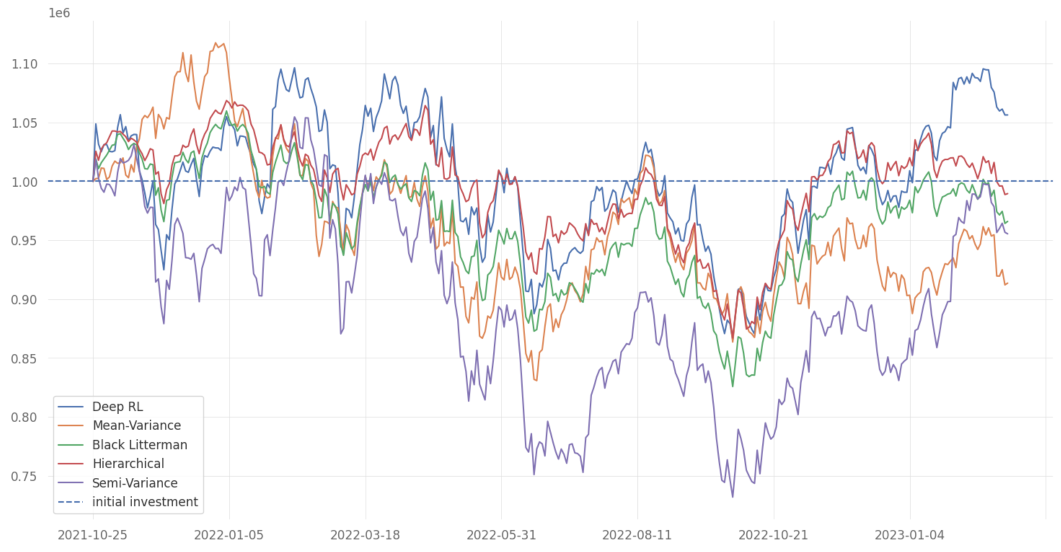


Figure 9: The Investment Returns of the Five Methods

Table 1: Evaluation

Methods	Sharpe ratio	Volatility	Expected return
Mean-variance optimization	-0.17	0.24	-0.027
Mean-semivariance optimization	0.036	0.30	-0.014
Black-Litterman allocation	-0.05	0.18	-0.010
Hierarchical risk parity	0.03	**0.16**	-0.0031
Deep Reinforcement Learning	**0.29**	0.24	**0.0163**

Based on Table 1, it can be concluded Deep Reinforcement learning shows the best performance based on the Sharpe ratio and Expected return. While, hierarchical risk parity shows best performance based on volatility, which is not surprising, as the main aim of this algorithm is to build no risk contributions of each asset based on hierarchical levels.

6 Conclusion

In the main conclusion, Deep reinforcement learning tends to show the best performance compared with classical methods for portfolio optimization. However, advanced classical methods like Black-Litterman allocation and Hierarchical Risk parity can compete with deep reinforcement learning methods in some metrics. Overall, it can be concluded that classical methods perform quite well for short-term periods, while deep reinforcement learning algorithm starts to take over when the trading period is long. So, what strategy and method to use, would depend on the trading period, objectives, and personal goals of the trader.

7 Supplementary

```
# Environment space
stock_dimension = len(train.tic.unique())
state_space = 1 + (2 + len(INDICATORS))*stock_dimension
# Commision fee and initial stock shares
buy_cost_list = sell_cost_list = [0.001] * stock_dimension
num_stock_shares = [0] * stock_dimension
# Environment settings
env_kwargs = {
    "hmax": 100,
    "initial_amount": 1000000,
    "num_stock_shares": num_stock_shares,
    "buy_cost_pct": buy_cost_list,
    "sell_cost_pct": sell_cost_list,
    "state_space": state_space,
    "stock_dim": stock_dimension,
    "tech_indicator_list": INDICATORS,
    "action_space": stock_dimension,
    "reward_scaling": 1e-4
}
# The environment
e_train_gym = StockTradingEnv(df = train, **env_kwargs)
```

Figure 10: Environment Settings

```python
class CustomA2CPolicy(ActorCriticPolicy):
    def __init__( self,

                  ⋮
        self.actor = nn.Sequential(
            nn.Conv1d(in_channels=1, out_channels=16, kernel_size=3,
    stride=2), nn.ReLU(),
            nn.Conv1d(in_channels=16, out_channels=32, kernel_size=3,
    stride=2), nn.ReLU(),
            nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3,
    stride=2), nn.ReLU(),
            nn.Flatten(),
            nn.Linear(64 * 35, 64), nn.ReLU(),
            nn.Linear(64, action_space.shape[0]), nn.Tanh()
        )
        self.critic = nn.Sequential(
            nn.Conv1d(in_channels=1, out_channels=16, kernel_size=3,
    stride=2), nn.ReLU(),
            nn.Conv1d(in_channels=16, out_channels=32, kernel_size=3,
    stride=2), nn.ReLU(),
            nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3,
    stride=2), nn.ReLU(),
            nn.Flatten(),
            nn.Linear(64 * 35, 64), nn.ReLU(),
            nn.Linear(64, 1)
        )
    def forward(self, obs):
        # Forward pass, please check section 3.2

                  ⋮
        return actions, values, log_probs
```

Figure 11: Network Implementation

Algorithm 1 Actor-Critic Algorithm

```
for each episode do
    while state has not ended do
        $a_t \leftarrow \pi_\theta(s_t)$
        $v_1 \leftarrow \hat{q}_w(s_t, a_t)$
        $s_{t+1} \leftarrow$ State.apply($a_t$)
        $\pi_\theta$.optimize($a_t, v_1$)            ▷ Optimize the Actor's parameters based on equation 3
        $a_{t+1} \leftarrow \pi_\theta(s_{t+1})$
        $v_2 \leftarrow \hat{q}_w(s_{t+1}, a_{t+1})$
        $r_{t+1} \leftarrow R(t+1, s_{t+1}, a_{t+1})$
        $\hat{q}_w$.optimize($r_{t+1}, v_1, v_2$)    ▷ Optimize the Critic's parameters based on equation 4
    end while
end for
```

References

[1] Antonio Briola, Jeremy D. Turiel, Riccardo Marcaccioli, and Tomaso Aste. Deep reinforcement learning for active high frequency trading. *ArXiv*, abs/2101.07107, 2021.

[2] Christian Erik Kaae, Joonas Aleksanteri Karppinen, and Johan Stax Jakobsen. Hierarchical risk parity–a hierarchical clustering-based portfolio optimization.

[3] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[4] Xiao-Yang Liu, Ziyi Xia, Jingyang Rui, Jiechao Gao, Hongyang Yang, Ming Zhu, Christina Dan Wang, Zhaoran Wang, and Jian Guo. Finrl-meta: Market environments and benchmarks for data-driven financial reinforcement learning. *NeurIPS*, 2022.

[5] Xiao-Yang Liu, Zhuoran Xiong, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading. *NeurIPS Workshop on Deep Reinforcement Learning*, 2018.

[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. *Deep RL Workshop, NeurIPS 2020*, 2020.

[7] Stephen Satchell and Alan Scowcroft. A demystification of the black-litterman model: Managing quantitative and traditional portfolio construction. In *Forecasting Expected Returns in the Financial Markets*, pages 39–53. Elsevier, 2007.

[8] Hongyang Yang, Xiao-Yang Liu, Shanli Zhong, and Anwar Walid. Deep reinforcement learning for automated stock trading: an ensemble strategy. *Proceedings of the First ACM International Conference on AI in Finance*, 2020.