On Non-Stationarity in Reinforced Deep Markov Models with Applications in Portfolio Optimization

Laurens Chin A Pauw

Delft University of Technology
Faculty of EEMCS

March 2024

A thesis presented for the degree of
Master of Science in Applied Mathematics -
Stochastics and Mathematics of Data Science

Supervisor:
Dr. Fenghui Yu

Thesis Committee:
Prof. Dr. Antonis Papapantoleon
Dr. Alexis Derumigny

Abstract

In this thesis, we aim to improve the application of deep reinforcement learning in portfolio optimization. Reinforcement learning has in recent years been applied to a wide range of problems, from games to control systems in the physical world and also to finance. While reinforcement learning has shown success in simulated environments (e.g. matching or exceeding human performance in games), its adoption in practical applications (non-simulated environments) has lagged. Dulac-Arnold et al. [2019] suggest this is caused by a discrepancy in the experimental set-up in research and the conditions in practice. Specifically, they present a list of challenges that make the application of reinforcement learning in real-world settings more difficult. One of these challenges is non-stationary environments, which is common in financial environments. It is a challenge since, given an observed state, the optimal action may not always be the same as it may change over time due to non-stationarity. Therefore, more specifically, the goal of this thesis is to overcome the challenge of non-stationarity in the application of reinforcement learning to portfolio optimization. In this thesis, we use reinforced deep Markov models (RDMM) introduced by Ferreira [2020] (applied to an optimal execution problem and later used by Cartea et al. [2021] for statistical arbitrage on simulated price movements of an FX triplet) for its data efficiency and ability to handle complex environments. RDMM involve a partially observable Markov decision process (POMDP) which is also the setting used by Xie et al. [2021] to model non-stationarity in reinforcement learning. We extend RDMM to incorporate non-stationarity, using the framework suggested by Xie et al. [2021], and apply it to portfolio optimization. Our implementation is sample efficient which allows for quick learning, by doing this we attempt to improve on another challenge of reinforcement learning — i.e. sample-inefficiency [Dulac-Arnold et al., 2019]. Moreover, our implementation can handle continuous state and action spaces.

We compare the performance of our algorithms to classical portfolio optimization techniques such as Mean-Variance (MV) and Equal Risk Contribution (ERC), and to popular reinforcement learning techniques such as Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC). We observe our implementation has higher sample-efficiency compared DDPG and SAC, and higher cumulative returns on the test set compared to MV, ERC, DDPG, and SAC.

Keywords: Reinforced Deep Markov Models, Model-Based Reinforcement Learning, Policy Gradient, Non-Stationarity, Portfolio Optimization, Partially Observable Markov Decision Processes, Variational Autoencoders, Deep Neural Networks

Contents

Chapter 1

Introduction

In this thesis, we explore and aim to improve the application of (deep) reinforcement learning to the problem of portfolio optimization. The field of reinforcement learning has gained significant traction in recent years, demonstrating its effectiveness in a variety of settings, including AlphaGo and large language models (LLMs). Its application in finance has shown potential, with implementations in various areas including market making, optimal execution [Ning et al., 2021], pricing and hedging options [Buehler et al., 2019], and portfolio optimization.

Combining deep neural networks with reinforcement learning has proven to be especially effective. This approach enables a decision-making framework using available data without the need for a predefined model that may often contain unrealistic assumptions regarding the problem or environment. This is particularly relevant in finance, where traditional problems often require simplifications or assumptions to arrive at an analytical solution—for example, the Almgren-Chriss model for optimal execution or the assumption of continuous-time delta-hedging in option pricing.

Within finance, we choose to apply reinforcement learning to portfolio optimization. One advantage is that data for portfolio optimization are usually easy to access for free. This is not necessarily the case for other financial data such as limit-order book (LOB) data used in market making. This allows us to perform a numerical experiment on historical data as opposed to simulated data which may be less realistic.

In the following subsections, we will provide a basic introduction to the reinforcement learning framework, concepts and definitions. This is followed by a brief introduction to portfolio optimization and the existing literature on reinforcement learning being applied to portfolio optimization. We then introduce general challenges that have inhibited the successful application of reinforcement learning to real-world tasks (as opposed to games and simulated environments) that require decision-making. Of those problems, we focus specifically on dealing with non-stationary environments.

1.1 Reinforcement Learning

Reinforcement learning, along with supervised and unsupervised learning, are distinct approaches of machine learning. It revolves around learning an agent to take actions in a certain environment in order to satisfy an objective – i.e. to maximize a quantity such as the cumulative reward. It is distinguished from supervised or unsupervised learning which may classify data or use regression to provide a prediction, respectively. Reinforcement learning, on the other hand, aims select an action to given input from the environment.

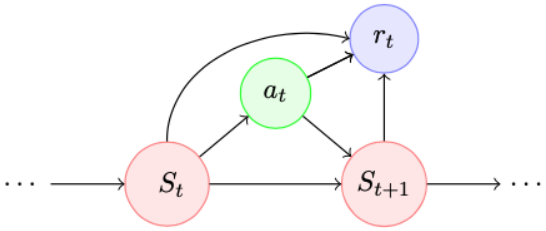


Figure 1.1: General Markov Decision Problem (MDP) for reinforcement learning [Cartea et al., 2021]

1.1.1 Markov Decision Process

A reinforcement learning environment is usually described by a Markov Decision Process (MDP). An MDP consists of states s_t, s_{t+1}, actions a_t and rewards r_t at time t. The action a_t depends on state s_t and influences the (stochastic) transition from state s_t to state s_{t+1}. The corresponding triplet (s_t, s_{t+1}, a_t) influences the reward r_t from taking action a_t and transitioning from s_t to s_{t+1} [Jaimungal, 2022].

The central goal of reinforcement learning can be summarized quite simply as [Jaimungal, 2022]:

> Find the mapping of states to actions that maximizes the (discounted) cumulative reward

Let us introduce a discrete time Markov Decision Process (MDP) with an infinite time horizon. The setting consists of a state space $\mathcal{S}$ and an action space $\mathcal{A}$. Since our goal is to find the mapping from states to actions, called the policy, that maximizes the discounted cumulative reward, we define the reward function as

$$R^\pi = \sum_{t=0}^{\infty} \gamma^t r_t^\pi, \tag{1.1}$$

where $\gamma \in (0, 1)$ is a discount factor, and r_t^π is the reward at time t when following policy π [Jaimungal, 2022]. The policy can be deterministic $\pi : \mathcal{S} \to \mathcal{A}$ which maps a state

to a deterministic action, or a randomized policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ which maps a state to a probability distribution over actions. Here we denote $\mathcal{P}(\mathcal{A})$ as the set of probability measures over the space $\mathcal{A}$ [Hambly et al., 2023].

We can now formulate the goal of reinforcement learning mathematically by defining the value function, for each state $s \in S$ [Hambly et al., 2023]

$$V(s) = \sup_{\pi \in \Pi} \mathbb{E}\left[R^{\pi} \mid s_0 = s\right]. \tag{1.2}$$

In addition, we define $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ to be the Markov transition function. Further, s_{t+1} is sampled from distribution $P(s_t, a_t) \in \mathcal{P}(\mathcal{S})$.

By Bellman's optimality principle, we can rewrite the value function to a Bellman equation. Bellman's principle of optimality is [Bellman, 1957]:

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Consequently, one can recursively rewrite the value function by relating the value function of subsequent time periods

$$V(s) = \sup_{a \in \mathcal{A}} \left[\mathbb{E}\left[r_t^a\right] + \gamma \mathbb{E}_{s_{t+1} \sim P(s,a)}\left[V(s_{t+1})\right]\right], \tag{1.3}$$

where r_t^a denotes the random reward from taking action a at time t. It is furthermore useful to define the Q-function, which measures the *quality* of an action in a given state

$$Q(s, a) = \left[\mathbb{E}\left[r_t^a\right] + \gamma \mathbb{E}_{s_{t+1} \sim P(s,a)}\left[V(s_{t+1})\right]\right], \tag{1.4}$$

from which it is clear that

$$V(s) = \sup_{a \in \mathcal{A}} Q(s, a). \tag{1.5}$$

Similarly, by Equation 1.3 we can also write the Q-function as a Bellman function

$$Q(s, a) = \mathbb{E}\left[r_t^a\right] + \gamma \mathbb{E}_{s_{t+1} \sim P(s,a)} \sup_{a' \in \mathcal{A}} Q(s_{t+1}, a'). \tag{1.6}$$

The optimal policy is then given by $\pi^* \in \arg\max_{a \in \mathcal{A}} Q(s, a)$, assuming it exists and is stationary.

A Markov decision problem becomes a reinforcement learning problem when the goal is to find an optimal policy π while the transition dynamics P and the reward function r are unknown. Those functions will be learned implicitly in reinforcement learning methods.

A reinforcement learning algorithm generally has at least one of the following components: a value function to determine the value of a state (or state-action pair) and a policy function. Consequently, reinforcement learning algorithms consist of two categories: value-based methods and policy-based methods.

1.1.2 Value-Based Reinforcement Learning

Value-based methods involve the estimation of the value function given in Equation (1.3) such as the Q-function which is given in Equation (1.6). What distinguishes value-based methods from policy-based methods is that only the value function is learned and not the policy. However, ultimately, the policy can be derived from the value function by taking the action that maximizes the value function, i.e. $\arg\max_{a\in\mathcal{A}} Q(s,a)$. The downside is, however, that computing $\arg\max_{a\in\mathcal{A}} Q(s,a)$ becomes computationally expensive as the dimensionality of the state or action spaces increases. This is especially expensive as value-based methods may require the aforementioned computation multiple times. Indeed, the computation of $\arg\max_{a\in\mathcal{A}} Q(s,a)$ becomes infeasible when the action space is continuous (or unbounded) [Jaimungal, 2022]. Consequently, value-based methods are used with bounded discrete state and action spaces—i.e. the set of states and actions is bounded and consists of discrete variables.

1.1.3 Policy-Based Reinforcement Learning

Despite the success of value-based reinforcement learning in practice, it has few theoretical guarantees in terms of convergence and some applications resulted in convergence issues. This was a reason behind the search for alternative methods that have more reliable and well-defined convergence properties [Sutton et al., 2000]. As a result, policy-based methods were introduced. These methods involve a directly parameterized policy function (instead of an implicit policy through a value function). The goal of policy-based reinforcement learning methods is to learn this policy function. Indeed, better convergence properties have been observed compared to value-based methods [Sewak and Sewak, 2019, Dabérius et al., 2019, Yu and Sun, 2020] because in value-based methods big oscillations tend to occur during the training process since actions may need to change dramatically for a small increase in the value function [Hambly et al., 2023]

Furthermore, an advantage of policy-based methods is that, unlike value-based methods, it does not suffer from an increasing dimensionality of state or action spaces. Indeed, policy-based methods can handle continuous action spaces, in which the actions are continuous variables or a vector whose elements are continuous variables. Indeed, Lillicrap et al. [2015] note that many tasks of interest, (mainly physical control tasks) have continuous and high dimensional action spaces.

In policy-based methods, there is a policy distribution $\pi(s,\cdot;\theta) \in \mathcal{P}(\mathcal{A})$ over the action space parameterized by the vector θ in a given state s (note that a policy can also be a deterministic function). Furthermore, there is ρ^π, denoting the stationary state distribution associated with policy π_θ [Hambly et al., 2023]. The policy objective function is given by [Hambly et al., 2023]

$$J(\theta) := \int_{\mathcal{S}} V_\theta^\pi(s)\rho^\pi(ds), \tag{1.7}$$

where the goal is to maximize this objective function, which can be done through a gradient ascent update on the parameter θ

$$\theta^{n+1} = \theta^n + \beta \nabla_\theta J(\theta), \tag{1.8}$$

where $\nabla_\theta J(\theta)$ is the gradient of the objective function and β is the learning rate.

To perform the gradient ascent step, the gradient $\nabla_\theta J(\theta)$ must be estimated. Using the stochastic policy gradient theorem [Sutton et al., 1999], gives an expression for the gradient

Theorem 1. *[Sutton et al., 1999] Assume $\pi(s, a; \theta)$ is differentiable with respect to θ. For any MDP for which a stationary distribution ρ^π exists, the policy gradient is*

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta} \left[\nabla_\theta \ln \pi(s, a; \theta) Q^\pi(s, a) \right]. \tag{1.9}$$

Conveniently, the expression does not require the gradient of the stationary distribution as it is unclear how to estimate it. In addition, the term $\nabla_\theta \ln \pi(s, a; \theta)$, is a Score function which, under a softmax or (multi-dimensional) Gaussian policy, has an analytically tractable derivative [Rao and Jelvis, 2022].

1.1.4 Model-Free and Model-Based Reinforcement Learning

Similar to the dichotomy between value-based methods and policy-based methods, there is another dichotomy in reinforcement learning — model-free and model-based reinforcement learning. Model-free reinforcement learns through direct interaction with been successful in video game playing and other simple control tasks, where the environment is relatively predictable and the goal is straightforward [Mnih et al., 2015]. This method, however, suffers from sample inefficiency.

Model-based reinforcement learning models the dynamics of state transitions and rewards, which, for example, could be modelled using supervised learning methods such as deep neural networks. This method was introduced to accelerate convergence and enhance data (sample) efficiency [Ferreira, 2020]. While the implementation is more complex than model-free, Nagabandi et al. [2018] show improved sample efficiency using model-based reinforcement learning in complex locomotion tasks.

1.1.5 On-Policy and Off-Policy Learning

Another concept in reinforcement learning is that of on-policy and off-policy. In off-policy learning there is a distinction between the policy that samples the data and the policy that is learned. This approach allows learning from more past experiences and not only the current decision. Off-policy is particularly useful in cases where acquiring new data is expensive or risky, as it can utilize historical or external data sources for learning. On the other hand, on-policy learning involves learning directly from the interactions and the

samples generated from the current policy. On-policy learning is known to have poor sample efficiency, whereas off-policy is known to be more sample-efficient Haarnoja et al. [2018a]. Off-policy learning is more sample efficient through the use of a replay buffer, which stores experiences consisting of states, actions and rewards, collected from interactions with the environment. These experiences come from different past policies and are sampled in batches from the replay buffer to update the current policy. Using large replay buffers stabilize and speeds up the learning process [Mnih et al., 2015, Haarnoja et al., 2018a].

1.2 Portfolio Optimization Problem

Portfolio optimization involves the construction of a portfolio from a pool of assets based on a specific objective. This problem was introduced by Markowitz [1952] when he claimed that not only the expected return of a portfolio should be considered, but also its "risk". Markowitz [1952] argued if only the expected return were to be maximized, then the portfolio would have to be fully allocated to the asset(s) with the highest expected return(s). Instead, to avoid high concentrations in a single or a small set of assets, Markowitz [1952] suggested that a portfolio should both maximize expected return while offering "diversification" which minimizes the portfolio's "risk". This risk was expressed in terms of the portfolio's variance. Consequently, the aim was to minimize the portfolio's variance for a given level of expected return or to maximize the expected return for a given level of variance.

We consider optimizing a portfolio of n assets, with allocation weight vector $w^\intercal = (w_1, \ldots, w_n) \in \mathbb{R}^n$, such that

$$w^\intercal \mathbf{1} = 1, \tag{1.10}$$

$$w \geq 0, \tag{1.11}$$

where $\mathbf{1} = (1, \ldots, 1)^\intercal \in \mathbb{R}^n$. The weights summing up to one and the non-negativity of the weights indicates we consider portfolios without leverage and short-selling.

Here we take the Mean-Variance portfolio optimization as an example. Let $\mu \in \mathbb{R}^n$ be the vector of expected returns of the n assets, and let $\Sigma \in \mathbb{R}^{n \times n}$ be the covariance matrix of the n-assets. The variance of the portfolio is given by

$$w^\intercal \Sigma w. \tag{1.12}$$

Furthermore, let $z \in \mathbb{R}$ be the target (desired) return. Then, the Mean-Variance (single-period) portfolio optimization problem can be formulated as

$$\min_w w^\intercal \Sigma \tag{1.13}$$

$$\text{s.t. } w^\intercal \mu = z. \tag{1.14}$$

1.3 Literature Review

In this section, we will review the literature regarding the application of reinforcement learning to portfolio optimization. We first broadly discuss some of the differences in implementation and then we discuss the reinforcement learning algorithms used. We also present a list of problems that have hindered the applicability of reinforcement problems in real-world applications. Lastly, we will explain the concepts of value-based or policy-based reinforcement learning, and model-free or model-based reinforcement learning.

1.3.1 Reinforcement Learning in Portfolio Optimization

There have been various papers on the application of reinforcement learning algorithms portfolio optimization, and stock trading more generally. With respect to stock trading, Zhang et al. [2020] use reinforcement learning to select an action from the discrete set $\{-1, 0, 1\}$ per stock to either short, stay neutral or long the stock at any chosen time interval. Moody and Saffell [1998] used reinforcement learning to extend this to two-asset portfolios consisting of a stock index (the SP500) and bonds (US T-bills). Similarly, Pendharkar and Cusatis [2018] also construct a two-asset portfolio of the SP500 and bond index (AGG) (or Treasury Bills). Pendharkar and Cusatis [2018], however, use different discrete actions (allocations), namely for a long-only portfolio given in Table 1.1.

The number of assets used in portfolio optimization using reinforcement learning was increased by Park et al. [2020], only ever so slightly, from two to three assets. Park et al. [2020] choose discrete actions from the set $\{-1, 0, 1\}$ however, these have a different meaning; namely, they denote the multiple of a fixed dollar amount (e.g. \$10,000) of shares to be bought or sold. While this may limit transaction costs, the resulting allocation may be far removed from an optimal allocation. Moreover, it adds the complexity of dealing with infeasible actions which occur once there is no capital left for buying, or when there is no more of the asset that can be sold. The authors Park et al. [2020] do note that the portfolio can be extended to deal with more than three assets at the cost of more computational effort. Liu et al. [2018] use a similar approach to Park et al. [2020] where they indicate actions are selected from the set $\{-k, 0, k\}$, where k denotes the number of shares to be bought or sold.

Other authors do use reinforcement learning in portfolio optimization for multi-asset portfolios [Jiang et al., 2017, Liang et al., 2018, Aboussalah et al., 2022, Sood et al., 2023]. Unlike before, in Jiang et al. [2017], Liang et al. [2018], Sood et al. [2023] actions are not selected from a set of discrete variables but take continuous values. Specifically, in Jiang et al. [2017], Liang et al. [2018], Sood et al. [2023] the actions are a vector $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$ which resemble portfolio weights where the non-negative weights sum up to one (using a softmax function).

The choice between a discrete or continuous action space depends on whether a value-based or a policy-based method is used. The authors Moody and Saffell [1998], Zhang et al.

Table 1.1: Reinforcement agent action set in Pendharkar and Cusatis [2018]

Asset	**1**	**2**	**3**	**4**	**5**
S&P 500 (%)	0	25	50	75	100
AGG or T-bill bond (%)	100	75	50	25	0

[2020], Pendharkar and Cusatis [2018], Park et al. [2020] who use a discrete set mainly use value-based learning with the exception of Liu et al. [2018] who do use a policy-based method, and Zhang et al. [2020] who compare a value-based method with a policy-based mehtod. Conversely, the authors who use a continuous action action space, Jiang et al. [2017], Liang et al. [2018], Sood et al. [2023], employ policy-based methods. The policy-based methods allow for more flexibility in the actions that can be chosen, hence it will be our preferred method.

The various value-based reinforcement learning methods used are Q-learning [Moody and Saffell, 1998, Pendharkar and Cusatis, 2018], Deep Q-Networks (Q-learning with deep neural networks) [Park et al., 2020, Zhang et al., 2020]. Q-learning involves learning the Q-function given in Equation (1.6) and will be explained in more detail in Section 2.1. The policy-based methods used are Deterministic Policy Gradient (DPG) Jiang et al. [2017] Deep Deterministic Policy Gradient (DDPG) [Liang et al., 2018, Liu et al., 2018, Aboussalah et al., 2022], Proximal Policy Optimization [Liang et al., 2018, Sood et al., 2023, Aboussalah et al., 2022], where the latter two are popular state-of-the-art policy gradient methods. DDPG will be explained in Section 2.6.

The states used in the learning methods for portfolio optimization include historical price data (open, high, low and close price) and/or the returns computed from this data: such as the close-to-close return between two days, or the ratio of the highest price to the close price of the same day per asset [Zhang et al., 2020, Park et al., 2020, Liu et al., 2018, Pendharkar and Cusatis, 2018, Liang et al., 2018, Sood et al., 2023, Jiang et al., 2017]. In addition, Park et al. [2020], Liu et al. [2018], Sood et al. [2023] also use the previous action (portfolio weight or number of shares held) as an input, which can have a relation on whether an asset needs to be bought or sold. Zhang et al. [2020] use the price data to also add technical indicators (e.g. MACD or RSI) which indicate price patterns, as states. More interestingly, Sood et al. [2023] use volatility indicators such as the VIX and a calculation which determines whether an asset's volatility has increased over a rolling period. The historical data used as states pertain to a specific period. This results in a tensor of dimension (n, t, f) where n is the number of assets, t is the length of the historical period of the data, and f is the number of features. Using neural networks, this tensor can either be flattened to a one-dimensional input to a neural network, or a Recurrent Neural Network (RNN) can be used to handle time-series data which is used by Zhang et al. [2020],

Jiang et al. [2017].

The simplest rewards for portfolio optimization is the return of the portfolio between two periods and hence it is used in Park et al. [2020], Liu et al. [2018], Liang et al. [2018], Jiang et al. [2017]. Moody and Saffell [1998], Sood et al. [2023], however, use a Sharpe ratio of the portfolio as rewards which resembles the objective of portfolio optimization more closely. Specifically, a differential Sharpe ratio is used which was introduced by Moody and Saffell [1998] which is essentially an approximation of the rate of change of the Sharpe ratio. This was introduced to ensure additivity of the rewards in the reward objective given in Equation (1.1).

1.3.2 Application Challenges of Reinforcement Learning

Dulac-Arnold et al. [2019, 2020, 2021] have observed numerous challenges that frequently occur when applying reinforcement learning to real-world problems (as opposed to simulated problems). These problems are [Dulac-Arnold et al., 2019, 2020, 2021]

1. Sample inefficiency: learning on live systems from limited samples.
2. Dealing with unknown and potentially large delays in rewards.
3. Learning and acting in high-dimensional state and action spaces.
4. Incorporating system constraints that cannot be violated.
5. Interacting with partially observable systems, which can alternatively be viewed as non-stationary.
6. Learning from multi-objective or poorly specified reward functions.
7. Being able to provide actions quickly, especially for systems requiring low latencies.
8. Learning policies offline (and off-policy).
9. Obtaining explainability of policies.

Of these problems, we deemed the sample efficiency (Problem 1) and non-stationarity (Problem 5) to be the most important and pertinent for the portfolio optimization problem. Indeed, the problem of non-stationary financial market environments or non-stationary financial data was mentioned to be a relevant problem in Liang et al. [2018], Yu et al. [2019], Aboussalah et al. [2022], Hambly et al. [2023], however, Liang et al. [2018], Aboussalah et al. [2022] did not attempt to solve it. Furthermore, Yu et al. [2019] did not explicitly model non-stationarity, but instead, unconvincingly, merely used the percentage change of an asset as a state variable, which was assumed to be a non-stationary time series. The precise definition of non-stationarity will be provided in Section 4.3.1.

Regarding sample efficiency, while this is generally an important problem for any algorithm, it is also particularly relevant for portfolio optimization. On the spectrum of high-frequency to low-frequency financial problems, portfolio optimization is a low-frequency

problem. Whereas, in the high-frequency domain, there is usually an abundance of data (in terms of samples), in the low-frequency domain there is usually a lack of data. Hence it is important to have a sample-efficient reinforcement learning method in portfolio optimization.

1.3.3 Reinforced Deep Markov Models

To improve sample efficiency through model-based reinforcement learning, Ferreira [2020] used a Reinforced Deep Markov Model (RDMM) applied to optimal trade execution. This model was subsequently used by Cartea et al. [2021] in statistical arbitrage on an FX-triplet. Cartea et al. [2021], however, train the model on simulated data FX-price data, which omits the necessity for model-based reinforcement learning as sufficient samples can be generated artificially.

The term reinforced deep Markov model (RDMM) was coined by Ferreira [2020]. It is an integration between a partially observable Markov decision process (POMDP), deep neural networks and policy gradient. A POMDP is an MDP (introduced in Section 1.1.1) with additional unobserved variables z_t, which are called latent variables. Alternatively, RDMM can also be thought of as the integration of a POMDP and a deep latent variable model (DVLM), where a DVLM is any model with latent variables and deep neural networks.

Relatedly, Xie et al. [2021] introduced a way to model non-stationarity using a POMDP and Soft Actor-Critic (SAC)(explained in Section 2.7). Their implementation, lifelong latent actor-critic (LILAC), was applied to simulated environments with discrete state and action spaces.

1.4 Our Contribution

We propose a policy-based Non-Stationary Reinforced Deep Markov Model (NSRDMM) to improve on value-based and policy-based reinforcement learning methods applied to portfolio optimization. The method is an extension of Reinforced Deep Markov Models (RDMM) [Ferreira, 2020] that is designed to provide a model for non-stationary Markov Decision Processes (MDPs) [Xie et al., 2020] with time-varying state transition distribution, reward distributions and policy distribution. Our contribution, is combining the RDMM to incorporate non-stationarity in the reward distribution and policy function (following the framework by Xie et al. [2021]) and applying it to portfolio optimization.

Similar to Cartea et al. [2021], we use Gaussian distributions to model the continuous state and action spaces. These Gaussian distributions are parameterized by neural networks which are also called mean-variance-estimation (MVE) networks. Specifically, for training the MVE networks, we use a warm-up period where only the mean of the normal distribution is learned, instead of the mean and variance simultaneously, as suggested by Sluijterman et al. [2023]. Furthermore, as in Ferreira [2020] (but unlike [Cartea et al., 2021]) we use a Gaussian policy function.

Another contribution is using a multi-objective model-based reward function, where the learned mean reward and variance of rewards are simultaneously maximized and minimized, respectively (Ferreira [2020] use a single objective). In the model-based implementation, we avoid the issue of having a Mean-Variance objective in the reward function which makes it computationally challenging to find the global optimum for the policy (indeed, convergence to the global optimum is not guaranteed) [Mannor and Tsitsiklis, 2013]. This is because the variance of rewards is nonlinear in expectation such that Bellman's principle of optimality cannot be applied [Tamar et al., 2016, Vigna, 2020, Wang and Zhou, 2020]. Our model-based multi-objective function avoids the need for Bellman's principle of optimality because we the use the learned mean and variance parameters of the reward distribution.

Furthermore, even though RDMM was introduced partly for its data efficiency, it has only been applied to simulated data Ferreira [2020], Cartea et al. [2021]. We train NSRDMM on historical data to show it can indeed be applied in real-world applications by overcoming the issue of sample-inefficiency. A further contribution is that, unlike [Ferreira, 2020, Cartea et al., 2021], we use a off-policy gradient updates of the policy function to further improve sample-efficiency as suggested by Haarnoja et al. [2018b] through using a replay buffer.

Lastly, the states (features) we use are covariance matrices (and momentum indicators) to provide a more close comparison to portfolio optimization methods such as Mean-Variance. Unlike previous literature which mainly used a combination of return data and technical indicators.

1.5 Outline

In Chapter 2, we will introduce model-free reinforcement learning algorithms, Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic, which will be used in the numerical experiment. In the subsequent Chapter 3, we introduce the portfolio optimization methods Mean-Variance (MV) and Equal Rsik Contribution (ERC), which are popular portfolio optimization techniques in practice, as benchmarks in the numerical experiment. In Chapter 4, we describe the NSRDMM model used for portfolio optimization, including the objective functions used. In the numerical experiment (Chapter 5) the neural network architectures, data, training, validation and testing is explained, along with the results of the portfolio optimization experiment.

Chapter 2

Reinforcement Learning Methods

In this chapter, we introduce various reinforcement learning methods and concepts. We first explain Q-learning which is an important value-based reinforcement learning method. We will then explain Actor-Critic methods which combine policy-based methods with a Q-function to stabilize learning. In addition, we introduce Monte Carlo Policy Gradient which will be useful for understanding Deep Deterministic Policy Gradient and Soft Actor-Critic (SAC). The latter two algorithms will be used in the numerical experiment.

2.1 Q-learning

Q-learning is a value-based reinforcement learning algorithm introduced by Watkins [1989] and is used to approximate the Bellman equation for the Q-function 1.3. For each iteration the agent:

1. observes the current state s,
2. selects action a,
3. observes the next state s', where $s' \sim P(s, a)$,
4. receives a reward r.

For iteration n, the Q-function update is as follows [Hambly et al., 2023]

$$Q^{n+1}(s,a) \leftarrow (1-\beta_n) \underbrace{Q^n(s,a)}_{\text{current estimate}} + \beta_n(s,a) \underbrace{\left[r(s,a) + \gamma \max_{a'} Q^n(s',a') \right]}_{\text{new estimate}}, \tag{2.1}$$

where β_n is the learning rate that determines the weight given to the current estimate and the new estimate of the Q-function. The new estimate of the Q-function, $r(s,a) +$

$\gamma \max_{a'} Q^n(s', a')$, consists of the reward $r(s, a)$ from taking action a in state s, whereas the latter term denotes the best the agent 'thinks' it can do from the subsequent state [Watkins and Dayan, 1992]. See Algorithm 1 for the Q-learning steps.

The following theorem states that under certain conditions, Q^n converges to Q^*, the true Q-function.

Theorem 2. *[Watkins and Dayan, 1992] Assume $|\mathcal{A}| < \infty$ and $|\mathcal{S}| < \infty$. Let $n^i(s, a)$ denote the index of the ith time that action a is taken in state s. Let $R < \infty$ be a constant. Given bounded rewards $|r| < R$ and learning rate $\beta_n \in [0, 1)$, and*

$$\sum_{i=1}^{\infty} \beta_{n^i(s,a)} = \infty, \sum_{i=1}^{\infty} (\beta_{n^i(s,a))})^2 < \infty \, \forall s, a, \tag{2.2}$$

then, $Q_n(s, a) \to Q^*(s, a)$ with probability 1.

While Theorem 2 provides an asymptotic result, it does not give any guarantee on the number of iterations, or samples (s, a, r, s'), needed to give an accurate result.

Algorithm 1: Q-learning [Hambly et al., 2023]

Input:
Total number of iterations N; the policy π; the learning rate $\beta_n \in (0, 1]$
$(0 \leq n \leq N - 1)$
Initialize $Q^0(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize s
for $n = 0, \ldots, N - 1$ **do**
  Sample action a from $\pi(s)$
  Observe r and s' after taking action a
  Update Q^n according to 2.1
  $s \leftarrow s'$

2.2 Monte Carlo Policy Gradient

Monte Carlo Policy Gradient (also named REINFORCE) is one of the simplest policy-based methods [Williams, 1992] that uses the policy-gradient theorem (Theorem 1). Since there is an expression for $\nabla_\theta \ln \pi(s, a; \theta)$, it is only necessary to estimate $Q^\pi(s, a)$ when using the theorem. REINFORCE uses the sum of discounted rewards $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ as an unbiased sample of $Q^\pi(s, a)$ [Rao and Jelvis, 2022]. Then, for each timestep, the gradient ascent step is [Rao and Jelvis, 2022]

$$\beta \gamma^t \nabla_\theta \ln \pi(s, a; \theta) G_t. \tag{2.3}$$

Algorithm 2: REINFORCE

Initialize θ
for *each episode* $\{s_0, a_0, r_0, \ldots, s_T, a_T, r_T\} \sim \pi(\cdot, \cdot; \theta)$ **do**
  for $t = 0, \ldots, T$ **do**
    $\theta \leftarrow \theta + \beta\gamma^t \nabla_\theta \ln \pi(s_t, a_t; \theta) G_t$
return (θ)

From Algorithm 2, we can see that for one episode (or "trajectory" of states, actions and rewards), the parameter θ is updated $T + 1$ times, and all future rewards in the episode are used in each update.

While REINFORCE gives an unbiased estimate for the gradient step, it may suffer from high variance (as it is a Monte Carlo method), resulting in a slow convergence. This is an instance of the bias-variance tradeoff.

2.3 Actor-Critic

To mitigate the high variance of REINFORCE, a function approximation for Q^π can be used. Unlike the unbiased sample for the Q-function, G_t, which may vary a lot, the function approximation for the Q-function will be updated gradually, thereby reducing the variance in the gradient update. We denote the function approximation of the Q-function by $Q(s, a; w)$, where w is the parameter vector of the function approximation. Consequently, there are two function approximations, $\pi(s, a; \theta)$, which is called the 'Actor' and the 'Critic' $Q(s, a; w)$. Intuitively, "the Actor updates policy parameters in a direction suggested by the Critic." [Rao and Jelvis, 2022]. Therefore, Actor-Critic methods involve two function approximations to lower the variance of the policy gradient ascent step. In

Algorithm 3: Actor-Critic [Hambly et al., 2023] [Rao and Jelvis, 2022]

Initialize policy parameter θ and Q-function parameter w
Initialize state s
Sample $a \sim \pi(s, \cdot; \theta)$
for $n = 0, 1, \ldots, N - 1$ **do**
  Take action a, observe r, s'
  Sample $a' \sim \pi(s, \cdot; \theta)$
  $\delta \leftarrow \mathrm{r} + \gamma Q(s', a'; w) - Q(s, a; w)$ (TD-error)
  $w \leftarrow w + \beta_w \delta \nabla_w Q(s, a; w)$
  $\theta \leftarrow \theta + \beta_\theta Q(s, a; w) \nabla_\theta \ln \pi(s, a; \theta)$
  $s \leftarrow s'$, $a \leftarrow a'$

addition, the variance can be lowered by subtracting a baseline function $B(s)$ from the Q-

value estimate in the policy gradient update without introducing a bias to the estimation. This can be shown as follows from rewriting the policy gradient equation 1.9

$$\nabla_\theta J(\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi(s, a; \theta) \cdot Q^\pi(s, a) \cdot da \cdot ds. \tag{2.4}$$

Subtracting the baseline function does not introduce bias since [Rao and Jelvis, 2022]

$$\int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi(s, a; \theta) \cdot B(s) \cdot da \cdot ds \tag{2.5}$$

$$= \int_{\mathcal{S}} \rho^\pi(s) \cdot B(s) \cdot \nabla_\theta \int_{\mathcal{A}} \pi(s, a; \theta) \cdot da \cdot ds. \tag{2.6}$$

$$= \int_{\mathcal{S}} \rho^\pi(s) \cdot B(s) \cdot \nabla_\theta 1 \cdot ds \tag{2.7}$$

$$= 0 \tag{2.8}$$

A good baseline function is the state value function $V(s)$ from Equation 1.2.

2.4 Deep Reinforcement Learning

Some of the reinforcement learning algorithms introduced in the previous sections involved a form of function approximation through a parametrization of the Q-function or policy function. A widely used form of function approximation is neural networks due to desirable properties and effectiveness in practice. One of the desirable properties of neural networks is that they are universal function approximators. Let us provide a version of the universal approximation theorem.

Theorem 3. *[Cybenko, 1989, Hornik et al., 1989, Hornik, 1991] Let σ be any continuous function that is not a polynomial. Then, the set of functions*

$$\tilde{f}(\mathbf{x}) = \sum_{i=1}^{m} a_i \sigma(\mathbf{w_i^T x} + b_i) \tag{2.9}$$

is dense in $C([0,1]^d)$. I.e. given any continuous function $f : [0,1]^d \to \mathbb{R}$ and let $\epsilon > 0$, there is a function like $\tilde{f}(x)$ (2.9), for which

$$|f(\mathbf{x}) - \tilde{f}(\mathbf{x})| < \epsilon, \quad \forall x \in [0,1]^d. \tag{2.10}$$

Besides the useful result of the universal approximation theorem, neural networks are known to deal well with the high dimensionality of features, which is desirable for reinforcement learning settings with large state and action spaces. Moreover, neural networks can be learned in an online fashion, which is useful in many settings where new data is received frequently.

2.5 Deep Q-Networks

In Deep Q-Networks (DQN), a neural network $Q(s, a; w)$ is used as a function approximation for the Q-function. We define the policy $\mu(s)$ to be a greedy policy, i.e. $\mu(s) = \arg\max_a Q(s, a)$. Let π be any stochastic policy, then the loss function for the function approximation is [Lillicrap et al., 2015]

$$\mathcal{L}(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi} \left[\left(Q(s_t, a_t; w) - y_t \right)^2 \right], \tag{2.11}$$

where

$$y_t = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}); w). \tag{2.12}$$

This loss function learns the Q-function off-policy. It should be noted that when using non-linear function approximations such as neural networks, the convergence of Q-learning is no longer guaranteed [Lillicrap et al., 2015].

2.6 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a policy-based method that was invented to extend Deep Q-learning to continuous action spaces. It is an extension built on the Deterministic Policy Gradient (DPG) introduced by Silver [Silver et al., 2014]. For DPG we denote a deterministic policy function by μ_θ. To improve a policy, it is common to use a greedy maximization of $\arg\max_{a \in \mathcal{A}} Q^\mu(s, a)$ with respect to θ. However, since it requires a global maximization at every update step, it becomes computationally infeasible for continuous action spaces [Lillicrap et al., 2015]. Instead, a computationally more feasible method than globally maximizing the Q-function is to update the policy parameters in the *direction* of $\nabla_\theta Q^{\pi_\theta}(s, a)$. Since each state may suggest a different update direction, the average is taken over the states with respect to the stationary state distribution ρ^π. Therefore, the policy improvement update is [Silver et al., 2014]

$$\theta^{n+1} = \theta^n + \beta \, \mathbb{E}_{s \sim \rho^\mu} \left[\nabla_\theta Q^\mu(s, \mu_\theta(s)) \right], \tag{2.13}$$

which, by the chain rule can be written as

$$\theta^{n+1} = \theta^n + \beta \, \mathbb{E}_{s \sim \rho^\mu} \left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a) \Big|_{a = \mu_\theta(s)} \right]. \tag{2.14}$$

Here, $\nabla_\theta \mu_\theta(s)$ is a Jacobian matrix where each column is the gradient $\left[\nabla_\theta \mu_\theta(s) \right]_d$ of the dth dimension of the action space. Note that the state distribution ρ^μ changes as the policy changes. Like the (stochastic) policy gradient theorem, this does not give any problems as the gradient of the state distribution does not need to be computed.

We now introduce the deterministic policy gradient theorem.

Theorem 4. *[Silver et al., 2014] Suppose $\nabla_\theta \mu_\theta$, $\nabla_a Q^\mu$ and the deterministic policy gradient exist, then*

$$\nabla_\theta J(\theta) = \int_{\mathcal{S}} \rho^\mu(s) \cdot \nabla_\theta \mu_\theta(s) \cdot \nabla_\theta Q^\mu(s, a)|_{a=\mu_\theta(s)} \cdot ds \tag{2.15}$$

$$= \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \cdot \nabla_\theta Q^\mu(s, a)|_{a=\mu_\theta(s)}\right] \tag{2.16}$$

The relation of the deterministic policy gradient to the stochastic policy gradient is that for many stochastic policies, the deterministic policy gradient is, in the limit, a specific case of the stochastic policy gradient [Silver et al., 2014]. The importance of Theorem 4, lies in the fact that it allows the usage of the policy gradient framework—like in the stochastic case—on deterministic policies while making the gradient easier to compute since it avoids an integral over the action space.

Since most optimization methods assume i.i.d. samples, which does not hold in sequential exploration, it is necessary to learn in batches as opposed to online. This can be achieved by using a large enough replay buffer which stores samples (s_t, a_t, r_t, s_{t+1}), where the oldest samples will be deleted when the replay buffer is full. In addition, likewise to DQN, updating the Q-function may be unstable. To mitigate instability a target Q-function and a target policy are used which is updated more slowly (with rate $\tau \ll 1$). To ensure exploration noise is required from, for example, a normal distribution as a deterministic policy does not provide any randomness for exploration. A caveat is that DDPG does require a large number of training episodes [Lillicrap et al., 2015] (Haarnoja et al. [2018a] state that it is relatively sample-efficient, but it is very sensitive to hyperparameters). Indeed, similar to DQN, there are no convergence guarantees for (D)DPG. However, despite these drawbacks, it is a method that has found success in numerous practical applications.

2.7 Soft Actor-Critic

Soft Actor-Critic (SAC) is an off-policy actor-critic algorithm. Its goal is to maximize the expected reward, while also maximizing the entropy—i.e. to perform a task well while acting as randomly as possible. Maximum entropy improves robustness (to model and estimation errors) and exploration. SAC shows better performance and sample complexity than other on-policy and off-policy algorithms.

The maximum entropy objective is [Haarnoja et al., 2018a]

$$J(\theta) = \sum_{t=0}^{T} \gamma^t \, \mathbb{E}_{s\sim\rho^\pi, a\sim\pi}\left[r(s_t, a_t) + \alpha\mathcal{H}(\pi(s, a))\right], \tag{2.17}$$

where α is the 'temperature' which determines the weight given to the entropy term $\mathcal{H}$ compared to the reward. Naturally, as $\alpha \to 0$, we recover the standard RL-objective function of reward maximization.

Algorithm 4: DDPG [Lillicrap et al., 2015]

Initialize critic network $Q(s, a; w)$, actor $\mu(s; \theta)$ and weights w, θ
Initialize target network Q' and μ' with weights $w' \leftarrow w$, $\theta' \leftarrow \theta$
Initialize replay buffer $\mathcal{D}$
for *episode = 1, . . . , M* **do**
  Initialize random process $\mathcal{N}$
  Initialize state s
  for t = *1,. . . ,T* **do**
    Select action $a_t = \mu(s_t; \theta) + \mathcal{N}_t$
    Take action a_t, observe r_t, s_{t+1}
    Store sample (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample a random minibatch of N transitions (s_t, a_t, r_t, s_{t+1}) from $\mathcal{D}$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}; \theta'); w')$
    Update critic by minimizing the loss

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (y_i - Q(s_i, a_i; w))^2$$

    Update the policy using the sampled policy gradient $\theta^{n+1} = \theta^n + \beta \nabla_\theta J(\theta)$ with

$$\nabla_\theta J \approx \frac{1}{N} \sum_{i=1}^{n} \nabla_a Q(s, a; w)|_{s=s_i, a=\mu(s_i)} \nabla_\theta \mu(s; \theta)|_{s=s_i}$$

    Update the target networks

$$w' \leftarrow \tau w + (1 - \tau) w'$$
$$\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$$

To derive the algorithm it is useful to first define the soft policy iteration—a general method to learn maximum entropy policies [Haarnoja et al., 2018a]. We first consider theoretical results for tabular setting (discrete and bounded action and state spaces) which will then be extended to the continuous case. Let π be some fixed policy, and let $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ be any Q-function, then the soft Q-value can be found by iteratively applying the Bellman backup operator $\mathcal{T}^{\pi}$ which is given by [Haarnoja et al., 2018a]

$$\mathcal{T}^{\pi} Q(s_t, a_t) := r(s_t, a_t) + \gamma \, \mathbb{E}_{s_{t+1} \sim P(s,a)} \left[V(s_{t+1}) \right], \tag{2.18}$$

where

$$V(s_t) = \mathbb{E}_{a_t \sim \pi} \left[Q(s_t, a_t) - \log \pi(a_t | s_t) \right], \tag{2.19}$$

where the expectation of $-\log \pi(s_t|a_t)$ denotes the entropy.

Lemma 1. (Soft Policy Evaluation) [Haarnoja et al., 2018a]. Let $\mathcal{T}^{\pi}$ be the soft Bellman backup operator defined in Equation 2.18 and let $Q^0 : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ (with $|\mathcal{A}| < \infty$. Define $Q^{k+1} = \mathcal{T}^{\pi} Q^k$. Then the sequence Q^k converges to the soft Q-function of policy π as $k \rightarrow \infty$.

In the policy improvement step, the policy is updated in the direction of the exponential of the Q-function. The policy π will be selected from a set of tractable policies Π (e.g. Gaussian policies). This is done by projecting the policy unto the set of admissible policies Π. The Kullback-Leibler divergence is used to project the policy as it turns out to be convenient. The policy update is done as follows [Haarnoja et al., 2018a]

$$\pi_{new} = \arg\min_{\pi' \in \Pi} D_{KL} \left(\pi(\cdot|s_t) \middle\| \frac{exp(Q^{\pi_{old}}(s_t, \cdot))}{Z^{\pi_{old}}(s_t)} \right), \tag{2.20}$$

where $Z^{\pi_{old}}(s_t)$ is a normalizing factor for the distribution, which is not necessarily tractable. Fortunately, it can be omitted as it has no effect on the policy gradient.

Theorem 5. Soft Policy Iteration [Haarnoja et al., 2018a]. Let $pi \in \Pi$, then π converges to $\pi^* \in \Pi$ from iterative applications of the soft policy evaluation and improvement step. Then $Q^{\pi^*}(s_t, a_t) \geq Q^{\pi}(s_t, a_t)$ for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$, assuming that $|\mathcal{A}| < \infty$.

Neural networks can be used to approximate the Q-function $Q(s, a; w)$ and the value function $V(s; \psi)$, and a Gaussian distribution with neural networks for the mean and covariance for the policy distribution $\pi(a|s; \theta)$. The SAC Algorithm 5 collects samples from the environment after which it performs a gradient step. Usually, multiple gradient steps are performed for each environment step. Gradient steps are performed for the value function, the Q-function and the policy.

The objective function of the soft value function is

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_{\psi}(s_t) - \mathbb{E}_{a_t \sim \pi_{\theta}} \left[Q_w(s_t, a_t) - \log \pi_{\theta}(a_t, s_t) \right] \right)^2 \right]. \tag{2.21}$$

Algorithm 5: Soft Actor-Critic [Haarnoja et al., 2018a]

Initialize parameter vectors θ, ψ, w
for *each iteration* **do**
  for *each environment step* **do**
    $a_t \sim \pi_\theta(a_t|s_t)$
    $s_{t+1} \sim P(s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
  for *each gradient step* **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $w_i \leftarrow w_i - \lambda_Q \hat{\nabla}_{wi} J_Q(w_i)$ for $i \in \{1, 2\}$
    $\theta \leftarrow \theta - \lambda_\pi \hat{\nabla}_\theta J_\pi(\theta)$
    $\psi' \leftarrow \tau\psi + (1-\tau)\psi'$

The unbiased estimator of the gradient of the soft value function is given by

$$\hat{\nabla} J_V(\psi) = \nabla_\psi (\, V_\psi(s_t) - Q_w(s_t, a_t) + \log \pi_\theta(a_t|s_t)) \tag{2.22}$$

The objective function of the soft Q-function is the TD error:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}} \left[\frac{1}{2} \left(Q_w(s_t, a_t) - r(s_t, a_t) - \gamma\, \mathbb{E}_{s_{t+1}\sim P(s,a)} \left[V'_\psi(s_{t+1}) \right] \right)^2 \right], \tag{2.23}$$

of which the gradient can be estimated by

$$\hat{\nabla}_w J_Q(w) = \nabla_w Q_w(s_t, a_t) \left(Q_w(s_t, a_t) - r(s_t, a_t) - \gamma\, V'_\psi(s_{t+1}) \right), \tag{2.24}$$

where V'_ψ is the target network. Two Q-functions are used to speed up training.

For the policy function, the target density is the Q-function which is a neural network that can be differentiated. Therefore it is convenient to reparameterize the policy by a neural network which can also be differentiated

$$a_t = f_\theta(\epsilon_t; s_t), \tag{2.25}$$

where ϵ is some noise vector (e.g. sampled from a Gaussian). The policy objective function from Equation 2.20 can be rewritten to

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim\mathcal{D}, \epsilon_t\sim\mathcal{N}} \left[\log \pi_\theta(f_\theta(\epsilon_t; s_t)|s_t) - Q_w(s_t, f_\theta(\epsilon_t; s_t)) \right] \tag{2.26}$$

The normalization term from Equation 2.20 does not depend on θ and can thus be omitted. Then, the gradient of J_π is

$$\hat{\nabla} J_\pi(\theta) = \nabla_\theta \log \pi_\theta(a_t|s_t) + (\nabla_{a_t} \log_\theta(a_t|s_t) - \nabla_{a_t} Q(s_t, a_t))\, \nabla_\theta f_\theta(\epsilon_t; s_t) \tag{2.27}$$

Chapter 3

Portfolio Optimization

In this chapter, we will introduce two portfolio optimization techniques that are often used in practice: Mean-Variance and Equal Risk Contribution. Both techniques will be used as a benchmark in the numerical experiment.

3.1 Mean-Variance

One of the common objectives of portfolio optimization consists of maximizing the expected return while minimizing its variance (risk). It usually involves maximizing the expected return for a given level of variance or minimizing the variance for a given level of expected return. This objective was introduced by Markowitz in his 1952 paper on *Portfolio Selection* [Markowitz, 1952]. Here, Markowitz introduced a single-period optimization problem. This single-period optimization problem was subsequently solved by Merton in 1972 [Merton, 1972].

The objective is formalized as follows. Let $\{x_t^w, 0 \leq t \leq T\}$ be the wealth process, i.e. the portfolio value at time t, under the allocation strategy $w = \{w_t, 0 \leq t \leq T\}$. Then, the Mean-Variance objective is [Wang and Zhou, 2020]

$$\min_w \text{Var}[x_T^w] \tag{3.1}$$

$$\text{s.t. } \mathbb{E}[x_T^w] = z, \tag{3.2}$$

where $z \in \mathbb{R}$ is the desired mean return set at time t and to be attained at time T of the investment horizon $[0, T]$. If, additionally, we require Equation 5.1 to hold we speak of the constrained mean-variance objective.

The downside of the mean-variance objective is that it suffers from time-inconsistency—i.e. the optimal solution at time t is not the optimal solution at time $s > t$. This is because the Bellman principle of optimality does not hold due to the non-linear function in the expectation of the mean-variance objective in 3.2 [Vigna, 2020].

If one relaxes the requirement of finding a time-consistent solution and instead looks at the optimal solution at $t = 0$, then the optimization problem can be solved by applying a Lagrange multiplier λ

$$\min_w \mathbb{E}[(x_T^w)^2] - z^2 - 2\lambda(\mathbb{E}[(x_T^w)] - z) = \min_w \mathbb{E}[(x_T^w - \lambda)^2] - (\lambda - z)^2. \tag{3.3}$$

This problem has an analytical solution $w^* = \{w_t^*, 0 \leq t \leq T\}$ depending on λ, which in turn is determined by $\mathbb{E}[x_T^{w^*}] = z$ [Wang and Zhou, 2020].

3.1.1 Global Minimum Variance

It should be noted that optimal Mean-Variance portfolios do suffer from drawbacks in practice. It has been observed that such portfolios tend to exhibit high concentrations in a small subset of the assets [Maillard et al., 2010]. In addition, the optimal mean-variance portfolio is very sensitive to its input— minor perturbations in its input parameters cause substantial differences in the portfolio composition. This is most notable for changes in the expected return [Merton, 1980].

Global minimum variance (GMV) omits the use of the expected return. Therefore, it is considered to be robust by practitioners [Maillard et al., 2010]. GMV is an unconstrained minimization of the variance; it seeks to minimize objective 3.2 without the expected return constraint in 3.3. Conveniently, this problem allows a unique closed-form solution.

Let Σ be the (constant) covariance matrix of the n assets. Then the variance of the wealth process x_T is $w^\intercal \Sigma w$. Then, the GMV optimization problem is [Back, 2010]

$$\min_w \frac{1}{2} w^\intercal \Sigma w \tag{3.4}$$

$$\text{s.t. } w_t^\intercal \mathbf{1} = 1. \tag{3.5}$$

With a Lagrange multiplier λ the Lagrangian of the above optimization problem is

$$\frac{1}{2} w^\intercal \Sigma w - \lambda(w_t^\intercal \mathbf{1} - 1). \tag{3.6}$$

From setting the partial derivative with respect to w to, it can be deduced that

$$\Sigma w - \lambda \mathbf{1} = 0 \implies w = \lambda \Sigma^{-1} \mathbf{1}. \tag{3.7}$$

Likewise, by setting the partial derivative with respect to the Lagrange multiplier λ equal to zero

$$w^\intercal \mathbf{1} = 1. \tag{3.8}$$

Substituting w from Equation 3.7 into Equation 3.8 gives $\lambda = 1/(\mathbf{1}^\intercal \Sigma^{-1} \mathbf{1})$. Hence, the global minimum variance solution w_{gmv}^* is given by

$$w_{gmv}^* = \frac{\Sigma^{-1} \mathbf{1}}{\mathbf{1}^\intercal \Sigma^{-1} \mathbf{1}}. \tag{3.9}$$

3.2 Equal Risk Contribution

Equal Risk Contribution (ERC) is a portfolio construction methodology that aims to allocate risk equally among the assets in a portfolio. Unlike traditional portfolio optimization methods that focus on maximizing returns for a given level of risk, ERC focuses solely on the risk aspect—therefore, ERC is assumed to be robust. The main idea of ERC is to construct a portfolio where each asset contributes equally to the overall risk, which is typically measured by volatility. ERC, therefore, is a specific (and widely used) instance of risk parity, which focuses on the allocation of risk.

The share of total portfolio risk attributable to an asset i, is the risk contribution of that asset. It is defined by the asset's weight multiplied by the marginal risk contribution—i.e. the infinitesimal change in the portfolio risk by changing the asset i's weight [Maillard et al., 2010].

Let us define a portfolio $w = (w_1, w_2, \ldots, w_n)$ of n assets. Let σ_i^2 represent the variance of asset i, let σ_{ij}^2 denote the covariance between assets i and j, and let Σ denote the covariance matrix. Let the risk of the portfolio be defined by $\sigma(w) = \sqrt{w^\intercal \Sigma w} = \sqrt{\sum_i w_i^2 \sigma_i^2 + \sum_i \sum_{j \neq i} w_i w_j \sigma_{ij}}$. Then the marginal risk contribution is [Maillard et al., 2010]

$$\delta_{w_i}\sigma(w) = \frac{\delta\sigma(w)}{\delta(w_i)} = \frac{w_i\sigma_i^2 + \sum_{j \neq i} w_j\sigma_{ij}}{\sigma(w)}. \tag{3.10}$$

It is called the marginal risk contribution as it represents the change in a portfolio's volatility for an infinitesimal change in an asset's weight. Then, the total risk contribution of asset i is, $\sigma_i = w_i \cdot \delta_{w_i}$. Whereas the risk of the overall portfolio is the sum of the individual contributions [Maillard et al., 2010]

$$\sigma(w) = \sum_i \sigma_i. \tag{3.11}$$

Using the covariance matrix instead, the marginal risk contribution is $\frac{\Sigma w}{\sqrt{w^\intercal \Sigma w}}$. This can be seen from $w^\intercal \frac{\Sigma w}{\sqrt{w^\intercal \Sigma w}} = \sqrt{w^\intercal \Sigma w} = \sigma(w)$.

Let us consider only portfolios without short-selling, i.e. $w \in [0,1]^n$. The aim is to find a portfolio that has equal risk contributions for each respective asset within the portfolio. This can be formalized as follows, we need to find the portfolio w^* where [Maillard et al., 2010]

$$w^* = \left\{ w \in [0,1]^n : \sum_i w_i = 1,\ w_i \cdot \delta_{w_i}\sigma(w) = w_j \cdot \delta_{w_j}\sigma(w)\, \forall i, j \right\}. \tag{3.12}$$

Let $(\Sigma w)_i$ be the ith row of the the vector resulting form the multiplication of Σ with w, and note that $\delta_{w_i}\sigma(w) \propto (\Sigma w)_i$. Then, similar to Equation 3.13

$$w^* = \left\{ w \in [0,1]^n : \sum_i w_i = 1,\ w_i \cdot (\Sigma w)_i = w_j \cdot (\Sigma w)_j\, \forall i, j \right\}. \tag{3.13}$$

3.2.1 Equal Correlation Case

Let us consider a scenario of $n > 2$ assets that have an equal correlation ρ. Then, from Equation 3.10 the total risk contribution becomes

$$\frac{w_i^2\sigma_i^2 + \rho\sum_{j\neq i} w_i w_j \sigma_i \sigma_j}{\sigma(w)}. \tag{3.14}$$

It can be shown that, assuming $\rho \geq -\frac{1}{n-1}$, the weight w_i that equalizes the total risk contribution of all assets is given by [Maillard et al., 2010]

$$w_i = \frac{\sigma_i^{-1}}{\sum_{j=1}^{n} \sigma_j^{-1}}. \tag{3.15}$$

This shows the assets' weights are scaled inversely to their volatility and normalized by the sum of all weights. Therefore, the equal correlation case has an analytical solution to the equal risk contribution problem.

3.2.2 Equal Risk Contribution Optimization Problem

Unless there is an equal correlation between all assets there is no closed-form solution. It therefore has to be solved numerically. The optimization problem can be formulated as [Maillard et al., 2010]

$$w^* = \arg\min f(w) \tag{3.16}$$

$$s.t. \quad w^\intercal \mathbf{1} = 1 \text{ and } w \in [0,1]^n, \tag{3.17}$$

where

$$f(w) = \sum_{i=1}^{n}\sum_{j=1}^{n}(w_i(\Sigma w)_i - w_j(\Sigma w)_j)^2. \tag{3.18}$$

This optimization problem can be solved using a sequential quadratic programming method [Maillard et al., 2010].

3.2.3 Comparison of Minimum-Variance, Equal Risk Contribution and $1/n$ Strategy

The $1/n$ portfolio, or equal-weight portfolio, is a simple heuristical portfolio allocation method that allocates a weight $w_i = \frac{1}{n}$ to each asset i. When $n = 2$, the weight $w_{1/n}^*$ is therefore equal to $\frac{1}{2}$. Similarly, when equating the total risk contributions in the two-asset case for the ERC portfolio, it is easy to derive that w_{erc}^*, too is equal to $\frac{1}{2}$ when $\sigma_1 = \sigma_2$ (see [Maillard et al., 2010]). For MV, the explicit unconstrained solution for the two-asset case is $w_1^* = (\sigma_2^2 - \rho\sigma_1\sigma_2)/(\sigma_1^2 + \sigma_2^2 - 2\rho\sigma_1\sigma_2)$. Here, too, the MV portfolio coincides with the equal-weight portfolio when the volatilities are equal, i.e. $\sigma_1 = \sigma_2$.

For the general case, the comparison between ERC, MV, and $1/n$ is as follows [Maillard et al., 2010]

$$\begin{aligned} w_i &= w_j && \text{(1/n)} \\ \delta_{w_i}\sigma(w) &= \delta_{w_j}\sigma(w) && \text{(MV)} \\ w_i \cdot \delta_{w_i} &= w_j \cdot \delta_{w_j}, && \text{(ERC)} \end{aligned}$$

where the MV case is equivalent to equalizing the marginal risk contributions [Maillard et al., 2010], rather than the total risk contributions in the ERC case. Because it can further be shown that

$$\sigma_{mv}(w) \leq \sigma_{erc}(w) \leq \sigma_{1/n}. \tag{3.19}$$

Therefore, the ERC portfolio is said to lie between the equal-weight and MV portfolio (see [Maillard et al., 2010] for details on the derivation).

Chapter 4

Non-Stationarity in Reinforced Deep Markov Models

In this chapter, we will introduce the variational autoencoding (VAE) framework by Kingma and Welling [2013], Kingma et al. [2019]. We will use VAEs to modify RDMM to our needs for portfolio optimization that takes into account non-stationarity.

4.1 Variational Autoencoders

In this section, useful concepts from VAEs are introduced. This includes, directed probabilistic graphical models, which we will use to design our model for portfolio optimization. Furthermore, we introduce latent variable models and the related problem of intractability of the posterior distribution. Lastly, we discuss variational inference in VAEs.

4.2 Directed Probabilistic Graphical Models

In the following section, we use a directed probabilistic graphical model to describe our Markov Decision Process(es). First, let us introduce these probabilistic models. Let $\mathbf{x}$ denote a set of observed random variables which has an unknown probability distribution $p^*(\mathbf{x})$. The aim is to estimate or learn this probability distribution with a chosen model $p_{\boldsymbol{\theta}}(\mathbf{x})$ such that [Kingma et al., 2019]

$$p_{\boldsymbol{\theta}}(\mathbf{x}) \approx p^*(\mathbf{x}), \tag{4.1}$$

where $\boldsymbol{\theta}$ are the parameters of the chosen model (we will use $\boldsymbol{\theta}$ to denote the set of all parameters for the decoding model which will be introduced). This can be extended to conditional models with another set of observable random variables $\mathbf{y}$, where a model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{y})$ is chosen such that [Kingma et al., 2019]

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{y}) \approx p^*(\mathbf{x}|\mathbf{y}). \tag{4.2}$$

Directed probabilistic graphical models consist of a directed acyclic graph where all variables have a topological order [Kingma et al., 2019]. The joint distribution $\mathbf{x} = (x_1, \ldots, x_N)$ of the observed variables in such models can be factorized into the conditional probability distributions distributions [Kingma et al., 2019]

$$p_{\boldsymbol{\theta}}(x_1, \ldots, x_N) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(x_i | Pa(x_i)), \tag{4.3}$$

where $Pa(x_i)$ is the set of parents for node x_i — i.e. the set of nodes with edges directed towards x_i (for a visual example see Figure 4.1). Nodes without parents are the prior probability distributions. Furthermore, each conditional probability distribution $p_{\boldsymbol{\theta}}(x_i|Pa(x_i))$ or $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{y})$ can be parameterized by a neural network.

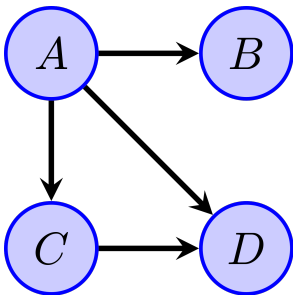


Figure 4.1: Directed graphical model with factorization $p(A, B, C, D) = p(A) \cdot p(B|A) \cdot p(C|A) \cdot p(D|A, C)$ [Wikipedia, 2024].

4.2.1 Latent Variable Models

The directed graphical models can be extended to incorporate latent variables $\mathbf{z} = (z_1, \ldots, z_N)$. These latent variables, while part of the model, are unobserved. The graphical model could then be described by the joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$. From the joint distribution, the marginal distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ of observed variables $\mathbf{x}$ can be computed by

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) \, d\mathbf{z}. \tag{4.4}$$

An example of a simple directed graphical model that includes latent variables can be given by [Kingma et al., 2019]

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \, p_{\boldsymbol{\theta}}(\mathbf{z}), \tag{4.5}$$

where $p_{\boldsymbol{\theta}}(\mathbf{z})$ is the prior probability distribution of $\mathbf{z}$. This model can also be extended to a conditional model where

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}|\mathbf{y}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, \mathbf{y}) \, p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{y}). \tag{4.6}$$

We note that when the conditional distributions are parameterized by neural networks the models are called deep latent variable models (DLVM).

4.2.2 Intractability

While the chosen model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p_{\boldsymbol{\theta}}(\mathbf{z})$ is tractable, the marginal probability is often intractable [Kingma et al., 2019]

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p_{\boldsymbol{\theta}}(\mathbf{z})\, d\mathbf{z}. \tag{4.7}$$

The integral namely becomes intractable for complicated models $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ which happens to be the case for neural networks with a non-linear activation function [Kingma and Welling, 2013] (which is what we will use). Since we have the following relation for the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p_{\boldsymbol{\theta}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})}, \tag{4.8}$$

we have that the tractability of the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is directly related to the tractability of the marginal distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ (remember that the model we choose $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p_{\boldsymbol{\theta}}(\mathbf{z})$ is tractable). Therefore, if the marginal distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ is intractable, then the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is intractable too.

4.2.3 Variational Inference

While the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is intractable, it is possible to estimate the posterior with a tractable inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ parameterized by the variational parameters ϕ. This model is called an encoder because it encodes the observed variables $\mathbf{x}$ to a distribution over the values $\mathbf{z}$ (that may have generated $\mathbf{x}$) [Kingma et al., 2019]. On the other hand, the model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is called the decoder as it decodes the latent variables $\mathbf{z}$ to the observed variables $\mathbf{x}$. The inference model is used to learn the variational parameters ϕ such that [Kingma et al., 2019]

$$q_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z}) \approx p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}). \tag{4.9}$$

Similar to Equation (4.3) Kingma et al. [2019] state that inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ can be almost any directed graphical model

$$q_{\boldsymbol{\phi}}(z_1, \ldots, z_N) = \prod_{i=1}^{N} q_{\boldsymbol{\phi}}(z_i|Pa(z_i), \mathbf{x}), \tag{4.10}$$

where the conditional distributions $q_{\boldsymbol{\phi}}(z_i|Pa(z_i), x)$ can also be modeled by deep neural networks. Note that the variational parameters ϕ are shared across all variables $\mathbf{z}$ as opposed to having variational parameters $\boldsymbol{\theta}_i$ for each z_i (i.e. $q_{\boldsymbol{\phi}_i}(z_i|Pa(z_i), \mathbf{x})$. This method is called variational autoencoders in which the sharing of parameters is what makes the

method more efficient as opposed to variational inference where the parameters are not shared. The posterior distribution is approximated by optimizing the evidence lower bound (ELBO) — a lower bound on the log-likelihood of the observations which we will derive specifically for our model that incorporates non-stationarity.

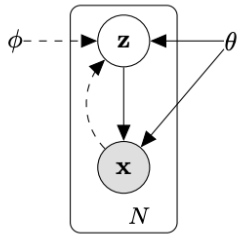


Figure 4.2: Visual representation of variational autoencoder (VAE) model [Kingma and Welling, 2013]. Solid lines denote the generative model; dashed lines denote the inference model.

4.3 Non-Stationary Reinforced Deep Markov Model

4.3.1 Non-Stationarity

In this section, we will introduce a probabilistic model that incorporates non-stationarity. First, we need to define non-stationarity. Let $(X_t : t \in \mathbb{Z})$ be a sequence of random variables.

Definition 1. [Van der Vaart, 2010] The stochastic process X_t is *strictly stationary* if, for any integer $h \in \mathbb{N}$, the unconditional joint probability distribution of the vector $(X_t, X_{t+1}, \ldots, X_{t+h})$ does not depend on t.

Therefore, the definition of strict stationarity implies that non-stationarity (in the strict sense), will have an unconditional joint distribution that is not independent of t.

An MDP is said to be *stationary* if both its transition probability function $p(s_{t+1}|s_t, a_t)$ and its reward function $r(s_t, a_t)$ do not change over time [Hambly et al., 2023]. Here, s_t and s_{t+1} denote the current and next state, respectively, and a_t represents an action at time t. An MDP is therefore considered *non-stationary* if either its transition probability function $p(s_{t+1}|s_t, a_t)$, its reward function $r(s_t, a_t)$, or both change over time. This means that the probabilities of moving from one state to another given an action, and/or the rewards received for such transitions depend on the time at which the action is taken. Consequently, time-varying transition distributions and reward distributions are particularly important [Hambly et al., 2023]. Moreover, as the state transition and reward distribution change, ideally, the policy function should change to remain optimal in a different environment.

4.3.2 Probabalistic Model for Non-Stationary Markov Decision Process

Xie et al. [2021] introduced a way to include non-stationarity in a partially observable Markov decision process (POMDP) which is similar to a latent variable model that also includes actions. To model non-stationarity, Xie et al. [2021] assume there is a sequence of $N \in \mathbb{N}$ MDPs, where each MDP $i \in \{1, \dots, N\}$ has a latent random variable z^i that describes the dynamics of the MDP. Within each MDP, the observable variables are the states s_t, the actions a_t, and the rewards r_t for each timestep $t \in \{1, \dots, T\}$. Therefore, in the non-stationary MDP framework, the latent variable z^i determines the state transitions $p_{\boldsymbol{\theta}}(s_{t+1}|s_t, a_t; z_i)$ and the reward distribution $p_{\boldsymbol{\theta}}(r_t|s_t, a_t; z_t)$ of MDP i [Xie et al., 2020]. Similar to the VAE setting, we include $\boldsymbol{\theta}$ to denote the parameters of the chosen probability model.

Therefore, the model consists of a Markov chain of latent variables z^i where there is a different MDP for each latent variable. Within each MDP, there $s^i_{1:T} = (s^i_1, \dots, s^i_T)$ be the sequence of observable states within MDP i. Similarly, let $a^i_{1:T} = (a^i_1, \dots, a^i_T)$ and $r^i_{1:T} = (r^i_1, \dots, r^i_T)$ be the sequence of observable actions and observable rewards within each MDP i, respectively. Since we assume there are N MDPs, let $s^{1:N}_{1:T} = \{s^1_{1:T}, \dots, s^N_{1:T}\}$, $a^{1:N}_{1:T} = \{a^1_{1:T}, \dots, a^N_{1:T}\}$, $r^{1:N}_{1:T} = \{r^1_{1:T}, \dots, r^N_{1:T}\}$ denote the set of states, actions and rewards sequences from each MDP. Furthermore, let $z^{1:N} = \{z^1, \dots, z^N\}$ be the set of latent variables that describe each MDP respectively.

The joint distribution over all variables of a chosen directed probabilistic graphical model can be factorized into a prior probability and conditional distributions. This model allows us to define a factorization of the joint probability of latent variables, states and rewards the actions $p_{\boldsymbol{\theta}}(z^{1:N}, s^{1:N}_{1:T}, r^{1:N}_{1:T}|a^{1:N}_{1:T})$. This can be compared to setting $\mathbf{z} = z^{1:N}, \mathbf{x} = (s^{1:N}_{1:T}, r^{1:N}_{1:T})$, and $\mathbf{y} = a^{1:N}_{1:T}$ in the conditional model from Equation (4.6). Then, our graphical model has the following factorization

$$p_{\boldsymbol{\theta}}(z^{1:N}, s^{1:N}_{1:T}, r^{1:N}_{1:T}|a^{1:N}_{1:T}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(z^i|z^{i-1}, s^{i-1}_{1:T}, r^{i-1}_{1:T}, a^{i-1}_{1:T})\, p_{\boldsymbol{\theta}}(s^i_{1:T}, r^i_{1:T}|z^i, a^i_{1:T}), \tag{4.11}$$

where we set the initial conditional probability $p_{\boldsymbol{\theta}}(z^1|z^0, s^0_{1:T}, r^0_{1:T}, a^0_{1:T})$ to be $p_{\boldsymbol{\theta}}(z^1)$, the prior distribution of the initial latent variable z^1, for ease of notation. According to Kingma et al. [2019], we may choose any directed graphical model in this framework. This graphical model is shown in Figure 4.3.

In Equation (4.11), we assume that the latent variable z^i is dependent on the latent variable of the previous MDP z^{i-1} and on the previous sequences of states, actions, rewards $(s^{i-1}_{1:T}, a^{i-1}_{1:T}, r^{i-1}_{1:T})$. We have chosen this dependence for two reasons. Firstly, we believe it resembles financial markets; for example, when the latent variable z^i describes a market regime, we think it is reasonable to assume it depends on the previous regime z^{i-1} instead of all preceding regimes $z^{1:i-1}$. Similarly, we believe it is a reasonable assumption that the data or observable variables of the previous MDP $(s^{i-1}_{1:T}, a^{i-1}_{1:T}, r^{i-1}_{1:T})$ provide information about the next (hidden) market regime z^i, instead of assuming all preceding observable

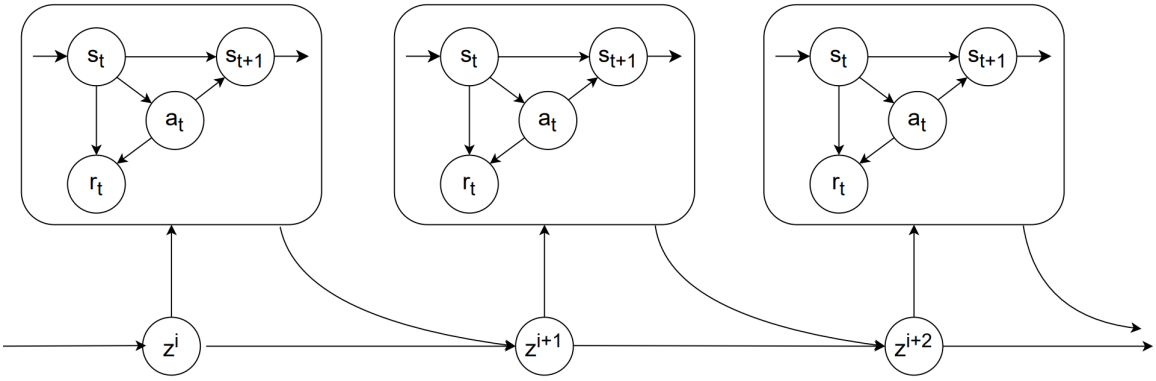


Figure 4.3: Visual representation of the directed graphical probabilistic latent variable model for non-stationarity. Each variable z^i decodes an MDP with states, actions, rewards (s_t, a_t, r_t) for $t \in \{1, \dots, T\}$ and its own state transition and reward distribution. The latent variable z^{i+1} is dependent on z^i and observations $(s^i_{1:T}, a^i_{1:T}, r^i_{1:T})$ from MDP i.

variables provide information on the next hidden market state. Secondly, this allows the implementation of an off-policy policy gradient method which uses a replay buffer (or database) to store all trajectories of states, actions, rewards and inferred latent variables. This is because we only need to store (and sample) observable variables and inferred latent variables of the current and previous MDP, respectively. This avoids the need to sample all preceding variables from a replay buffer which is significantly more complicated to implement.

The joint probability $p_{\boldsymbol{\theta}}(s^i_{1:T}, r^i_{1:T} | z^i, a^i_{1:T})$ from Equation (4.11) can be further factorized into the state transition distribution and the reward distribution

$$p_{\boldsymbol{\theta}}(s^i_{1:T}, r^i_{1:T} | z^i, a^i_{1:T}) \tag{4.12}$$

$$= p_{\boldsymbol{\theta}}(s_1) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(s^i_{t+1} | s^i_t, a^i_t; z^i)\, p_{\boldsymbol{\theta}}(r^i_t | s^i_t, a^i_t; z^i) \quad \text{for } i = 1, \dots, N. \tag{4.13}$$

In this factorization, we condition both the state transition function $p_{\boldsymbol{\theta}}(s^i_{t+1} | s^i_t, a^i_t; z^i)$ and $p_{\boldsymbol{\theta}}(r^i_t | s^i_t, a^i_t; z^i)$ on latent variable z^i, this subdivides the overall non-stationary problem into multiple stationary MDPs where the reward and transition probabilities are stationary conditional on the latent variables $z^{1:N}$.

Following the VAE framework, we aim to approximate the posterior distribution of the latent variables given the observations $p_{\boldsymbol{\theta}}(z^{1:i} | s^{1:i}_{1:T}, a^{1:i}_{1:T}, r^{1:i}_{1:T})$ at episode $i \in \mathbb{N}$. Where we use the notation i to indicate that the model can be used in an online learning setting. As stated by Kingma et al. [2019], we may choose almost any directed graphical model for the variational approximating distribution of the posterior. We select the posterior model, with parameters $\boldsymbol{\phi}$, $q_{\boldsymbol{\phi}}(z^i | z^{i-1}, s^i_{1:T}, r^i_{1:T}, a^i_{1:T})$, such that $Pa(z^i)$ the set of parents of z^i, in-

cludes the previous latent variable z^{i-1} and the current observed trajectory $(s^i_{1:T}, r^i_{1:T}, a^i_{1:T})$. Therefore, the factorization of the posterior approximating distribution is given by

$$q_{\boldsymbol{\phi}}(z^{1:i}|s^{1:i}_{1:T}, r^{1:i}_{1:T}, a^{1:i}_{1:T}) = \prod_{j=1}^{i} q_{\boldsymbol{\phi}}(z^j|z^{j-1}, s^j_{1:T}, r^j_{1:T}, a^j_{1:T}), \tag{4.14}$$

where the initial term of the factorization $q_{\boldsymbol{\phi}}(z^1|z^0, s^1_{1:T}, r^1_{1:T}, a^1_{1:T})$ is set given by $q_{\boldsymbol{\phi}}(z^1|s^1_{1:T}, r^1_{1:T}, a^1_{1:T})$, for ease of notation. As a consequence, we can factorize the expectation

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(z^{1:i}|\cdot)}\left[\cdot\right] = \prod_{j=1}^{i} \mathbb{E}_{q_{\boldsymbol{\phi}}(z^j|\cdot)}\left[\cdot\right], \tag{4.15}$$

where we shorten $q_{\boldsymbol{\phi}}(z^{1:i}|s^{1:i}_{1:T}, r^{1:i}_{1:T}, a^{1:i}_{1:T})$ to $q_{\boldsymbol{\phi}}(z^{1:i}|\cdot)$, and $q_{\boldsymbol{\phi}}(z^j|z^{j-1}, s^j_{1:T}, r^j_{1:T}, a^j_{1:T})$ to $q_{\boldsymbol{\phi}}(z^j|\cdot)$ for ease of notation in the following derivations.

4.3.3 Evidence Lower Bound

Likewise, to the VAE framework, we aim to find a lower bound on the log-likelihood of the observable variables. This evidence lower bound (ELBO), is what we seek to maximize through the parameters θ and ϕ. We aim to find this lower bound on the log-likelihood of the conditional model $\log p_{\boldsymbol{\theta}}(s^{1:i}_{1:T}, r^{1:i}_{1:T}|a^{1:i}_{1:T})$. The ELBO of this log-likelihood is given by

$$\log p_{\boldsymbol{\theta}}(s^{1:i}_{1:T}, r^{1:i}_{1:T}|a^{1:i}_{1:T}) \tag{4.16}$$

$$\geq \sum_{j=1}^{i} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}(z^j|\cdot)} \sum_{t=1}^{T} \log p_{\boldsymbol{\theta}}(s^j_{t+1}, |s^j_t, a^j_t; z^j)\, p_{\boldsymbol{\theta}}(r^j_t|s^j_t, a^j_t; z^j) \right] \tag{4.17}$$

$$- D_{KL}\left(q_{\boldsymbol{\phi}}(z^j|z^{j-1}, s^j_{1:T}, r^j_{1:T}, a^j_{1:T}) || p_{\boldsymbol{\theta}}(z^j|z^{j-1}, s^{j-1}_{1:T}, r^{j-1}_{1:T}, a^{j-1}_{1:T}) \right)$$

To derive the lower bound, we first rewrite the log-likelihood $\log p_{\boldsymbol{\theta}}(s^{1:i}_{1:T}, r^{1:i}_{1:T}|a^{1:i}_{1:T})$ to its expectation with respect to $q_{\boldsymbol{\phi}}(z^{1:i}|s^{1:i}_{1:T}, r^{1:i}_{1:T}, a^{1:i}_{1:T})$ (where $z^{1:i} = \{z^1, \ldots, z^i\}$ denotes the

set of latent variables)

$$\log p_{\boldsymbol{\theta}}(s_{1:T}^{1:i}, r_{1:T}^{1:i} | a_{1:T}^{1:i}) \tag{4.18}$$

$$= \log \int_{z^1} \cdots \int_{z^i} p_{\boldsymbol{\theta}}(s_{1:T}^{1:i}, r_{1:T}^{1:i}, z^{1:i} | a_{1:T}^{1:i})\, dz^1 \ldots dz^i \tag{4.19}$$

$$= \log \int_{z^{1:i}} p_{\boldsymbol{\theta}}(s_{1:T}^{1:i}, r_{1:T}^{1:i}, z^{1:i} | a_{1:T}^{1:i})\, dz^{1:i} \tag{4.20}$$

$$= \log \int_{z^{1:i}} \frac{p_{\boldsymbol{\theta}}(s_{1:T}^{1:i}, r_{1:T}^{1:i}, z^{1:i} | a_{1:T}^{1:i})}{q_{\boldsymbol{\phi}}(z^{1:i} | s_{1:T}^{1:i}, r_{1:T}^{1:i}, a_{1:T}^{1:i})} q_{\boldsymbol{\phi}}(z^{1:i} | s_{1:T}^{1:i}, r_{1:T}^{1:i}, a_{1:T}^{1:i})\, dz^{1:i} \tag{4.21}$$

$$= \log \mathbb{E}_{q_{\boldsymbol{\phi}}(z^{1:i}|\cdot)} \left[\frac{p_{\boldsymbol{\theta}}(s_{1:T}^{1:i}, r_{1:T}^{1:i}, z^{1:i} | a_{1:T}^{1:i})}{q_{\boldsymbol{\phi}}(z^{1:i} | s_{1:T}^{1:i}, r_{1:T}^{1:i}, a_{1:T}^{1:i})} \right] \tag{4.22}$$

$$\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(z^{1:i}|\cdot)} \left[\log \frac{p_{\boldsymbol{\theta}}(s_{1:T}^{1:i}, r_{1:T}^{1:i}, z^{1:i} | a_{1:T}^{1:i})}{q_{\boldsymbol{\phi}}(z^{1:i} | s_{1:T}^{1:i}, r_{1:T}^{1:i}, a_{1:T}^{1:i})} \right] \tag{4.23}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(z^{1:i}|\cdot)} \left[\log \prod_{j=1}^{i} \frac{p_{\boldsymbol{\theta}}(z^j | z^{j-1}, s_{1:T}^{j-1}, r_{1:T}^{j-1}, a_{1:T}^{j-1})\, p_{\boldsymbol{\theta}}(s_{1:T}^{j}, r_{1:T}^{j} | z^j, a_{1:T}^{j})}{q_{\boldsymbol{\phi}}(z^j | z^{j-1}, s_{1:T}^{j}, r_{1:T}^{j}, a_{1:T}^{j})} \right] \tag{4.24}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(z^{1:i}|\cdot)} \left[\sum_{j=1}^{i} \log p_{\boldsymbol{\theta}}(s_{1:T}^{j}, r_{1:T}^{j} | z^j, a_{1:T}^{j}) - \log \frac{q_{\boldsymbol{\phi}}(z^j | z^{j-1}, s_{1:T}^{j}, r_{1:T}^{j}, a_{1:T}^{j})}{p_{\boldsymbol{\theta}}(z^j | z^{j-1}, s_{1:T}^{j-1}, r_{1:T}^{j-1}, a_{1:T}^{j-1})} \right] \tag{4.25}$$

$$\begin{aligned} &= \sum_{j=1}^{i} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}(z^j|\cdot)} \log p_{\boldsymbol{\theta}}(s_{1:T}^{j}, r_{1:T}^{j} | z^j, a_{1:T}^{j}) \right] \\ &\quad - \mathbb{E}_{q_{\boldsymbol{\phi}}(z^j|\cdot)} \left[\log \frac{q_{\boldsymbol{\phi}}(z^j | z^{j-1}, s_{1:T}^{j}, r_{1:T}^{j}, a_{1:T}^{j})}{p_{\boldsymbol{\theta}}(z^j | z^{j-1}, s_{1:T}^{j-1}, r_{1:T}^{j-1}, a_{1:T}^{j-1})} \right] \end{aligned} \tag{4.26}$$

$$\begin{aligned} &= \sum_{j=1}^{i} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}(z^j|\cdot)} \log p_{\boldsymbol{\theta}}(s_{1:T}^{j}, r_{1:T}^{j} | z^j, a_{1:T}^{j}) \right] \\ &\quad - D_{KL} \left(q_{\boldsymbol{\phi}}(z^j | z^{j-1}, s_{1:T}^{j}, r_{1:T}^{j}, a_{1:T}^{j}) || p_{\boldsymbol{\theta}}(z^j | z^{j-1}, s_{1:T}^{j-1}, r_{1:T}^{j-1}, a_{1:T}^{j-1}) \right) \end{aligned} \tag{4.27}$$

$$\begin{aligned} &= \sum_{j=1}^{i} \left[\mathbb{E}_{q_{\boldsymbol{\phi}}(z^j|\cdot)} \sum_{t=1}^{T} \log p_{\boldsymbol{\theta}}(s_{t+1}^{j}, | s_t^j, a_t^j; z^j)\, p_{\boldsymbol{\theta}}(r_t^j | s_t^j, a_t^j; z^j) \right] \\ &\quad - D_{KL} \left(q_{\boldsymbol{\phi}}(z^j | z^{j-1}, s_{1:T}^{j}, r_{1:T}^{j}, a_{1:T}^{j}) || p_{\boldsymbol{\theta}}(z^j | z^{j-1}, s_{1:T}^{j-1}, r_{1:T}^{j-1}, a_{1:T}^{j-1}) \right). \end{aligned} \tag{4.28}$$

The lower bound for Equation (4.23) is by Jensen's inequality due to moving the log inside the expectation. Then, using Equation (4.11) and Equation (4.14), we can factorize both $p_{\boldsymbol{\theta}}(s_{1:T}^{1:i}, r_{1:T}^{1:i}, z^{1:i} | a_{1:T}^{1:i})$ and $q_{\boldsymbol{\phi}}(z^{1:i} | s_{1:T}^{1:i}, r_{1:T}^{1:i}, a_{1:T}^{1:i})$ in Equation (4.24). Furthermore, we use Equation (4.15) to take the expectation inside the summation in Equation (4.26). In Equation 4.26 we use the Kullback-Leibler divergence (KL-divergence), which is given by

$$D_{KL}(q \parallel p) = \int q(x) \log \left(\frac{q(x)}{p(x)} \right) dx, \tag{4.29}$$

and which measures the dissimilarity or "distance" between two distributions p and q. Finally, using Equation (4.13) to the previous Equation, we obtain the state transition distribution and the reward distribution in Equation (4.28).

By optimizing the ELBO in Equation (4.28) with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, we simultaneously maximize the likelihood $p_{\boldsymbol{\theta}}(s_{t+1}^j, |s_t^j, a_t^j; z^j)$ of state transitions, the likelihood $p_{\boldsymbol{\theta}}(r_t^j|s_t^j, a_t^j; z^j)$ of rewards, and we minimize the KL-divergence between the approximating distribution $q_{\boldsymbol{\phi}}(z^j|z^{j-1}, s_{1:T}^j, r_{1:T}^j, a_{1:T}^j)$ and the posterior $p_{\boldsymbol{\theta}}(z^j|z^{j-1}, s_{1:T}^{j-1}, r_{1:T}^{j-1}, a_{1:T}^{j-1})$.

4.3.4 Model Distributions

Since we have chosen to use continuous state, action and reward spaces for portfolio optimization (due to their lack of restriction compared to discrete spaces), we choose Gaussian distributions for our model distributions, as these can handle continuous variables. Furthermore, Kingma et al. [2019] claim that despite using simple prior or conditional model distributions such as Gaussian distributions, the marginal distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ can still be considerably complex in the sense that it may contain arbitrary dependencies. Before we introduce the model distributions, we first introduce the variable h^j used in Cartea et al. [2021], to summarize the sequence of observable variables or the trajectory of MDP j: $(s_{1:T}^j, r_{1:T}^j, a_{1:T}^j)$. It is a summary as it contains fewer elements than the complete sequence $(s_{1:T}^j, r_{1:T}^j, a_{1:T}^j)$. The computation of this summary variable is covered in Subsection 5.2. We now specifically use the parameters $\boldsymbol{\theta}_z, \boldsymbol{\theta}_s, \boldsymbol{\theta}_r$ to denote the following decoding multivariate Gaussian distributions similar to Cartea et al. [2021]

$$z^{j+1}|z^i, s_{1:T}^j, r_{1:T}^j, a_{1:T}^j \overset{\mathbb{P}}{\sim} \mathcal{N}\left(\mu_{\boldsymbol{\theta}_z}(z^j, h^j), \Sigma_{\boldsymbol{\theta}_z}(z^j, h^j)\right), \tag{4.30}$$

$$s_{t+1}|s_t, a_t; z^j \overset{\mathbb{P}}{\sim} \mathcal{N}\left(\mu_{\boldsymbol{\theta}_s}(s_t, a_t; z^j), \Sigma_{\boldsymbol{\theta}_s}(s_t, a_t; z^j)\right), \tag{4.31}$$

$$r_t|s_t, a_t; z^j \overset{\mathbb{P}}{\sim} \mathcal{N}\left(\mu_{\boldsymbol{\theta}_r}(s_t, a_t; z^j), \Sigma_{\boldsymbol{\theta}_r}(s_t, a_t; z^j)\right), \tag{4.32}$$

The parameters $\boldsymbol{\phi}$ parameterize the encoder (inference model) which is given by

$$z^{j+1}|s_{1:T}^{j+1}, r_{1:T}^{j+1}, a_{1:T}^{j+1} \overset{\mathbb{Q}}{\sim} \mathcal{N}\left(\mu^{\phi}(h^{j+1}), \Sigma^{\phi}(h^{j+1})\right). \tag{4.33}$$

In addition, following the VAE framework, we choose to use neural networks to describe the distributions parameterized by $\boldsymbol{\theta}_z, \boldsymbol{\theta}_s, \boldsymbol{\theta}_r$ and $\boldsymbol{\phi}$. These neural networks respectively output a mean vector $\mu_{\boldsymbol{\theta}_z}, \mu_{\boldsymbol{\theta}_s}, \mu_{\boldsymbol{\theta}_r}$, and a covariance matrix $\Sigma_{\boldsymbol{\theta}_z}, \Sigma_{\boldsymbol{\theta}_s}, \Sigma_{\boldsymbol{\theta}_r}$. Since these neural networks model Gaussian distributions, they are called mean-variance-estimation (MVE) networks. A visual representation of a one-dimensional MVE network is given in Figure 4.4.

4.3.5 Policy Distribution

Furthermore, the actions are the output of a policy function (distribution) which can be any arbitrary function (distribution). A general policy function for our problem is given

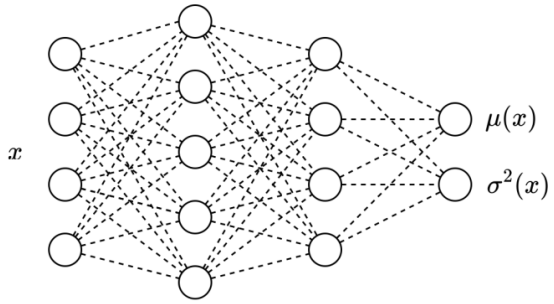


Figure 4.4: Mean-variance-estimation (MVE) network of one-dimensional normal distribution which outputs a mean parameter μ and a variance parameter σ^2 μ [Sluijterman et al., 2023].

by

$$a_t = \pi\left(s_t, z^i\right), \tag{4.34}$$

where a_t is the action taken at timestep t. The policy function π takes as input the observable state s_t and latent variable z^i which ensures sure it takes into account non-stationarity through the hidden state that describes the MDP. This is essentially a policy which can adapt to different financial market regimes characterized by $z^{1:i}$. The policy function can also be Gaussian, which due to the inherent randomness allows for exploration in the reinforcement learning framework. The policy distribution we choose is therefore given by

$$a_t|s_t, z^i \overset{\mathbb{P}}{\sim} \mathcal{N}\left(\mu^{\boldsymbol{\nu}}(s_t, z^i), \Sigma^{\boldsymbol{\nu}}(s_t, z^i)\right). \tag{4.35}$$

A Gaussian policy with a time-decaying variance is shown to be optimal for entropy-regularized mean-variance objective in portfolio optimization Wang and Zhou [2020]. We choose to not add any more complexity and omit the time-decaying variance in the Gaussian distribution.

4.3.6 Objective Functions

The decoding and encoding neural networks are trained by maximizing the Evidence Lower Bound (ELBO) 4.28. Conveniently, we have a closed-from objective function due to the multivariate Gaussian density function. Using the (log of) d-dimensional multivariate Gaussian distribution

$$f(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right), \tag{4.36}$$

the first term of the ELBO 4.28 is given by [Cartea et al., 2021]

$$\sum_{t=1}^{T-1} \log p(s_{t+1}^j, |s_t^j, a_t^j; z^j)\, p(r_t^j | s_t^j, a_t^j; z^j) \tag{4.37}$$

$$= \sum_{t=1}^{T-1} \log p(s_{t+1}^j, |s_t^j, a_t^j; z^j) + \log p(r_t^j | s_t^j, a_t^j; z^j) \tag{4.38}$$

$$\begin{aligned} = -\frac{1}{2} \sum_{t=1}^{T} & \Big[d_s \log(2\pi) + \log\det \Sigma_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^j) \Big] \\ & + \Big(s_{t+1}^j - \mu_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^j) \Big)^\intercal \Big(\Sigma_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^j) \Big)^{-1} \Big(s_{t+1}^j - \mu_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^j) \Big) \\ & + d_r \log(2\pi) + \log\det \Sigma_{\boldsymbol{\theta}_r}(s_t^j, a_t^j; z^j) \\ & + \Big(\Big(r_t^j - \mu_{\boldsymbol{\theta}_r}(s_t^j, a_t^j; z^j) \Big)^\intercal \Big(\Sigma_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^j) \Big)^{-1} \Big(r_t^j - \mu_{\boldsymbol{\theta}_r}(s_t^j, a_t^j; z^j) \Big) \Big], \end{aligned} \tag{4.39}$$

where d_s, d_r are the dimensions of the state and reward distributions, respectively. In the ELBO 4.28, we take the expectation of Equation 4.37 with respect to $q_{\boldsymbol{\phi}}(z^j|h^j) \sim \mathcal{N}(\mu^{\boldsymbol{\phi}}, \Sigma^{\boldsymbol{\phi}})$. Instead of sampling directly from $q_{\boldsymbol{\phi}}(z^j|h^j) \sim \mathcal{N}(\mu^{\boldsymbol{\phi}}, \Sigma^{\boldsymbol{\phi}})$ which introduces noise in the gradients of the network [Kingma et al., 2019], we can use the reparameterization trick introduced by Kingma and Welling [2013] to reduce the variance of the gradients. We use a differentiable reparameterization to allow stochastic gradient descent (SGD) on the objective function. The reparameterization is given by

$$z^{(m)} = \mu + L\epsilon^{(m)}, \tag{4.40}$$

where L is the Cholesky decomposition with non-zero diagonal of $\Sigma_{\boldsymbol{\phi}}$ such that $\Sigma_{\boldsymbol{\phi}} = LL^\intercal$. The reparameterization trick allows us to write

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(z|\cdot)}[f(z)] = \mathbb{E}_{\mathcal{N}(\epsilon;0,I)}[f(\mu + L\epsilon)] \simeq \frac{1}{M} \sum_{m=1}^{M} f(\mu + L\epsilon^{(m)}), \tag{4.41}$$

where $\epsilon^{(m)}) \sim \mathcal{N}(0, I)$. Let $z^{(j,m)}$ denote the m–th sample of z^j from the reparameterization trick. Hence the expectation in Equation (4.39) can be rewritten using Equation (4.41) to

$$\begin{aligned} -\frac{1}{2M} \sum_{j=1}^{i} \sum_{t=1}^{T} \sum_{m=1}^{M} & \Big[d_s \log(2\pi) + \log\det \Sigma_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^{(j,m)}) \\ & + \Big(s_{t+1}^j - \mu_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^{(j,m)}) \Big)^\intercal \Big(\Sigma_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^{(j,m)}) \Big)^{-1} \Big(s_{t+1}^j - \mu_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^{(j,m)}) \Big) \\ & + \log(2\pi) + \log\det \Sigma_{\boldsymbol{\theta}_r}(s_t^j, a_t^j; z^{(j,m)}) \\ & + \Big(r_t^j - \mu_{\boldsymbol{\theta}_r}(s_t^j, a_t^j; z^{(j,m)}) \Big)^\intercal \Big(\Sigma_{\boldsymbol{\theta}_s}(s_t^j, a_t^j; z^{(j,m)}) \Big)^{-1} \Big(r_t^j - \mu_{\boldsymbol{\theta}_r}(s_t^j, a_t^j; z^{(j,m)}) \Big) \Big]. \end{aligned} \tag{4.42}$$

For the second term of the ELBO 4.28, we use the expression for the KL-divergence between two d-dimensional multivariate Gaussian distributions, which is given by

$$D_{KL}(\mathcal{N}_1 \parallel \mathcal{N}_2) = \frac{1}{2}\left(\log\frac{|\Sigma_2|}{|\Sigma_1|} - d + \mathrm{Tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T\Sigma_2^{-1}(\mu_2 - \mu_1)\right), \tag{4.43}$$

where $|\Sigma|$ denotes the determinant of Σ, and $\mathrm{Tr}(\cdot)$ denotes the trace of a matrix. Then applying Equation (4.43 to the latter (KL-divergence) term of the ELBO in Equation (4.28) gives

$$\sum_{j=1}^{i} D_{KL}\left(q_{\boldsymbol{\phi}}(z^j|s^j_{1:T}, r^j_{1:T}, a^j_{1:T}) \parallel p(z^j|z^{j-1}, s^{j-1}_{1:T}, r^{j-1}_{1:T}, a^{j-1}_{1:T})\right) \tag{4.44}$$

$$= \sum_{j=1}^{i} -\frac{1}{2}\left[\log\frac{\det\Sigma_{\boldsymbol{\theta}_z}(z^{j-1}, h^{j-1})}{\det\Sigma_{\boldsymbol{\phi}}(h^j)} + \mathrm{Tr}\left(\Sigma_{\boldsymbol{\theta}_z}(z^{j-1}, h^{j-1})^{-1}\Sigma_{\boldsymbol{\phi}}(h^j)\right)\right. \tag{4.45}$$

$$\left. +(\mu_{\boldsymbol{\theta}_z}(z^{j-1}, h^{j-1}) - \mu_{\boldsymbol{\phi}}(h^j))^T(\Sigma_{\boldsymbol{\theta}_z}(z^{j-1}, h^{j-1})^{-1})(\mu_{\boldsymbol{\theta}_z}(z^{j-1}, h^{j-1}) - \mu_{\boldsymbol{\phi}}(h^j)) - d_z\right].$$

For the policy network $\boldsymbol{\theta}_r$, we can set the objective function to satisfy a mean-variance objective. For this, we use the expected value and variance of the rewards given by the decoding network $\boldsymbol{\theta}_r$. The distribution of the rewards of the portfolio allocation is learned by the network θ_r and expressed through parameters $\mu_{\boldsymbol{\theta}_r}, \Sigma_{\boldsymbol{\theta}_r} \in \mathbb{R}$. Conveniently, we can use these parameters as the objective function for the policy function. For the policy function, the objective we wish to maximize is

$$\frac{\mu_{\boldsymbol{\theta}_r}}{\sqrt{\Sigma_{\boldsymbol{\theta}_r}}}, \tag{4.46}$$

which is known as the ex-ante Sharpe ratio [Sharpe, 1994] (where the risk-free rate is omitted $\mu_{\boldsymbol{\theta}_r}$). This corresponds to the goal of portfolio optimization which is to maximize the expected rewards while minimizing the standard deviation of rewards. Explicitly, the objective function for the policy network $\boldsymbol{\nu}$ can be expressed as

$$\sum_{j=1}^{i}\sum_{t=1}^{T}\frac{\mu_{\boldsymbol{\theta}_r}(s_t, a_t; z^j)}{\sqrt{\Sigma_{\boldsymbol{\theta}_r}(s_t, a_t; z^j)}} = \sum_{j=1}^{i}\sum_{t=1}^{T}\frac{\mu_{\boldsymbol{\theta}_r}(s_t, \pi^{\nu}(s_t, z^j); z^j)}{\sqrt{\Sigma_{\boldsymbol{\theta}_r}(s_t, \pi^{\nu}(s_t, z^j); z^j)}}. \tag{4.47}$$

The objective function can also be adjusted to resemble the global minimum variance portfolio (GMV). In this case the objective is to minimize

$$\Sigma_{\boldsymbol{\theta}_r}, \tag{4.48}$$

which results in minimizing the following objective function

$$\sum_{j=1}^{i}\sum_{t=1}^{T}\Sigma_{\boldsymbol{\theta}_r}(s_t, \pi^{\nu}(s_t, z^j); z^j). \tag{4.49}$$

Using the Sharpe ratio in Equation (4.46) may have drawbacks. For negative mean returns, the Sharpe ratio is not necessarily informative as it can be increased by increasing the variance, which is the opposite of our goal. A solution can be to instead use a clipped Sharpe ratio as the objective function for the policy network, i.e. we seek to maximize

$$\max(0, \frac{\mu_{\boldsymbol{\theta}_r}}{\sqrt{\Sigma_{\boldsymbol{\theta}_r}}}), \tag{4.50}$$

where one may "clip" the positive values too by a constant c, to maximize

$$\min\left(\max(0, \frac{\mu_{\boldsymbol{\theta}_r}}{\sqrt{\Sigma_{\boldsymbol{\theta}_r}}}), c\right). \tag{4.51}$$

Clipping of the rewards or the objective functions is common in deep reinforcement learning such as in DQN [Mnih et al., 2015] and Proximal Policy Optimization [Schulman et al., 2017] to stabilize the learning process. A variance-penalized objective has also been used in reinforcement learning studies which in our case can be expressed as

$$\mu_{\boldsymbol{\theta}_r} - \gamma\Sigma_{\boldsymbol{\theta}_r}, \tag{4.52}$$

where the parameter γ controls the variance penalty. This formulation also avoids the problem that arises when the mean reward estimate $\mu_{\boldsymbol{\theta}_r}$ is negative, the objective still aims to minimize the variance.

4.4 NSRDMM Algorithm

The Non-Stationary Reinforced Deep Markov Model (NSRDMM) is shown in 6. It collects trajectories in a replay buffer, from which, in turn, trajectories are sampled to determine the gradient updates. This is a form of off-policy learning that has better sample efficiency than on-policy gradient methods [Haarnoja et al., 2018a].

Algorithm 6: Non-Stationary RDMM

Input:

Initialize encoder $\boldsymbol{\phi}$, decoder $\boldsymbol{\theta}_z, \boldsymbol{\theta}_s, \boldsymbol{\theta}_r$, and policy $\boldsymbol{\nu}$

Initialize empty replay buffer $\mathcal{D}$

Trajectory length T

Sample z^1 from prior $\sim p_{\boldsymbol{\theta}_z}(z^1)$

for j = *2,...,* N **do**

- Sample $z^j \sim p_{\boldsymbol{\theta}_z}(z^j | z^{j-1}, s^{j-1}_{1:T}, r^{j-1}_{1:T}, a^{j-1}_{1:T})$
- Collect trajectory $(s^j_{1:T}, a^j_{1:T}, r^j_{1:T})$ from $\pi_{\boldsymbol{\nu}}(a^j_t | s^j_t, z^j)$
- Update replay buffer $\mathcal{D}[j] \leftarrow (s^j_{1:T}, a^j_{1:T}, r^j_{1:T}, z^{j-1}, z^j)$
- Sample a batch of k trajectories from $\mathcal{D}$
- Update encoding network $\boldsymbol{\phi}$ (including GRU) using Adam to minimize Equation 4.45
- Update decoding network $\boldsymbol{\theta}_z$ using Adam to minimize Equation 4.45
- Sample $z^{(m)} = \mu + L\epsilon^{(m)}$ with $\epsilon^{(m)}) \sim \mathcal{N}(0, I)$ for $\mathbb{Q}$-expectation
- Update decoding networks $\boldsymbol{\theta}_s, \boldsymbol{\theta}_r$ using Adam to maximize Equation 4.42
- Update policy network $\pi^{\boldsymbol{\nu}}$ using Adam to maximize chosen policy objective

Chapter 5

Numerical Experiment

In this section, we will compare the performance of our model-based NSRDMM algorithm to other model-free reinforcement learning methods such as SAC and DDPG, and traditional analytical portfolio allocation methods such as Mean-Variance and Equal Risk Contribution.

The setup of our experiment involves a monthly rebalancing of a portfolio, which happens to be a common rebalancing frequency in the financial industry. In each case, we will consider a fixed portfolio of n pre-selected assets, which will be rebalanced monthly over the full test period. Therefore, we omit the selection process of assets in the portfolio, which in practice can be generated by any discretionary or systematic selection criteria, and merely focus on the optimization of the given portfolio — i.e. we select the optimal weight allocation of each of the n assets for each month. Furthermore, we consider portfolios without leverage, i.e. we consider a weight vector $w_t^\intercal = (w_1, \dots, w_n) \in \mathbb{R}^n$, where $0 < w_i < 1 \ \forall i \in \{1, \dots, n\}$, such that

$$w_t^\intercal \mathbf{1} = 1, \tag{5.1}$$

where $\mathbf{1} = (1, \dots, 1)^\intercal \in \mathbb{R}^n$. We note that the policy in NSRDMM can also be adjusted to handle portfolio allocations with leverage (which will be explained in Section 5.1.2).

5.1 Data

The dataset consists of the return data of all assets which have been in the SP500 for over the last 20 years (this happens to be the case for a bit fewer than 400 assets). The return data of each stock is freely available on Yahoo Finance. The companies currently in the SP500 can be sourced from Wikipedia. Specifically, we use return data from February 2004 to January 2024. The data is split into a training set, a validation set and a test set. The training set is from February 2004 to February 2014, the validation set is from March 2014 to March 2015, and the test set is from April 2016 to January 2024.

For our NSRDMM, and reinforcement learning more generally, we need to select the data we use for our states and rewards. For readability, we will simply denote, in the following sections, at time $t \in \{1, \ldots, T\}$, the states s_t^i, the actions a_t^i and the rewards r_t^i for any MDP $i \in \{1, \ldots, N\}$, as s_t, a_t and r_t.

5.1.1 States

The states are the features that will be used as input for the policy function. Therefore, it is important to select state variables that are meaningful to an allocation policy. In MV portfolio optimization, those are usually the mean return and covariance matrix between the assets. Since it is difficult to estimate the mean return (a phenomenon referred to as the mean-blur problem by Luenberger [1998]), using only the covariance matrix (which is the case for GMV and ERC) is considered to be more robust. Consequently, we believe it is reasonable to use the covariance as feature entries, to compare NSRDMM to MV, GMV and ERC. Following standard practice, we use daily log-returns to calculate the covariance matrix, which are defined as

$$x_t = \log\left(\frac{P_t}{P_{t-1}}\right), \tag{5.2}$$

where P_t and P_{t-1} represent the close price on day t and day $t-1$, respectively. Furthermore, to calculate the volatility of returns or covariance between returns, we assume an average return of zero which is common practice by traders according to Taleb [1997], Sinclair [2013] due to the noisiness in estimating the sample mean (this is reminiscent of the mean-blur problem described in Luenberger [1998]). Hence, the volatility of a stock with log-return x_t is estimated by

$$\sigma = \sqrt{\frac{1}{n}\sum_{t=1}^{n} x_t^2}. \tag{5.3}$$

Similarly, the covariance matrix between returns x and y is given by

$$\mathrm{Cov}(x, y) = \frac{1}{n}\sum_{t=1}^{n} x_t y_t. \tag{5.4}$$

Another consideration is the lookback period for estimating the covariance matrix. This is a subjective consideration, which involves making a trade-off between long memory or assigning more importance to more recent patterns. To allow for responsiveness to recent changes in the covariance, we choose to use an exponential weighted moving average (EWMA). This method provides an exponential decay to historical return data and therefore weights the returns in proportion to their recency [Taleb, 1997]. The exponentially weighted moving average of the variance is given by

$$\sigma_t^2 = \lambda\sigma_{t-1}^2 + (1-\lambda)x_t^2, \tag{5.5}$$

where σ_t^2 is the variance at time t, x_t^2 is the squared log-return at time t, and λ is a decay factor ($0 < \lambda < 1$). Substituting the same expression for σ_{t-1}^2 into Equation (5.5) gives

$$\sigma_t^2 = \lambda(\lambda\sigma_{t-2}^2 + (1-\lambda)x_{t-1}^2) + (1-\lambda)x_t^2. \tag{5.6}$$

Iterating this process recursively gives us the EWMA variance for log-returns x_t

$$\sigma_t^2 = \lambda^t\sigma_0^2 + (1-\lambda)\sum_{i=0}^{t-1}\lambda^i x_{t-i}^2. \tag{5.7}$$

If t is sufficiently large we may choose to leave out $\lambda^t\sigma_0$ from Equation (5.7). Then, similarly, the EWMA covariance between returns x and y can be written as

$$\mathrm{Cov}(x,y) = (1-\lambda)\sum_{i=0}^{t-1}\lambda^i x_{t-i}y_{t-i}. \tag{5.8}$$

The decay factor λ allows us to set our lookback period, as the span (or lookback period) is given by

$$span = \frac{2}{(1-\lambda)}, \tag{5.9}$$

where we choose $\lambda = 0.99$, to get a span of 200 trading days out of roughly 252 trading days per year, which we deem appropriate for a monthly rebalancing frequency.

Since NSRDMM contains (deep) neural networks, the model provides the flexibility to add *any* other feature deemed relevant to improve the portfolio allocation policy, unlike MV and ERC which can only use the covariance matrix (and expected returns). For simplicity, we will consider only one type of feature that can easily be computed from the close price data which is already in our dataset, namely, we can use momentum features. Momentum features are based on the premise that assets moving upwards (downwards) are likely to continue moving upwards (downwards). Standard features for momentum are crossovers between exponential moving averages of two different spans. We choose to compute the exponential moving crossover with a span of one month (λ_1) compared to two months (λ_2) which is given by

$$Crossover_t = EWMA_{\lambda_1,t} - EWMA_{\lambda_2,t}. \tag{5.10}$$

This crossover gives an indication of whether the short-term price trend is above or below its longer-term average, therefore indicating an asset's momentum. Additionally, we choose to add another crossover feature with a span of 3 months and 6 months, respectively.

Therefore, the dimensionality of the state space for a portfolio of n assets is $n \times n$ (covariance matrix) plus $2n$ for the two momentum features per asset, resulting in a flattened vector of dimension $n(n+2)$.

5.1.2 Actions

The actions a_t are sampled from the policy $\pi^{\boldsymbol{\nu}}$. In this numerical experiment, the actions are similar to the weight vector $w_t \in \mathbb{R}^n$ defined in Equation (5.1). That is, $a_t^\intercal = (a_1, \ldots, a_n) \in \mathbb{R}^n$, where $0 < a_i < 1 \; \forall i \in \{1, \ldots, n\}$, such that

$$a_t^\intercal \mathbf{1} = 1, \tag{5.11}$$

where $\mathbf{1} = (1, \ldots, 1)^\intercal \in \mathbb{R}^n$. This constraint is satisfied by a softmax output function on the policy network $\pi^{\boldsymbol{\nu}}$. Given an input vector $\mathbf{z} = (z_1, \ldots, z_k) \in \mathbb{R}^k$, the softmax function: $\mathbb{R}^k \mapsto (0, 1)^k$ is given by

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \quad \text{for } i = 1, \ldots, n, \tag{5.12}$$

which due to the normalization in the denominator ensures the weights sum up to one. We note that other output functions can be used too to include the use of leverage. For example, the l_1 norm $\| \cdot \|_1$, can be used to normalize the weights on both the long and short sides of a market-neutral portfolio (which has an equal dollar amount in longs and shorts) to obtain a gross leverage of 100%. The weights can subsequently be scaled by a constant to set the gross leverage.

5.1.3 Rewards

The rewards r_t used for training the reward network $\boldsymbol{\theta}_r$ are chosen to be one-month forward returns from the portfolio rebalancing date. Let $R_{i,t}$ denote the simple one-month-forward return of asset $i \in \{1, \ldots, n\}$ at time t, i.e.

$$R_{i,t} = \frac{P_{i,t+1}}{P_{i,t}}, \tag{5.13}$$

where $P_{i,t+1}$ is asset i's price in one month time from t and $P_{i,t}$ is its price at time t. Then, if we define the return vector $R_t = (R_{1,t}, \ldots, R_{n,t})^\intercal \in \mathbb{R}^n$, we calculate the portfolio rewards r_t corresponding to action a_t as

$$r_t = a_t^\intercal R_t. \tag{5.14}$$

5.2 Network Architectures

In this section we will describe the architectures of all artificial neural networks used in the NSRDMM section for this numerical experiment.

5.2.1 Gated Recurrent Unit (GRU)

The summary variable h^j, also called a hidden state, of the trajectory is provided through a Gated Recurrent Unit (GRU) [Cho et al., 2014], which is a recurrent neural network (RNN). It is considered to be a simplified version of an LSTM network (it has fewer parameters), but its performance is comparable [Chung et al., 2014]. Due to the recurrent nature, it can be is useful for processing time-dependent sequences of data. Its architecture gives a GRU the useful property of being able to "memorize" previous states.

A GRU consists of a reset gate, an update gate, a proposed hidden state, and an actual hidden state. Let $t \in \{0, \dots, T\}$ and let $h^j = h_T$. The reset gate ρ_t is computed by [Cho et al., 2014]

$$\rho_t = \sigma\left(W_r x_t + V_r h_{t-1}\right), \tag{5.15}$$

where σ is the sigmoid function, x_t is the current input, h_{t-1} is the previous hidden state and, W_ρ, V_ρ are the weight matrices. Similarly, the update gate u_t is computed by [Cho et al., 2014]

$$u_t = \sigma\left(W_z x + V_z h_{t-1}\right). \tag{5.16}$$

The actual hidden state h is then computed by [Cho et al., 2014]

$$h_t = u_t h_{t-1} + (1 - u_t)\tilde{h}_t, \tag{5.17}$$

which involves a weighting of the previous hidden state and the proposed hidden state given by [Cho et al., 2014]

$$\tilde{h}_t = \phi\left(W x_t + V\left(\rho_t \odot h_{t-1}\right)\right), \tag{5.18}$$

where $\odot$ is the Hadamard product, and where ϕ is an activation function commonly chosen to be $\tanh(\cdot)$. The reset gate allows the network to disregard the previous hidden state by setting it close to 0 and instead focus solely on the current input. This mechanism enables the network to eliminate any outdated information [Cho et al., 2014]. The update gate determines the degree how much the previous hidden state influences the current state. Cho et al. [2014] assert that having different reset and update gates for each state, the GRU is able to learn dependencies across different time scales. Reset gates will be active for short-term dependencies, where update gates will be active for longer-term dependencies. A visual overview of a GRU is shown in Figure 5.1. To learn more complex dynamics, the GRU can be extended to have multiple layers. In a multilayer GRU, with $l > 1$ layers, the hidden state $h_t^{(l-1)}$ serves as the input $x_t^{(l)}$ for the l-th layer. In addition, we have reset, update and proposed gates $r_t^{(l)}$, $z_t^{(l)}$, $\tilde{h}_t^{(l)}$ respectively, for the l-th layer. In this numerical experiment, we choose to use a depth of 2 to 4 layers in the GRU, which is selected in the trade-off to allow for learning complex dynamics while limiting the model's complexity and the potential to overfit.

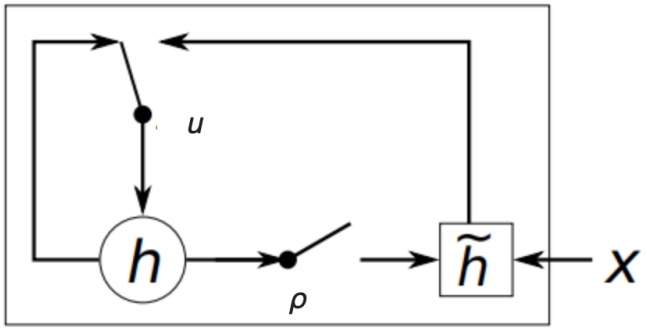


Figure 5.1: Visual representation of a Gated Recurrent Unit (GRU). The update gate u determines how much the new hidden state is updated with the proposed hidden state $\tilde{h}$. The reset gate ρ determines the extent to which the previous hidden state is forgotten for the proposed hidden state [Cho et al., 2014].

Furthermore, to mitigate the potential for overfitting we use a dropout rate. In a dropout-regularized GRU, the input to each neuron is calculated by

$$\left[h_t^{(l-1)} \cdot \delta_t^{(l-1)}\right]_j, \tag{5.19}$$

where $\delta_t^{(l-1)} \sim \text{Bernoulli}(p_{dropout})$ and $[\cdot]_j$ is the $j-th$ element of the vector — i.e. each neuron has a probability of being dropped out of $p_{dropout}$. We set $p_{dropout}$ to a default value of 0.2.

In addition, we make use of layer normalization, which is computed by normalizing the sum of all inputs into a layer's neurons by its mean and variance. This mitigates the problem of covariate shift, which is when changes in the outputs of one layer are highly correlated to the sum of all inputs into the following layer. Layer normalization has been shown to reduce training times and, in recurrent networks, to stabilize the dynamics of hidden states [Ba et al., 2016]. Let x be the vector of inputs into a layer's neurons, then the layer normalization is given by [PyTorch, 2024]

$$\text{LN}(x) = \gamma \odot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{5.20}$$

where μ, σ are the mean and variance of the sum of elements in the vector x, respectively; ϵ is a small constant, and γ, β are the scale and bias vectors, respectively. The elements of the scale and bias vector allow each neuron the flexibility to learn its own scale and bias parameter after normalization.

5.2.2 Mean-Variance-Estimation Networks

The decoding networks $\boldsymbol{\theta}_z, \boldsymbol{\theta}_s, \boldsymbol{\theta}_r$ and encoding network $\boldsymbol{\phi}$ are a specific type of artificial neural networks, called mean-variance-estimation (MVE) networks. Each of these MVE networks outputs a mean vector $\mu \in \mathbb{R}^n$ and a covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$ of a multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$. As this would require $n + n^2$ outputs, the model complexity increases quickly as the dimensionality of the Gaussian distribution increases. It is somewhat more convenient for the network to output the Cholesky decomposition L of the covariance matrix Σ as it requires $n(n+1)/2$ instead of n^2 outputs. This is because the Cholesky decomposition L is a lower-triangular matrix L which has $n(n+1)/2$ non-zero elements. From the Cholesky decomposition we can computer the covariance matrix

$$\Sigma = LL^\intercal. \tag{5.21}$$

We can compute the Cholesky decomposition because Σ is positive semi-definite by definition. However, the covariance matrix resulting from $LL^\intercal$ need not be unique [Pinheiro and Bates, 1996]. We, therefore, use the log-Cholesky parametrization because this ensures the diagonal of L is positive which implies that the covariance matrix is unique. If we suppose that the neural network outputs the log-Cholesky matrix, this means it outputs the diagonal elements $(\log l_{11}, \ldots, \log l_{nn})$ with all other elements of the matrix L, i.e. l_{ij} $\forall i, j \in \{1, \ldots, n\}$ where $i \neq j$. Consequently, after exponentiating the diagonal elements we obtain the Cholesky decomposition L with a positive diagonal. Instead of using a log-Cholesky decomposition and exponentiating the diagonal element, it is also common to apply the softplus function to the diagonal elements [Sluijterman et al., 2023]. The softplus function is given by

$$\text{softplus}(x) = \log(1 + e^x), \tag{5.22}$$

and also ensures positivity of its outputs. In our experiment, we choose the Cholesky decomposition with the softplus function to transform the diagonal elements.

The Cholesky parameterization, however, does not fully prevent the issue of the network quickly increasing in complexity as the dimension of the Gaussian distribution increases. To mitigate this issue for high dimensional Gaussian distributions, one may choose make the simplifying assumption that the covariance matrix Σ is nearly diagonal. Then, the neural network only needs n outputs to parameterize the covariance matrix (besides the n outputs for the mean vector μ). We also choose to use this assumption when we add the momentum features (for each of the n assets) to our state space as it increases the dimensionality.

All neural networks use three hidden layers. This depth ensures the network is as 'deep' neural network, with the ability to learn complex patterns while limiting the architecture's complexity and number of parameters. The number of nodes in each layer are between 30 and 120, depending on the dimensions of the input and output of the specific network.

Furthermore, we choose the Rectified Linear Unit (ReLU) as our activation function and the Kaiming normal weight initialization designed to work well with the ReLu activation function [He et al., 2015]. Furthermore, we choose the widely used Adam to be the optimizer for our neural networks. Adam is known for its effectiveness through the use of "momentum" and its update is given by [Kingma and Ba, 2014]

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t, \tag{5.23}$$

where $\boldsymbol{\theta}$ are the parameters of the neural network, η is the learning rate, $\hat{m}_t$ is the first moment estimate (a decayed average of past gradients), and $\hat{v}_t$ is the second moment estimate (a decayed average of past squared gradients). The term ϵ is a small scalar added to the denominator to ensure numerical stability.

To ensure stability, and more specifically, to avoid gradients from exploding (which happened on some occasions during training of the networks) we implement gradient clipping on gradient g, with the clipping value c_{clip} to 1. This works as follows, if $\|g\|_2 \geq c$, then we set

$$g := c\frac{g}{\|g\|_2}. \tag{5.24}$$

Similar to the GRU, we also use layer normalization and a dropout rate of 0.2 to mitigate the potential to overfit.

We also include a warm-up period for training the MVE networks as proposed by Nix and Weigend [1994]. [Sluijterman et al., 2023] show that learning the mean paramter μ, while keeping the variance (covariance) fixed during a warm-up period improves learning. This is because the network may have trouble learning the mean function for regions where it has large errors initially [Sluijterman et al., 2023]. In this scenario, the loss function can be improved by increasing the variance while not necessarily improving the mean.

5.2.3 Training

Policy gradient methods (and reinforcement learning more generally) are known to suffer from sample inefficiency. Even though our proposed NSRDMM is model-based which can improve sample efficiency. As a consequence one needs either a lot of historical data or one needs to simulate data. Unfortunately, the portfolio optimization problem is a relatively low-frequency problem in finance as opposed to market making. This means that per feature, there is usually much less data (fewer samples) available. Wang and Zhou [2020] offer a clever solution to this problem. Wang and Zhou [2020] randomly choose n stocks from the SP500 which is repeated 100 times to create 100 portfolios/datasets. This gives us $\binom{n}{k}$ different portfolios where k is the number of stocks in the SP500.

In our training, we choose to allocate weights to a portfolio of $n = 10$ assets on a monthly basis. For each trajectory $(s^j_{1:T}, a^j_{1:T}, r^j_{1:T})$, where $j \in \{1, \ldots, i\}$, we set $T = 12$. This means that we have monthly states, actions, and rewards (s_t, a_t, r_t) at month $t = 1, \ldots, 12$. Since

our training data involves a period of 10 years from 2004 to 2014 we have, for each portfolio of $n = 10$ assets, 10 years of monthly training data.

We note that NSRDMM has the flexibility to simulate data too by sampling z, s, r from the decoding networks $\boldsymbol{\theta}_z, \boldsymbol{\theta}_s, \boldsymbol{\theta}_r$. However, we choose to use historical data to remain as realistic as possible.

5.2.4 Early Stopping

Besides dropout, we also use early stopping to prevent the model from overtraining and overfitting. Early stopping involves stopping the training of the neural networks when its loss on the validation set starts to increase. This indicates the model is overfitting on the training data as it loses its generalizability by performing worse on unseen data. The networks were run for a maximum of 10,000 gradient steps; the networks $\boldsymbol{\theta}_r$ and $\boldsymbol{\phi}$ were stopped from training after 3000 gradient steps as the loss on the validation set plateaued and started to increase.

5.2.5 Hyperparameters

The validation set was also used to tune parameters such as the learning rate, and the warm-up period. While there are sophisticated methods for hyperparameter tuning including Bayesian optimization we refrained from using this due to the computational resources necessary to train the algorithm for multiple runs on the training and validation set using different parameters. The code was implemented in PyTorch and required CUDA GPUs through Google Colab Pro to make training and testing the algorithm feasible (see code in Appendix B). Therefore, we resorted to a rough grid search using mainly default hyperparameter initializations which were subsequently tweaked in the direction which was deemed necessary.

The decoding networks, $\boldsymbol{\theta}_z, \boldsymbol{\theta}_s, \boldsymbol{\theta}_r$, the encoding network $\boldsymbol{\phi}$ and the policy network $\pi^{\boldsymbol{\nu}}$ used a learning rate of 10^{-4} (unless stated otherwise in specific cases). This learning rate allowed for stable learning curves on the training and validation set. Moreover, the number of gradient steps for the warm-up period for the networks $\boldsymbol{\theta}_z, \boldsymbol{\theta}_s, \boldsymbol{\theta}_r$ and $\boldsymbol{\phi}$ was set to 1500, as all these networks' losses decreased and stabilized in under 1500 steps (some networks somewhat sooner than the others albeit not by a large margin) on the validation set. Since, the parameters $\mu_{\boldsymbol{\theta}_r}, \Sigma_{\boldsymbol{\theta}_r}$ were used in the objective function for policy network $\pi^{\boldsymbol{\nu}}$, the policy network could only be trained after the warm-up period of the network $\boldsymbol{\theta}_r$. This is because only after the warm-up period was the parameter $\Sigma_{\boldsymbol{\theta}_r}$ learned. Our policy network started learning after the warm-up period of 1500 gradient steps and had its own warm-up period of 500 gradient steps (which corresponds to the 2000^{th} gradient step of training).

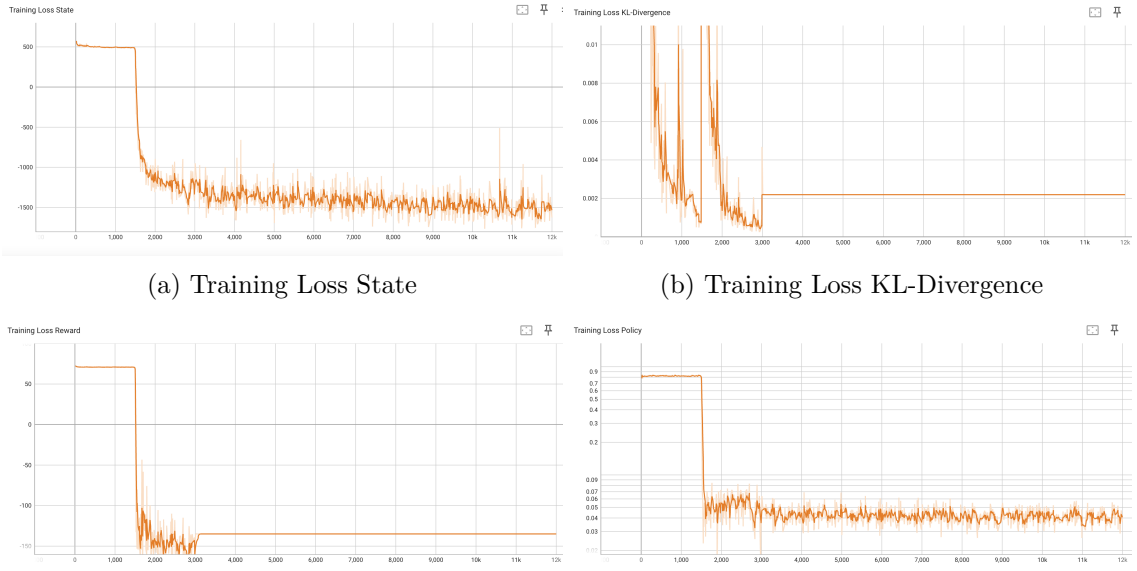
Other parameters in the model include the number of batches sampled from the replay buffer $\mathcal{D}$, which is the database that stores all sampled trajectories shown in Algorithm 6.

The number of batches sampled is the number of samples used to compute the gradient updates, which we set equal to 20. We also include a sampling period, where we only fill the replay buffer with 200 trajectories. This sampling period is used to avoid using the same trajectories for gradient computations over and over during the first few gradient updates. Moreover, we set the q-sample size M, as provided in Equation (4.42), to 20.

5.3 Results

In this section, we will share the loss plots on the training set of each network, and the portfolio performance per algorithm on the 20 distinct test sets (portfolios). Performance is mainly measured in terms of (log) cumulative returns, the Sharpe ratio; we also provide the maximum drawdown. The SAC and DDPG algorithms were trained for 10,000 gradient steps, learning rates of 0.001 and a batch size of 20.

We assume furthermore that the risk-free rate is 0 and that there is an absence of transaction costs. We also assume that stocks can be bought at the close price without paying the bid/ask spread. Furthermore, we assume that the exact weight allocation can be bought and does not need to be rounded off to a specific number of shares.



(a) Training Loss State

(b) Training Loss KL-Divergence

(c) Training Loss Reward

(d) Training Loss Policy

Figure 5.2: Training losses (y-axis) for number of gradient steps (x-axis) for the networks: (a) state decoder $\boldsymbol{\theta}_s$, (b) $\boldsymbol{\theta}_z$, $\boldsymbol{\phi}$ and GRU with shared KL-divergence loss, (c) reward decoder $\boldsymbol{\theta}_r$, (d) policy network $\pi^{\boldsymbol{\nu}}$. In (b) and (c) due to its validation loss, early stopping was used at 3000 iterations.

In Figure 5.2 the losses of all networks on the training set are shown. The loss functions look reasonably stable, with the obvious exception of the first iteration after the warm-up period where the variance of the networks is not held to an arbitrary fixed value. The learning of networks $\boldsymbol{\theta}_z$, $\boldsymbol{\phi}$ and GRU with a shared KL-divergence loss, and the reward decoder $\boldsymbol{\theta}_r$, were stopped at 3000 iterations due to early stopping (the validation loss plateaued or became higher). For the policy function, Objective (4.52) was used as it had

the most stable loss function in training and validation unlike the clipped Sharpe Objective (4.51) which was much noisier. The penalty parameter γ in Objective (4.52 was set to 1.

The Figures (5.3, 5.4a, 5.4b, 5.5a, 5.5b) summarize the log-cumulative returns of 20 portfolios over a test period from 2016-05-31 to 2023-12-29. The lines represent the median log-cumulative return — i.e. it is the cumulative sum of the median log-return at each timestep. Similarly, the shaded area represents the minimum and maximum log-cumulative return, also by taking the cumulative sum of the minimum and maximum log-return, respectively. This is used to show the distribution of the portfolio optimization method in terms of cumulative returns over 20 portfolios in one visualization instead of 20. It also provides insight into the median performance and the variability of performance. The red-shaded area belongs to the same portfolio optimization method as the red line and the blue-shaded area is from the same portfolio optimization method as the blue line.

In Figure 5.3, we compare two trained NSRDMM's. The NSRDMM only takes the covariance matrix as input, whereas NRSDMM-Momentum also uses two momentum features as state variables per asset.

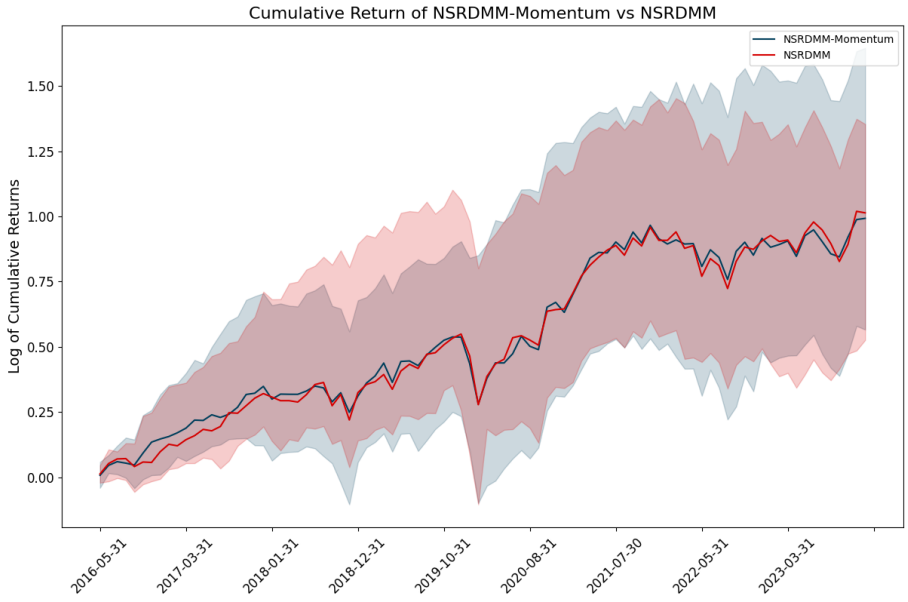


Figure 5.3: Log of cumulative return comparison between NSRDMM-Momentum and NSRDMM.

From Figure 5.3, it can be seen that the median cumulative portfolio returns have similar performance at the end of the test period. It can also be seen that the best-performing and worst-performing portfolios of NSRDMM-Momentum outperform those of

NSRDMM (presented in the shaded area).

Compared to MV and ERC, in Figures (5.4a, 5.4b), although the NSRDMM-Momentum achieves higher cumulative returns in its best portfolio, the worst performing portfolio has lower cumulative returns than both MV and ERC. Consequently, MV and ERC exhibit a lower dispersion of cumulative returns across the portfolios.

In Figures (5.5b, 5.5a), we see that NSRDMM-Momentum outperforms the model-free reinforcement learning algorithms SAC and DDPG both in terms of both cumulative return and dispersion of cumulative return at the end of the test period.

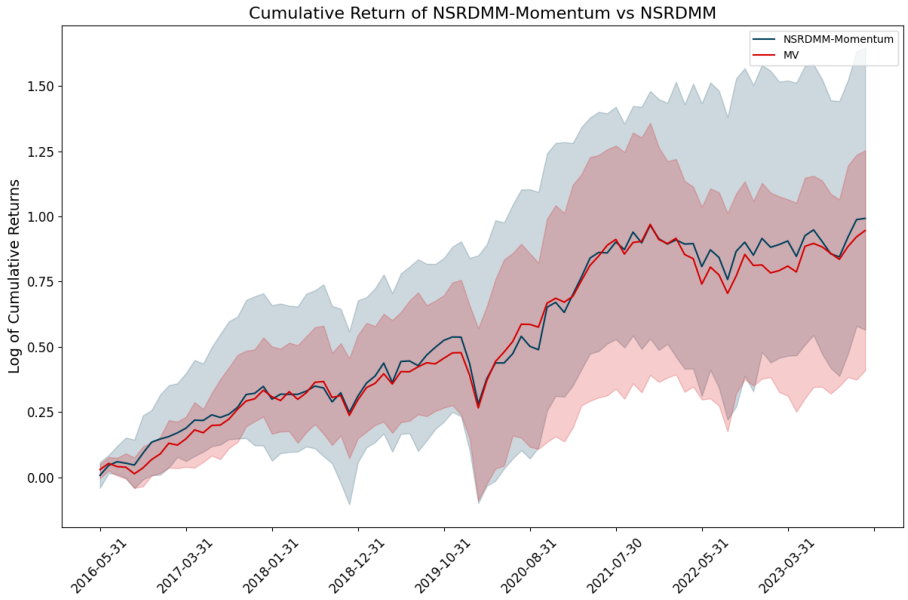
In Tables 5.1, 5.2, and 5.3 the full log-cumulative return data, Sharpe ratios, and maximum drawdown are given, respectively. The full results in the tables also show that the NSRDMM-Momentum performs best in terms of cumulative returns. In addition, the Sharpe ratios and maximum drawdowns also corroborate the observation that the MV and ERC have lower dispersion of returns than NSRDMM-Momentum and hence also lower than SAC and DDPG.

We believe the results can be explained by the distribution of portfolio weights shown in Figure 5.6 where the weights of MV and ERC vary little over the portfolios, followed by NSRDMM-Momentum and NSRDMM, whereas SAC and DDPG have the largest dispersion of weights over the portfolios. While this does not benefit SAC and DDPG, we may argue that in NSRDMM-Momentum, the increased weight allocation to certain assets influences the performance seen in terms of cumulative returns, albeit at the cost of a lower Sharpe ratio than GMV and ERC. These increased allocations, in turn, could be caused by NSRDMM "recognizing" that an asset may outperform and hence it could be interpreted that NSRDMM also acts as a trading algorithm besides merely a portfolio optimization method.

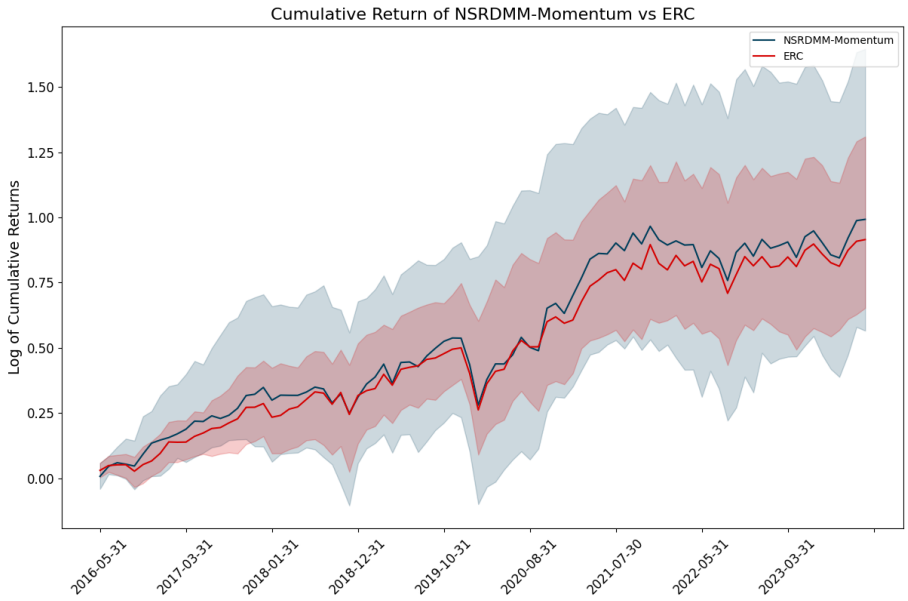
We emphasize that NSRDMM outperforms model-free methods SAC and DDPG in both cumulative returns and Sharpe ratio. Moreover, the model-free methods lead to very high and possibly unstable portfolio allocation weights. We believe this shows that NSRDMM has superior sample efficiency for a similar amount of policy gradient steps (10,000). From the loss on the validation set, we believe NSRDMM could have had good performance for fewer than 5000 policy gradient steps (or 7000 total gradient steps when taking into account the warm-up period of the policy network).

Table 5.1: Log-cumulative returns of the different portfolio optimization methods.

Test	**NSRDMM-Mo**	**NSRDMM**	**MV**	**ERC**	**SAC**	**DDPG**
1	0.7473	0.5258	0.4098	0.6526	0.1473	0.3814
2	0.9423	0.8876	1.0263	1.0564	1.0289	0.3864
3	1.0822	0.8778	0.9233	0.8934	0.1933	1.1153
4	0.7518	0.6950	0.88	0.7633	0.8843	0.6925
5	1.6455	1.3195	1.1737	1.3098	1.0527	1.3303
6	0.9534	1.2909	0.8132	0.7379	0.6178	0.8104
7	1.2827	0.7529	0.8724	0.9240	0.9570	0.4201
8	1.2695	1.1054	1.1281	1.1012	0.6606	1.2013
9	1.0266	1.1401	1.2536	1.0760	1.1487	1.2440
10	0.5734	0.7133	1.1727	0.8017	1.1151	0.6500
11	0.9028	1.1913	1.0474	1.1064	1.0589	0.9027
12	0.9853	0.9713	0.8342	0.8292	1.2191	0.5320
13	1.1955	1.1545	1.1904	0.9899	0.7948	1.3968
14	0.8751	1.0560	0.8272	0.8891	0.7459	1.2761
15	0.8480	0.8807	0.9687	0.9444	1.3187	1.0836
16	0.9999	1.1858	0.7444	0.8690	0.9784	0.7969
17	1.0585	0.7282	1.0203	0.8051	1.0730	0.9770
18	0.5659	1.3290	1.0305	0.9642	1.2160	0.6901
19	1.1082	0.8410	0.8677	0.9059	0.7663	0.9983
20	1.1987	1.3540	0.8854	0.9609	1.2708	0.7206
Mean	1.0006	1.000	0.9534	0.9290	0.9123	0.8802

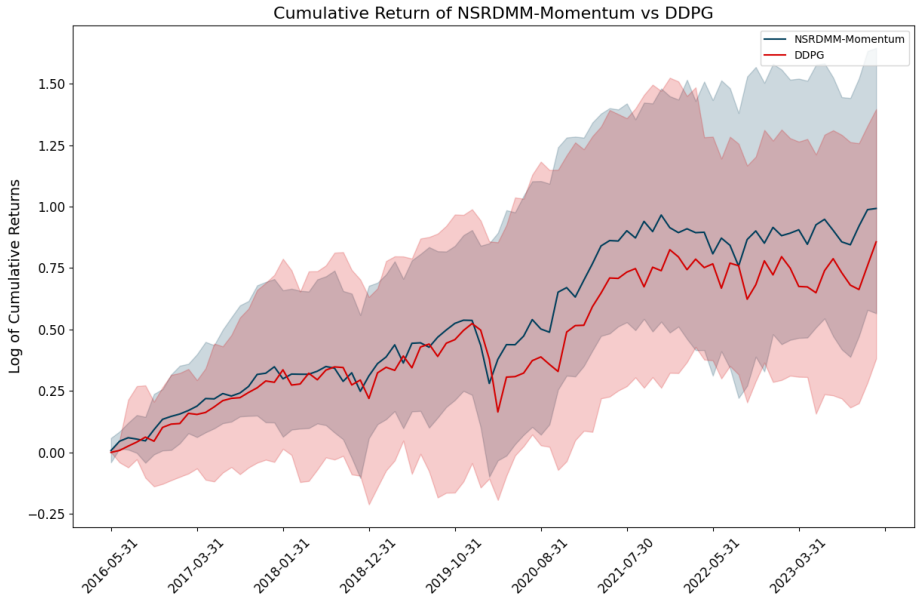


(a) NSRDMM-Momentum vs Mean-Variance

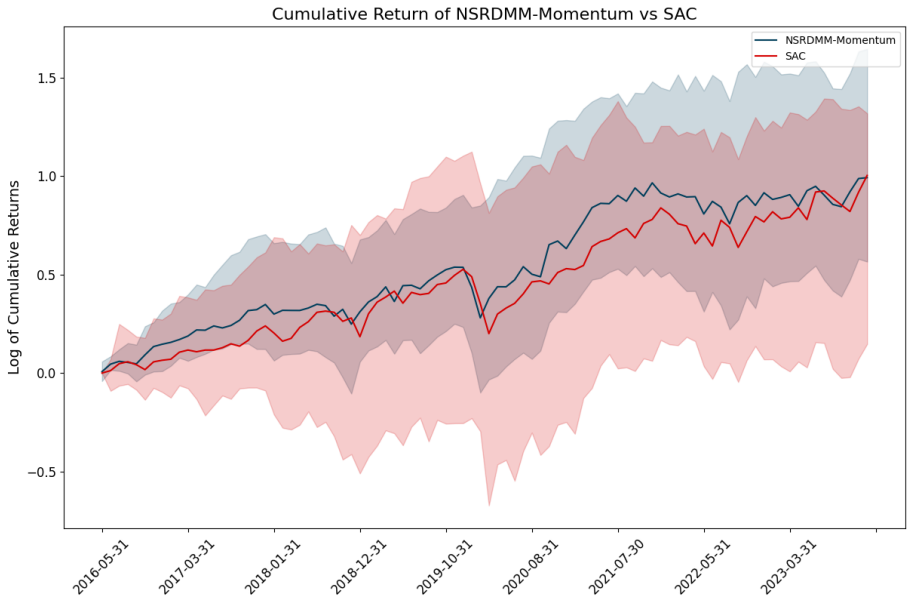


(b) NSRDMM-Momentum vs Equal Risk Contribution

Figure 5.4: Log of cumulative return comparisons (Part 1)



(a) NSRDMM-Momentum vs DDPG



(b) NSRDMM-Momentum vs SAC

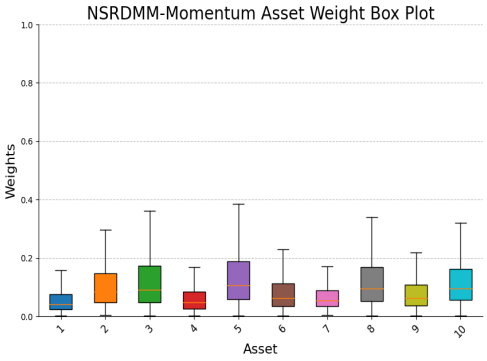
Figure 5.5: Log of cumulative return comparisons (Part 2)

Table 5.2: Sharpe ratios of the different portfolio optimization methods.

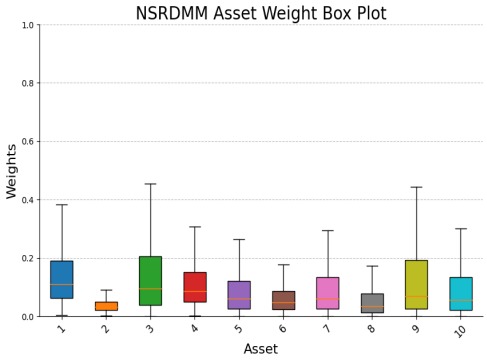
Test	**NSRDMM-Mo**	**NSRDMM**	**MV**	**ERC**	**SAC**	**DDPG**
1	0.5666	0.3998	0.381	0.5872	0.2080	0.3445
2	0.7215	0.7067	0.8565	0.8994	0.7395	0.3668
3	0.6512	0.4777	0.6559	0.5799	0.2339	0.6840
4	0.6512	0.5355	0.7976	0.7034	0.6858	0.5735
5	1.2233	1.0017	0.9495	1.1111	0.6824	1.0452
6	0.6589	0.8719	0.5364	0.5699	0.4538	0.5950
7	0.8564	0.5481	0.6815	0.7675	0.6372	0.3543
8	0.9566	0.8808	0.9705	0.9133	0.5225	0.8807
9	0.7101	0.7843	0.939	0.8088	0.7985	0.9031
10	0.3746	0.4103	0.8501	0.6240	0.6903	0.4800
11	0.6710	0.8478	0.9046	1.0347	0.5822	0.6740
12	0.6235	0.5783	0.6347	0.6232	0.7384	0.4208
13	0.7686	0.7869	0.8362	0.7941	0.5475	0.9946
14	0.6402	0.6539	0.6303	0.7665	0.6564	0.8698
15	0.6353	0.6131	0.7999	0.8266	1.0078	0.7858
16	0.7827	0.8717	0.5778	0.7551	0.7365	0.6170
17	0.7299	0.4993	0.8013	0.6187	0.7766	0.7297
18	0.3540	0.8949	0.8256	0.7357	0.8424	0.5218
19	0.8989	0.7526	0.7494	0.8080	0.6221	0.8891
20	0.7235	1.0205	0.7144	0.7569	0.6999	0.4829
Mean	0.7099	0.7067	0.7546	0.7641	0.6431	0.6606

Table 5.3: Maximum drawdown for the different test portfolios.

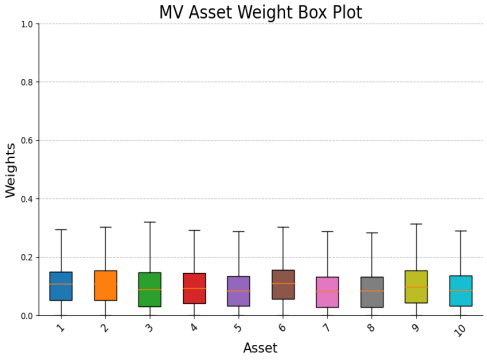
Test	**NSRDMM-Mo**	**NSRDMM**	**MV**	**ERC**	**SAC**	**DDPG**
1	0.2545	0.2733	0.2225	0.2525	0.3717	0.3646
2	0.2192	0.1937	0.1814	0.1882	0.1815	0.2362
3	0.3129	0.4002	0.1949	0.3325	0.4292	0.2928
4	0.1750	0.2290	0.2344	0.1661	0.1814	0.2515
5	0.1727	0.1416	0.1621	0.1463	0.2297	0.1957
6	0.2639	0.2538	0.3169	0.2599	0.3650	0.3149
7	0.2497	0.2762	0.2082	0.1862	0.2451	0.3480
8	0.3016	0.2964	0.2568	0.2892	0.3636	0.3281
9	0.2367	0.2146	0.2427	0.2136	0.2572	0.2314
10	0.2948	0.3186	0.3366	0.2383	0.4284	0.2797
11	0.2198	0.2813	0.2192	0.2084	0.3405	0.2787
12	0.3196	0.2994	0.2756	0.2408	0.2784	0.3018
13	0.3366	0.2931	0.2927	0.2183	0.3492	0.2763
14	0.2377	0.2623	0.2314	0.2080	0.2335	0.2408
15	0.2573	0.2483	0.1834	0.2260	0.1520	0.2974
16	0.2162	0.2090	0.2414	0.2104	0.2651	0.2741
17	0.2777	0.2878	0.2937	0.2895	0.2833	0.2877
18	0.3176	0.2600	0.2469	0.2723	0.2955	0.2755
19	0.2509	0.2089	0.2369	0.2422	0.2669	0.2106
20	0.3227	0.2120	0.2393	0.2524	0.2806	0.3310



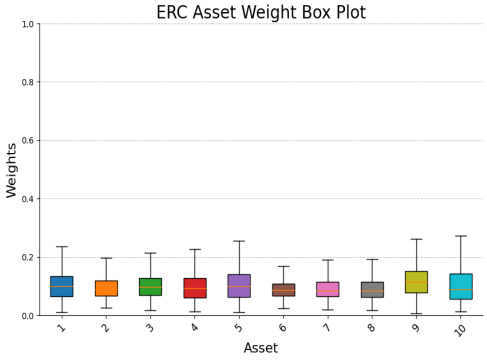
(a) NSRDMM-Momentum



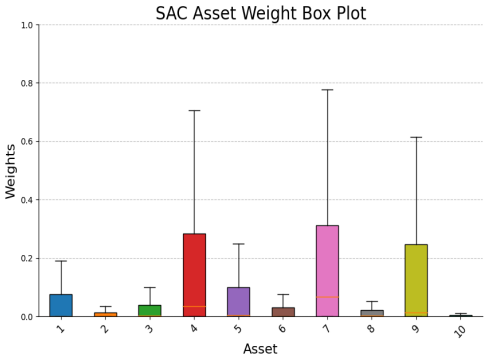
(b) NSRDMM



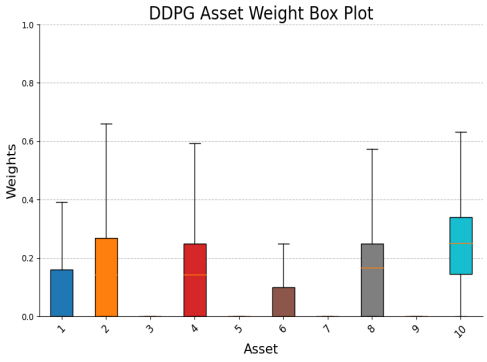
(c) MV



(d) ERC



(e) SAC



(f) DDPG

Figure 5.6: Asset Weight Box Plots

Chapter 6

Conclusion

In conclusion, this thesis has developed and evaluated NSRDMM, a model-based off-policy reinforcement learning algorithm that incorporates non-stationarity inherent in financial markets. Through comparative analysis with both traditional model-based approaches (MV/ERC) and state-of-the-art model-free methods (DDPG/SAC), NSRDMM has not only demonstrated its capability to achieve superior cumulative returns over MV, ERC, DDPG, and SAC but also excelled in Sharpe Ratio performance against DDPG and SAC, albeit with room for improvement against MV and ERC. Furthermore, NSRDMM showcased significant sample efficiency surpassing that of DDPG and SAC which is important for many applications.

6.1 Recommendations for Future Work

We believe the flexibility of adding features to NSRDMM can improve the performance of the model compared to classical methods such as MV and ERC. For example, the integration of additional features such as the VIX and other measures of volatility may enhance the performance of the algorithm. As features are added to the model, it may behave more as a trading strategy compared to a pure portfolio optimizing method. This is related to the problem of multi-objective reward functions [Dulac-Arnold et al., 2019, 2020, 2021] as there is a balancing act between maximizing the expected return and minimizing variance. It is a useful avenue to further explore a framework to analyze and control this trade-off.

While we chose to use a Gaussian policy, a Gaussian policy with decaying variance suggested by the entropy-regularized framework by Wang and Zhou [2020] has not been tested in NSRDMM. It would be interesting to see an integration and how it could improve performance in the MV-objective.

Further recommendations include enhancements that could include implementing a hyperparameter tuning method such as Bayesian optimization. Additionally, there are

countless ways to design neural network architectures and improvements could be made by trying different depths, numbers of nodes, activation functions etc. It would be a promising idea to replace the fully connected neural networks with recurrent neural networks (RNNs) which could be particularly useful for processing time-series data. Moreover, the current success of large language models and transformers might pose promising avenues for improvement in various ways. For example, LLMs can be used for time series prediction. In addition, inspired by the recommendation from Sluijterman et al. [2023], a modified MVE network could be used that applies distinct regularization techniques for the prediction of the mean and variance. This could improve the modelling of probabilistic graphical models and hence lead to more accurate learned model distributions.

Lastly, we have not used NSRDMM's ability to simulate data from the trained decoding networks. An interesting suggestion for future research would be to test whether learning a policy on such simulated data is effective and how it compares to real data.

Bibliography

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

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Kerry Back. *Asset pricing and portfolio choice theory*. Oxford University Press, 2010.

Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, 19(8):1271–1291, 2019.

Álvaro Cartea, Sebastian Jaimungal, and Leandro Sánchez-Betancourt. Deep reinforcement learning for algorithmic trading. *Available at SSRN 3812473*, 2021.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Kevin Dabérius, Elvin Granat, and Patrik Karlsson. Deep execution-value and policy based reinforcement learning for trading and beating market benchmarks. *Available at SSRN 3374766*, 2019.

Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*, 2020.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

Tadeu A Ferreira. Reinforced deep markov models with applications in automatic trading. *arXiv preprint arXiv:2011.04391*, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Ben Hambly, Renyuan Xu, and Huining Yang. Recent advances in reinforcement learning in finance. *Mathematical Finance*, 33(3):437–503, 2023.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Sebastian Jaimungal. Reinforcement learning and stochastic optimisation. *Finance and Stochastics*, 26(1):103–129, 2022.

Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*, 2017.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

Zhipeng Liang, Hao Chen, Junhao Zhu, Kangkang Jiang, and Yanran Li. Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*, 2018.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Xiao-Yang Liu, Zhuoran Xiong, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522*, 2018.

David G Luenberger. *Investment science*. Oxford university press, 1998.

Sébastien Maillard, Thierry Roncalli, and Jérôme Teïletche. The properties of equally weighted risk contribution portfolios. *The Journal of Portfolio Management*, 36(4):60–70, 2010.

Shie Mannor and John N Tsitsiklis. Algorithmic aspects of mean–variance optimization in markov decision processes. *European Journal of Operational Research*, 231(3):645–653, 2013.

Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. ISSN 00221082, 15406261.

Robert C. Merton. An analytic derivation of the efficient portfolio frontier. *The Journal of Financial and Quantitative Analysis*, 7(4):1851–1872, 1972. ISSN 00221090, 17566916.

Robert C Merton. On estimating the expected return on the market: An exploratory investigation. *Journal of financial economics*, 8(4):323–361, 1980.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

John Moody and Matthew Saffell. Reinforcement learning for trading. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11. MIT Press, 1998.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

Brian Ning, Franco Ho Ting Lin, and Sebastian Jaimungal. Double deep q-learning for optimal execution. *Applied Mathematical Finance*, 28(4):361–380, 2021.

David A Nix and Andreas S Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, volume 1, pages 55–60. IEEE, 1994.

Hyungjun Park, Min Kyu Sim, and Dong Gu Choi. An intelligent financial portfolio trading strategy using deep q-learning. *Expert Systems with Applications*, 158:113573, 2020.

Parag C Pendharkar and Patrick Cusatis. Trading financial indices with reinforcement learning agents. *Expert Systems with Applications*, 103:1–13, 2018.

José C Pinheiro and Douglas M Bates. Unconstrained parametrizations for variance-covariance matrices. *Statistics and computing*, 6:289–296, 1996.

PyTorch. torch.nn.LayerNorm. `https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html`, 2024. Accessed: 2024-02-27.

Ashwin Rao and Tikhon Jelvis. *Foundations of reinforcement learning with applications in finance*. CRC Press, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Mohit Sewak and Mohit Sewak. Policy-based reinforcement learning approaches: Stochastic policy gradient and the reinforce algorithm. *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*, pages 127–140, 2019.

William F. Sharpe. The sharpe ratio. *Journal of Portfolio Management*, 21(1):49, Fall 1994.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

Euan Sinclair. *Volatility trading*. John Wiley & Sons, 2013.

Laurens Sluijterman, Eric Cator, and Tom Heskes. Optimal training of mean variance estimation neural networks. *arXiv preprint arXiv:2302.08875*, 2023.

Srijan Sood, Kassiani Papasotiriou, Marius Vaiciulis, and Tucker Balch. Deep reinforcement learning for optimal portfolio allocation: A comparative study with mean-variance optimization. *FinPlan 2023*, page 21, 2023.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Richard S Sutton, Satinder Singh, and David McAllester. Comparing policy-gradient algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 2000.

Nassim Nicholas Taleb. *Dynamic hedging: managing vanilla and exotic options*, volume 64. John Wiley & Sons, 1997.

Aviv Tamar, Dotan Di Castro, and Shie Mannor. Learning the variance of the reward-to-go. *The Journal of Machine Learning Research*, 17(1):361–396, 2016.

Aad W Van der Vaart. Time series. *VU University Amsterdam, lecture notes*, 2010.

Elena Vigna. On time consistency for mean-variance portfolio selection. *International Journal of Theoretical and Applied Finance*, 23(06):2050042, 2020.

Haoran Wang. Large scale continuous-time mean-variance portfolio allocation via reinforcement learning. *arXiv preprint arXiv:1907.11718*, 2019.

Haoran Wang and Xun Yu Zhou. Continuous-time mean–variance portfolio selection: A reinforcement learning framework. *Mathematical Finance*, 30(4):1273–1308, 2020.

Christopher JCH Watkins. Learning from delayed rewards. 1989.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

Wikipedia. Graphical model — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Graphical_model`, 2024.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Annie Xie, James Harrison, and Chelsea Finn. Deep reinforcement learning amidst lifelong non-stationarity. *arXiv preprint arXiv:2006.10701*, 2020.

Annie Xie, James Harrison, and Chelsea Finn. Deep reinforcement learning amidst continual structured non-stationarity. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11393–11403. PMLR, 18–24 Jul 2021.

Mengran Yu and Shiliang Sun. Policy-based reinforcement learning for time series anomaly detection. *Engineering Applications of Artificial Intelligence*, 95:103919, 2020.

Pengqian Yu, Joon Sern Lee, Ilya Kulyatin, Zekun Shi, and Sakyasingha Dasgupta. Model-based deep reinforcement learning for dynamic portfolio optimization. *arXiv preprint arXiv:1901.08740*, 2019.

Zihao Zhang, Stefan Zohren, and Roberts Stephen. Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2020.

Appendix A

Optimality of Gaussian Polcies

In solving the MV-objective through Equation (3.3), [Wang, 2019, Wang and Zhou, 2020], derive that a Gaussian policy is an optimal policy. [Wang, 2019, Wang and Zhou, 2020] introduce a randomized control process $u = \{u_t, 0 \leq t \leq T\}$ where $u_t \in \mathbb{R}^d$ denotes the invested amount in each of the d assets. This control process leads to a measure-valued (or distributional) control process with density function $\pi = \{\pi_t, 0 \leq t \leq T\}$. They describe the dynamics of the wealth process for a portfolio of d assets follows

$$dX_t^\pi = \left(\int_{\mathbb{R}^d} \rho^\intercal \sigma u \pi_t(u) du \right) dt + \left(\int_{\mathbb{R}^d} u^\intercal \sigma^\intercal \sigma u \pi_t(u) du \right)^{\frac{1}{2}} dB_t, \tag{A.1}$$

where ρ is a vector that represents the "market price of risk", $\sigma \in \mathbb{R}^d \times d$ is an invertible volatility/covariance matrix, and B_t is a one-dimensional Brownian motion on a filtered probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Then, [Wang and Zhou, 2020, Wang, 2019] introduce a temperature parameter λ and a differential entropy term to model the trade-off between exploration and exploitation. Then the "entropy-regularized, exploratory MV problem" for each $w \in \mathbb{R}$ is given by [Wang, 2019, Wang and Zhou, 2020]

$$\min_{\pi \in \mathcal{A}(x_0,0)} \mathbb{E}\left[(X_t^\pi - w)^2 + \lambda \int_0^T \int_{\mathbb{R}^d} \pi_t(u) \ln \pi_t(u) du dt \right] - (w - z)^2, \tag{A.2}$$

where $\mathcal{A}(x_0, 0)$ is the set of controls admissible. Then, after defining the value function V using Bellman's principle of optimality and noting that it satisfies the Hamilton-Jacobi-Bellman equation, [Wang, 2019, Wang and Zhou, 2020] derive that the optimal feedback control is Gaussian with its distribution given by

$$\pi^*(t, x, w) = \mathcal{N}\left(-\sigma^{-1}\rho(x - w), (\sigma^\intercal \sigma)^{-1} \frac{\lambda}{2} e^{\rho^\intercal \rho (T-t)} \right), \tag{A.3}$$

from which a vector $u \in \mathbb{R}^d$ is sampled denoting the amount to be invested in each asset, and where the Lagrange multiplier w is given by $w = \frac{z e^{\rho^\intercal \rho T} - x_0}{e^{\rho^\intercal \rho T} - 1}$. Furthermore, [Wang, 2019,

Wang and Zhou, 2020] show that the Gaussian policy can be parameterized as $\pi(t, x, w) = \mathcal{N}\left(\alpha(x - w), \Sigma e^{\beta(T-t)}\right)$, where $\alpha \in \mathbb{R}^d, \beta \in \mathbb{R}$ and where Σ is a $d \times d$ matrix which is positive definite, as it converges in distribution to the optimal policy π^*.

Appendix B

All code can be found in this GitHub repository: https://github.com/lchinapauw/RL-portfolio-allocation.