REINFORCEMENT LEARNING FOR CARDINALITY CONSTRAINED PORTFOLIO OPTIMIZATION

FINAL REPORT

Rafay Kalim
Department of Mechanical and Industrial Engineering
University of Toronto
Toronto, Canada
rafay.kalim@mail.utoronto.ca

Stefano Scaini
Department of Mechanical and Industrial Engineering
University of Toronto
Toronto, Canada
stefano.scaini@mail.utoronto.ca

Yan Pan Chung
Department of Mechanical and Industrial Engineering
University of Toronto
Toronto, Canada
normanyp.chung@mail.utoronto.ca

December 23, 2023

ABSTRACT

This report details our efforts to enhance Cardinality Constrained Mean-Variance Optimization (MVO) in portfolio management through the application of Reinforcement Learning, specifically utilizing online Proximal Policy Optimization (PPO) and Dueling Deep Q-Networks (Dueling DQN). Significant progress has been achieved in formulating state representations, defining action spaces, and implementing the RL algorithm. Challenges related to reward stability and convergence have been identified and effectively addressed. Preliminary experiments highlights the algorithm's potential to adapt and optimize portfolios compared to the baseline method. Looking ahead, the project aims to refine the model by fine-tuning parameters and conducting extensive testing on a larger real dataset. For further details and access to the project code, the GitHub repository can be found at https://github.com/rafayk7/rl-ccmvo.

1 Introduction

Mean Variance Optimization (MVO), pioneered by Markowitz in 1952, was the first such attempt to create mathematically optimal investment decisions [8]. This technique aims to balance the trade-off between risk and reward, subject to the investor's needs. The goal of MVO and all other portfolio optimization techniques is to obtain $\boldsymbol{\phi} \in \mathbb{R}^n$, a vector which contains the percentage of total wealth to allocate to each of the n assets. Here, 'optimal' is defined by the user's objective. This requires the estimation of $\boldsymbol{\mu} \in \mathbb{R}^n$, the reward measure, and $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$, the risk measure. Once we have a risk and reward measure, we can formulate an optimization problem to obtain $\boldsymbol{x}$. Letting λ be a risk-return tradeoff parameter, MVO is stated as

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} - \lambda \boldsymbol{\mu}^T \boldsymbol{x} \\ \text{s.t.} \quad & \mathbf{1}^T \boldsymbol{x} = 1, \\ & (\boldsymbol{x} \geq 0) \end{aligned} \tag{1}$$

A well-known disadvantage to MVO is its sensitivity to errors in the estimation of its input parameters, $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ [9]. This can lead to unstable portfolios. Instability in portfolio weights leads to high transaction costs, especially for investors with frequent rebalancing. In an effort to reduce MVO's sensitivity to its input parameters, and thus reduce

high transaction costs, the Cardinality Constrained MVO (CC-MVO) is proposed. CC-MVO attempts to encourage sparsity in the portfolio vector ϕ by the addition of the cardinality constraint $||\phi||_0 \leq K$. Here, K is an integer limit to the number of assets allowed to have a non-zero asset allocation. The l-0 norm is non-convex, however this can be formulated as a convex Mixed Integer Program to find the solution. However, Bienstock (1996) proved that in general, solving Quadratic Mixed Integer Programs like CC-MVO (program 2) is NP-Hard [3]. This implies that there is no guarantee that Mixed Integer Program solvers will converge in polynomial time, and that they do not scale well to large problems. This motivates the need for heuristics to solve CC-MVO.

2 Related Work

2.1 Heuristics for Cardinality Constrained Optimization

There have been a few attempts at creating heuristics and faster branch-and-bound algorithms to solve problems like CC-MVO. Bertsimas and Cory-Wright (2022) provide a Benders-type cutting plane algorithm, and two heuristics for warm-starting the algorithm and show their method significantly outperforms traditional branch-and-bound techniques [2]. Further, Streichert and Tanaka-Yamawaki (2006) formulate the portfolio optimization problem through a multi-objective lens, and propose an improved evolutionary algorithm-based heuristic to solve for optimal portfolios [12]. For a review of other heuristics for CC-MVO, we refer to Ruiz-Torrubiano et. al (2010) [10].

2.2 Machine Learning for Optimization

Recent advances in combinatorial optimization have led to the use of machine learning algorithms to automatically develop heuristics. Specifically, Dai et. al (2017) propose the use of reinforcement learning to solve combinatorial optimization problems over graphs [7]. Their proposed methods are able to achieve much better results than traditional optimization techniques.

2.3 Reinforcement Learning for Optimization

Reinforcement learning (RL) has emerged as a promising avenue for addressing portfolio optimization challenges, as evidenced by the works of Aboussalah et. al(2021)[1] and Sood et. al [11]. Aboussalah et. al conducted a thorough examination of five RL algorithms—Deterministic Policy Gradient (DPG), Stochastic Policy Gradient (SPG), Deep Deterministic Policy Gradient (DDPG), Trust Region Policy Optimization (TRPO), and Proximal Policy Optimization (PPO). Employing cross-sectional analysis, Aboussalah et. al scrutinized the performance of these algorithms in comparison to traditional Mean-Variance Optimization (MVO) techniques, offering a cross-sectional analysis of an interested stock and its peer stock at defined points in time. Concurrently, Sood et. al contributed to the field by introducing an RL algorithm that leverages the Sharpe Ratio as a reward metric for the RL agent over a specified time horizon. Despite the strides made by Aboussalah et. al and Sood, a notable gap in the literature persists which is the absence of studies exploring the integration of RL algorithms with the MVO approach in portfolio optimization.

3 Methodology

Our project will aim to solve the CC-MVO problem. In words, this problem can be stated as "Given n assets, which $K < n$ assets will provide the minimum variance portfolio?" Given an asset covariance matrix $Q \in R^{n \times n}$, an asset expected return vector $mu \in R^n$ and letting $x \in R^n$ represent a portfolio of assets, one such formulation of CC-MVO is

$$\begin{aligned} \min_{x} \quad & x^T Q x - \lambda \mu^T x \\ \text{s.t.} \quad & \mathbf{1}^{\boldsymbol{T}} x = 1 \\ & ||x||_0 = K \end{aligned} \tag{2}$$

Where $\mathbf{1} \in R^n$ represents a vector of ones. The use of the $l0$-norm here enforces the cardinality constraint. However, the $l0$-norm is non-convex and thus this program is not tractable to be solved through traditional methods. This can be re-written as a MIQP by introducing an integer decision variable $z \in \{0, 1\}$. The program then becomes:

$$\begin{aligned} \min_{x} \quad & x^T Q x - \lambda \mu^T x \\ \text{s.t.} \quad & \mathbf{1^T} x = 1 \\ & \mathbf{1^T} z = K \\ & x \leq z \end{aligned} \tag{3}$$

3.1 Overview

We aim to follow Dai et. al (2017) and propose a reinforcement learning (RL) algorithm to solve this problem [7]. The RL agent acts as a "stock chooser" and is rewarded based on the performance of the portfolio. The current state is represented by mu, Q and x_t (the current portfolio at time t). For the base state, we will have the current portfolio be empty, i.e. $x_0 = \{\}$. The action that the agent can take is to choose one asset out of the n available assets to add to the existing portfolio (state). With regards to our reward, we will have two rewards depending on the progress by the agent. When $card(x^t) < K$ (i.e. when the problem is not yet finished), we can not simply use the MVO objective value as a reward as adding an additional asset will always improve the objective value, and thus the agent may not be as sensitive to choose the correct assets. So, instead we propose to use the derivative of the objective function with respect to the added asset. To that end, the MVO objective function is given by

$$f_{MVO}(x) = x^T Q x - \lambda \mu^T x$$

Differentiating the negative of this with respect to x_i, we get

$$-\frac{df_{MVO}}{dx_i} = -(Qx)_i + \lambda \mu_i \tag{4}$$

This will be our reward when $card(x_t) < K$. The negative sign is added as the agent will maximize its reward, but a lower value for the derivative is better. When we reach our desired cardinality, our reward can simply be $-f_{MVO}(x_t)$ (as the agent will maximize reward).

A high-level overview of our proposed algorithm is provided in Algorithm 2.

Algorithm 1 Reinforcement Learning Algorithm Pseudocode

Inputs:
  $\boldsymbol{\mu}$, $\boldsymbol{Q}$, K
Initialize:
  $x_0 \leftarrow \{\}$
while $card(\boldsymbol{x_t}) < K$ **do**
  Agent performs an action a_i
  $x_{t+1} \leftarrow x_t \cup \{a_i\}$
  Solve the relaxed variant (without $l0$-norm) of Program 2, obtain f^*_{MVO}
  Give reward as per Equation 4
end while
Give reward as $-f^*_{MVO}(x_t)$
Update RL parameters
return $\boldsymbol{x}$

We hope that the RL algorithm during inference will be able to choose near-optimal portfolios faster than the state-of-the-art solvers like Gurobi.

3.2 Machine Learning Approaches

3.2.1 Traditional Artificial Neural Network (ANN) Approach

In the traditional ANN approach, we aim to develop a neural network which accepts Q and μ for n assets, and outputs a single value for the estimated objective value of the MVO problem. This ANN approach is developing a policy function that we can then use as a warm-start for the reinforcement learning approach. The loss function will simply be the

mean-squared error of $|M\hat{V}O - MVO|$. Where $M\hat{V}O$ is the estimated risk-weighted return of the ANN and MVO is the true risk-weighted return solved by an optimizer.

In order to solve this approach, 26000 of random portfolio compositions (of size < K) were initialized based on training and testing synthetic data. These portfolios consisted of the (μ) values for each stock in the randomized portfolio, and their linear Q covariance matrix.

For each portfolio, the MVO optimization problem was solved using a CVXPY solver. The optimal risk-weighted returns for each portfolio as solved by cvxpy, were returned and used as the true target values for training the ANN.

The portfolios in the training and testing datasets were randomized at different sizes (i.e. varying level of stocks) in order to train the ANN at each portfolio size up to the cardinally limit K. Since these portfolio's are of altering input sizes, they were all padded with zero to standarized the input dimensions.

A densly-connected multi-layer-perceptron network was selected for the ANN of choice. This is a simple linear network that has an input size of 2550 neurons (the first 50 neurons for the padded μ vector and the last 2500 neurons for the padded linear Q matrix).

Multiple configurations were tested, but optimal performance is achieved with a less complex architecture consisting of two hidden layers (sizes 256, and 32) and a relu activation functions at each layer. Additionally, L1 and L2 regularization (1 percent) was utilized on the first hidden layer to help reduce over fitting. The final output layer has one neuron.

Training was conudcted using gradient descent, in batch sizes of 64. Learning was stopped early at 50 epochs. The final RMSE achieved by the network was 2.7953, which shows that the ANN does not learn perfectly. However, we hope that this warm-start for the Reinforcement Learning agent helps improve performance.

3.2.2 Dueling Deep-Q-Network

Initially, we experimented with the conventional DQN in our custom environment, but encountered instability due to fluctuating rewards and a lack of agent convergence. This instability stemmed from insufficient information for the agent to learn the Q function, given our state representation of the MVO of K assets and a single index representing the action. To address these challenges, we adopted the Dueling Deep-Q-Network (Dueling DQN) model. Dueling DQN

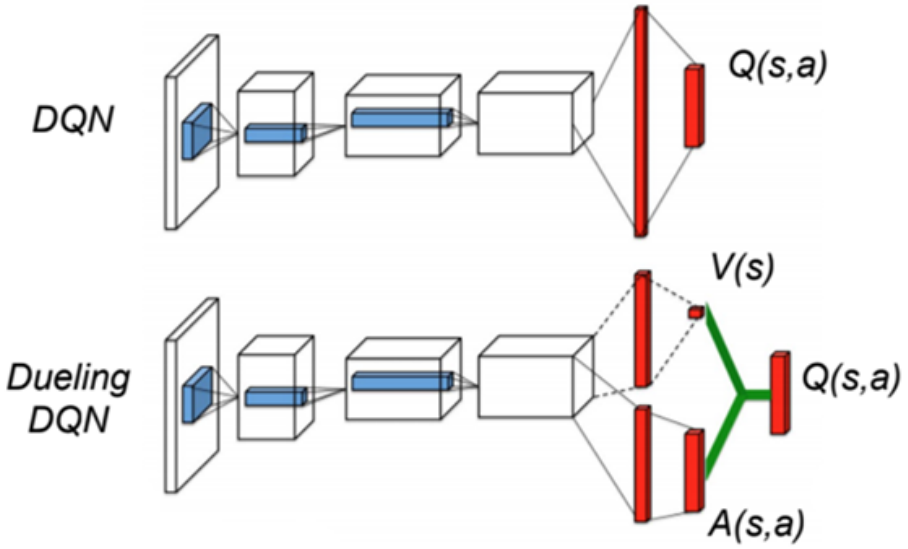


Figure 1: Traditional DQN (top) and Dueling DQN (bottom)

excels by separating state values and action advantages estimation, providing the agent with more information about its actions and the environment/state. Unlike our traditional DQN, the Dueling DQN operates in a distinct environment where it selects K assets at the start and refines the MVO values over multiple iterations. The action of the agent is to input a selected binary representation of the K asset in the size of n.The advantage of the dueling architecture lies partly in its ability to learn the state-value function efficiently. With every update of the Q values in the dueling architecture, the value stream V is updated [13]. This separation improves learning efficiency, stability, and adaptability, especially

in situations where action values vary. Dueling DQN proves particularly effective when actions share similar values in specific states, leading to enhanced generalization and reduced variance in training updates.

In the traditional DQN approach, we adopt a Single Stream DQN model. The input to the network comprises the mean and covariance matrix of the n assets, with the agent starting from an empty asset selection. The agent iteratively chooses one asset at each step until the portfolio includes K assets. Rewards are determined as the derivative of the Mean-Variance Optimization (MVO) concerning the last selected asset in the previous step. This process encourages the agent to dynamically adapt its asset selection strategy, optimizing the risk-return trade-off throughout the training process.However the result of Single Stream DQN model is not stable and the agent might lack a comprehensive understanding of the consequences and implications of its chosen actions.

Contrastingly, Dueling DQN takes a different route. It initiates with a random portfolio containing K selected assets. At each step, the portfolio undergoes updates to maximize the reward. The action space is represented in a binary format corresponding to the total assets available. Rewards are computed based on the transition from the old state to the new state. Specifically, the reward is set to 1 in the case of a positive transition, 0.5 if there is no change, and 0 if the transition is negative. By employing this approach, we aim to capture a broader spectrum of portfolio dynamics, allowing for more nuanced decision-making in response to changes in the asset collection.

Algorithm 2 Dueling DQN Pseudocode

Inputs:
$\boldsymbol{\mu}$, $\boldsymbol{Q}$, K
Initialize:
$x_0 \leftarrow$ Random Initial Portfolio
while iteration $\leftarrow 1$ to i **do**
Agent performs an action $A_1 : \{a_1, a_2, \ldots, a_K\}$
reward = $\begin{cases} 1 & if(s_{t+1} - s_t) \geq 0 \\ 0.5 & if(s_{t+1} - s_t) = 0 \\ -1 & if(s_{t+1} - s_t) \leq 0 \end{cases}$
Give reward r based on your defined equation
Output V (value) and A (advantage) for the Q value function
Update the Q value function
end while
Accumulate total reward for i iteration r_T
return x_K, A_i

During model training process, We conduct a training experiment of using the μ and Q data obtained from a synthetic dataset containing 1000 assets. The focus of our analysis is on the performance of the Double Deep Q-Network (DDQN) algorithm. The results provide valuable insights into the learning dynamics and effectiveness of DDQN in optimizing asset selection.

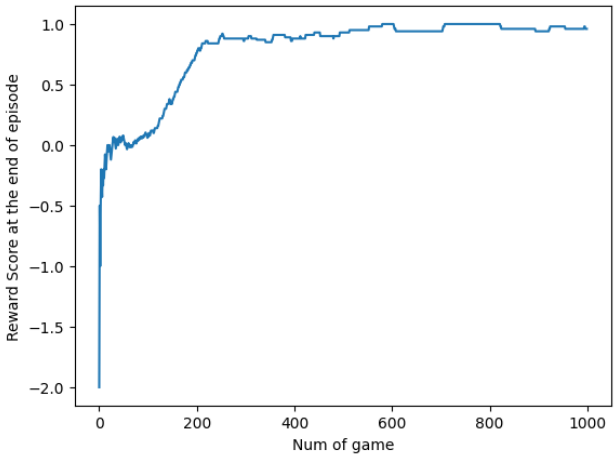


Figure 2: Average reward of the DDQN agent through 1000 games

In Figure 2, the trajectory of the average reward of the DDQN is depicted. Notably, the average reward consistently converges to 1. This convergence implies that the transition of Mean-Variance Optimization (MVO) is consistently increasing. Specifically, after approximately 500 epochs, we observe a notable uptrend in the average reward. This behavior underscores the efficacy of DDQN in adapting and improving its asset selection strategy over the course of training.

These results showcase the capability of DDQN to learn and adapt, leading to enhanced Mean-Variance Optimization. The observed convergence and improvement in MVO values reflect the algorithm's ability to navigate the solution space effectively and make informed decisions for asset selection. These promising outcomes lay the groundwork for further exploration and fine-tuning of DDQN in the domain of portfolio optimization.

3.2.3 Online Proximal Policy Optimization

As part of the group's continued efforts to create an RL Agent for stock selection for the MVO problem, a proximal policy optimization approach was implemented.

PPO is a state-of-the-art variant of the Actor-Critic method. It uses probability distributions over action spaces to select final actions. The theory that ground our work on PPO is outlined in J. Schulman et al [13].

As technical background into PPO, we we briefly discuss some of the components that differentiate it.

Unlike traditional temporal difference estimators, it uses a generalized advantage estimation function. Updating the policy takes into account an advantage, which dictates how much improvement came from that action relative to the average improvement:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

$$\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

In the above equations, γ is a smoothing factor for the Trajectory (in our code we assumed 0.95), and λ is the discount factor of the next state, as future rewards may be equally or less equally important to current rewards. T is the trajectory, and it defines the entire length of a memory sequence. The real meaning behind this generalized estimator is that it evaluates the between future and current state-values for every step in the sequence, generalizing across a trajectory.

To update the Critic network, the Critic Loss function is defined:

$$\tfrac{1}{2}\mathbb{E}[(G_t - V_p red(s_t))^2]$$

$$\text{where } G_t = \hat{A}_t + V_t arget(s_t)$$

The critic loss is responsible for training the state-value function, represented by the critic network. This loss is the difference between the true Value (from memory) and the predicted Value (from network). This Loss in PPO is slightly different in that it includes the estimated advantage.

The heart of PPO, however, lies in the actor loss, also known as the Surrogate Objective Function (SOF). We define the SOF as follows:

$$L^{CLIP}(\theta) = hat\mathbb{E}_{\approx}[min(r_t(\theta)hatA_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon]$$

$$\text{where } r_t = \tfrac{\pi_{current}(a|s)}{\pi_{old}(a|s)}$$

The SOF is a clipping function. It's purpose is to limit the update of the policy, represented by the state-action of the actor network. The probability ratio, which is the the probability of taking an action given a state in the new policy, divided by the probability of taking the same action in the same state given the old policy, can become extremely large, making the network unstable. Since the objective is to maximize the SOF (gradient ascent), we dont not want to have Loss value that is too high. In all cases, we can to have a r_T that is close to one. This is what the clipping does in the SOF. If the r_t is much higher than 1, it binds it to a range of $1 \pm \epsilon$, where epsilon is a small number.

This is illustrated in the graph below, which illustrates the clipping function for one time-step (note, that this is summed up across all time steps in the trajectory).

When the advantage is positive, we want to encourage the probability of repeating those actions again, however, we do not want too large of a loss (as this due to a large r_t) This is where the clipping is important, it limits the r_t to be less than 1 + ϵ. However, when the advantage is negative, we want to discourage the probability of repeating that action, so prohibit the loss from being too high, by clipping the r_t to be 1 - ϵ the lower bound.

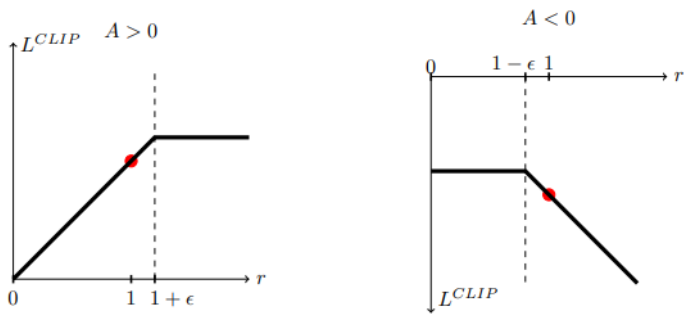


Figure 3: Surrogate objective clipping

This approach builds on the traditional actor-critic method by adding increased stability to the learning process and good sample efficiency. PPO aims to balance between improving the policy and ensuring that the policy does not change too quickly. The core of PPO is finding a policy to optimize the surrogate objective function, which encourages the model to move in positive-advantage directions and implements a clipping function to limit possible policy changes.

Below is a diagram for the specific implementation of the PPO alogirithm for our portfolio selection problem:

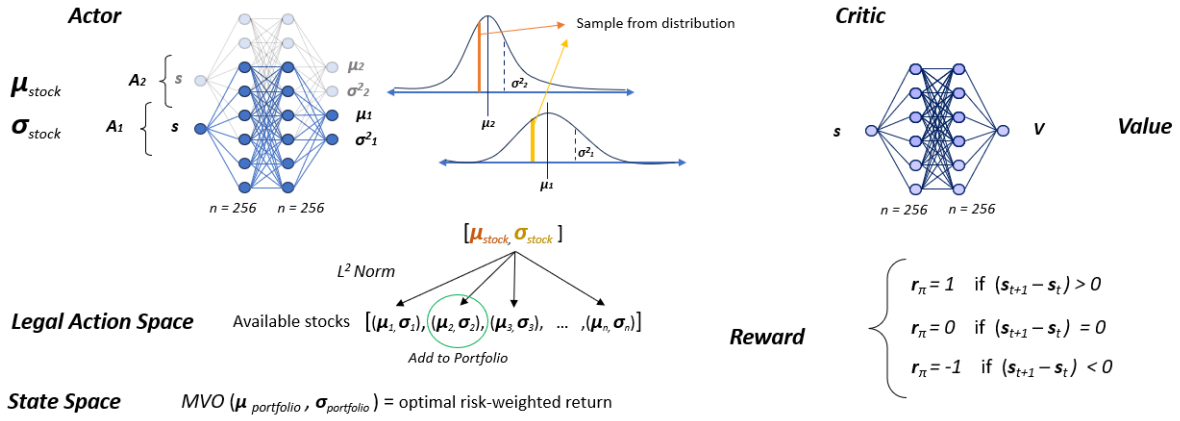


Figure 4: PPO implementation for MVO stock selection

The implementation of the PPO algorithm for selecting stock-by-stock an optimal portfolio is as follows: The state-space is comprised solely of the current MVO portfolio value. This is fed to the Actor linear neural network (two hidden layers of 256 neurons each). The actor is has two independent actions in the state space, each of a continuous value. The first action is the ideal return (μ_stock) of a hypothetical stock, and the second action is the ideal variance (σ_stock) of a hypothetical stock. In order to determine these values, the actor network outputs the parameters to a Gaussian-normal distribution for each independent action (μ_1,σ_1^2) and (μ_2,σ_2^2). These distributions are randomly sampled in order to determine what the final actions will be. At the end of the actor cycle, a hypothetical (μ_stock) and (σ_stock) will be selected. In order to select a real-world stock, one that we have available in our basket to choose from, we use the Euclidean distance, (L^2 Norm) from the hypothetical stock to all the other available stocks. The closest real stock to the hypothetical is chosen as the stock to add to the current portfolio. This new portfolio is passed to the cvxpy MVO optimizer to get the new optimal MVO value. This represents the new state.

While the actor network is evaluating the state-action function, the critic is evaluating the state-value function. The same state is passed into the critic network (two hidden layers of 256 neurons each), and the a continuous value of that state is outputted.

At each time step, a reward is calculated for the new state. Initially, we were using the derivative of the MVO change as our reward; however, it was noticed that for minute improvements to the MVO value, the reward diminishes, incentivizing learning less and less as incremental improvement is made. Instead, we use a piece-wise reward:

$$\begin{cases} reward = 1 & (s_{t+1} - s_t) \geq 0 \\ reward = 0 & (s_{t+1} - s_t) = 0 \\ reward = -1 & (s_{t+1} - s_t) \leq 0 \end{cases}$$

This reward function as the advantage of equally rewarding improvements of any size, which helps encourage the Agent to reach higher incremental optimality.

In this instance of the PPO implementation, the model runs 'online'. That is, it undergoes retraining for every time a new portfolio needs to be built from a new basket of stocks. If a portfolio needs to be on a weekly basis, then a full retrain is instantiated every week. The Agent is capable of returning optimal portfolios for every size portfolio less than cardinality.

To further improve the Agent, hyperparameter tuning is available for exploring. Additionally, there is the potential to expand upon the state-space by adding in a lower-dimensional representation of the available Q-matrix of the current portfolio. Lastly, instead of using Euclidean distances to determine which stocks to select, a Sharpe Ratio can be used, which is the average return of a stock divided by its variance.

3.2.4 Linear Regression

In order to benchmark the performance of our PPO implementation against a simpler algorithm, we also trained a linear regressor on the problem.

Linear regression takes the following form:

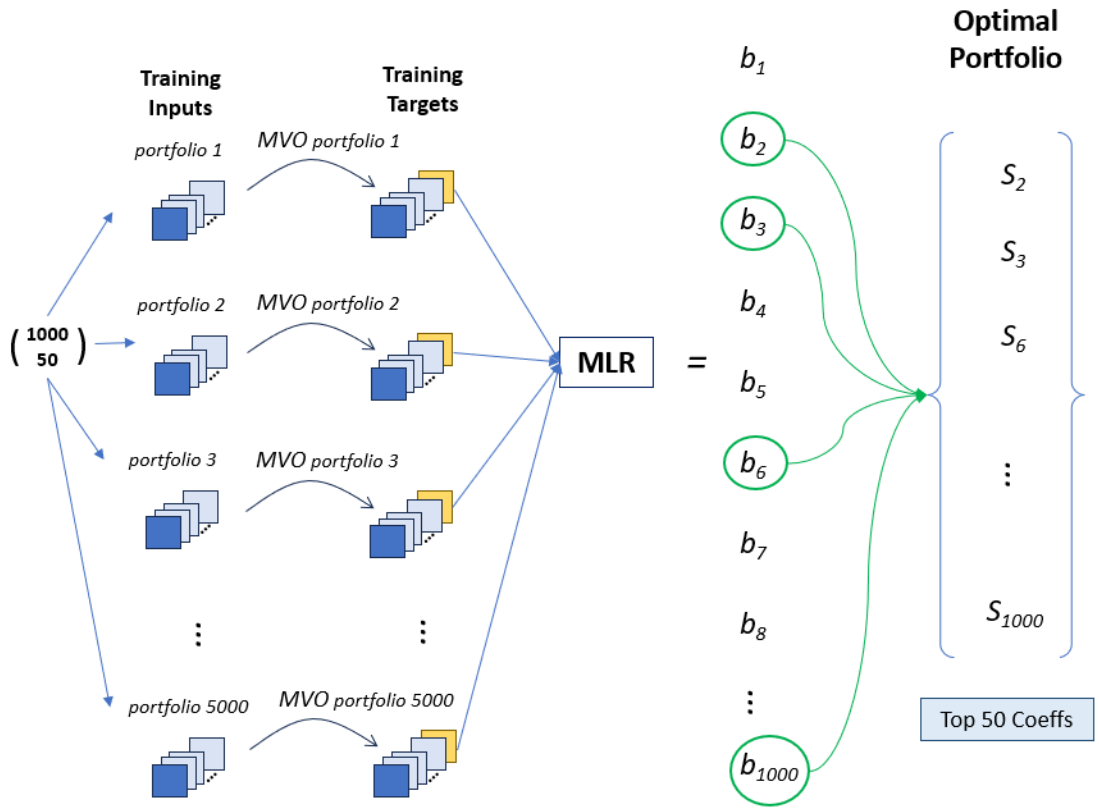
$$y = B^T X + \epsilon = b_0 + b_1 x_1 + b_2 x_2 + ... + b_k x_k + \epsilon$$

where ϵ is some error term

$X = [x_1, x_2, ..., x_k]$ is a k dimensional vector of all the decision variables under consideration. In our case, X is a binary vector of 0's and 1's, of dimension 1000, as there are 1000 stocks to choose from.

$B = b_0, b_1, b_2 ... b_k$ is the set of training coefficients. These coefficients represent the absolute coefficient of determination R^2 to a specific decision variable x_i and the target y.

Below is a diagram for the specific implementation of Linear Regression algorithm for our portfolio selection problem:



In order to train the model, we perform the mathematical operation $(\frac{1000}{50})$ 5000 times (sampled without replacement). This initializes 5000 randomly generated portfolios of size 50 from the 1000 stocks in our basket. This acts as a our training dataset. This also means that the cardinality constraint is met in the following way:

$$\sum_{i=1}^{k} X = x_0 + x_1 + x_2 + ... + x_k = 50$$

For each of these generated training portfolios, we pass them into the cvxpy solver to obtain the optimal MVO value. This is our target vector, representing the true optimal portfolio returns.

According to the diagram, we can then train our multiple linear regressor, on our entire dataset. Once we've acquired a trained model, we then take the top 50 largest positive coefficients from the trained model. These coefficient indices can then be used as the indices to select our optimal 50-stock portfolio.

This instance of the multiple linear regressor also runs 'online'. Meaning we retrain it every a new portoflio needs to be built from a new basket of stocks.

To further improve the MLR model, we can lift the cardinality restrain when generating the randomized portoflios, exposing each training instance to a larger or smaller set of stocks. In the end, we can take the top k coefficients from the model that we need to meet cardinality. Furthermore, we can create a larger training dataset by increasing the number of traning portfolios from 5000 to some higher number.

4 Dataset

We have the opportunity to use two sources as our dataset. Since we are working with financial data, it is quite easy to obtain this data from online sources. For example, AlphaVantage (www.alphavantage.co) readily provides weekly close prices for a number of real-life assets. As a first test, we choose a well diversified set of 288 US assets that span various different industries. This choice of assets is consistent with literature [5]. Data is obtained from 2014 to 2022.

As a reminder, the input to both the MVO model and to our reinforcement learning agent will be the return vector μ and the covariance matrix Σ. A natural question to ask is how $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are estimated. A naive method is by taking the

average of observed historical data. Let $\hat{\boldsymbol{\xi}} \in \mathbb{R}^{nxT}$ be the matrix of observed historical returns (in our case weekly), with T historical occurrences and n assets. Then, the estimate of the expected asset return for asset i is

$$\hat{\mu_i} = \mathbb{E}[\boldsymbol{\xi}] = \frac{1}{T}\sum_{k=1}^{T}\hat{\xi}_k \tag{5}$$

Similarly, for the asset covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{nxn}$, where $\sigma_{i,j}$ can be computed as:

$$\sigma_{ij} = \frac{1}{T-1}\sum_{t=1}^{T}(\hat{\xi}_{it} - \hat{\mu}_i)(\hat{\xi}_{jt} - \hat{\mu}_j) \tag{6}$$

One issue with this approach is that we would like to test our optimization algorithm on large problems. Obtaining and storing data for a large number of assets ($n \in \{2000, 3000, 5000\}$ may be memory-inefficient. Thus, we also plan to run experiments on synthetic data. For each problem size, we generate 10 instances of random covariance matrices ($\hat{\boldsymbol{\Sigma}}$) and return vectors ($\hat{\boldsymbol{\mu}}$) using a random portfolio generator provided by Sertalp (n.d.) [4]. This generator is based on the theory provided by Hirschberger et al. (2007) [6]. They study statistical properties of covariance matrices and provide a robust approach to generating them that reflect the trends seen in real-life assets. Thus, they provide a simple yet robust approach to generating large problem instances.

5 Baseline Methods

CC-MVO is solved by Gurobi 10.0.3 with presolve and heuristics kept on for a fair comparison. To do so, we must first formulate CC-MVO as a Mixed Integer Program. As a reminder, CC-MVO is defined as

$$\begin{aligned} \min_{\boldsymbol{\phi}} \quad & \boldsymbol{\phi}^T\boldsymbol{\Sigma}\boldsymbol{\phi} - \lambda\boldsymbol{\mu}\boldsymbol{\phi} \\ \text{s.t.} \quad & \boldsymbol{\phi} > 0, \\ & \mathbf{1}^T\boldsymbol{\phi} = 1, \\ & ||\boldsymbol{\phi}||_0 \leq K \end{aligned} \tag{7}$$

We define new auxillary binary variables $\boldsymbol{z} \in^n$ where $z_i \in \{0, 1\}$, allowing us to re-write the above as

$$\begin{aligned} \min_{\boldsymbol{\phi}} \quad & \boldsymbol{\phi}^T\boldsymbol{\Sigma}\boldsymbol{\phi} - \lambda\boldsymbol{\mu}\boldsymbol{\phi} \\ \text{s.t.} \quad & \boldsymbol{\phi} > 0, \\ & \mathbf{1}^T\boldsymbol{\phi} = 1, \\ & \mathbf{1}^T\boldsymbol{z} = K, \\ & \boldsymbol{\phi} \leq \boldsymbol{z} \end{aligned} \tag{8}$$

This is now in a form that is implementable in Gurobi. The time limit for Gurobi is set to 600s per instance.

6 Experimental Results

We have run experiments to compare our models with Gurobi. Our primary metrics are the Mean Variance Objective Values obtained by the strategies and time taken for inference. We showcase our results in figures 5 and 6. For RL, we used 2014-2018 for training and thus the results shown below are on out of sample data from 2019 to 2022. All experiments were run on the 2022 Macbook Air with M1 chip and 8GB of memory.

Our results indicate that the simple Linear Regression model is able to achieve very-close-to-optimal solutions and at runtimes faster than Gurobi. This is very impressive for such a simple idea, and one that has not been explored in the literature. With regards to our RL-based models, we unfortunately see that the Online methods tend to have a very large runtime (due to the online component) but do however achieve competitive MVO performance. The offline methods achieve roughly the same performance but the runtime is still higher than that of Gurobi. Overall, we conclude that the linear regression method is competitive with Gurobi, but the other methods we tried did not work as well.

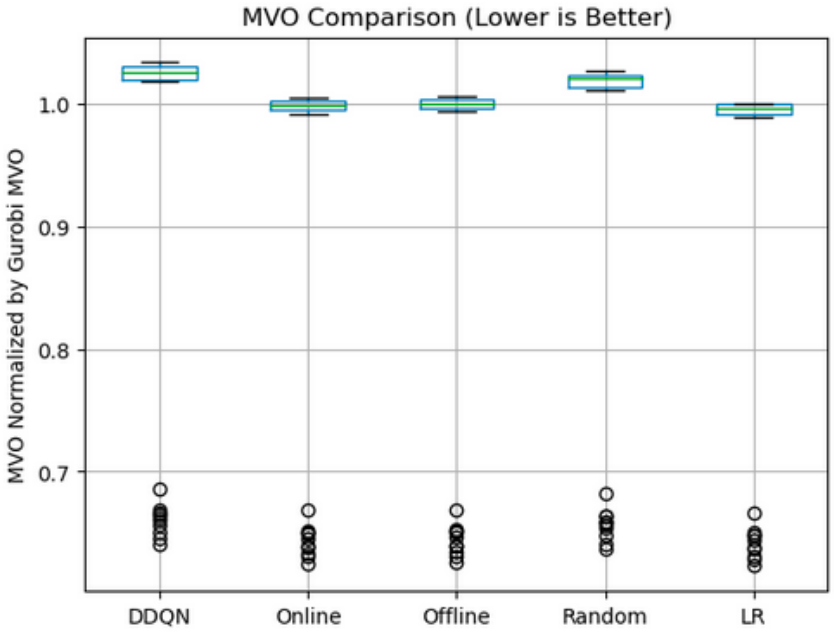


Figure 5: Out of Sample MVO Values Comparison

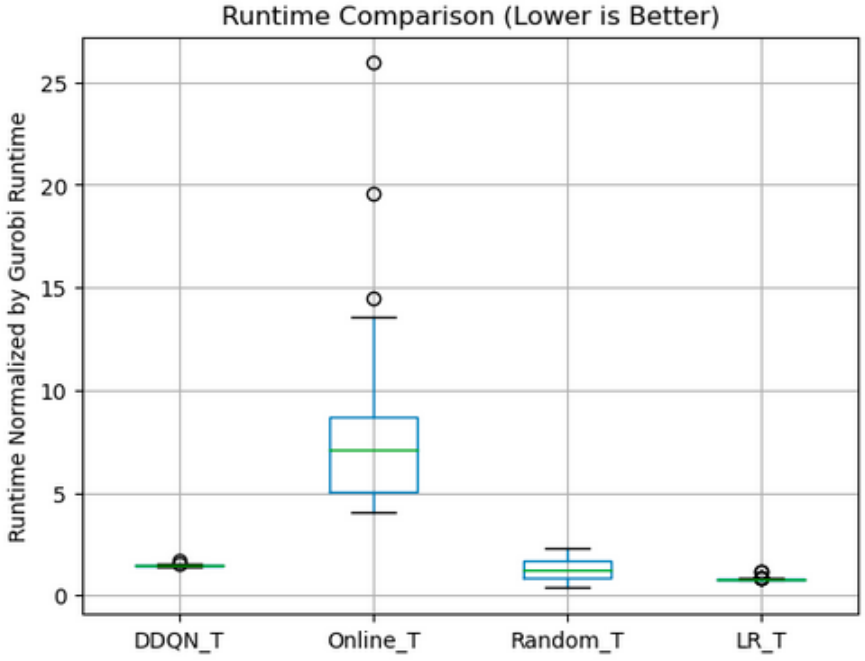


Figure 6: Time Comparison of Gurobi / RL

7 Conclusion and Future Work

In this project, we tried different reinforcement learning approaches to solving the cardinality constrained mean variance optimization problem. We found that the approaches we tried and our implementation was not able to yield superior results to Gurobi, our benchmark.

There are definitely many ways that we can improve on our work to try to make this work in the future. Some ideas can include trying more difficult optimization problems, such as multi-period portfolio optimization. This is a more challenging task for Gurobi and this can allow the reinforcement learning approach's scalability to become useful.

References

[1] Ziyun Xu Amine Mohamed Aboussalah and Chi-Guhn Lee. What is the value of the cross-sectional approach to deep reinforcement learning? *Quantitative Finance*, 22(6):1091–1111, 2022.

[2] Dimitris Bertsimas and Ryan Cory-Wright. A scalable algorithm for sparse portfolio selection. *Informs journal on computing*, 34(3):1489–1511, 2022.

[3] Daniel Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical programming*, 74:121–140, 1996.

[4] Sertalp B. Çay. Random portfolio dataset generator. `http://sertalpbilal.github.io/randomportfolio/`.

[5] Giorgio Costa and Roy H Kwon. Risk parity portfolio optimization under a markov regime-switching framework. *Quantitative Finance*, 19(3):453–471, 2019.

[6] Markus Hirschberger, Yue Qi, and Ralph E Steuer. Randomly generating portfolio-selection covariance matrices with specified distributional characteristics. *European Journal of Operational Research*, 177(3):1610–1625, 2007.

[7] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[8] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[9] Richard O Michaud. The markowitz optimization enigma: Is 'optimized'optimal? *Financial analysts journal*, 45(1):31–42, 1989.

[10] Rubén Ruiz-Torrubiano, Sergio García-Moratilla, and Alberto Suárez. Optimization problems with cardinality constraints. In *Computational Intelligence in Optimization: Applications and Implementations*, pages 105–130. Springer, 2010.

[11] Srijan Sood, Kassiani Papasotiriou, Marius Vaiciulis, Tucker Hybinette Balch, J. P. Morgan, and AI Research. Deep reinforcement learning for optimal portfolio allocation: A comparative study with mean-variance optimization.

[12] Felix Streichert and Mieko Tanaka-Yamawaki. The effect of local search on the constrained portfolio selection problem. In *2006 IEEE International Conference on Evolutionary Computation*, pages 2368–2374. IEEE, 2006.

[13] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.