Asset Allocation using Reinforcement Learning

Stefano Giacomazzi Dantas

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

August 2020

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Science

© 2020 Stefano Giacomazzi Dantas

Abstract

The asset allocation problem can be defined as the decision making process of periodic redistribution of capital into different financial assets. The goal is to maximize the return obtained while restraining the risk.

Most of the classical methods for asset allocation rely on assumptions regarding the distributions of return and risks, as well as models for the correlation structures between various resources/assets. However, estimation of these quantities often fails to produce good results as financial markets are extremely challenging to model due to their dynamic, nonlinear, nonstationary, and noisy nature.

On the other hand, representing the market as a discrete-time stochastic system allows the usage of reinforcement learning as an alternative framework. In this case, no assumptions are made about statistical moments of financial time series or prediction of future prices.

The goal of this thesis is to develop a reinforcement learning model that is able to optimize a portfolio given a particular set of information. We present two extensions for the Recurrent Reinforcement Learning algorithm and compare their performance with other reinforcement learning baselines, including existing extensions of this same algorithm. The proposed model outperforms all the baselines in terms of cumulative return in our evaluations. In addition, we also analyze the model's capability of selecting better assets within an asset class, generating more returns while maintaining the same risk level.

Sommaire

Le problème de l'allocation d'actifs peut être défini comme le processus décisionnel de redistribution continue du capital entre différents actifs financiers. L'objectif est de maximiser le rendement obtenu tout en limitant le risque.

La plupart des méthodes classiques d'allocation d'actifs reposent sur des hypothèses de distribution de rendement et de risques, ainsi que sur les structures de corrélations entre différents actifs. Cependant, ces estimations échouent souvent à produire de bons résultats car les marchés financiers sont extrêmement difficiles à modéliser en raison de leur nature dynamique, non linéaire, non stationnaire et bruyante.

D'autre part, représenter le marché comme un système stochastique à temps discret permet d'utiliser l'apprentissage par renforcement pour une approche différente. Dans ce cas, aucune hypothèse n'est faite sur les moments statistiques des séries chronologiques financières ou sur la prévision des prix futurs.

L'objectif de cette thèse est de développer un modèle d'apprentissage par renforcement capable d'optimiser un portfolio à partir d'un ensemble particulier d'informations. Nous présentons deux extensions pour l'algorithme d'apprentissage récurrent par renforcement et comparons leurs performances avec d'autres bases d'apprentissage par renforcement, y compris les extensions de ce même algorithme créées par d'autres auteurs. Le modèle proposé a surpassé tous les niveaux de référence en termes de rendement cumulatif dans nos évaluations. En outre, nous analysons également la capacité du modèle à sélectionner de meilleurs actifs au sein d'une classe d'actifs, générant plus de rendements tout en maintenant le même niveau de risque.

Acknowledgments

First and foremost, I would like to express my gratitude and thanks to my supervisor Prof. Mark Coates for his support, patience, and guidance throughout the progression of my thesis. His knowledge and advice were fundamental for concluding this step in my life, especially after I pivoted my research focus to a different topic.

I would also like to thank Professor Milica Popovic for all her support during my time as an exchange student at McGill five years ago. Her help was extremely important in guiding my decision to come to McGill to pursue my Master's degree.

Additionally, I would like to thank all my colleagues in the Computer networks lab for their companionship. In particular, I would like to thank Florence Robert-Regol and Soumyasundar Pal for all the helpful discussions and their help during the first year of my Master's.

Moreover, I would like to thank Arthur Carvalho, Gabriel Bayomi, Gabriel Castellano, Gustavo Cid, Henrique Orefice, João Antônio, Leonardo Albuquerque, Luiz Felipe, Letícia Brito, Pedro Campos and Renata Lopes for all the laughs and support throughout this journey. "We all take different paths in life, but no matter where we go, we take a little of each other everywhere."

I would also like to thank João Pedro, for all his help and friendship since the first days of our exchange program.

This thesis would not have been possible without the help of my partner, Luisa Marini. Her support, in particular during the final moments of this journey, was essential in finishing my thesis. I would also like to thank my sister for all her encouragement and conversations during the many ups and downs of this period.

Last but not least, I would like to thank my parents for giving me all the necessary conditions to follow my dreams, and for raising me to believe that anything is possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traditional resource allocation techniques that rely on simple heuristics often fail to achieve good performance on complex systems, such as virtual servers, cloud services and telecommunication networks [118]. In the field of finance, resource allocation applications are essential to optimize investment decisions. These problems contain scarce resources as well as interconnected and time-dependent tasks. Recently, machine learning approaches are being used with success to manage and optimize these systems.

1.1 Motivation

In the field of finance, machine learning techniques are gaining ground quickly. In recent years, significant impact has been made on different applications, such as stock prediction [27, 49, 99], bankruptcy prediction [4, 64], stock trading [26, 33] and portfolio allocation [18, 60, 77].

In this thesis, we are interested in applying machine learning techniques for portfolio optimization, a form of resource allocation. Resource allocation problems are of high practical importance, arising in many fields, such as fleet and personnel management, scheduling of computer programs, manufacturing production control, control of mobile telecommunication networks and finance applications [1, 6, 24, 34].

Real world resource management problems are challenging for several reasons: 1) the underlying systems are complex and often impossible to model accurately; 2) decisions should be made online with noisy inputs and work well under diverse conditions; 3) some performance metrics are often hard to optimize.

In this context, machine learning can provide a viable alternative to human-generated heuristics that are often used in this field [81]. More specifically, the sub-field of reinforcement learning is often used to tackle these problems. It solves sequential decision making problems by interacting directly with the environment.

Some successful examples of applying reinforcement learning to decision-making domains have been presented recently, such as playing games [87, 106], routing telecommunication networks [52] and medicine [100].

In this thesis, we are interested in the *portfolio management* problem, which is the decision making process of periodically reallocating an amount of funds into a number of different financial investment products, aiming to maximize the return while restraining the risk [50]. One of the most effective ways to formulate this problem is by using a discrete-time (partially observable) Markov Decision Process (MDP) [9, 93]. Hence, it becomes a stochastic optimal control problem where the system being controlled in discrete time is the portfolio with multiple assets, and the controls are the fractions of capital allocations. This representation enables the development of trading agents using the reinforcement learning framework.

1.2 Thesis Organization and Contributions

This section outlines the structure of this thesis and summarizes the technical contributions that are proposed.

— *Chapter 2 - Background: Reinforcement Learning*
We present the background material on reinforcement learning that is needed for this thesis. We start by presenting the fundamentals of Reinforcement Learning, and the definition of a Markov Decision Process. Then, we discuss policies and the methods used to estimate them. To conclude, we present the differences between model-based and model-free algorithms.

— *Chapter 3 - Financial Theory*
In this chapter, we present a brief overview of financial concepts that are required for this thesis. Then, we introduce the task of portfolio optimization, two methods to solve it, and some common metrics that are used to evaluate a portfolio's performance.

— *Chapter 4 - Literature Review*
We present a concise literature review of reinforcement learning applications in finance, with a special focus on asset allocation.

— *Chapter 5 - Methodology and Proposed Extensions*
We formalize the problem of asset allocation in order to be compatible with the reinforcement learning framework. We define the main assumptions, types of agent used, and how the environment is modelled. Next, we validate the baselines using this methodology. Finally, We propose two extensions for the Recurrent Reinforcement Learning algorithm and discuss their practical implications and benefits.

— *Chapter 6 - Results*
We start the chapter with a brief discussion on expected returns and covariance estimation, which are essential components of classical portfolio optimization methods. Next, we present a brief discussion of the reinforcement learning models used in this thesis. Finally, we present the results for different experiments.

— *Chapter 7 - Conclusion*
We provide a summary of the main contributions of the thesis, discuss the implications of the results and make some suggestions for future work.

1.2.1 Contributions

— *Chapters 5 and 6 - Methodology, Proposed Extensions, and Experiments*
Prof. Mark Coates provided guidance with the research plan and experimental procedure. I designed and implemented the algorithm. I also conducted the experiments to evaluate the proposed methods.

Chapter 2

Background: Reinforcement Learning

Reinforcement learning is a sub-field of machine learning which aims to complete a given task by maximizing the rewards obtained by a software agent that interacts with an environment. Unlike supervised learning, the model does not learn by using input/output pairs provided by a supervisor. Instead, the agent learns by trial and error. One of the challenges is to find the balance between exploring new actions and exploiting already known paths.

During the interaction with the environment, the agent receives a numerical reward based on its action and state. The goal of the agent is to maximize the cumulative return obtained. The environment can keep evolving in a stochastic manner, so an action taken by the agent can influence the circumstances that it will encounter in the future. Therefore, another common challenge is balancing how short-sighted the agent's desire to obtain an immediately reward is in contrast to obtaining a delayed, but larger, reward.

We start this chapter by presenting in Section 2.1 a brief overview of other machine learning paradigms, known as supervised and unsupervised learning, followed by the fundamentals of Reinforcement Learning (Section 2.2), and the definition of a Markov Decision Process (Section 2.3). Next, we discuss policies in more detail in Section 2.4, followed by two classes of methods used to estimate them, value-based methods in Section 2.5, and policy-based methods in Section 2.6. We conclude the chapter with Section 2.7, which presents the differences between model-based and model-free algorithms.

2.1 Other Machine Learning Paradigms

In this section, we present a brief description of other machine learning paradigms.

2.1.1 Supervised Learning

Perhaps the most studied paradigm of machine learning, supervised learning is characterized by the existence of labeled examples provided by a knowledgeable external supervisor

[110]. In other words, each example seen by the model during training has a corresponding "true output", a label in the case of a classification task or a real value in case of a regression.

The goal of the supervised system is generalizing its predictions to unseen data, i.e., data not presented in the training set. This sub-field of machine learning is widely used nowadays with applications in many different industries. However, it also has some limitations, especially the dependency on labeled examples.

In some problems, it is impractical to obtain examples of desired behavior. For instance, in interactive problems, the training data must be able to represent all the different situations that an agent might encounter and specify its expected behaviour in these situations. For this type of problem, an agent's ability to learn from its own experience is extremely valuable.

2.1.2 Unsupervised Learning

The other main paradigm of machine learning is known as unsupervised learning. Unlike supervised learning, there is no supervisor to provide labeled examples. Instead, the goal of this paradigm is finding structure hidden in collections of unlabeled data [110].

Both unsupervised and reinforcement learning techniques do not depend on labeled data. The main difference between these two methodologies is their goal. The goal of reinforcement learning is to maximize the cumulative reward obtained by an agent; the goal of unsupervised learning is to discover hidden structure in the data.

2.2 Elements of Reinforcement Learning

The learning procedure of a reinforcement learning agent has two main distinctions compared to other paradigms: the agent learns which actions are desirable by trying them, and some of its actions may affect not only the immediate reward but also subsequent rewards. Reinforcement learning emphasizes learning by interacting directly with an environment in terms of states, actions and rewards.

2.2.1 Reward

The reward R_t is a scalar feedback signal that represents how well the agent is doing at step t. The goal of the agent is to maximize the total reward received over the whole episode of interaction with the environment.

The reward is then used to describe to the agent what are good and bad outcomes. It is the primary basis for altering the behaviour of the agent; if a selected action is followed by low reward, then the agent may change its action in that situation in the future. The learning procedure is based on the reward hypothesis [45].

Hypothesis 1 (Reward Hypothesis)**.** *All goals can be described by the maximization of expected cumulative reward.*

As a consequence, it is essential to select an appropriate reward signal for each application, as it is used to describe the agent's goal. Let $R_{t+1}, R_{t+2}, R_{t+3}, ...$ denote the sequence of rewards obtained after time step t. We define the cumulative return G_t as the sum of rewards:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T, \tag{2.1}$$

where T is the final time step. In situations where the notion of final step is clear, i.e., when the agent's interaction with the environment is naturally divided into sub-sequences, this approach is often used. However, in many cases the identification of individual episodes is not straightforward. Instead, the agent and environment interact continuously throughout time and there is no final time step.

The main implication of the return formulation in (2.1) is that the agent could achieve a sub-optimal solution and still achieve a high reward by just repeating the same action forever. The definition of return can be improved by introducing a discount rate γ , $0 \leq \gamma \leq 1$, and considering the **discounted cumulative reward**, defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\,. \tag{2.2}$$

The parameter γ determines what is the present value of future rewards. If γ is close to 0, the agent will be more concerned about immediate rewards, as future rewards are deeply discounted. On the other hand, if γ is close to 1, the return will place more emphasis on future rewards, because even after discounting a higher reward in the future can be worth more than a smaller reward in the present. In other words, the agent becomes more farsighted.

2.2.2 Agent and Environment

One of the main characteristics of reinforcement learning is the direct interaction between the **agent**, the term used to refer to the controller, and the **environment**, which is the system with which the agent interacts. At each step t, the agent takes an action $a_t \in \mathcal{A}$, receives a reward R_t, and makes an observation O_t. On the other hand, the environment receives the action A_t and outputs the observation O_{t+1} and reward R_{t+1}. This relationship is illustrated in Figure 2.1. The sequence of observations, actions and rewards is called **history** H_t:

$$H_t = O_1, R_1, a_1, ..., O_t, R_t, a_{t-1}$$

.

Another fundamental element of reinforcement learning is the **state** $s_t \in \mathcal{S}$. It contains the information used to determine what happens at each time step. Formally, the state is a function of the history. There are two distinct states: the environment state s_t^e and the agent state s_t^a. The former is the internal representation of the environment and is used to determine the next observation and reward. The latter can be any function of the history $f(H_t)$, and contains the information used by the agent to select the next action.

In some cases, the agent is able to observe the environment state. Therefore, $O_t = s_t^a = s_t^e$. These are known as fully observable environments. If the environment state s_t^e is not

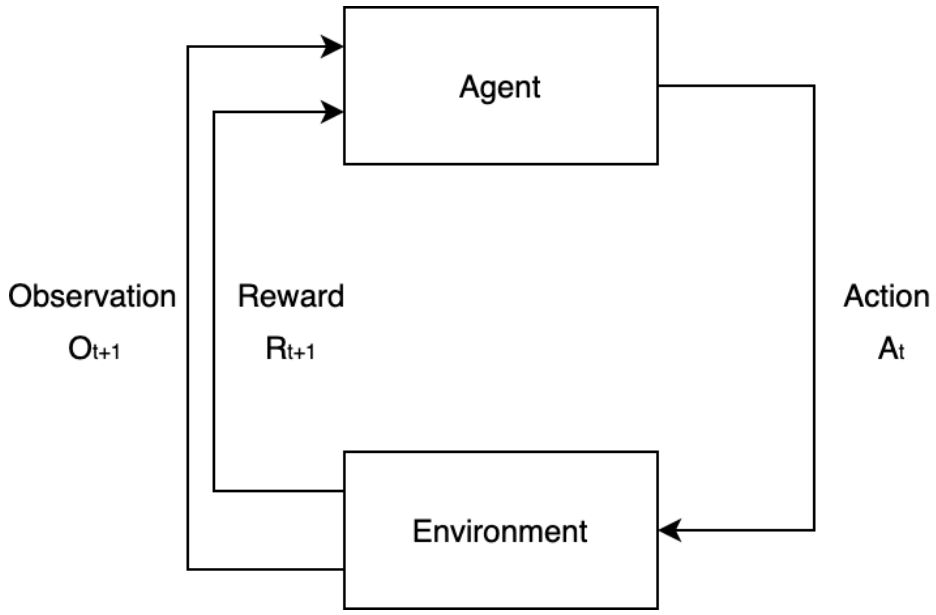

Figure 2.1: Agent and environment dynamics

observable, which is often the case, the agent state acts as an indirect representation of the environment. In other words, the agent indirectly observes the environment. As a consequence, the agent state s_t^a should contain all relevant information for the agent to take action a_{t+1}.

To simplify notation, we will represent the agent state as s_t. The set of possible states $\mathcal{S}$ can be either discrete $\mathcal{S} = \{k_1, k_2, ...\}$ or continuous $\mathcal{S} \subseteq \mathbb{R}^K$.

2.3 Markov Decision Processes

In sequential decision making problems, an agent interacts with an environment by selecting a series of actions in order to achieve its goal. One of the most commonly used frameworks to model this task is the Markov Decision Process (MDP). In this formulation, actions influence not just immediate rewards, but also subsequent states and actions.

2.3.1 Markov Property

In a finite MDP, the sets of states ($\mathcal{S}$), actions ($\mathcal{A}$), and rewards have a finite number of elements. In this case, the random variable that represents the state s_t has a discrete probability distribution which depends only on the preceding state and action. That is, for any given state, the previous state contains all information needed to define its probability distribution. This property is known as the **Markov Property**. Formally, a state s_t is Markov if:

$$P[s_t|s_{t-1}, s_{t-2}, ..., s_1] = P[s_t|s_{t-1}]\,. \tag{2.3}$$

This property implies that the previous state captures all relevant information from the history H_t.

2.3.2 Definition

An MDP is a stochastic dynamical system characterized by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{R} \rangle$ [45], where:

— $\mathcal{S}$ is the state space, a finite set of states which satisfy the Markov condition.
— $\mathcal{A}$ is the action space, a finite set of actions.
— $\mathcal{P}$ is a state transition probability matrix, which determines the probability of moving to a given state s_{t+1} after taking action a at state s. Thus $\mathcal{P}_{s,s'} = P[S_{t+1} = s' | s_t = s, a_t = a]$.
— $\mathcal{R}$ is a reward function.
— γ is the discount rate.

A Markov Decision Process is defined by its state, action and environment dynamics in the next time step. Given an action a and state s, the probability of a given state-reward pair s', r is:

$$p(s', r \mid s, a) = P[s_{t+1} = s', R_{t+1} = r \mid s_t = s, a_t = a]. \tag{2.4}$$

Here p defines a probability distribution for each state s and action a. In other words, all possible combinations are described by p. Hence,

$$\sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{2.5}$$

The MDP framework is a powerful abstraction for solving sequential decision making problems by learning from interaction. The problem is reduced to three components: actions, states and rewards. This methodology may not be applicable to all types of decision-learning problems, but it has proved useful in multiple application domains [110].

2.4 Policies

As stated previously, the purpose of an RL agent is formalized as maximizing the cumulative reward received. In order to achieve its goal, the agent has to decide at each time step which action $A_t \in \mathcal{A}$ to take, depending on the agent's current state s_t.

This decision is based on the agent's **policy**. A policy is a mapping from state to action and defines the agent's behaviour. For example, let π be the policy at time t, then $\pi(a|s)$ defines the probability of taking action $a_t = a$ in state $s_t = s$. The policy can also be deterministic, i.e., $\pi(s) = a, s \in \mathcal{S}, a \in \mathcal{A}$.

As a consequence, the goal of a RL problem can also be defined in terms of its policy. The purpose then becomes finding an optimal policy which maximizes the cumulative reward acquired. Different reinforcement learning methods specify different ways to adapt the policy as a result of interaction with the environment.

A policy can be improved in two different ways:

— **Directly**: The policy can be learned directly by using the reward obtained to update the agent's behaviour. Such methods are called **policy-based**.
— **Indirectly**: Instead of learning the policy directly, in this methodology an agent learns a value (or action-value) function and uses it to guide its actions. Techniques adopting this approach are known as **value-based** methods.

2.5 Value Functions

Value-based reinforcement learning algorithms are based on estimating value functions, which define how good it is for the agent to be in a given state. The notion of "how good" is defined in terms of expected future rewards.

Since the agent's behaviour depends on its policy, the value function is also a function of the policy π. In this case, $v_\pi(s)$ is the value of state s under a policy π, i.e., the expected return when starting in state s and following the policy π from there on. We define $v_\pi(s)$ as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s]\,, \tag{2.6}$$

where $\mathbb{E}_\pi[\cdot]$ is the expected return given that the agent follows π.

Similarly, we can define the action-state value function $q_\pi(s, a)$ as the expected return for taking action a in state s and following the policy π afterward:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid s_t = s, A_t = a] = \sum_{k=0}^{\infty} \gamma^k R(s_k, a_k). \tag{2.7}$$

We can separate equation (2.7) into two parts:

$$\begin{aligned} q_\pi(s,a) &= \mathcal{R}(s,a) + \sum_{k=1}^{\infty} \gamma^k R(s_k, a_k) \\ &= \mathcal{R}(s,a) + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} R(s_k, \pi(s_k)) \\ &= \mathcal{R}(s,a) + \gamma v_\pi(s'). \end{aligned} \tag{2.8}$$

The first component is the immediate reward $\mathcal{R}(s, a)$ obtained by taking action a in state s. The second term is the expected return by following policy π starting from s'.

Both value functions q_π and v_π can be estimated from experience. A fundamental property of value functions is that they satisfy some recursive relationships. For any policy π and any state s, the value of s and the value of its possible future states follow the condition:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid s_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid s_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(s') \mid s_t = s]. \end{aligned} \tag{2.9}$$

The actions a are taken from $\mathcal{A}$, the states s and s' from $\mathcal{S}$, and the reward r from $\mathcal{R}$. Equation (2.9) is known as the Bellman equation for the value function following policy π and it forms the basis of several ways to learn and compute value functions. [1]

Value functions and state-value functions are used as indirect ways to find an optimal policy, i.e., finding a policy that achieves the highest cumulative reward. For MDPs, we can use these functions as a way to order different policies. An arbitrary policy π is defined to be better than another policy π' if its value function v_π is greater than $v_{\pi'}$ for all states. Formally, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)\ \forall s \in \mathcal{S}$.

There always exists at least one optimal policy, a policy that is better than or equal to all other policies [110]. Optimal policies are denoted by π^*. The optimal value function is then defined as:

$$v_*(s) = \max_\pi v_\pi(s). \tag{2.10}$$

Similarly, we can also define the optimal action-state value function as:

$$q_*(s,a) = \max_\pi q_\pi(s,a). \tag{2.11}$$

Methods that rely on value functions aim to maximize these functions as way of improving their behaviour (policy) and, consequently, maximizing the reward obtained. After the optimal $v_*(s)$ or $q_*(s,a)$ is found, the agent only has to act greedily with respect to these functions.

2.6 Policy Gradient Methods Overview

An alternative approach is to improve the policy directly without relying on value functions. The goal of policy optimization is to maximize a function $J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}[\sum_{t=0}^{T} R_t] = \underset{\tau\sim\pi_\theta}{E}[R(\tau)]$ by updating the parameterized policy π_θ. This is achieved by using gradient ascent, iteratively adapting the parameters, with the k-th update being:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k} \tag{2.12}$$

The gradient of the policy performance $\nabla_\theta J(\pi_\theta)$ is called **policy gradient**. In order to actually use this algorithm we need an expression for numerically computing this term, which was derived in [111].

Let the probability of a given trajectory $\tau = (s_0, a_0, ..., s_{T+1})$ following the policy π_θ be defined as:

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)\,, \tag{2.13}$$

1. We do not elaborate in greater details on the Bellman equation in this thesis, an extensive discussion can be found in [110].

where s_t is the state at time t, a_t is the action taken at time t and ρ_0 is the initial state probability. The policy gradient is then given by

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \underset{\tau\sim\pi_\theta}{E}[R(\tau)] \tag{2.14}$$

$$= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) \qquad \text{Expanding expectation} \tag{2.15}$$

$$= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) \qquad \text{Bringing gradient under integral} \tag{2.16}$$

We can use the log-derivative trick to write the derivative of the trajectory as

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta \log P(\tau|\theta). \tag{2.17}$$

Inserting this in (2.16) and rewriting as an expectation, we have:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}[\nabla_\theta \log P(\tau|\theta)R(\tau)] \tag{2.18}$$

Since the environment has no dependence on θ, the gradient of the log-probability of the trajectory becomes simply

$$\nabla_\theta \log P(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{2.19}$$

The final expression for the policy gradient is then:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right] \tag{2.20}$$

2.6.1 REINFORCE: Monte-Carlo Policy Gradient

One of the simplest policy gradient algorithms is the Monte-Carlo Policy Gradient, also known as **REINFORCE** [122]. It addresses the calculation of the expectation of the $\nabla_\theta J(\pi_\theta)$ term by obtaining samples such that the expectation of this sample gradient is proportional to the actual gradient of the performance measure.

If we let an agent act according to a policy π_θ, we can estimate its policy gradient by collecting enough trajectories $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ and averaging them using

$$\hat{g} = \frac{1}{N}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau) \tag{2.21}$$

The actions are decided by sampling the policy π_θ. The key idea underlying this method is to increase the probability of taking actions that lead to higher returns and decrease the probabilities of actions that lead to lower returns. Each update in the gradient is proportional

to the product of the sum of returns (G_t) and the gradient of the probability of taking those actions.

In the case where the action space is discrete, the output of the policy network is a probability distribution over actions, which is used to sample new actions based on their probabilities.

In continuous action spaces, the actions are also chosen based on their probability. However, instead of being a probability distribution, the output of the policy network is the mean vector of a multivariate Gaussian distribution. The variance of this distribution can be fixed or can decrease over time or can be learned. In this thesis, we consider a fixed variance, denoted by an exploration parameter σ_e.

The REINFORCE algorithm has good theoretical convergence properties as the expected update over an period T is in the same direction as the performance gradient. However, this method suffers from high variance and, consequently, slow learning [110].

2.6.2 Actor-Critic algorithms

One tool that is often used to reduce the variance in REINFORCE is to subtract a baseline function $b(s_t)$ from the return G_t. This operation leaves the expected value of the update unchanged, but it can have a large effect on its variance [110].

The gradient update equation then becomes

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \ \left(\sum_{t'=t}^{T} R_{t'} - b(s_t)\right)\right]. \tag{2.22}$$

A natural choice for the baseline is an estimate of the state value, $\hat{v}(s_t;\phi)$, where ϕ is a weight vector that can be learned via a Monte Carlo method.

The idea of incorporating both value function and direct policy is the key point of actor-critic methods [2], which aim to take advantage of both value-based and policy-based techniques while eliminating some of their drawbacks [70].

Actor-critic methods consist of two models, which may optionally share parameters:

— **Critic**: updates the value function parameters ϕ according to the reward received.

— **Actor**: updates the policy parameters θ for $\pi_\theta(s|a)$, in the direction suggested by the critic.

Approaches that combine policy and value based method have achieved state-of-the-art performance for solving reinforcement learning problems. In the following sections we present two of the most prominent policy gradient algorithms used in deep reinforcement learning: trust region policy optimization (TRPO) and proximal policy optimization (PPO).

2. Although the REINFORCE method with baseline briefly introduced in this section learns both policy and value functions, it is not consider an actor-critic method as its value function is used only as a baseline, not as a critic. A more detailed discussion can be found in [110].

2.6.3 Trust Region Policy Optimization

One of the limitations of simple policy gradient methods such as REINFORCE is that training can become unstable if the step sizes α are not chosen carefully. A small change in the parameter space can have a large impact on performance, making this algorithm susceptible to training instabilities. For this reason, the learning rate is often very small, thus reducing the sample efficiency of the algorithm. Another limitation is that the whole trajectory is sampled before updating the policy. A trajectory may contain hundreds or thousands of steps, so updating just once for each trajectory is not sample efficient.

The Trust Region Policy Optimization (TRPO) [103] algorithm aims to solve these limitations by updating policies while satisfying a special constraint on how close the new and old policies are allowed to be, expressed in terms of Kullback-Leibler (KL) divergence. The improvement in policies is guaranteed by iteratively maximizing a lower bound function for the expected reward $\eta(\pi_\theta)$ of a given policy π_θ. This is based on a type of majorization-minimization (MM) algorithm [55].

For each iteration, the algorithm finds a surrogate function M which is a lower bound function for the expected reward η. This should be a good representation of η at the current policy and easy to optimize. After obtaining M, we find the optimal point θ' for it and use it to specify the parameters θ_{i+1} of the current policy. Then, we re-evaluate a new lower bound M_{i+1} for the new policy and repeat the process. The procedure is illustrated in Figure 2.2.

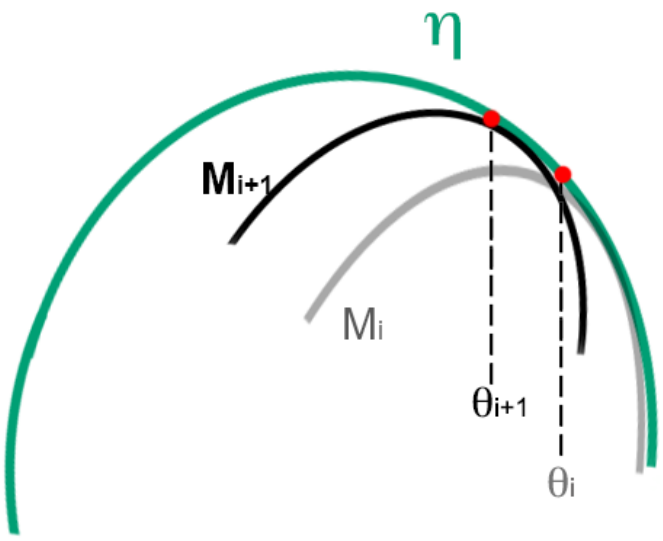

Figure 2.2: Illustration of the MM iterations

We define the advantage function A_π as the difference between the action-state value

function and the state value function:

$$q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l R(s_{t+1})\right], \tag{2.23}$$

$$v_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l R(s_{t+1}),\right] \tag{2.24}$$

$$A_\pi(s, a) = q_\pi(s, a) - v_\pi(s), \tag{2.25}$$

where $a_t \sim \pi(a_t, s_t)$, $s_t \sim P(s_{t+1}|s_t, a_t)\ \forall t > 0$, and γ is the discount rate. The advantage function A_π is important because it allows us to express the expected reward of a policy using another policy. The expected cost accumulated over timesteps of a policy $\widetilde{\pi}$ in terms of the advantage over π is:

$$\mathbb{E}_{\tau\sim\widetilde{\pi}}\left[\sum_{l=0}^{\infty} \gamma^l A_\pi(s_t, a_t)\right] = \sum_s \rho_{\widetilde{\pi}}(s) \sum_a \widetilde{\pi}(a|s) A_\pi(s, a) \tag{2.26}$$

where the actions A_t are sampled from $\widetilde{\pi}$ and ρ_π is the sum of discounted visitation frequencies:

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots \tag{2.27}$$

We can rewrite the left-hand side of equation (2.26) as:

$$\begin{aligned}
\mathbb{E}_{\tau\sim\widetilde{\pi}}\left[\sum_{l=0}^{\infty} \gamma^l A_\pi(s_t, a_t)\right] &= \mathbb{E}_{\tau\sim\widetilde{\pi}}\left[\sum_{l=0}^{\infty} \gamma^l (R(s_t) + \gamma v_\pi(s_{t+1}) - v_\pi(s_t))\right] \\
&= \eta(\widetilde{\pi}) + \mathbb{E}_{\tau\sim\widetilde{\pi}}\left[\sum_{l=0}^{\infty} \gamma^{l+1} v_\pi(s_{t+1}) - \sum_{l=0}^{\infty} \gamma^l v_\pi(s_t)\right] \\
&= \eta(\widetilde{\pi}) - \mathbb{E}_{\tau\sim\widetilde{\pi}}\left[v_\pi(s_0)\right] \\
&= \eta(\widetilde{\pi}) - \eta(\pi)
\end{aligned} \tag{2.28}$$

Using equation (2.28) we can define a local approximation function L:

$$L_\pi(\widetilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \widetilde{\pi}(a|s) A_\pi(s, a) \tag{2.29}$$

Due to the complex dependency of $\rho_{\widetilde{\pi}}(s)$ on $\widetilde{\pi}$, $\rho_\pi(s)$ is used as an approximation instead. [3] This equation indicates that an improvement in $L(\pi)$ will also improve η. However, it does not provide any guidance on how big of a step to take. With this in mind, Schulman et al. use KL divergence in [103] to find a lower bound on the policy improvement.

3. For a parameterized policy π_θ, if $\pi(a|s)$ is a differentiable function of θ, then $L_\pi(\widetilde{\pi})$ is a good first order approximation to $\eta(\widetilde{\pi})$ [62].

Let $D_{KL}(P|Q) = \mathbb{E}_x log\frac{P(x)}{Q(x)}$ denote the KL divergence. The lower bound for a new policy $\widetilde{\pi}$ can be expressed as:

$$\begin{aligned} \eta(\widetilde{\pi}) &\geq L_{\pi}(\widetilde{\pi}) - CD_{KL}^{max}(\pi, \widetilde{\pi}) \\ \text{where } \ \epsilon &= \max_{s,a}|A_{\pi}(s,a)|, \\ C &= \frac{4\epsilon\gamma}{(1-\gamma)^2}, \\ D_{KL}^{max}(\pi, \widetilde{\pi}) &= \max_{s} \ D_{KL}(\pi(\cdot|s)||\widetilde{\pi}(\cdot|s)) \end{aligned} \tag{2.30}$$

Equation (2.30) shows that we can generate a monotonically improving sequence of policies. In other words, $\eta(\pi_0) \leq \eta(\pi_1) \leq \eta(\pi_2) \leq ....$ Let $M_i(\pi) = L_{\pi_i}(\pi) - CD_{KL}^{max}(\pi_i, \pi)$ denote the lower bound after the i-th iteration. Then

$$\begin{aligned} \eta(\pi_{i+1}) &\geq M_i(\pi_{i+1}) \\ \eta(\pi_i) &= M_i(\pi_i), \text{ therefore,} \\ \eta(\pi_{i+1}) - \eta(\pi_i) &\geq M_i(\pi_{i+1}) - M(\pi_i) \end{aligned} \tag{2.31}$$

Thus, by maximizing M_i we can guarantee that the true objective function η is non-decreasing at each iteration. The optimization problem can be written as follows:

$$\underset{\theta}{\text{maximize}}[L_{\theta_{old}}(\theta) - D_{KL}^{max}(\theta_{old}, \theta)] \tag{2.32}$$

However, equation (2.32) poses two challenges. First, finding the maximum KL divergence among all policies is intractable. Second, the penalty coefficient C would make the step sizes very small in practice. So, the optimization is rewritten using the average KL instead of the maximum and a trust region constraint on the KL divergence between the new and old policy is enforced. The new problem becomes

$$\begin{aligned} &\underset{\theta}{\text{maximize}} \ L_{\theta_{old}} \\ &\text{subject to } \bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta \, . \end{aligned} \tag{2.33}$$

Here $\bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) := \mathbb{E}_{s\sim\rho}[D_{KL}(\pi_{\theta_1}(.|s)||\pi_{\theta_2}(.|s))]$. Moreover, using the definition of L in equation (2.29) and importance sampling, the optimization can be written as:

$$\begin{aligned} &\underset{\theta}{\text{maximize}} \ \mathbb{E}_{s\sim\rho_{\theta_{old}}, a\sim q}\left[\frac{\pi_{\theta}(a|s_n)}{q(a|s_n)} A_{\theta_{old}}(s_n, a)\right] \\ &\text{subject to } \bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta \end{aligned} \tag{2.34}$$

Despite these modifications, the theoretical TRPO update remains challenging to implement and some approximations are needed. The constraint and objective can be Taylor expanded around θ, but even with this approximation there are still some computations that are computationally expensive and complex.

2.6.4 Proximal Policy Optimization

The Proximal Policy Optimization [104] (PPO) is motivated by the same idea behind TRPO: take the largest improvement step on a policy without causing any performance collapse. PPO is based on first-order methods that keep new policies close to old policies. It performs at least as well as TRPO empirically and is significantly simpler to implement.

One of the most significant differences between PPO and TRPO is the lack of the KL-divergence term or additional constraints. Instead, PPO uses a clipping operation in the objective function in order to keep the new policy not too far from the old. The policy weights θ are updated via:

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathbb{E}}_{s,a\sim\pi_{\theta_k}} \left[L(s, a, \theta_k, \theta) \right] , \tag{2.35}$$

where L is defined as

$$L(s, a, \theta_k, \theta) = \min \left(\psi(\theta) A^{\pi_{\theta_k}}(s, a), \text{clip} \left(\psi(\theta), 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) , \right) \tag{2.36}$$

and $\psi(\theta)$ is the probability ratio $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$.

The *clip* function is used to keep the ratio $\psi(\theta)$ within the range $[1 - \epsilon, 1 + \epsilon]$, i.e., $1 - \epsilon \leq \psi(\theta) \leq 1 + \epsilon$. The objective function of PPO takes the minimum between the original ratio and its clipped version. Hence, we avoid increasing the policy update to extremes for better rewards. The PPO has some of the benefits of trust region policy optimization (TRPO), while being much simpler to implement, more general, and having better sample complexity [104].

2.7 Model-Based and Model-Free RL

The process of learning and improving a policy from experience is a key aspect of **model-free** reinforcement learning (RL), as there is no explicit model of the environment. This is often useful in real-world applications, where the complete knowledge regarding the dynamics of the environment is impossible to obtain or very costly to compute.

Nonetheless, if the dynamics of the environment are known (or can be approximated relatively well), the agent's decisions can be based on a predictive model of this environment and the MDP can be solved using dynamic programming [85] or other methods. This framework is known as **model-based** RL and its approach to solving a task differs from conventional model-free algorithms.

Instead of using experience to learn a value function (or policy), model-based RL uses experience to learn a model of the environment. Then, using the learned model, the agent constructs a policy or value function [45]. A model $\mathcal{M}$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{R} \rangle$ that is parameterized by a function ϕ. Additionally, the state and action space are assumed to be known. The state transitions and reward of a model $\mathcal{M} = \langle \mathcal{P}_\phi, \mathcal{R}_\phi \rangle$

can be represented as:

$$s_{t+1} \sim \mathcal{P}_\phi(s_{t+1}|s_t, a_t) \tag{2.37}$$
$$R_{t+1} = \mathcal{R}_\phi(R_{t+1}|s_t, a_t) \tag{2.38}$$

The goal is to estimate $\mathcal{M}$ from experience $\{s_1, a_1, R_2, ...\}$. This estimation can be done using supervised learning, as we have input-output pairs. Given s_T, a_T we want to find s_{T+1}, R_{T+1}. Learning the mapping from state and action to reward is a regression problem, whereas learning the state transition probabilities is a density estimation problem. The parameters of ϕ are then found by minimizing an empirical loss function such as mean-squared error for regression or KL-divergence for density estimation.

Some advantages of model-based RL are that it can learn a model using supervised learning methods, account for model uncertainty and provide a greater sample efficiency when compared to model-free RL. However, since it has to learn a model then construct a policy, this framework suffers from two sources of approximation error. Moreover, it is often burdensome to approximate the environment.

2.8 Summary

Reinforcement learning is a machine learning paradigm that differs from supervised and unsupervised learning by its focus on learning from direct interaction with its environment. This is achieved using the Markov decision process framework, which uses states, actions and rewards to define the interaction between an agent and its environment.

We introduced in this chapter the concepts of policies and value functions. These are major elements of reinforcement learning methods, and they are used to define the behaviour of an agent. We also discussed two main variants of policy gradient methods.

Lastly, we provided a brief overview of different classes of RL. Policy-based methods learn a policy directly while value-based methods learn the policy indirectly by using value (or action-value) functions. We explained how RL algorithms can be divided into model-based and model-free methods, depending on the availability and usage of knowledge concerning the dynamics of the problem's environment.

Chapter 3

Financial Theory

The relationship between time and money is one of the most fundamental in finance and economics. The basic assumption is that an individual always prefers present consumption to future consumption. Therefore, these individuals require a greater quantity of consumption in the future if they are postponing their consumption in the present. This is the basic idea of interest and investments.

All applications in finance involve this notion of money and time. Consequently, most of the problems encountered are modelled as inference and operations on time series, i.e., manipulation and analysis of sequences of observations indexed by time.

This chapter presents a brief overview of financial theory, starting with basic financial concepts (Section 3.1) such as assets and returns. Next, we introduce the task of portfolio optimization (Section 3.2). Finally, in Section 3.3 we present the most commonly used metrics for evaluating a portfolio's performance.

3.1 Financial Concepts

This section presents some key definitions of financial terms that are used throughout the thesis.

3.1.1 Asset

An *asset* represents an item of economic value from which future economic benefits are expected to be obtained. Some examples of assets are stocks, bonds, gold, loans, and cash.

As a practical consideration, we assume that assets considered in this work are liquid. This means that assets can be bought and sold with no loss of value during the transaction. Moreover, we consider that historical data is available for usage and analysis.

3.1.2 Portfolio

A *portfolio* is a collection of multiple financial assets. It can include assets from different classes, such as stocks, bonds and other products. For a portfolio with N assets, we define its weights as:

$$\mathbf{w}_t = [w_{1,t},\ \dots\ , w_{N,t}]^T \in \mathbb{R}^N \quad \text{where} \quad \sum_{i=1}^{N} w_{i,t} = 1 \tag{3.1}$$

Portfolios can be considered as a more general representation of financial assets since a portfolio can also be used to represent an individual asset. For instance, an asset j can be represented as a portfolio using the weights $w_{j,t} = 1$ and $w_{i,t} = 0\ \forall t, i \neq j$.

3.1.3 Financial Time Series

The prices of financial series are dynamic due to the nature of the economy. The variations are a result of the supply and demand balance that keeps changing over time. Time series analysis and modelling together constitute a good framework for representing and inferring market dynamics.

Let $p_{i,t} \in \mathbb{R}$ be the price of asset i at time t. Hence, for a time interval of interest divided into T discrete smaller time intervals, the prices of asset i are:

$$\mathbf{p}_{i,0:T} = \begin{bmatrix} p_{i,0} \\ p_{i,1} \\ \vdots \\ p_{i,T} \end{bmatrix} . \tag{3.2}$$

Similarly, we can define the prices for N assets at time t as:

$$\mathbf{p}_t = \begin{bmatrix} p_{1,t} & p_{2,t} & \dots & p_{N,t} \end{bmatrix} . \tag{3.3}$$

The prices for all time steps and N assets can then be represented as a matrix of prices:

$$\mathbf{P} = \begin{bmatrix} p_{1,t} & p_{2,t} & \dots & p_{N,t} \\ \vdots & \dots & \ddots & \vdots \\ p_{1,T} & p_{2,T} & \dots & p_{N,T} \end{bmatrix} . \tag{3.4}$$

3.1.4 Returns

For investors, the prices of assets are generally not directly useful in the process of decision making (although there are exceptions). The prices are usually analyzed by considering their changes over time. In other words, the focus is on the *return* of a given asset.

The *gross return* can be used to derive a scaling factor that relates the current value after a given time interval to the amount originally invested. For example, a return of 10% over a month means that the current amount after one month is 10% higher than the initial amount

invested. Formally, the gross return is the ratio of the price at time t_0 and the price at time t_f:

$$R^{(g)}_{t_f,t_0} = \frac{p_{t_0}}{p_{t_f}}\,, \tag{3.5}$$

where t_0 and t_f are the times at the beginning and end of the considered period, respectively. Often the return is expressed in daily or monthly terms. If each time step represents one month, the monthly return for every t is given by:

$$R^{(g)}_t = \frac{p_t}{p_{t-1}}\,. \tag{3.6}$$

Despite its simplicity, the gross return is not widely used in practice. Instead, a slight variation is more common, known as the *simple return* $R^{(s)}_t$:

$$R^{(s)}_t = \frac{p_t - p_{t-1}}{p_{t-1}} = R^{(g)}_t - 1\,. \tag{3.7}$$

The main advantage of using the simple return is its interpretability. A positive value indicates that the asset is more valuable than before and vice-versa.

Let $R_{i,t}$ represent the return of asset i at time t. Similar to the matrix of prices, we can define the returns for all the assets in a portfolio $\forall t$ as:

$$\mathbf{R} = \begin{bmatrix} R^{(s)}_{1,t} & R^{(s)}_{2,t} & \dots & & R^{(s)}_{N,t} \\ \vdots & \dots & \ddots & \vdots & \\ R^{(s)}_{1,T} & R^{(s)}_{2,T} & \dots & & R^{(s)}_{N,T} \end{bmatrix} \in \mathbb{R}^{T\times N}\,, \tag{3.8}$$

where T is the length of the time period and N is the number of possible assets in the portfolio. As portfolios can be considered to be a more general representation of an asset, we can define the simple and gross return for a portfolio in similar fashion.

The return of the portfolio is simply the linear combination of its weights and the assets' returns for a given time step. Formally,

$$\mathcal{R}_t = \sum_{i=1}^{T} w_{i,t} R^{(s)}_{i,t} = \mathbf{R}_t \mathbf{w}_t\,. \tag{3.9}$$

where $\mathbf{R}_t$ is the t-th row of return matrix $\mathbf{R}$ and $\mathbf{w}_t$ the corresponding allocation for time step t.

The simple return offers an intuitive way of representing profit or loss in an investment. However, its asymmetry makes comparison of strategies harder to interpret. As an alternative, *logarithmic returns* (log returns) are often used due to several advantageous properties. As an illustration, consider the returns presented in Table 3.1. The simple return of 20% followed by -20% does not result in the same initial price. In fact, the original value is only reached after an additional 4% increase. On the other hand, it is easy to see that the log returns

Table 3.1: Illustration of the asymmetry of simple returns and symmetry of log return

Time	Price	Simple Return	Log Return
0	100	-	-
1	120	0.20	0.18
2	96	-0.20	-0.22
3	100	0.04	0.04

represent in a clear way that the price in time 2 is lower than in time 0. The log return at time t is defined as

$$r_t = \ln(R_t^{(g)}) = \ln\left(\frac{p_t}{p_{t-1}}\right) \tag{3.10}$$

where ln is the natural logarithm.

Logarithmic returns have a direct link with the compounding function, because the sum of successive returns can be interpreted as the continuously compounded return [54]. This property is advantageous when considering returns over several time periods as the cumulative log return is simply the sum of the individual log returns. Continuously compounded returns are time additive and it is easier to derive the time series properties of additive processes than multiplicative processes.

Another advantage of using log returns is that asset prices are often modeled via Brownian motion [23, 36], which makes the logarithmic returns of the security normally distributed. Additionally, for small magnitudes, logarithmic returns and simple returns have approximately the same value.

We can derive the return of the portfolio in log terms equation (3.9)

$$r_t^{log} = \ln(1 + \mathbf{R}_t \mathbf{w}_t) \tag{3.11}$$

Figure 3.1 shows how both simple and log returns are centered around zero. However, despite the shape of their distribution, they are not normal. External events and crises frequently provoke returns that are multiple standard deviations away from the mean. Such returns would be extremely unlikely if normal distributions were accurate models.

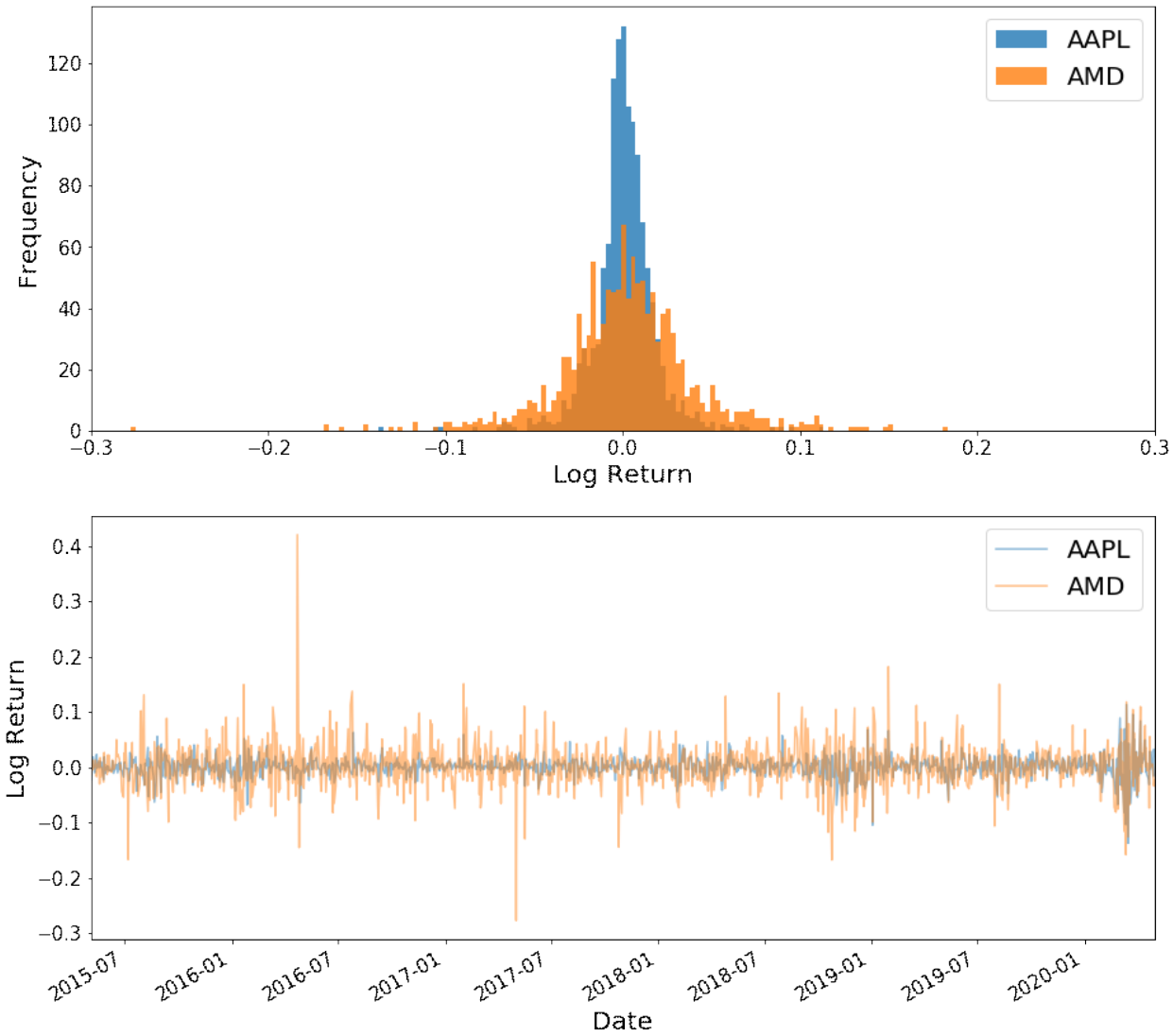

Figure 3.1: Simple and log (daily) returns of two example stocks – Apple and AMD over the time period July 2015 - Mar. 2020.

3.2 Portfolio Optimization

In the investment industry, dynamic portfolio optimization is one of the problems most frequently encountered [35]. It can be defined as the process of selecting the best asset distribution (portfolio) out of the set of all other asset combinations being considered, according to some objective. Typically, the aim is to maximize some desirable factors such as expected return while minimizing others such as risk.

Recall that a portfolio of N assets at time t is defined as:

$$\mathbf{w}_t = [w_{1,t},\ \dots\ , w_{N,t}]^T \in \mathbb{R}^m \quad \text{for} \quad \sum_{i=1}^{M} w_{i,t} = 1\,, \tag{3.12}$$

where $w_{i,t}$ denotes the percentage allocated on the asset i at time t. Our goal is to find an optimal allocation $\mathbf{W}^*$, which denotes for each time step what the allocation $\mathbf{w_t}$ should be such that a cumulative performance function is maximized (or minimized). The main goal of using portfolios is combining assets with different properties in order to mitigate some negative aspects while amplifying positive aspects of the market.

3.2.1 Modern Portfolio Theory

Markowitz's Modern Portfolio Theory (MPT) [83] is the dominant paradigm in portfolio optimization. It consists of computing the *Mean-Variance Efficient Frontier (EF)*. This is defined as the selections of investments that yields the highest return with respect to the risk-free rate for any given level of risk measured in terms of standard deviation.

Specifically, it considers a universe of N investments with returns r, empirical estimates of the expected returns $\boldsymbol{\mu} = [\mu_1,\ \dots\ , \mu_N]^T \in \mathbb{R}^N$, standard deviation σ, covariance matrix $\boldsymbol{\Sigma} = [\sigma_{ij}]$ (where $\sigma_{ii} = \sigma_i^2$ and $\sigma_{ij} = \rho\sigma_i\sigma_j$ for i $\neq$ j), and the portfolio weight vector $\mathbf{w} \in \mathbb{R}^N$. The objective is to minimize the risk (variance) of the portfolio, subject to the constraints that the weights of the portfolio sum up to one and the portfolio's return at least achieves the target return. Formally, the problem can be specified as:

$$\min_{\mathbf{w}} \qquad \frac{1}{2}\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w} \tag{3.13a}$$

$$\text{subject to } \mathbf{w}^T\mathbf{1}_N = 1\,, \tag{3.13b}$$

$$\mathbf{w}^T\boldsymbol{\mu} \geq r_p\,, \tag{3.13c}$$

where $\mathbf{1}_N$ is a column vector of ones and r_p is the target return.

The solution to the portfolio optimization is obtained by minimizing the Lagrange function with respect to $\mathbf{w}$. The Lagrange function is:

$$L(\mathbf{w}, \gamma, \lambda) = \frac{1}{2}\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w} - \gamma(\mathbf{w}^T\mathbf{1}_N - 1) - \lambda(\mathbf{w}^T\boldsymbol{\mu} - r_p)\,, \tag{3.14}$$

where γ and λ are scalar multipliers.

The first order conditions for minimizing $L(\mathbf{w}, \gamma, \lambda)$ are the following linear equations:

$$\frac{\partial L(\mathbf{w}, \gamma, \lambda)}{\mathbf{w}} = \boldsymbol{\Sigma}\mathbf{w} + \gamma\mathbf{1}_N + \lambda\boldsymbol{\mu} = 0\,, \tag{3.15}$$

$$\frac{\partial L(\mathbf{w}, \gamma, \lambda)}{\lambda} = \mathbf{w}^T\boldsymbol{\mu} - r_p = 0\,, \tag{3.16}$$

$$\frac{\partial L(\mathbf{w}, \gamma, \lambda)}{\gamma} = \mathbf{w}^T\mathbf{1}_N - 1 = 0\,, \tag{3.17}$$

or, in matrix form:

$$\begin{bmatrix} \boldsymbol{\Sigma} & \boldsymbol{\mu} & \mathbf{1}_N \\ \boldsymbol{\mu}^T & 0 & 0 \\ \mathbf{1}_N^T & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \lambda \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ r_p \\ 1 \end{bmatrix} . \tag{3.18}$$

The portfolio weights are determined by solving the linear system in (3.18). The solution is the portfolio with minimum variance with expected return r_p.

There are other variations for constructing the optimal portfolio. One alternative is to maximize return while providing an upper-bound for the variance (risk):

$$\max_{\mathbf{w}} \quad \mathbf{w}^T \boldsymbol{\mu} \,, \tag{3.19a}$$

$$\text{subject to } \mathbf{w}^T \mathbf{1}_N = 1 \,, \tag{3.19b}$$

$$\mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w} \leq \sigma_p^2 \,, \tag{3.19c}$$

where σ_p is the maximum allowable *volatility* (standard deviation of the portfolio returns).

Additionally, we can define a trade-off function between risk and return that can be used to find optimal portfolios based on a coefficient ϕ that captures risk aversion:

$$\min_{\mathbf{w}} \quad \phi \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w} - \mathbf{w}^T \boldsymbol{\mu} \tag{3.20a}$$

$$\text{subject to } \mathbf{w}^T \mathbf{1}_N = 1 \,. \tag{3.20b}$$

Larger values of ϕ result in a portfolio with a higher sensitivity to volatility.

Finally, we can also consider as an objective function the ratio of expected returns and their standard deviation [1]. In this case, the optimization problem becomes:

$$\max_{\mathbf{w}} \quad \frac{\mathbf{w}^T \boldsymbol{\mu}}{\sqrt{\mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w}}} \tag{3.21a}$$

$$\text{subject to } \mathbf{w}^T \mathbf{1}_N = 1 \,. \tag{3.21b}$$

Figure 3.2 presents an example of the mean-variance portfolio. Ten assets were randomly selected from the S&P500 index [2]. Their returns and volatility over the considered period can be seen in the figure. Note how the optimal allocation, selected by using the mean-variance approach, offers the best achievable return for a specified allowable risk (volatility).

1. This quantity is known as the Sharpe Ratio, and it will be discussed in further detail in the following sections.

2. The S&P 500 is a stock market index that measures the stock performance of 500 large companies listed on stock exchanges in the United States.

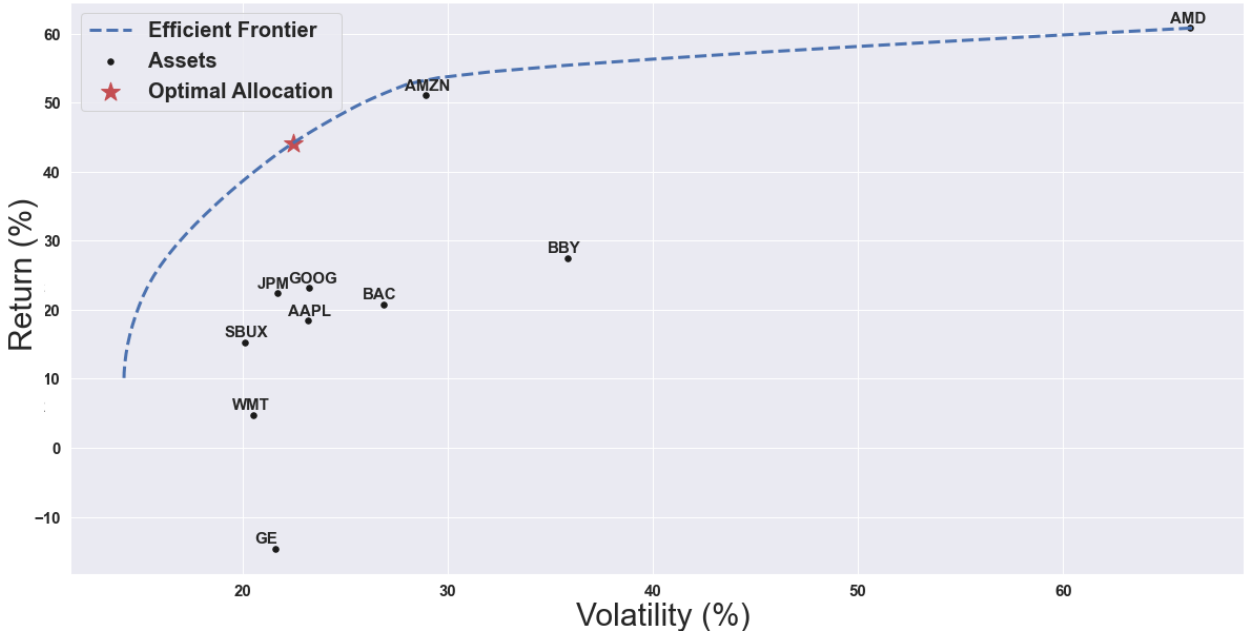

Figure 3.2: Efficient Frontier of a portfolio with 10 assets

The selected assets and their weights in the mean-variance portfolio are presented in Table 3.2. Note that the portfolio is not composed solely of the best performing asset (in terms of return), but consists of a combination of different assets in order to maximize the return while keeping the volatility relatively low.

Table 3.2: Hypothetical portfolio composed of ten stocks using mean-variance optimization.

Ticker	Company Name	Return (%)	Volatility(%)	Weight in portfolio
AAPL	Apple	23.19%	18.44%	12.5%
AMD	Advanced Micro Devices	60.79%	66.20%	0%
AMZN	Amazon	51.03%	28.92%	48.0%
BAC	Bank of America	20.77%	28.86%	0%
BBY	Best Buy	27.48%	35.84%	17.1%
GE	General Electric	-14.58%	21.57%	0%
GOOG	Google	23.17%	23.22%	0%
JPM	JP Morgan	22.36%	21.70%	8%
SBUX	Starbucks	15.29%	20.10%	0%
WMT	Walmart	4.78%	20.50%	14.2%

Despite having higher return and lower volatility, the portfolio's weight of AAPL is lower than WMT. This is a consequence of how different assets relate to each other. If two assets are highly correlated, their price movements tend to behave similarly. From an investor's point of view, this makes an investment riskier, as both assets have the same price direction. The mean variance portfolio tries to incorporate this by using the covariance of the assets.

The correlation matrix is shown in Figure 3.3. Note how the correlation of WMT with other assets is, on average, lower than its counterparts, which makes it an attractive alternative for diversification in order to reduce the risk of the portfolio.

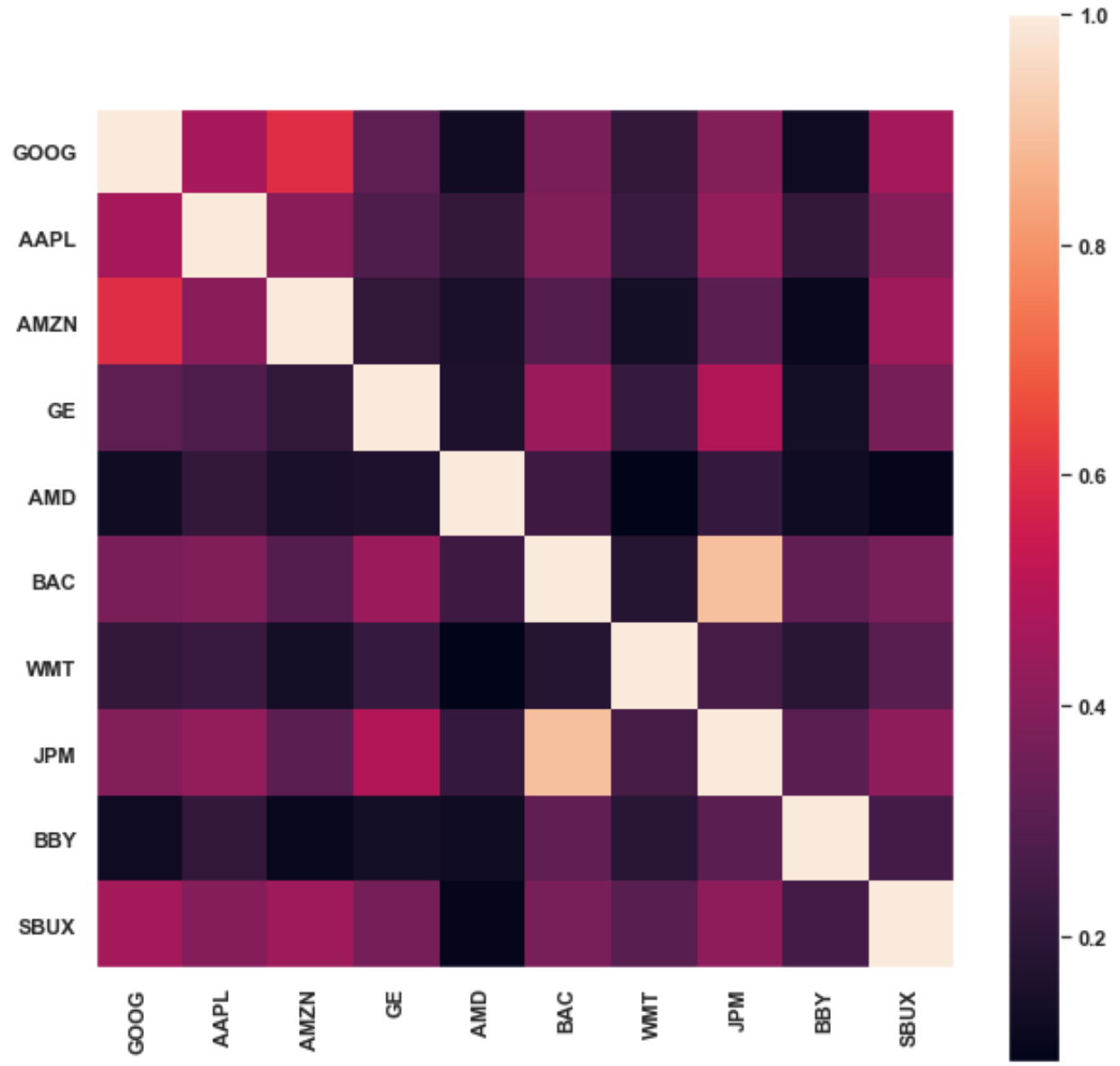

Figure 3.3: Correlation matrix of the 10 selected assets

The mean-variance analysis is a very commonly used technique for optimizing a portfolio. Its simplicity is one of the key strengths of this framework, as it just requires the means, variances, and the covariance of asset returns for finding the trade-off between return and risk. However, this model has several shortcomings [65].

The first limitation is the usage of variance as a metric for risk. Due to the nature of the

variance calculation, both positive and negative returns have the same effect. Nevertheless, large positive returns are usually not considered as a risk for the investor. For example, consider the returns of two hypothetical portfolios, as presented in Table 3.3.

Table 3.3: Quarterly returns of two portfolios

	First Quarter	Second Quarter	Third Quarter	Last Quarter
Portfolio 1	5.0%	1.5%	3.0%	2.5%
Portfolio 2	10.0%	0.5%	0.3%	12.5%

Both portfolios had positive returns in all quarters, but from a investor's point of view, the portfolios are quite different. Portfolio 2 is much more attractive, as the average return is higher and there are no severe losses. However, this portfolio might not be selected in a Markowitz framework as it presents high variability, despite its higher return.

One of the reasons that variance is used as a risk metric is the assumption that returns are normally distributed. If that was the case, this metric would be more appropriate as the normal distribution is symmetric. It is known that historical equity returns are not normally distributed. The empirical distribtuions tend to have fat-tails, especially on the negative side.

The second limitation is the challenge of estimating returns. These values have to be estimated (forecast) if we want to use this framework to derive the optimal allocations for a future period. Consequently, the model is affected by estimation errors, and the optimal allocations for the true and estimated returns can show considerable disparity.

With respect to variance as a risk estimator, there are some alternatives that can be used instead of the variance. One of them is the *semivariance*, which only considers deviation in one direction. The semivariance for a target return of 0% only considers the deviation of negative returns and is defined as:

$$Semi_{\sigma^2} = \frac{1}{T}\sum_{t=1}^{T} \min(0, r_t)^2 \,, \tag{3.22}$$

where r_t is the return of the asset on time t and min is the minimum between 0 and r_t.

Another alternative is using a probabilistic approach to model risk. One of the most commonly used metrics is known as *value-at-risk* (VaR). It measures the probability of losing a certain amount over a time interval. For instance, if the 5% VaR during a year is 10%, it is equivalent to saying that the probability of losing more than 10% is less than 5%. In other words, there is a 95% probability that the loss will be lower than 10%. The VaR can be estimated from historical data; its estimation does not require any assumptions about the distribution of returns.

3.2.2 Black-Litterman Model

A more sophisticated alternative to the mean-variance framework is the Black-Litterman model [16]. It uses a Bayesian approach to combine particular views of an investor regarding

the expected returns of one or more assets with a prior distribution (the market equilibrium expected return) to form a new estimate of expected returns.

This model is particularly interesting because most portfolio managers have expertise in a specific industry, sector or class of assets. Therefore, they may have a more accurate view of these portions of the market when compared to all the other market components. The Black-Litterman model can take into account these subjective views to determine the asset allocation.

The equilibrium state of the Black-Litterman model is based on the capital asset pricing model (CAPM) [37]. According to the CAPM, the excess return of an asset can be expressed as:

$$E(r_i) = r_f + \beta_i(E(r_M) - r_f)\,, \tag{3.23}$$

where r_i, r_M and r_f are the returns of the asset i, the market and the risk-free asset, respectively. The parameter β_i is the covariance between the returns of the asset considered and the market divided by the variance of market returns. Furthermore, let $w_m = (w_{m,1}, ..., w_{m,N})$ be the weights of each asset that form the market return with N assets. The expected excess returns of asset i can be written as

$$\Pi_i = E(r_i) - r_f \equiv \beta_i(E(r_M) - r_f)\,. \tag{3.24}$$

Using the definition of β_i, we have:

$$\Pi_i = \frac{cov(r_i, r_M)}{\sigma_M^2}(E(r_M) - r_f)\,. \tag{3.25}$$

Finally, by using the fact that the return of the market can be expressed as $r_M = \sum_{k=1}^{N} w_{M,k} r_k$, we can rearrange the terms as follows:

$$\Pi_i = \frac{(E(r_M) - r_f)}{\sigma_M^2} \sum_{k=1}^{N} cov(r_i, r_M) w_{M,k} \tag{3.26}$$

Let $\delta = \frac{(E(r_M)-r_f)}{\sigma_M^2}$ be defined as the risk premium of the market over its variance. Then the expected excess returns can then be written in matrix-form as

$$\mathbf{\Pi} = \delta \mathbf{\Sigma} w_M,$$

where $\mathbf{\Pi} \in \mathbb{R}^N$, $\mathbf{\Sigma} \in \mathbb{R}^{N\ x\ N}$ are defined as

$$\mathbf{\Sigma} = \begin{bmatrix} cov(r_1, r_1) & \dots & cov(r_1, r_N) \\ \vdots & \ddots & \vdots \\ cov(r_N, r_1) & \dots & cov(r_N, r_N) \end{bmatrix}, \qquad \mathbf{\Pi} = \begin{bmatrix} \Pi_1 \\ \vdots \\ \Pi_N \end{bmatrix}. \tag{3.27}$$

Black and Litterman assumed that the market is expected to be in the equilibrium state on average but may deviate from it depending on other market conditions. This deviation

can be modelled as normal noise $\epsilon_\Pi \in \mathbb{R}^N$ with mean zero and covariance matrix $\mathbf{\Sigma}$. The prior distribution of the returns can be expressed as

$$\boldsymbol{\mu} \sim \mathcal{N}(\Pi, \tau\mathbf{\Sigma}), \tag{3.28}$$

where $\tau \in (0, 1]$ represents the confidence on the estimate of excess return. In other words, it reflects the investor's belief in the CAPM model. Small values of τ represent high confidence in the estimate of returns.

The Black-Litterman model is able to incorporate investors' views regarding the market equilibrium. There are two type of views that can be incorporated. The first is absolute view. For example, if one portfolio manager believes that "Asset A" will be have a return of 15% over the next year, it is possible to incorporate this view in absolute terms. The other view concerns performance relative to another asset. For example, we can incorporate a view that "Asset B" will outperform "Asset C" by 10% without knowing the exact returns of either asset, but only the difference in performance.

These views are integrated by two components in the model. Let K represent the total number of views, which accounts for both absolute and relative views. The first component is a vector $q \in \mathbb{R}^K$ with the expected returns. In our example, this would be a vector $q = [0.15\ 0.10]^T$. The second component is a matrix $\mathbf{P} \in \mathbb{R}^{K\times N}$, where N is the number of assets considered. This matrix indicates how each asset is related to a given view, where each row represents a different view. The entry (k, i) represents how the i-th asset is relevant to the k-th view. If a particular asset is irrelevant to a given view, its entry is zero. The matrix $\mathbf{P}$, along with the vector q, for the example are illustrated below.

$$\begin{bmatrix} 0.15 \\ 0.10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \mu_A \\ \mu_B \\ \mu_C \end{bmatrix} \tag{3.29}$$

Similarly to predicted returns, it is also possible to incorporate uncertainty in the views $\epsilon_q \in \mathbb{R}^K$. The expression then becomes:

$$\mathbf{Q} = \mathbf{P}\boldsymbol{\mu} + \epsilon_q\,. \tag{3.30}$$

One of the assumptions is that the deviations (errors) from the views are independent of each other, which will result in a diagonal covariance matrix $\mathbf{\Omega} \in \mathbb{R}^{K\times K}$. The uncertainty is incorporated by setting the covariance $\mathbf{\Omega}$ of the distribution $\epsilon_q \sim \mathcal{N}(0, \mathbf{\Omega})$. This allows the investor to add different confidence levels to different views.

There are different ways to incorporate the views [102] and arrive at the expected return from the model

$$\hat{\boldsymbol{\mu}}_{BL} = [(\tau\mathbf{\Sigma})^{-1} + \mathbf{P}^T\mathbf{\Omega}^{-1}\mathbf{P}]^{-1}[(\tau\mathbf{\Sigma})^{-1}\mathbf{\Pi} + \mathbf{P}^T\mathbf{\Omega}^{-1}\mathbf{Q}]\,. \tag{3.31}$$

The Bayesian interpretation is that the posterior distribution of the expected returns is a normal distribution with parameters:

$$\mathcal{N}([(\tau\mathbf{\Sigma})^{-1} + \mathbf{P}^T\mathbf{\Omega}^{-1}\mathbf{P}]^{-1}[(\tau\mathbf{\Sigma})^{-1}\mathbf{\Pi} + \mathbf{P}^T\mathbf{\Omega}^{-1}\mathbf{Q}], [(\tau\mathbf{\Sigma})^{-1} + \mathbf{P}^T\mathbf{\Omega}^{-1}\mathbf{P}]^{-1}) \tag{3.32}$$

The Black-Litterman model is illustrated in Figure 3.4. The final distribution is a combination of the prior equilibrium distribution and the view distribution. The weights can then be found using classical mean-variance optimization. If both q and $\mathbf{\Omega}$ are zero, i.e., the investor has no views, the expected return from the model is simply $\mathbf{\Pi}$. However, in the case where the investor incorporates views into the model, the robustness of the model can increase.

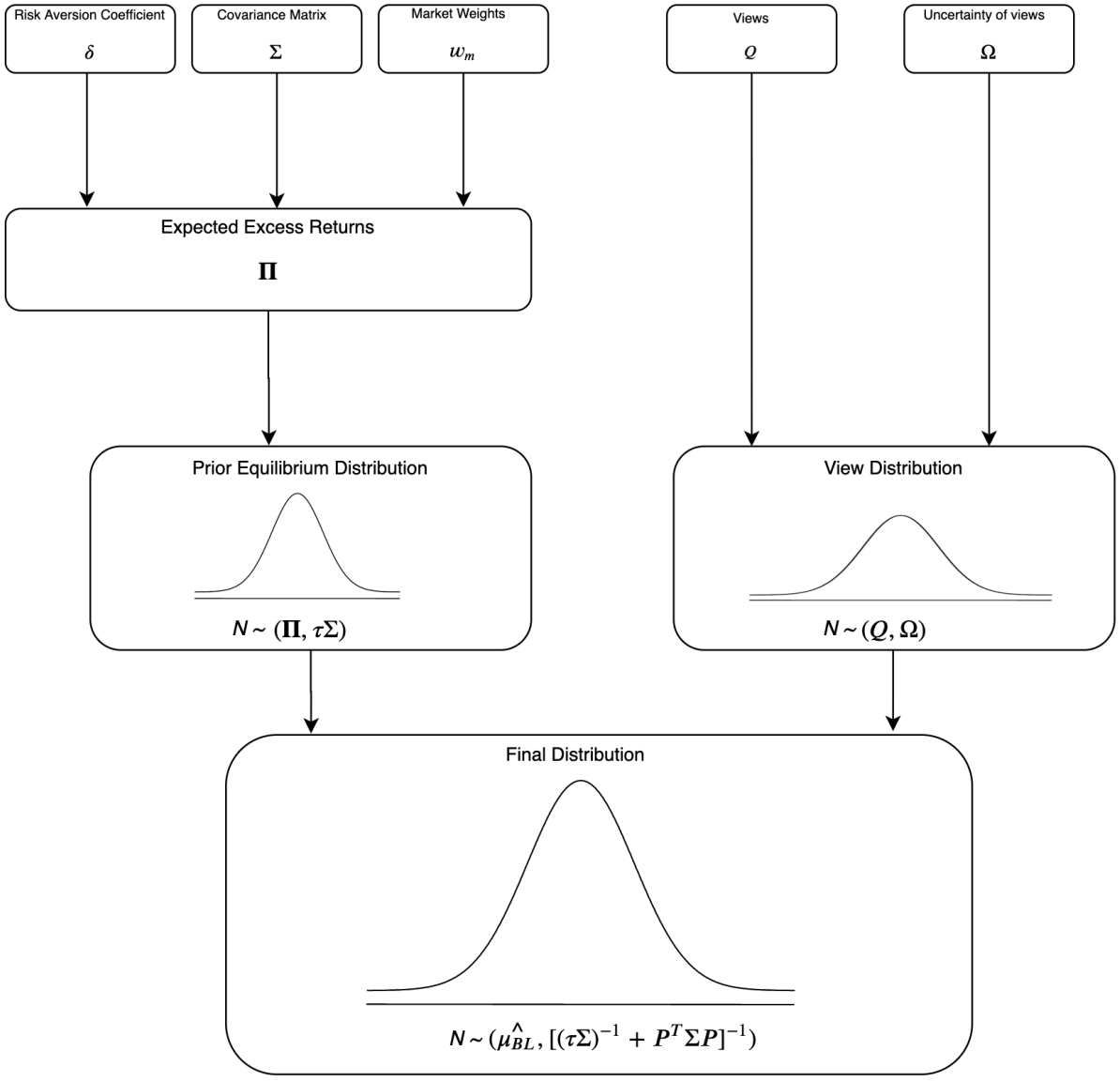

Figure 3.4: Schematic of Black-Litterman Portfolio

Estimation of Extra Parameters

When compared to the standard mean-variance portfolio, the Black-Litterman (BL) model requires some extra input parameters, namely τ, which reflects the uncertainty of implied

returns, the subjective views determined by $\mathbf{Q}$ and their uncertainty, defined by $\mathbf{\Omega}$. The BL model combines implied returns and subjective return estimates to create combined returns estimates.

The parameter τ controls how distinctly the optimized portfolio may differ from the reference portfolio. There are some works dedicated entirely to discussing this parameter [2, 117]. For values close to zero, the combined returns converge to implied returns and the BL optimized portfolio converges to the reference portfolio. On the other hand, for large values of τ, the BL portfolio converges to a mean-variance portfolio with the subjective views as return estimates. According to Bessler et al., the values used for τ typically range between 0.025 and 0.300. In our experiments in this thesis, we choose a default value for τ of 0.05.

In terms of subjective return estimates and their reliability, there is no clear answer in the literature on how to derive them. Numerous studies simply assume that estimates are provided by an external source or specialist [51, 56] and their uncertainty can be represented using confidence intervals [16]. However, these approaches make the performance of the BL model hard to compare to other methods, such as the MV portfolio. As an alternative, the sample means of returns can be used as the subjective return estimates [13].

Finally, to estimate the confidence on these estimates, expressed in terms of the matrix $\mathbf{\Omega}$, we use the approach proposed by [84], which assumes that the reliability of views is proportional to the variance of the priors:

$$\mathbf{\Omega} = \mathbf{P}\mathbf{\Sigma}\mathbf{P}^T \frac{1}{c}, \tag{3.33}$$

where $1/c$ is the proportionality factor, usually a small value close to zero.

3.3 Performance Criteria

There are several ways to evaluate an allocation for a given investment horizon. For example, an investor could focus on maximizing the final wealth or surpassing a benchmark index while minimizing the volatility.

Despite some limitations of the mean-variance framework, some of its main ideas are often used by investors. The notion of risk and reward is very important. For the same risk, the investor wants to maximize the expected return. In the same vein, for the same expected return, the investor wants to minimize the volatility. Some evaluation metrics that take into account these aspects are introduced in the following sections.

3.3.1 Cumulative Returns

The simple return is defined as the ratio of the price at the end of the period and the price at the beginning of the period. However, we often want to evaluate a strategy over a longer time horizon. Hence, the definition of return has to incorporate several time steps in order to represent the final performance of the investment. This is achieved by compounding the

returns over time. Formally, we define the cumulative simple return from time 1 to time T as

$$cR_T = \frac{p_T}{p_1} - 1 = \prod_{t=1}^{T}(R_t^{(g)}) - 1 = \left[\prod_{t=1}^{T}(R_t + 1) - 1\right] \tag{3.34}$$

Similarly, we can define the cumulative log return as

$$cr_T = ln(\frac{p_T}{p_1}) = \ln\left(\prod_{t=1}^{T}(R_t^{(g)})\right) = \sum_{t=1}^{T} r_t \tag{3.35}$$

The equation above shows one of the advantages of using the log returns, as we have a sum of returns instead of the product over all previous time steps.

3.3.2 Sharpe Ratio

The idea of selecting the portfolio with the highest expected return compared to others with the same risk or choosing the portfolio with lowest volatility compared to others with the same expected return is intuitive. However, in most cases the risk and reward for each portfolio will be different and selecting the most attractive portfolio might not be straightforward.

The Sharpe Ratio [105] addresses this limitation, by incorporating both risk (standard deviation) and reward (cumulative return). It is defined as:

$$S_T = \sqrt{T}\frac{\mathbb{E}(\mathcal{R}_t)}{\sqrt{Var(\mathcal{R}_t)}}\,, \tag{3.36}$$

where T is the length of the time period, $\mathbb{E}$ is the expected value operator and Var is the variance, both over the same time period T.

By using the Sharpe ratio, a risk-adjusted return metric, one can compare portfolios with different characteristics. It is important to notice that the risk in the Sharpe ratio is defined as the standard deviation of returns, which presents several limitations, as discussed above.

3.3.3 Drawdown

As an alternative to measure risk, the drawdown (DD) is often used as a risk indicator in the investment industry [44]. The drawdown measures the decline of returns compared to its historical peak and it was first introduced in the context of Brownian motion [114]. The drawdown can be defined as:

$$DD_t = -\max_{u\in[0,t]}(cR_u) - cR_t\,, \tag{3.37}$$

where max is the maximum operator and cR_t the cumulative return. It is often useful to represent the drawdown in percentage terms by dividing DD_t by $\max_{u\in[0,t]}(cR_u)$.

Another way to incorporate the drawdown as a risk metric is to consider how long it takes, on average, to recover from a drawdown, i.e., to reach the level of cumulative return of the last peak. This is known as the average drawdown time.

In the field of active portfolio management, the reduction of drawdown has received considerable attention as a risk metric [20, 46]. In practice, however, investors are usually concerned with the maximum drawdown (MDD) a portfolio can suffer over the return horizon T, which is defined as:

$$MDD_T = -\max_{t \in [0,T]}(DD_t) \tag{3.38}$$

This means that the maximum drawdown is represented by a single value in a given return horizon T. Nonetheless, it is important to consider the distribution of the maximum drawdowns. By examining it, one can form reasonable expectations about the risks of a given portfolio over a given investment horizon.

3.3.4 Calmar Ratio

The Calmar Ratio ($CalR_t$) is similar to Sharpe ration, but instead of using the standard deviation to reflect risk, it uses uses maximum drawdown [5].

$$CalR_t = \frac{\mathbb{E}(\mathcal{R}_t)}{MDD_t} \tag{3.39}$$

3.3.5 Hit Ratio

The Hit Ratio represents the proportion of simple returns from a portfolio $\mathcal{R}_t$ that were higher than a benchmark return br_t over a time period T. Formally, we can define the hit ratio as:

$$HR_T = \frac{\sum_{t=0}^{T} \mathbf{1}(\mathcal{R}_t \geq br_t)}{T} \tag{3.40}$$

where $\mathbf{1}$ is the indicator function and T the length of the period considered.

Most of the time, investors want to consider additional factors other than just performance when evaluating portfolios. The goal is to understand the risk exposure of a specific strategy. In some cases, the risk exposure might not compensate the possibility of higher returns. For this reason, risk-adjusted metrics offer an attractive way to evaluate portfolios.

3.4 Summary

In this chapter, we introduce key definitions of some financial concepts that are used by investors and financial industry. Moreover, we introduce the modern portfolio theory, also known as mean-variance or Markowitz portfolio optimization, which was the first mathematical framework introduced to optimize a portfolio with respect to returns and risks.

Some improvements of this framework were later proposed, one example is the Black-Litterman portfolio, also introduced in the chapter. Additionaly, we present some of the most used metrics to evaluate the performance of a portfolio.

Chapter 4

Literature Review

The reinforcement learning framework has several inherent advantages for financial applications when compared to other machine learning paradigms. In particular, it facilitates the combination, in one integrated procedure, of the portfolio creation process with a performance metric that is closely related to an investor's objective.

In Section 4.1 we present a thorough literature review of reinforcement learning applications in finance. More specifically, we focus on the task of asset allocation, starting with works that use value-based algorithms, and then proceeding to survey policy-based techniques. Finally, we conclude the chapter in Section 4.2 with some key observations based on the literature review.

4.1 Reinforcement Learning Applications in Finance

In the next subsections, we present some of the most relevant works that have applied the reinforcement learning framework in finance. The key idea in most of the papers is to treat the asset allocation problem as an MDP. We will divide the works into the categories of value-based algorithms, policy-based algorithms and other variants.

4.1.1 Value-based Algorithms

The application of reinforcement learning to portfolio management was first introduced in [93]. Neuneier presented an approach in which an agent traded into two strategies: choosing at each time step between currency pairs (U.S. dollar or Deutsche Mark [1]) and choosing between a risky asset (DAX Index [2]) or staying in a risk free asset (Deutsche Mark). The agent decided in which asset to invest based on its action-value function. More specifically, it was based on the Q-Learning algorithm [119]. The key idea of this algorithm is to maximize the action-value function q in order to optimize a policy, as in Eq. (2.11). The action-value

1. German currency used before the creation of Euro
2. The DAX is a stock market index consisting of the 30 major German companies trading on the Frankfurt Stock Exchange

function is updated according to the following expression:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \,. \tag{4.1}$$

Neuneier used DAX prices and other market variables to model the state space and the reward was modelled as the return obtained after discounting transaction costs. This model outperformed a neural network benchmark, which was trained to predict whether the DAX would yield a positive return on the following day.

This approach was later improved by considering an intermediate deterministic step after each change in allocations [94]. This step takes into account transaction costs and reduces the overall number of transactions at each time step. In other words, the agent holds its position for longer periods. One key advantage of using RL is the flexibility of the reward function which can be conditioned on states, actions and incorporate risk aversion of the investor. For instance, Neuneier enhanced his previous model by incorporating risk aversion in the reward function. More precisely, the agent used the log return of the portfolio as reward function $R_t = \log(\frac{P_t}{P_{t-1}})$, where P_t is the portfolio value at time t. Due to the asymmetry of simple returns, log returns offer a more compelling reward function.

Similarly to [94], [40] trades the same currency pair and uses the same definition of state and immediate reward. It proposes a hybrid model that combines two approaches: an algorithm based on Q-Learning that maximizes the return and strives to maximize the Sharpe Ratio, using the same approach as [21]. A neural network is used as the function approximator for the state space. The model surpasses all the benchmarks proposed: the model defined in [93], a trading system based on supervised learning that optmizes the Sharpe ratio, and a trading system based on return forecasts. One of the limitations of this work is that the system only deals with a discrete action space.

In an attempt to simulate a human investor's behavior, Lee et al. [73] defined a multi-agent Q-learning (MQ) model with four agents. It divides the task of stock trading into two parts: timing, which requires two agents (buy and sell signals), and pricing, which requires another two agents (buy and sell orders). The authors argue that the investor usually considers different aspects when buying and selling. Thus, the separation is necessary to allow the agents to have different state representations.

Instead of using the raw price data as state representation, the signal agents use moving averages to represent the long-term price movements. The buy signal agent identifies buying opportunities while the sell signal agent observes the development of the buy orders and decides when to sell them. The order agents use information from the Japanese candlestick representation to define the state space. A candlestick is made up of four prices: the open, high, low and closing prices during a specific time interval. The order agents are triggered by the signal agents and their goal is to determine the buy and sell prices.

The proposed model outperforms the benchmarks, which include a supervised learning model and some simpler variants of MQ. However, the model has high computational costs as four different agents must be trained and it takes 5,000,000 episodes to converge to an optimum policy. Another limitation is the fact that the model only supports single-asset trading.

The Q-learning algorithm is also used by Bertoluzzo and Corazza in [12], but they also evaluate a different algorithm, Kernel-based reinforcement learning (KbRL) [95]. In order to specify KbRL, we first define $S^a = \{(s_k^a, r_k^a, \hat{s}_k^a,)|k = 1, ..., t-1\}$ as a collection of historical transitions where action a was taken, with $a \in \mathcal{A}$, $s \in \mathcal{S}$ and $r_k^a \in \mathbb{R}$. Let $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ be a Lipschitz continuous function satisfying $\int_0^1 \phi(x)d(x) = 1$. Let $k_\tau(s, s')$ be a kernel function defined as:

$$k_\tau(s, s') = \phi\left(\frac{||s - s'||}{\tau}\right), \tag{4.2}$$

where $\tau \in \mathbb{R}$ is the kernel's width. The normalized kernel function associated with action a can also be defined as:

$$k_\tau^a(s, s_i^a) = \frac{k_\tau(s, s_i^a)}{\sum_{j=1}^{n_a} k_\tau(s, s_j^a)}, . \tag{4.3}$$

The action-value function can be determined as:

$$\hat{Q}(s_t, a_t) \leftarrow \sum_{i=1}^{n_a} k_\tau^a(s, s_i^a)\left[r_i^a + \gamma \hat{V}(s_i^a)\right]. \tag{4.4}$$

Ormoneit and Sen have shown in [95] that, if $n_a \to \infty$ for all $a \in \mathcal{A}$ and the kernel's width τ decreases at an "admissible" rate, the probability of choosing a suboptimal action based on $\hat{Q}(s, a)$ converges to zero. Therefore, the KbRL has strong theoretical guarantees. Nevertheless, the model constructed by it grows with the number of sample transitions, resulting in high computational costs [7].

Bertoluzzo and Corazza used the Sharpe ratio calculated in the last L (5 and 22) days as reward function. The authors claim that the Q-learning performed better than kernel-based reinforcement learning, but the results vary across the 1000 performed replications of artificial data and they only performed tests on a single real world stock. Therefore, it is still unclear if the result holds for other assets.

All papers described until this point either focus on the trading of one currency pair or deciding whether or not to allocate on a risky asset (as opposed to allocating on a risk-free asset). The number of trading assets is expanded in [58], where a RL-based asset allocation strategy is proposed. Similar to [73], the framework is composed of multiple parts. The first part is called *local traders*, which is implemented by a neural network that is trained to forecast patterns and identify trading opportunities based on threshold values. The focus in [58] is on four meaningful patterns, which are derived from moving averages of prices. For each pattern there is a local trader that consists of a predictor and a local policy. Predictors estimate the future price of shares, and local policies specify how to use the predicted values given by its predictor.

The second part uses Q-learning to distribute the investments among the local traders. The states are modelled by the output of each local trader as well as the available cash. The actions represent the amount of money (discretized to four different values) that each local trader can invest per trade. It is important to notice that the action space grows exponentially with respect to the number of traders. Also, the performance of this system depends on the

forecasts of future prices, which is a challenging task, making this model hard to use as a benchmark for systems that do not rely on explict forecasts.

Tan et al. [113] also expand the number of assets, using 5 stocks from the S&P index. However, the proposed system is always fully invested in the chosen stock, i.e., it only holds one asset at a time. An Adaptive Network Fuzzy Inference System (ANFIS) [57] is used as a trading agent. RL is employed in two different ways. The first application is related to determining optimal parameters for the cycle identification (period of moving averages) component that is used to process the price information and to tune the hyperparameters of the ANFIS component. The second application of RL is related to the actual trading. It uses as the state the output from the ANFIS system and historical components.

The backtesting results of [113] are encouraging as the proposed model outperforms the market by 50% over 13 years of testing. However, more rigorous evaluation is required, as only five stocks were used.

In general, value-based methods suffer from the curse of dimensionality of the action space. The usage of Q-Learning when there are a finite number of discrete action is straightforward since the *max* operation poses no problem. However, this is usually not the case for asset allocation problems since the action space for a portfolio of N assets is defined by the weight vector $\in \mathbb{R}^N$. Using optimization algorithms to solve $\max_a Q_k(s_t, a_t)$ for each time step is infeasible.

However, more recently, [98] proposes a portfolio trading strategy using Deep Q-Learning (DQL) [87]. The authors argue that action spaces that are directly related to the portfolio weights, i.e., $a_t = \mathbf{w}_t$, do not provide a direct guide to portfolio managers that is applicable to a real-world trading scenario, which includes transaction costs and many different ways to transition from the current portfolio weight to the portfolio weight in the next period.

In [98], the state space is defined as the previous allocation and five market features for each asset i: the rate of change of the closing price of asset i in period t, the ratio of the opening price in period t to the closing price in period $t-1$ for asset i, the ratio of the closing price to the highest price of asset i in period t, the ratio of the closing price to the lowest price of asset i in period t, and the rate of change of the volume of asset i in period t.

The action space is composed of three possible actions for each asset: buy, sell or hold. Thus, the number of possible actions is 3^N, where N is the number of assets considered, and $a_t \in \mathbb{Z}^N : a_{t,i} \in \{-1, 0, 1\}\ \forall i, t$. Moreover, the orders are carried out at a fixed trading size, i.e., the agent can only sell or buy a predetermined amount. Park et al. argue that this limitation is similar to the restrictions on hedge funds that allow portfolio traders to only trade below a certain amount each day. Thus, they believe is not an unrealistic assumption.

However, this choice of action space imposed several challenges for the authors. First, some actions are infeasible depending on the state. For instance, the agent cannot buy more assets if there is no available capital or cannot sell assets that it did not buy beforehand. In order to deal with the infeasibility of some actions, the authors propose a rule-based mapping. Given an an originally infeasible action, the algorithm derives a similar action set including only actions that are feasible, and that do not have any opposite direction of each asset compared to the directions under the original action.

To illustrate, consider the case where the action of buying both $asset_1$ and $asset_2$ is infeasible due to the lack of available funds. A similar set proposed by the mapping would be all the actions that do not involve selling these two assets (i.e., holding $asset_1$ and buying $asset_2$, buying $asset_1$ and holding $asset_2$, and holding both $asset_1$ and $asset_2$).

Then, the rule finds the most valuable action (i.e., the action with the largest Q-value) in the similar action set, and it maps the infeasible action to the identified feasible action. This mapping function can be considered a heuristic approach for solving a quadratic programming to handle RL in a constrained action space [14].

The value for $\hat{Q}(s_t, a_t)$ is approximated by a deep neural network. The algorithm is closely related to [87], but the authors introduce several changes. Motivated by [112], instead of just taking one action per time step, the model simulates several actions and updates their Q values. However, this results in a severe computational overhead. Park et al. argue that this technique can relax the data shortage issue that arises when deriving a multi-asset trading strategy and multi-core parallel computing can prevent this computational time from greatly increasing.

With respect to the deep neural network architecture, the authors first employ an LSTM autoencoder [108] to extract relevant information from the market data. Then, the output of this network is concatenated with the previous portfolio position and fed through a fully connected neural network.

The authors compared their results with 4 different traditional strategies using two portfolios with 3 assets each. The model outperforms the proposed benchmark. However, the number of assets evaluated is fairly low, the benchmark strategies are relatively simple and the performance gain is not very significant. Moreover, using Q-Learning introduces several challenges that could be avoid by using policy-based methods.

4.1.2 Policy-based Algorithms

The policy gradient method was first applied to finance by Moody et al. in [90]. Their approach is called Recurrent Reinforcement Learning (RRL), and is based on the following steps: the system takes actions (makes trades), receives feedback based on its performance (such as profit, utility or risk-adjusted return) and then adjusts its internal parameters to increase its future rewards. Therefore, the policy is optimized directly.

Let $\mathbf{w}_t$ denote the portfolio weights. In the case of only one asset with fixed position sizes, $\mathbf{w}_t$ would only assume two values, either long or short [3], i.e., $\mathbf{w}_t \in \{-1, 1\}$. The position at time t is a function of the previous allocation $\mathbf{w}_{t-1}$ and past returns $\mathbf{r}_t \in \mathbb{R}^M$. Both quantities can be expressed as the state matrix $\mathbf{S_t} \in \mathbb{R}^{N \times M+2}$, where N is the number of assets considered, M is the number of time lags considered and the other two extra columns are the bias, which is set to 1, and the past allocation percentage of each asset.

According to the RRL framework, a single asset trading system has the following decision

3. The short operation involves borrowing shares and selling them afterward. These shares have to be repurchased later, so a profit is made when the repurchasing price is lower than the price when they were first sold.

function:

$$\mathbf{w}_t = \mathbf{f}(\boldsymbol{\theta_t}; \mathbf{S_t}), \tag{4.5}$$

where $\boldsymbol{\theta_t} = [\theta_b, \theta_1, ..., \theta_\omega] \in \mathbb{R}^{M+2}$ denotes the learned policy parameters at time t. In the case of a trading system with only one asset and fixed position size, the allocation function can be written as:

$$\begin{aligned} \mathbf{w}_t &= \text{sign}(\theta_b + \theta_1 \times r_t + \theta_2 \times r_{t-1} + \cdots + \theta_M \times r_{t-M} + \theta_\omega \times \mathbf{w}_{t-1}) \\ \mathbf{w}_t &= \text{sign}(\mathbf{S_t}\boldsymbol{\theta_t}^T) \end{aligned} \tag{4.6}$$

where sign is the sign function, θ_ω is the corresponding weight of the previous allocation and θ_b is the weight of the bias term. In this setting, the agent will take a long position when $\mathbf{w}_t > 0$ and short position when $\mathbf{w}_t < 0$.

This model can be easily extended to allow continuous allocations by replacing the sign function with other non-linear functions $f(.)$, such as $\tanh(\cdot)$ when short orders are allowed, or $\text{sigmoid}(\cdot)$ in a long-only portfolio.

The RRL algorithm can also be extended to deal with multiple assets. Denoting $\mathbf{w}_{i,t}$ as the portfolio weight of the i-th asset at time t, the allocations in a long-only portfolio with N assets have to satisfy the following conditions:

$$\mathbf{w}_{t,i} \geq 0 \text{ and } \sum_{i=1}^{N} w_{t,i} = 0, \ \forall t\,. \tag{4.7}$$

One straightforward way to impose these constraints without performing a constrained optimization is to use a softmax function in the output of the model:

$$\mathbf{w}_{t,i} = \frac{exp(\mathcal{T} f_i(.))}{\sum_{i=1}^{N} exp(\mathcal{T} f_i(.))} \tag{4.8}$$

where $f_i(.)$ is the i-th column of $f(\mathbf{S_t}\boldsymbol{\theta_t}^T)$, which represents the score function of asset i (output before normalization), and $\mathcal{T}$ is the temperature parameter [4], often set to 1. Hence, the contribution of each asset is normalized in order to satisfy the allocations constraints. Moreover, it is possible

Since our application is concerned with a long-only portfolio, we employed the *sigmoid* function. The allocation at each time step is then determined by

$$\mathbf{w}_{t,i} = \frac{\exp(\text{sigmoid}_i(\mathbf{S_t}\boldsymbol{\theta_t}^T)}{\sum_{i=1}^{N} \exp(\text{sigmoid}_i(\mathbf{S_t}\boldsymbol{\theta_t}^T)}, \tag{4.9}$$

where $\text{sigmoid}(x) = \dfrac{1}{1+e^{-x}}$.

4. For low temperatures, the allocations are less concentrated, i.e., closer to an uniform portfolio. For higher temperatures, the allocation is more aggressive in the sense that assets with higher score get assigned greater weights.

As an RL-based method, the RRL algorithm aims to maximize a performance function J_t by adjusting its policy parameters $\boldsymbol{\theta_t}$ in a continuous manner. These functions are related to the portfolio profits (or losses) obtained after a sequence of continuous allocation during T time steps. The optimization task is thus:

$$\max \; J(\mathcal{R}_t; \boldsymbol{\theta_t}) \,, \tag{4.10}$$

where $\mathcal{R}_t$ is the portfolio return, as defined in eq. (3.9). The performance function can be expressed as a function of the sequence of portfolio returns $J(\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_T)$, which we denote as $\mathbf{J_T}$.

This maximization for a complete sequence of T allocations can be performed offline using batch optimization algorithms, or online using stochastic optimization methods. Using the chain rule, we can write the gradient of $\mathbf{J_T}$ with respect to the policy parameters $\boldsymbol{\theta}$ of the model after a sequence of length T as

$$\frac{d\mathbf{J_T}(\theta)}{d\boldsymbol{\theta}} = \sum_{t=1}^{T} \frac{d\mathbf{J_T}}{d\mathcal{R}_t} \left\{ \frac{d\mathcal{R}_t}{d\mathbf{w}_t} \, \frac{d\mathbf{w}_t}{d\boldsymbol{\theta}} + \frac{d\mathcal{R}_t}{d\mathbf{w}_{t-1}} \, \frac{d\mathbf{w}_{t-1}}{d\boldsymbol{\theta}} \right\} \tag{4.11}$$

The performance function U_T can be optimized in batch mode by repeatedly computing its value on forward passes and updating the policy parameters θ by using an optimization method. For example, the update rule for gradient ascent becomes:

$$\begin{aligned} \boldsymbol{\theta_{k+1}} &= \boldsymbol{\theta_k} + \alpha \Delta\theta \,, \\ \text{with} \;\; \Delta\theta &= \frac{d\mathbf{J_T}(\theta)}{d\boldsymbol{\theta}} \,, \end{aligned} \tag{4.12}$$

where α is the learning rate.

It is worthwhile to note that due to the inherent recurrence, the derivatives $\dfrac{d\mathbf{w}_t}{d\theta}$ depend upon the entire sequence of previous time steps. Therefore, it requires the use of recurrent algorithms such as backpropagation through time (BPTT) [121] or real-time recurrent learning (RTRL) [123] to correctly compute the weight updates.

The derivative of $\mathbf{w}_t$ with respect to the policy parameters θ can be written as:

$$\frac{d\mathbf{w}_t}{d\boldsymbol{\theta}} = \frac{\partial \mathbf{w}_t}{\partial \boldsymbol{\theta}} + \frac{\partial \mathbf{w}_t}{\partial \mathbf{w}_{t-1}} \frac{d\mathbf{w}_{t-1}}{d\boldsymbol{\theta}} \tag{4.13}$$

Additionally, we can use a simple online approximation of equation (4.11) by considering during the forward pass only the terms that depend on the most recent return:

$$\frac{dJ_t(\theta)}{d\boldsymbol{\theta}} = \frac{dJ_t}{d\mathcal{R}_t} \left\{ \frac{d\mathcal{R}_t}{d\mathbf{w}_t} \, \frac{d\mathbf{w}_t}{d\boldsymbol{\theta}} + \frac{d\mathcal{R}_t}{d\mathbf{w}_{t-1}} \, \frac{d\mathbf{w}_{t-1}}{d\boldsymbol{\theta}} \right\} \tag{4.14}$$

Similarly to the offline optimization, the policy parameters can be updated using gradient ascent

$$\begin{aligned} \boldsymbol{\theta_{t+1}} &= \boldsymbol{\theta_t} + \alpha \Delta \boldsymbol{\theta_t} \\ \text{with} \;\; \Delta\boldsymbol{\theta_t} &= \frac{dJ_t(\boldsymbol{\theta_t})}{d\boldsymbol{\theta_t}}, \end{aligned} \tag{4.15}$$

and the stochastic version of equation (4.13):

$$\frac{d\mathbf{w}_t}{d\boldsymbol{\theta_t}} \approx \frac{\partial \mathbf{w}_t}{\partial \boldsymbol{\theta_t}} + \frac{\partial \mathbf{w}_t}{\partial \mathbf{w}_{t-1}} \frac{d\mathbf{w}_{t-1}}{d\boldsymbol{\theta_t}} \,. \tag{4.16}$$

In a follow-up paper, Moody and Saffell [88] show that RRL produces better trading strategies than systems that use Q-Learning. The performance was evaluated on trading currency-pairs and risky assets. They argue that value-based methods are natural for problems where immediate feedback on performance is not readily available at each point in time. However, for many financial decision making problems, results accumulate over time, and one can immediately measure short-term performance and use it to update the policy of the agent.

Building upon the RRL algorithm, in [43] Gold explores the effects of using more complex decision functions for trading currency pairs. More specifically, the proposal introduces a hidden layer, allowing the agent to learn more complex decision rules. The decision function becomes:

$$\begin{aligned} \mathbf{w}_t &= sign(\mathbf{h_t}\boldsymbol{\rho}_t^T + \mathbf{v}') \\ \mathbf{h_t} &= tanh(\mathbf{s_t}\boldsymbol{\theta_t}^T) \,, \end{aligned} \tag{4.17}$$

where $\boldsymbol{\rho}$ and $\mathbf{v}'$ represent the weights and bias of the second layer, respectively, and $\mathbf{s_t}$ is modelled as the last M exchange rate variations of a given currency-pair.

The model was evaluated for 25 different currency-pairs using hourly-data. Instead of using the conventional method of splitting training and testing sets by time, Gold splits 10 pairs for training and 15 for testing. However, such an evaluation strategy is not recommended as it could cause look-ahead bias.

The overall performance was positive, but Gold attributes the high performance to the simplicity of the price model used in the simulations, as just a single price quote is used at each half hour and the trader is guaranteed to transact at that price. In reality, foreign exchange prices are very noisy [89], so many of the prices used in simulation are probably not available in real time.

Gold also analyses the impact of changing the network parameters (number of neurons) and training parameters (learning rate, number of epochs used for training and length of the training). Overall, the optimal parameters varied widely depending on the currency-pair. Moreover, the author found that the two-layer neural network performance was worse than the performance achieved by a single layer network, due to overfitting. Nonetheless, as the first to model the policy as a neural network, [43] can be considered as an early precursor of the deep recurrent reinforcement learning methods such as [38].

Dempster and Leemans [29] expand the RRL algorithm in another direction by embedding an RRL trader into a larger trading system with three layers. The first layer of the system is composed of the RRL agent with two proposed performance tweaks. First, the authors rescaled the model's weights as soon as one of the weights hit a certain threshold value. This was done to avoid weights with large values, but a more reasonable choice could be using regularization in the gradient update [15]. The second proposed change aims to improve the

position updating scheme by recalculating the output of the trading model twice: once after the new inputs have been received and again after the weights are updated.

The second layer performs risk management by introducing a stop-loss for every trade, which is set and adjusted so that it is always a given level under or above the best price reached during that position. If the price reaches this value, the order is stopped (closed). In addition, a "cool-down period" is introduced after a position is stopped to avoid another unexpected loss in succession. Finally, this layer also terminates the agent's operations if the maximum drawdown is reached during the period.

The final layer aims to optimize the investor's utility function by updating the previous layers' parameters. The authors define a custom risk measure Λ:

$$\Lambda = \frac{\sum_{t=0}^{T} \mathcal{R}_t^2 \mathbf{1}(\mathcal{R}_t < 0)}{\sum_{t=0}^{T} \mathcal{R}_t^2 \mathbf{1}(\mathcal{R}_t > 0)}\,, \tag{4.18}$$

and a utility function U defined by

$$U(\bar{\mathcal{R}}, \Lambda, v) = a(1-v)\bar{\mathcal{R}} - v\Lambda\,, \tag{4.19}$$

where $\mathcal{R}_t$ is the portfolio's return, $\bar{\mathcal{R}}$ is the average portfolio return in the period of length T considered, v is the system risk aversion and a is a constant.

The utility U aims to penalize large negative rewards via Λ and control the risk via the v parameter. The function U is also used to tune several parameters, such as the stop-loss level and the learning rate. The model was evaluated on minute-to-minute EUR/USD exchange rate, achieving an impressive 26% annualized return. One of the most important contributions of this paper is to present a multi-layer system with a recurrent reinforcement learning trader.

Instead of adding more layers, Maringer and Ramtohul [82] proposed a regime-switching model. They argue that linear models might not be suitable to capture all the intricate aspects of financial data but, on the other hand, black-box approaches such as neural networks should be avoided because it is important to understand how economic variables affect financial markets.

With this in mind, Maringer and Ramtohul define models that can have different regimes, where each regime may have different dynamics. This framework is better adapted to deal with dramatic changes in behaviour in financial series, such as those caused by major crises or changes in government policy [48].

Maringer and Ramtohul considered two threshold-based regime-switching methods, the threshold autoregressive (TAR) and smooth transition autoregressive (STAR) models. These can be defined as

$$G_t = \begin{cases} \mathbf{1}[q_t > c], & \text{for TAR} \\ [1 + exp(-\gamma[q_t - c])]^{-1}, & \text{for STAR}\,, \end{cases} \tag{4.20}$$

where q_t is the transition/indicator variable, c is a threshold value, and γ defines the smoothness of the transition. The values G_t for the threshold model are binary values, while for the smooth transition version, G_t can take any value in the range [0,1].

The regime-switch recurrent reinforcement learning (RSRRL) model is given by:

$$\begin{aligned} \mathbf{w_t} &= y_{t,1}G_t + y_{t,2}(1 - G_t) \\ y_{t,j} &= \tanh(\mathbf{s_t}\theta_t^T) \text{ for } j = \{1, 2\} \end{aligned} \tag{4.21}$$

where $y_{t,j}$ is the same as in standard RRL. The model can be interpreted as a combination of two RRL agents with different weights and regimes. The final output is defined by the weighted average of the output of each model and they can be trained by the same procedure described in [91]. The weights G_t are determined according to the volatility of returns.

The RSRRL was evaluated using daily data from 12 stocks that compose the Dow Jones Industrial Average (DJIA) index. The authors showed that the performance of RSRRL is superior to RRL in the majority of cases, specially in situations where the data sets are characterised by distinctly different regimes such as financial crises or economic policy changes. However, the results are not sufficient to conclude that RSRRL is consistently superior to standard RRL, as RRL outperformed the regime-switching variants in some cases. Moreover, other data frequencies should be used and different stocks included in order to have a more significant comparison.

Similarly, Zhang and Maringer also used multiple RRL agents [125], but instead of averaging their output according to the regime-switching framework, the parameters of all models were averaged to determine the final model's weights. The intuition was that by considering different agents trading the same asset, the noise would be reduced and the model would be more robust.

More specifically, Zhang and Maringer propose two update scheme. The first scheme is called "average elitist", and in this method several models are initialized with random weights and trained using the training data. The training period also covers an evaluation period, which is used to rank the best $n \in \mathbb{N}$ agents according to their Sharpe ratio.

Then, a new RRL agent is initialized using the average parameters of the "elite" members for the testing period. More formally, letting $t_0 = T_{train} + 1$ be the first testing step, the policy parameters are initialized as:

$$\hat{\boldsymbol{\theta}}_{\boldsymbol{t0}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{\theta}_{\boldsymbol{t0,i}}. \tag{4.22}$$

The updates of the policy parameters are:

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{\boldsymbol{t+1}} &= \hat{\boldsymbol{\theta}}_{\boldsymbol{t}} + \alpha\Delta\hat{\theta}_t \\ \text{with } \Delta\hat{\theta}_t &= \frac{1}{n}\sum_{i=1}^{n} \frac{dJ_t(\boldsymbol{\theta}_{\boldsymbol{t,i}})}{d\boldsymbol{\theta}_{\boldsymbol{t,i}}}, \end{aligned} \tag{4.23}$$

The second scheme is called "multiple elitist", which is based on training different RRL agents using the "average elitist" technique, and aggregating their parameters based on the correlation among assets. Let m be a group of M_a highly correlated assets. The gradient

update of the elite agent associated with m is given by

$$\Delta\check{\theta}_t^m = \left(\phi\Delta\check{\theta}_t^m + (1-\phi)\frac{1}{M_a - 1}\sum_{j\neq m}\Delta\hat{\theta}_t^{\,j} \right), \tag{4.24}$$

where ϕ is the parameter that controls the influence of gradients from outside m. The weights are initialized using Eq. (4.22) and updated using

$$\check{\boldsymbol{\theta}}_{t+1}^m = \check{\boldsymbol{\theta}}_t^m + \alpha\Delta\check{\theta}_t^m. \tag{4.25}$$

The Pearson correlation coefficient is used to identify highly correlated stocks in these groups.

The agents were evaluated using daily prices from the companies of the S&P 500 index and each of the RRL trading systems consisted of 100 individual agents. The "average elitist" approach selected the top 5% performing agents. The presented results show that the system is able to beat the buy&hold strategy and a random agent. However, the results are not very impressive when compared to these simple strategies, considering the amount of extra computation. There is no comparison with standard RRL. Moreover, estimating the correlation between assets can lead to large errors if the number of assets is large compared to the number of observations [61]. This may be one of the reasons why the authors did not find any meaningful improvement by employing the multiple elitist when compared to its average version.

Some extensions of RRL have also been proposed in the literature to deal with high-frequency trading. However, these extensions are based on preprocessing or using other information as input. We present only a brief overview of these papers as these applications are concerned with high-frequency data, and our main goal is to deal with low-frequency asset allocation.

Gabrielsson and Johansson in [39] use RRL with data from Japanese candlesticks to trade the S&P 500 equity index futures contract. According to their results, the agent trained on the candlestick outperformed the basic RRL trader. An alternative approach was used by Deng et al. in [31] to deal with high-frequency trading. Sparse coding is used to create features, based on the argument that it has significant advantages in noise reduction and task-driven dictionary learning.

More recently, Almahdi and Yang proposed in [3] a recurrent reinforcement portfolio allocation model using particle swarm optimization (PSO) methods [5] [63]. The trading signals for each individual asset are generated by the RRL agent, which is trained using the Calmar ratio (a risk-adjusted metric) as the reward function. Since both short and long operations are allowed, the score function of each asset i is given by:

$$f_{t,i} = \tanh(\mathbf{s}_{t,i}\boldsymbol{\theta_t}^T). \tag{4.26}$$

5. Particle swarm optimization is a population based approach similar to genetic algorithms where a collection of particles moves in the solution space and at each step the algorithm evaluates the fitness function for every particle moving in the space.

As the RRL will only act indicating if the signal is to long or short a given asset, the authors imposed the following rule:

$$f_{t,i} = \begin{cases} -1, & \text{if } \tanh(\mathbf{s}_{t,i}\boldsymbol{\theta_t}^T) < 0 \\ +1, & \text{if } \tanh(\mathbf{s}_{t,i}\boldsymbol{\theta_t}^T) > 0 \end{cases} \tag{4.27}$$

Next, the weights of the portfolio are obtained by using the Calmar ratio as the fitness function for the PSO. The particle swarm optimization procedure described is closely related to [25].

The assets with the K highest fitness are selected and then normalized, according to their fitness value. Some other constraints are also taken into consideration, such as cardinality, floor and ceiling values. This results in PSO weights $pso_{t,i}$ and the final portfolios weights are obtained as follows:

$$w_{t,i} = f_{t,i}\ pso_{t,i}, \tag{4.28}$$

respecting the constraint $\sum_i^K |w_{t,i}| = 1, \forall t$.

This model was evaluated using data from 2011 to 2015 from the S&P 100 index and the authors experimented with four variants of the PSO algorithm. The parameter K was set to 10, so the portfolio was always composed of 10 different stocks. The results show that one variant consistently outperformed the other variants and the benchmark. One of the limitations of [3] is that the number of equities is fixed throughout the whole period. This could limit the risk diversification under periods of financial stress. Moreover, information is discarded by imposing Eq. (4.27). With this approach, a score of 0.99 and 0.01 are treated the same way. Finally, it is unclear why the model is evaluated over only 5 years of data.

One of the first attempts to apply deep reinforcement learning in an asset allocation problem is proposed by Deng et al. in [30]. The system consists of two components. First, a fuzzy layer is used to reduce the uncertainty in the original financial data [69, 97]. The fuzzy sets are defined in groups based on their trends (increasing, decreasing, or no trend). The second component is the trader system, implemented as a recurrent neural network with 4 layers. The reward is simple returns and the agent actions are mapped by a tanh function, so the trader can stay long or short. The model was backtested on minute by minute data of three Chinese futures markets, and on daily data for the S&P 500 index. As opposed to most previous works, this paper compared their proposed model with models from other papers. The result was compared to the shallow version of their proposed algorithm, to standard RRL [88], to RRL using sparse coding [31] and to the Buy&Hold strategy.

Guo et al. [47] use a Convolutional Neural Network (CNN) to directly approximate the policy function with the goal of maximizing the expected logarithmic return of a portfolio. The proposed algorithm is called Robust Log-Optimal Strategy with Reinforcement Learning and is based on two steps. The first step is to estimate the optimal portfolio allocation v_t using estimations for the mean and covariance matrix of the assets' returns. These estimations are done by computing the Pearson correlation coefficient between the current price returns and historical returns, and choosing the time period in which the coefficient is the highest.

The second step is to use a CNN to approximate the policy in order to obtain the final weight allocation. The state is composed of the initial portfolio estimations and other input

features such as opening, highest and lowest prices. The model was evaluated on 100 randomly selected constituent stocks of the CSI 300 index.

A deep learning approach was also used in [59]. The input features consist of asset prices and the agent policy is approximated by a deep neural network in order to compute a set of portfolio weights as the agent's action. The gradient of the state-action value function is simply the portfolio's adjusted sample return after taking an action. This action is determined in a constrained matter based on the softmax voting scores from the output layer.

The neural network consists of a CNN with two hidden layers: a convolution layer and a fully-connected layer. The height of the convolution layer's filter is equal to the number of assets. Pooling is not applied after the convolution layer as it causes loss of location information. Following the convolution layer there is a fully connected layer, and a softmax output layer. The proposed method outperformed most of the benchmark strategies when trading a portfolio with 12 cryptocurrencies [6].

4.1.3 Actor-critic algorithms

In this section, we present papers that have used actor-critic methods in portfolio optimization problems. This methodology combines value-based and policy-based algorithms, and is the basis of state-of-the-art RL methods [103, 104].

Zhang et al. evaluated three deep reinforcement learning algorithms in the task of trading future contracts: deep Q-learning [87], deep policy gradient, and Advantage Actor Critic (A2C) [86]. They represented the state space using past returns along with two technical indicators: the moving average convergence divergence (MACD) and the relative strength index (RSI) indicators [11].

The policy and Q-function are approximated using a two-layer LSTM network [41], and a separated model is trained for each asset class. The reward function is set as profits obtained, representing a risk-insensitive trade.

The model evaluation was fairly comprehensive, as they evaluated the performance of each of the three models using 9 years of data and multiple different metrics, including the annualized returns, volatility, volatility of negative returns, Sharpe ratio, Calmar ratio, maximum drawdown, percentage of positive trade returns and the ratio between positive and negative trade returns.

Moreover, the performance of the RL agents was also compared to other stategies, such as Buy&Hold, and methods based on technical analysis. The RL agents were able to outperform the benchmarks, but only one asset was traded at a time. It would be interesting to expand the number of assets and experiment with other reward functions.

Li et al. [76] propose a novel Adaptive Deep Deterministic Reinforcement Learning approach for the portfolio allocation problem. The key idea of their model is to adjust the amplitude of changes according to the signal of the prediction errors (difference between the actual and expected rewards). In this case, the learning rate can be adjusted from one time

6. Cryptocurrencies are electronic and decentralized alternatives to government-issued money, with Bitcoin as the best-known example.

step to the next.

The model is mainly based on the Deep Deterministic Policy Gradient (DDPG) [107], which aims to combine a policy gradient network (actor) with a Q-learning network (actor). Inspired by [74], the authors modified the Q-Learning update to the following:

$$Q_\pi(s_{t+1}, a_{t+1}) = Q_\pi(s_t, a_t) + \begin{cases} \alpha^- \delta(t), & \text{if } \delta(t) < 0 \\ \alpha^+ \delta(t), & \text{if } \delta(t) > 0\,, \end{cases} \tag{4.29}$$

where $\delta(t)$ is the prediction error, calculated as follows:

$$\delta(t) = R_t - Q_\pi(s_t, a_t)\,, \tag{4.30}$$

which represents the difference between the actual and expected rewards.

The state space comprises past returns and past allocation, the actions are the weights of the portfolio and the reward is the cumulative return. Random Gaussian noise is added to encourage the actor to explore the action space and a softmax rule is used to control the exploration-exploitation for the Q-Learning.

The proposed method was evaluated in terms of Sharpe ratio and cumulative return trading 30 stocks from the DJIA index using 17 years of data. The authors set $\alpha^- = 0$ and $\alpha^+ = 1$ in Eq. (4.29), making the Q-learning only update its values when $\delta(t)$ is positive. These changes enhance the model when compared to the standard DDPG, as the reported results show significant improvement. Moreover, the proposed model also outperformed the mean-variance portfolio and Buy&Hold strategies.

One of the main advantages of policy-based RL methods is that they can be applied directly to large continuous domains. However, one of the drawbacks of approximating the optimal policy with a deep neural network is the large number of parameters, which can make the model suffer from suboptimal solutions.

Moreover, the fact that model-free RL solves portfolio optimization problems by adjusting its policy parameters for each new state and action makes the dynamics of its learning fundamentally dependent on the samples used. Since historical prices used for training are just a realization of a complex process of market price, model-free RL is particularly sample inefficient in this case.

Another concern with approximating the policy function with an overly complex function (like a deep neural network) is overfitting. This is a particular concern in our case, since the availability of data is limited.

4.1.4 Model-based Reinforcement Learning

Unlike model-free algorithms, model-based RL methods uses simulated transitions to learn a model, resulting in increased sample efficiency and, consequently, increased stability. The main challenge lies in simulating transitions of a complex problem. A novel alternative is proposed by [124]. The authors use a Generative Adversarial Network (GAN) to generate synthetic market data to overcome sample inefficiency. This is achieved by training a Recurrent

GAN (RGAN) using real historical price data to produce more data. The synthetic data is then used to make price predictions and train the off-policy actor-critic DDPG algorithm in an imitation learning framework. Despite its promising results, the implementation is highly-complex and lacks interpretability.

4.2 Key Observations

After reviewing the existing literature on the topic of reinforcement learning for portfolio allocation, we can identify some desirable characteristics of the design and evaluation process of a robust asset allocation model that uses reinforcement learning:

— *Comprehensive Evaluation:* the proposed model is evaluated using several metrics, including but not limited to cumulative return, maximum drawdown, Sharpe ratio and volatility. Additionally, the model should be tested during different market conditions.

— *Comparison with other reinforcement learning methods:* the proposed model's performance metrics are compared with other similar RL algorithms.

— *Comparison with classical portfolio techniques:* the proposed model performance metrics are also compared to classical portfolio techniques, such as those described in Section 3.2.

— *Interpretability:* a model whose decisions can be interpreted offers a higher degree of transparency and less uncertainty to a human expert. This factor also ensures that automated trading systems are more tractable, making it easier to fine-tune its parameters whenever performance starts to deteriorate.

We aim to address these aspects in our proposed methodology and contribution, described in the next chapter.

4.3 Summary

We presented a literature review of the application of reinforcement learning to the portfolio allocation problem. Over the past two decades, the RL research community has made considerable advances in this field.

However, there are still room for improvement. In particular, a more rigorous and standard evaluation procedure would lead to more reliable assessment of new proposals.

Chapter 5

Methodology and Proposed Extensions

In Chapters 2 and 3, we introduced basic concepts of reinforcement learning and financial theory. Our goal is to use reinforcement learning in a portfolio optimization task. Nevertheless, we need to first formulate our application problem of interest so that it is an suitable form for the RL framework.

The problem of asset allocation consists in dynamically deciding in which assets to invest the available capital at each time step t in order to maximize the expected total return or another relevant performance measure, such as the risk-adjusted return. It can be formalized as a discrete-time stochastic optimization problem and the most natural formulation is a Markov Decision Process (MDP).

This chapter starts by casting the problem of asset allocation as a reinforcement learning task. Section 5.1 discusses the assumptions underpinning the formulation. In Section 5.2 we describe the types of agent that are used, and in Sections 5.3, 5.4 and 5.5, we define the key aspects of the environment. In Section 5.6, we discuss suitable reinforcement learning frameworks and outline an approach for validation of baselines. Finally, we propose two extensions to the RRL algorithm in Section 5.7.

5.1 Assumptions

In order to evaluate an asset allocation model, we have to backtest it, i.e., estimate its performance as if it had been deployed in the past, without knowing any "future" market information. This requires a series of assumptions regarding past conditions, which we present below. Despite being simplifications, these assumptions are quite realistic if we are considering assets that trade at a high volume [60].

5.1.1 Zero Slippage

The difference between the expected price of an order (buy or sell) and the actual price that it is executed is referred to as *Slippage*. This often occurs in periods of high market volatility or when the total value of the order is large compared to the trading volume of

an asset. In our models and experiments, we assume *zero slippage*. Under the zero slippage assumption, all orders are executed at the expected price.

5.1.2 Zero Market Impact

Very large orders can affect the market behaviour, provoking what is called a herd effect. [1] We assume that the agent orders are sufficiently small that they have zero market impact, i.e., they have no influence on the market.

5.1.3 Sufficient Liquidity

All assets considered are assumed liquid, i.e, they can be converted into cash quickly, and under the same conditions.

5.1.4 No Transaction Costs

When assessing the performance of a portfolio allocation strategy it is usually essential to incorporate transaction costs [60, 90]. These are expenses related to trading and include brokerage fees and commissions. Consideration of such costs is critical in preventing an agent from gravitating towards very frequent changes in portfolio structure. The fees associated with excessive changes can often outweigh the benefits of choosing a better balance of assets.

In this thesis, we restrict the portfolio allocation strategy to make changes at most once per month, meaning that the transaction costs have a much reduced impact on the performance. Moreover, commission-free brokers are becoming more popular recently [96]. For these reasons, in our experiments we assume that all agents can change their allocations at no extra cost.

5.2 Types of Agents

We consider two different kinds of models. The first type is based on the portfolio optimization algorithms presented in Chapter 2. We refer to the agents in these models as classical portfolio (CP) agents. They rely on assumptions concerning expected returns and intra-asset correlation to determine the portfolio allocation. The second type does not depend on these assumptions. Instead, it is based on the reinforcement learning framework. We refer to the agents in the models belonging to the second type as RL agents.

The following discussion regarding reward, action and space states only concerns the RL agents, as the framework of CP agents does not depend on these elements.

1. A herd effect exists in the financial market when a group of investors ignores their own information and instead only follows the decisions of other investors.

5.3 Action Space

We consider a financial market consisting of N different assets, traded only at discrete times $t \in \{0, 1, 2..., T\}$, and with prices defined by a price vector $\mathbf{p_t} = [p_{1,t}, \ ... \ , p_{N,t}]$.

The agent observes the state of the market at each time step t, and based on this information it chooses how to redistribute the portfolio allocations. The change in wealth (profit or loss) from the allocation at time $t-1$ is also observed and is used to update the policy of the agent. The agent's goal is to maximize a given performance measure.

In order to allocate assets, the RL agent has to determine the portfolio vector $\mathbf{w}_t$ for all T time steps. Therefore, the action a_t at time t is the assignment of a portfolio weight to each of the N assets:

$$a_t \equiv \mathbf{w}_t = [w_{1,t}, \ ... \ , w_{N,t}]^T. \tag{5.1}$$

As a consequence, the action space $\mathcal{A}$ is defined on a continuous N-dimensional real space $\mathbb{R}^N$:

$$a_t \in \mathcal{A} \subseteq [0, 1]^N, \ \ \forall t \ \text{ subject to } \ \sum_{i=1}^{N} a_{i,t} = 1 \tag{5.2}$$

We are assuming a *long-only* portfolio, i.e., the weights are non-negative since short-selling is not allowed.

5.4 State Space

The agent interacts with the environment, defined as the financial market. This environment is extremely complex, containing all available information in the markets and including its participants' expectations towards future events and earnings. Hence, it is impossible to provide all this information for the agent. Nonetheless, asset prices are publicly available and it is believed that they contain enough information to drive investment decisions [68, 79].

At each time step, the agent can observe the asset prices and use this information to allocate the portfolio weights accordingly. Let $\mathbf{p_t} = [p_{1,t}, \ ... \ , p_{N,t}]$ denote the closing prices for all N assets at time t and denote the log-return of asset i at time t as $r_{i,t} = log(\frac{p_{i,t}}{p_{i,t-1}})$. The log return vector can then be defined as

$$\mathbf{r_t} = [r_{i,t}, \ ... \ , r_{N,t}]. \tag{5.3}$$

Additionally, we want the model to consider its previous allocation ($\mathbf{w}_{t-1}$) as one of the state components. This helps the model to avoid dramatic fluctuations in the allocations from one time step to the next. This is particularly important when transaction costs are taken into account. The state s_t at time t is defined as

$$\mathbf{S_t} = \begin{bmatrix} w_{1,t-1} & r_{1,t} & r_{1,t-1} & \dots & r_{1,t-M} \\ w_{2,t-1} & r_{2,t} & r_{2,t-1} & \dots & r_{2,t-M} \\ \vdots & \vdots & \ddots & \dots & \\ w_{N,t-1} & r_{M,t} & r_{M,t-1} & \dots & r_{M,t-M} \end{bmatrix} \tag{5.4}$$

where $r_{i,t-M}$ is the log-return of the i-th asset between times $t-M-1$ and $t-M$. Hence, the state is a function of the previous action and past returns. [2]

5.5 Reward

The last component remaining to be defined is the reward signal. This scalar value has to fully specify the objective of the agent since the goal of a RL agent is to maximize the cumulative reward G_t, as defined by equation (2.1). As a result, the reward function can have significant a impact on the strategies employed by the agents. The goal of a portfolio manager is usually to maximize the return while minimizing the risk. Moody et al. [90] suggests two reward function that can be used in RL applications: the log return and the differential Sharpe ratio.

5.5.1 Log return

Let R_t be the reward obtained at time t, $\mathbf{R}_t$ be the t-th row of return matrix $\mathbf{R}$ and $\mathbf{w}_t$ be the corresponding action (allocation) for time step t. If we consider the log-return as reward function, the goal of the RL agent is defined as:

$$\underset{\theta}{\text{maximize}} \sum_{t=1}^{T} \mathbb{E}[\gamma^t ln(1 + \mathbf{R}_t \mathbf{w}_t)] \tag{5.5}$$

The agent using the log return obtained by its allocation in the previous step is performing a multi-step maximization of the cumulative log return, defined by the sum of the log returns obtained during the evaluation period (see eq. (3.35)). The agent trained using log returns as the reward is only focused on maximizing the returns, and the volatility is not penalized.

5.5.2 Differential Sharpe Ratio

In practice, a good portfolio manager should always consider the risk exposure when constructing a portfolio [35]. One of the most commonly used metrics that combines the risk and reward trade-off is the Sharpe ratio [105], defined as:

$$SR_T = \sqrt{T} \frac{\mathbb{E}(\mathcal{R}_t)}{\sqrt{Var(\mathcal{R}_t)}}, \tag{5.6}$$

where T is the length of the time period, $\mathcal{R}_t$ is the portfolio return, $\mathbb{E}$ is the expected value operator and Var the variance, both over the same time period T.

In order to use the Sharpe ratio as a reward function for on-line training, we need to compute the influence on the Sharpe ratio of the return at time t. The differential Sharpe ratio (DSR), introduced by [90], allows to approximate this influence.

2. In some cases, the state can also include an extra column of ones to represent the bias term.

Let us define two auxiliary variables, $A_T = \frac{1}{T}\sum_{t=1}^{T}\mathcal{R}_t$ and $B_T = \frac{1}{T}\sum_{t=1}^{T}\mathcal{R}_t^2$. The Sharpe ratio can then be written as:

$$SR_t = \frac{A_t}{\sqrt{B_t - A_t^2}}. \tag{5.7}$$

Moreover, we can express A_t and B_t in terms of their exponential moving estimates

$$\begin{aligned} A_t &= \eta\mathcal{R}_t + (1-\eta)A_{t-1} = A_{t-1} + \eta(\mathcal{R}_t - A_{t-1}) \\ B_t &= \eta\mathcal{R}_t + (1-\eta)B_{t-1} = B_{t-1} + \eta(\mathcal{R}_t^2 - B_{t-1}) \end{aligned} \tag{5.8}$$

The quantity η controls the magnitude of the influence of the return $\mathcal{R}_t$ on the Sharpe ratio S_t. The DSR is obtained by expanding (5.7) to first order in decay rate η:

$$SR_t \approx SR_{t-1} + \eta\frac{dSR_t}{d\eta}|_{\eta\to 0} \tag{5.9}$$

As only the first-order term in (5.9) depends upon the return $\mathcal{R}_t$ at time t, the differential Sharpe ratio D_t can be defined as:

$$D_t \equiv \frac{dSR_t}{d\eta} = \frac{B_{t-1}(\mathcal{R}_t - A_{t-1}) - \frac{1}{2}A_{t-1}(\mathcal{R}_t^2 - B_{t-1})}{(B_{t-1} - A_{t-1}^2)^{\frac{3}{2}}} \tag{5.10}$$

We can understand the influences of risk and reward by observing the terms in the numerator. The first term becomes positive when $\mathcal{R}_t > A_{t-1}$, i.e., when the return obtained exceeds the moving average of past returns. On the other hand, the second term is negative if $\mathcal{R}_t^2$ exceeds the moving average of past squared returns B_{t-1}, so even positive returns can be penalized if the return is too abrupt.

5.6 Validation of Reinforcement Learning Agents

In this section, we start by discussing which RL methods are best suited to our problem. Subsequently, we specify and describe the reinforcement learning algorithms that will be used as baselines in our experiments. We use agents trained using the REINFORCE [122] and PPO [104] algorithms.

5.6.1 Suitable RL Framework

In the reinforcement learning framework, the goal is to find an optimal policy such that the obtained cumulative reward is maximized. This can be achieved either by using the agent's experience to improve a policy or to improve a model of the environment and then derive a policy.

The inherent uncertainty of the financial market presents a great challenge when modelling it as the environment, since just fitting a model to predict asset prices is itself a challenging problem. In the RL frameworks that attempt to model the environment, the significant

approximation error, induced by both the modeling and the optimization, can propagate over time and hinder performance. As a consequence, most RL approaches in financial applications use model-free frameworks, as seen in the Chapter 4. Hence, we will consider this type of methodology in the rest of this work.

In the model-free RL framework, algorithms can be divided into value-based, policy-based, or a combination of the two. In value-based RL methods (such as Q-Learning [120] and SARSA[101]), the agent's action is determined by finding the optimal policy π^* indirectly via an optimal state-action value function $q^*(s, a)$ that defines how good an action is after following the actual policy in a given state. This approach is straight-forward in problems with discrete state and action spaces, as all state-action values can be stored in a look-up table. If the number of possible states is too large to be stored or if the state space is continuous, using a look-up table is no longer possible. In this case, we can use a approximation function, such as a neural network or a linear model, to approximate $q(., a)\ \forall a \in \mathcal{A}$.

However, this approach cannot be directly applied to continuous domains since finding the optimal action in a continuous space would require an iterative optimization process at every time step. One alternative is discretizing the action space, but it would suffer from the curse of dimensionality and it is not ideal for an asset allocation application where the dimension of the action space is linked with the number of assets considered.

Consequently, policy-based approaches seem more suitable for the portfolio allocation problem. These methods learn the policy parameters θ based on the gradient of the performance measure $J(\theta)$ with respect to the policy parameter. A benefit of using policy gradient methods is the ability to implement stochastic policies, which allow the agent to deal with the exploration/exploitation trade-off [3] in an intuitive manner. The action is chosen by sampling a probability distribution based on the policy, which allows the agent to explore new actions.

With these aspects in mind, we choose two different policy gradient methods to use as baselines. The first is the REINFORCE algorithm, which is one of the simplest PG algorithms that can be parametrized by a deep neural network. This aspect is important since we are interested in understanding the benefits and shortcomings of employing a "black-box" model. Since the REINFORCE presents several shortcomings, we also include the PPO algorithm, a state-of-the-art policy gradient method, as a baseline.

5.6.2 REINFORCE Evaluation

In order to validate the REINFORCE algorithm implementation, we perform an experiment using this algorithm with a small set of real financial data. More specifically, the agent is trained using data from equities index and bonds from two major economies: the United States (S&P 500 index and U.S. 10 Year Treasury Note) and Germany (DAX index and Germany 10 Year Government Bond). The algorithm to train the REINFORCE agent is presented in Algorithm 1.

3. The exploration-exploitation trade-off arises in decision-making systems where there is a frequent pay-off uncertainty. In essence, the system has to decide whether to repeat decisions that have worked well so far (exploit), or to make novel decisions, hoping to obtain even greater rewards (explore).

Algorithm 1 REINFORCE (Monte Carlo Policy Gradient)

Inputs: N assets, states $\mathbf{S_t} \in \mathbb{R}^{N \times M}$, prices $\mathbf{P}$, exploration parameter σ_e
Initialize: Policy parameters θ_0, initial allocation weights $\mathbf{w}_0$
Initialize trajectory buffer: $\mathcal{D}$
Returns: Optimal weights θ^*
while θ has not converged or maximum number of epochs has not been reached **do**
  $\Delta_\theta = 0, G = 0$
  for $t = 1, ..., T$ **do**
    get state s_t
    sample asset allocation from policy $\mathbf{w_t} \sim \pi(.|\mathbf{S_t}, \theta, \sigma_e)$
    evaluate reward $R_t(\mathbf{w_t})$
    accumulate rewards $G_t \longleftarrow R_t + G_t$
    accumulate log gradients $\Delta_\theta \longleftarrow \nabla_\theta \log \pi_\theta(\mathbf{w_t}|\mathbf{S_t}) R_t + \Delta_\theta$ (Eq. 2.21)
  end for
  update policy parameters θ (Eq. (2.20))
end while
$\theta^* = \theta$

It is important to emphasize that the reported performance for the training set **does not** have a direct connection with the out-of-sample performance. The weights that are obtained during the training procedure are updated at every epoch, which means that the algorithm ends up incorporating future information during training after the first pass through the data. For example, the agent's weights during a given epoch at time $t = 4$ have already been updated considering information from $t > 4$ in previous epochs. We conduct this exercise merely to validate the REINFORCE approach and explore the impact of different parameter settings on training performance.

The goal is to evaluate if the agent is able to outperform the benchmark, in this case, a balanced $(1/N)$ portfolio [4]. In other words, we want to verify if the agent is able to learn profitable strategies. Despite its simplicity, the 1/N portfolio proved to be quite competitive in terms of out-of-sample performance when compared to other optimization methods [28].

In Figure 5.1, we present the results for the REINFORCE agents trained using monthly returns data from 1980 to 2012 (approximately 80% of the available data). The policy was approximated by a fully-connected (FC) neural network that can be defined as follows:

$$\begin{aligned} \mathbf{h_1} &= ReLU(\mathbf{W}^{(1)}[\mathbf{S}_t \parallel \mathbf{w_{t-1}}] + b^{(1)}) \\ \mathbf{h_2} &= ReLU(\mathbf{W}^{(2)}\mathbf{h_1} + b^{(2)}) \\ \mathbf{w_t} &= Softmax((\mathbf{W}^{(3)}\mathbf{h_2} + b^{(3)})) \end{aligned} \tag{5.11}$$

where $\parallel$ denotes the concatenation operation, ReLU the rectified linear unit activation function [92], and $\mathbf{W}$ and b are the policy parameters.

4. A balanced portfolio always divide its resources equally, e.g., for a portfolio with N assets, all the portfolio weights are equal to $\frac{1}{N}$.

The maximum number of epochs is set to 10,000 as the performance ceases to improve after approximately 8,000 epochs. The network is trained using the Adaptive neural network optimization algorithm (Adam) [66], with learning rate $= 5 \times 10^{-5}$ as optimizer and weight decay [80]. Adam is a common choice for training neural networks and we use weight decay to prevent overfitting by penalizing weights with higher magnitudes. We opt for this simple neural network architecture as the amount of data available is limited, because we are working with monthly data. However, this architecture is still able to solve common RL test problems, such as OpenAI gym [5] [17].

Other architectures such as Long-Short Term Memories (LSTMs) [53] and gated recurrent units (GRUs) [22] are often used for time-series. However, the fine tuning of parameters can be very time-consuming. The usage of FC layers is fairly common in RL as the main focus is the optimization of a policy function. For example, FC layers were used to evaluate TRPO and PPO when they were proposed. Finally, a FC network is the most general architecture and does not introduce a strong inductive bias [8], which can affect the performance of the baselines.

The reward is set to the portfolio return obtained during the last time step and the state $\mathbf{S}_t$ is modelled as the monthly assets' returns from the past 6 months. After training, the agent obtained an annualized return of 10.6 %, slightly higher than the 9.98% from the benchmark.

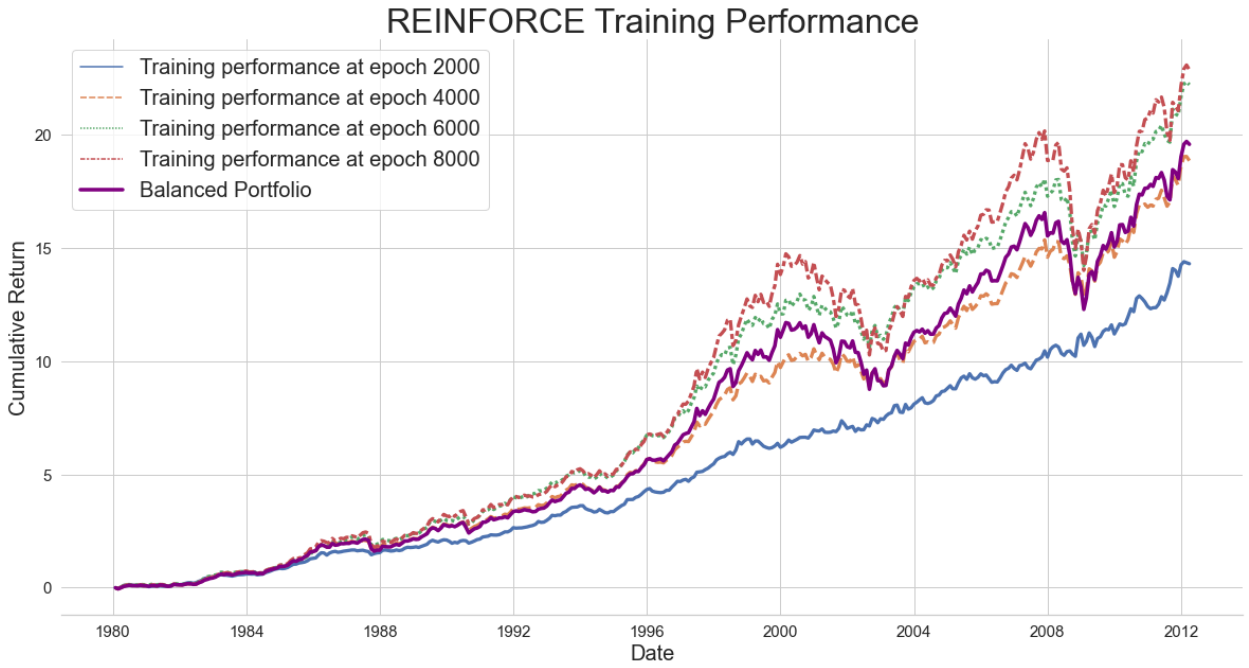

Figure 5.1: Reinforce training performance during different stages of training.

It is clear that it takes several thousands epochs before the agent is able to beat the

5. OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms.

benchmark. In fact, the REINFORCE algorithm is known for its poor sample efficiency. Therefore, it is common to take thousands of epochs until the agent performs better than a random policy or a specific benchmark.

In the next subsection we present the validation results of a significantly more sample-efficient method, the Proximal Policy Optimization algorithm.

5.6.3 PPO Evaluation

For the purpose of evaluating the PPO performance, we repeat the same experiment performed on a small set of market data. The pseudo-code for PPO is presented in Algorithm 2.

Algorithm 2 Proximal Policy Optimization (PPO)

Inputs: N assets, states $\mathbf{S_t} \in \mathbb{R}^{N \times M}$, prices $\mathbf{P}$, exploration parameter σ_e
Initialize: Policy parameters θ_0, Critic parameters ϕ_0, initial allocation weights $\mathbf{w}_0$
Initialize trajectory buffer: $\mathcal{D}$
Returns: Optimal weights θ^*
while θ has not converged or maximum number of epochs has not been reached **do**
  for $k = 0, 1, 2, ...$ **do**
    Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment for T timesteps.
    Compute rewards R_t.
    Compute advantage estimates, $\hat{A}_t$ based on the current value function V_{ϕ_k}.
    Update the policy by maximizing the PPO objective (Eq. 2.36):

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(L(s, a, \theta_k, \theta)\right),$$

    using a stochastic gradient ascent method.
    Fit value function by minimizing the mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2,$$

    using a stochastic gradient descent method.
  end for
end while
$\theta^* = \theta$

The assets are the same as the ones used in the REINFORCE experiment, as well as the architecture of the neural network, the reward function and the optimizer used. The only difference is that the PPO model also employ a critic network, which is almost equal to the

actor network. The only difference is on the final layer, as the critic outputs the estimate of the value function instead of the assets' weights. Figure 5.2 shows the results.

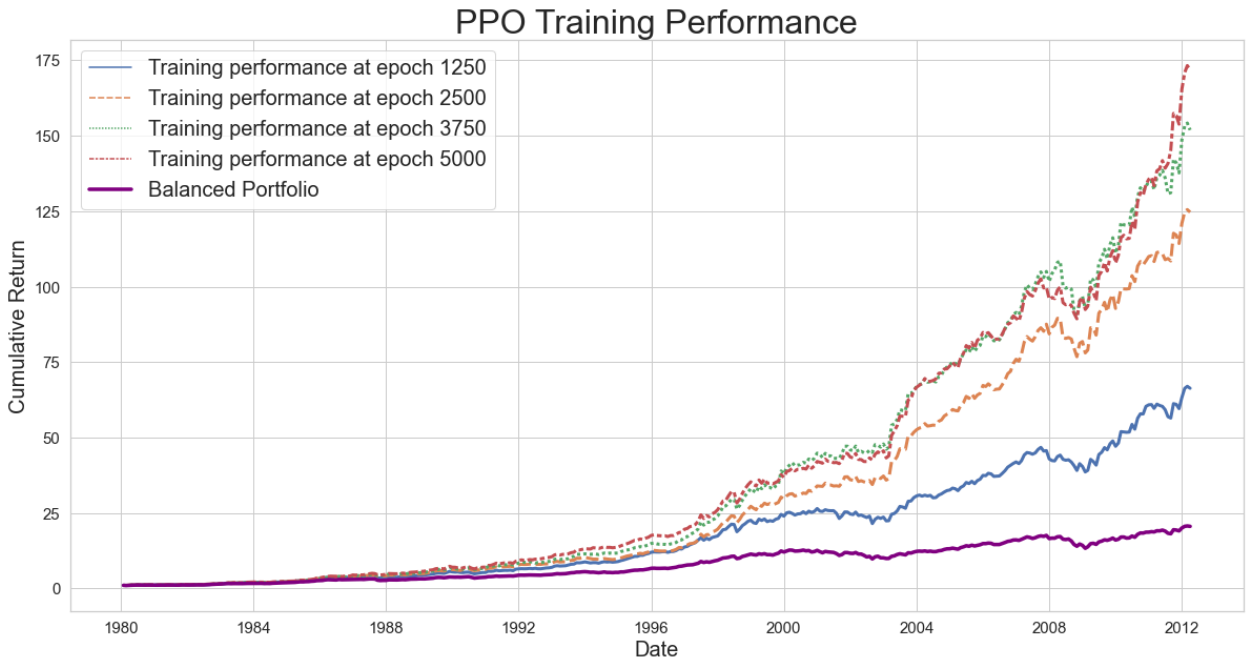

Figure 5.2: PPO Training Performance at different epochs

It becomes clear that the PPO can outperform the benchmark, even at the beginning of the training. It achieved an annualized return of 17.5%, fairly superior to the one achieved by the REINFORCE. Moreover, the difference in terms of sample efficiency is also quite noticeable. Even after 8000 iterations, the REINFORCE agent still underperforms the PPO agent trained for only 1250 epochs.

Nevertheless, more experiments are needed in order to draw solid conclusions, as training performance can often be misleading. Financial markets have remarkably complex dynamics, and even a model that performs well on out-of-the-sample data in the past can fail to achieve good results in the future. Therefore, it is important to understand that impressive training results do not imply favorable out-of-the-sample performance. In fact, even good backtesting results do not guarantee profits in real applications.

5.7 Proposed Extensions

Despite all proposed extensions to the RRL model, none of them explicitly considered each asset in a portfolio separately, i.e., the weight for a given input parameter is the same irrespective of the asset. Different types of assets often have different behaviours that should be taken into account. For the works that used neural networks to approximate the policy, if

the asset class is ignored, the model loses its interpretability. With this in mind, we propose two extensions in this direction.

First, we consider a matrix of weights $\Theta \in \mathbb{R}^{N \times M+2}$, instead of the conventional weight vector $\theta \in \mathbb{R}^{M+2}$. Additionally, we propose training different agents for different asset classes or different economic sectors, and an agent with all assets to determine the final classes' weights.

Finally, Moody and Saffell does not refer to any kind of regularization, but we implement L2-regularization (with $\lambda = 10^{-2}$) during training to avoid overfitting.

5.7.1 Matrix form

The components of the weight vector $\boldsymbol{\theta}$ can be interpreted as the importance of each state constituent. For example, if the largest weight component corresponds to the previous monthly return, it means that this quantity has the most influence in the decision process [6]. In other words, it is possible to understand what drives the final allocation of the model.

This can be particularly valuable in the financial domain, as most of the time real capital is being invested based on computer algorithms. Therefore, the interpretability is a major advantages of the RRL when compared to the previous deep learning models. Before committing capital and risking loss, an investor often wants to understand what drove the investment decision.

REINFORCE and PPO depend on neural networks to approximate their policy. Despite their outstanding performance in several domains such as as text [32], images [71], and graphs [67, 116], they are still a black-box type of model. Their behavior and underlying mechanisms are difficult to explain, in spite of recent efforts in promoting interpretability of deep learning models [19].

According to the work of Lipton [78], the word 'interpretability' in the context of machine learning can be have two different meanings. The first is concerned with how the model works (which is referred to as transparency) and the second consists of understanding and extracting information from trained models. The RRL model is considered interpretable in both senses.

The simplicity of RRL can be one great benefit when using it in a real-world application where decision makers might need to understand the model's behaviour. However, the multi-asset version proposed by Moody et al. assumes that the policy can be described as a vector $\boldsymbol{\theta_t} \in \mathbb{R}^{M+2}$, where M is the number of market variables for each asset, which implies that a given state component has the same importance irrespective of the nature of the asset.

This simplification can hinder the model's ability to deal with assets with different dynamics. For example, let θ_2 be the weight associated with the return two time steps ago. For each asset i, the contribution of $r_{i,t-2}$ would be given by $r_{i,t-2} \times \theta_2$. It is clear that for every asset, $r_{i,t-2}$ has the same impact on the derived portfolio weight.

However, one of the key aspect of portfolio risk management is including assets from a

6. This is only true if the inputs have the same scale. Otherwise, higher weights might not be indicative of greater importance.

variety of different classes, e.g., bonds, equities, precious metals and commodities. These assets are usually exposed to different risks. Since risk and return are closely related, "safer" investments (lower volatility) like bonds usually have lower returns when compared to more volatile alternatives, such as equities. Thus, one should not expect the past returns of different assets to be treated with the same importance.

With this in mind, we propose a simple alternative to incorporate flexibility for each asset. Instead of approximating the policy using a vector $\boldsymbol{\theta} \in \mathbb{R}^{M+2}$, we let each asset-indicator pair have a different weight. The policy becomes a matrix $\boldsymbol{\Theta} \in \mathbb{R}^{N \times M+2}$ defined as:

$$\boldsymbol{\Theta} = \begin{bmatrix} b_1 & \theta_{1,1} & \dots & \theta_{1,M} & \omega_1 \\ b_2 & \theta_{2,1} & \dots & \theta_{2,M} & \omega_2 \\ \vdots & \dots & \ddots & \vdots & \\ b_N & \theta_{N,1} & \dots & \theta_{N,M} & \omega_N \end{bmatrix}_{N \times M+2} \tag{5.12}$$

where $\theta_{i,j}$ represents the weight of the j-th state constituent of the i-th asset, and b_i and ω_i are the corresponding bias and past allocation weight, respectively.

The main difference when compared to the vector version is that the allocation now depends on the Hadamard product [7] between $\boldsymbol{\Theta}$ and $\boldsymbol{S}_t$. More specifically, it depends on the column sums of these two matrices. We can define a score function of asset i as:

$$f_i = \text{sigmoid}_i(\sum_j (\boldsymbol{S_t} \circ \boldsymbol{\Theta_t})_{ij}). \tag{5.13}$$

We can write the column sums of the Hadamard product in terms of the diagonal of a matrix multiplication as follows [109]:

$$\sum_j (\boldsymbol{S_t} \circ \boldsymbol{\Theta}_t)_{ij} = (\boldsymbol{\Theta}^T \boldsymbol{S}_t)_{ii} = \text{diag}(\boldsymbol{\Theta}^T \boldsymbol{S_t}). \tag{5.14}$$

The decision function that determines the allocation on asset i at time t becomes:

$$\mathbf{w}_{t,i} = \frac{\exp(\text{sigmoid}_i(\text{diag}(\boldsymbol{\Theta}^T \boldsymbol{S_t}))}{\sum_{i=1}^{N} \exp(\text{sigmoid}_i(\text{diag}(\boldsymbol{\Theta}^T \boldsymbol{S_t}))}. \tag{5.15}$$

The weights can be updated using the same procedures described in Eq.4.11 and 4.12, i.e., updating the parameters according to their influence on the performance function.

The importance of having different weights for different assets is illustrated in the next two figures. We report the out-of-sample performance of both the RRL vector form and weight form in allocating a portfolio with only two assets (gold and commodities) in Fig. 5.3. The details of the experiment are presented in the Appendix B.

7. For two matrices $\mathbf{A} = \{a_{ij}\}$ and $\mathbf{B} = \{b_{ij}\}$ with the same dimensions $m \times n$, their Hadamard product is the $m \times n$ matrix of element wise products $\mathbf{A} \circ \mathbf{B} = \{a_{ij} b_{ij}\}$.

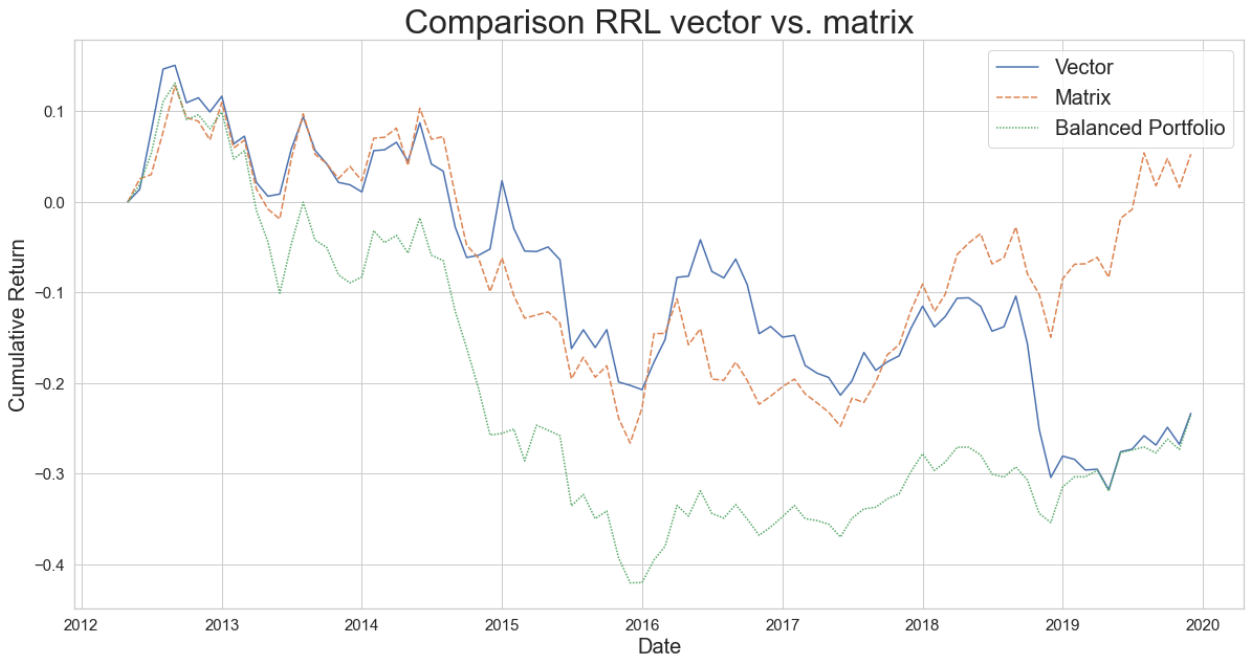

Figure 5.3: Comparison of performance between RRL using a vector of weights and using a matrix of weights.

Note that only the matrix agent had a positive performance in the period. In order to understand the difference behaviour of the two RRL variants, we can analyze the weights for each model, presentend in Fig. 5.4.

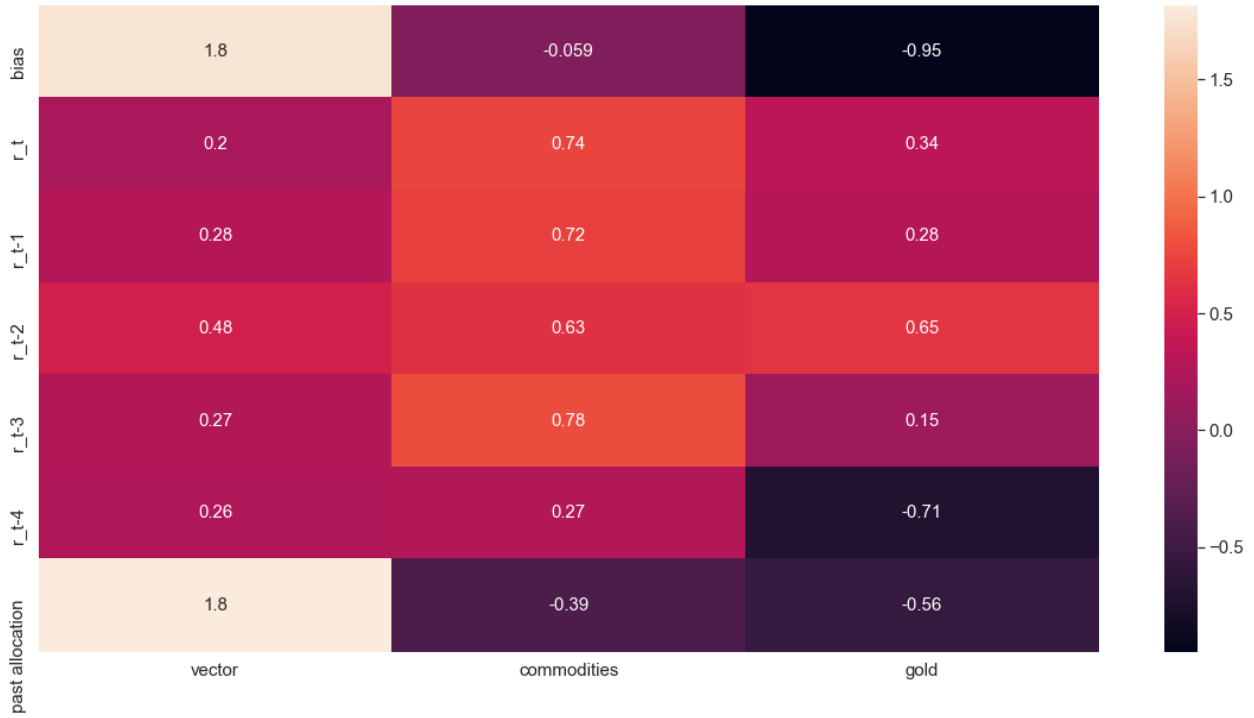

Figure 5.4: Weights of the RRL vector and RRL matrix presented as a heatmap.

All the weights in the vector variant are positive, meaning that the agent will act on the asset's momentum. In other words, if prices of a specific asset continue to increase, the tendency to allocate more in that asset is also higher. This is also the case for the commodities' weights for the matrix variant. However, note that the weights for the most recent time lags are significantly higher, making the short-term momentum response stronger. At the same time, the weights associated with gold are very different. The bias term is significantly negative, which creates a resistance in allocating in this asset. Also, the price momentum has much less impact due to the negative weight on the r_{t-4} component.

These subtle differences help to illustrate how the matrix form of RRL can provide a better model for allocating assets.

5.7.2 Decoupling

The second proposed extension is based on decoupling the allocation process in different classes of assets or sectors. For example, we can divide stocks into different sectors (technology, energy, utilities, etc) if we were to trade only equities from a specific stock market, or divide assets into classes, such as bonds, equities index, commodities, gold, if we are allocating macro assets.

In the decoupling framework, an independent agent is trained for each asset class, and an additional model is trained with all asset classes. Essentially, the objective of the decoupled models is to allocate all the resources within its asset class, and the coupled agent decides

the final allocation percentage on each asset class. This procedure is illustrated in Fig. 5.5.

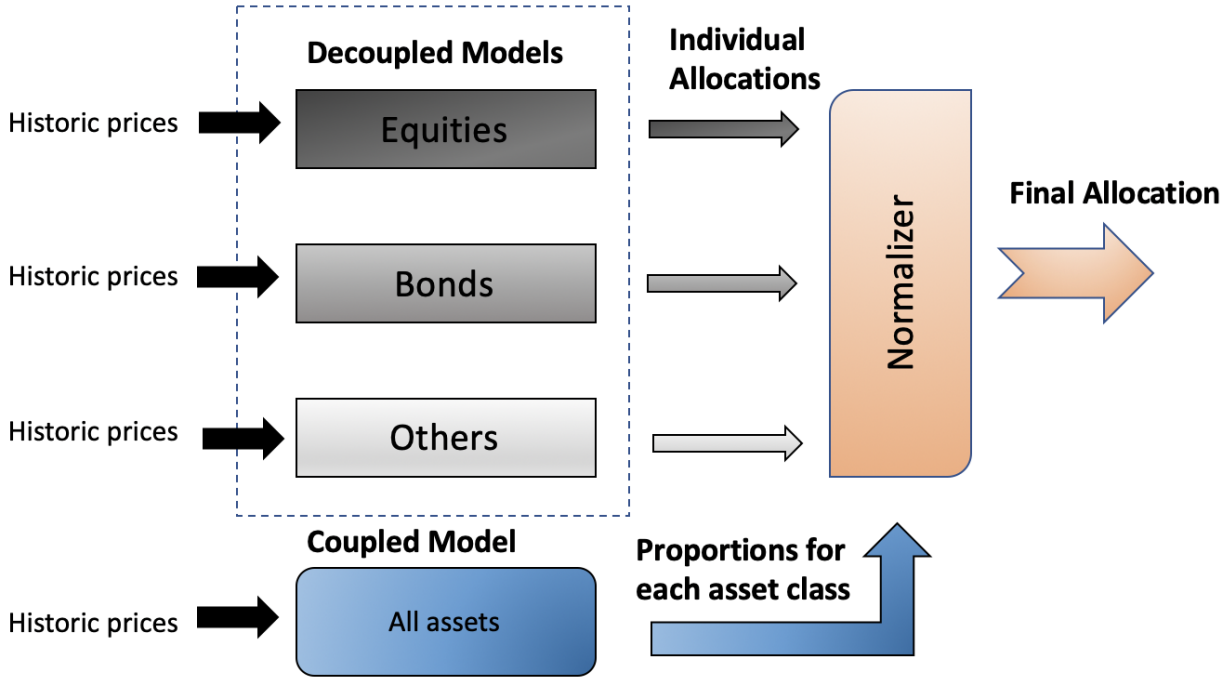

Figure 5.5: Decoupled version architecture.

For example, the equities model will always be allocated 100% in equities, but the coupled version might decide to only invest 10% in equities. Therefore, the final allocation will be 10% in equities according to the proportions defined by the decoupled equities model. If the decoupled version had allocated 50% in German equities and 50% in Canadian equities, the final allocation would be 5% in each.

The decoupling processing allows flexibility with respect to two characteristics: the individual dynamics and risk exposure of each asset class. With regards to risk exposure, we can be aggressive on the decoupled model's allocation by setting a high temperature for this agents softmax function but more conservative on the final allocation by setting a lower temperature for the coupled model. For example, even if the bond's model decides to allocate 100% to a specific market, the coupled version can scale it down according to its convictions.

As the decoupled models are independent, different indicators and time lags can be defined for each of them. Equities' prices and dynamics tend to vary faster than bonds, so longer time lags for equities might not be as informative as they are for bonds. Another advantage is being able to incorporate indicators of different asset classes into the model.

These characteristics were not used in this work, but they can be useful when using states that contain more complex indicators, instead of just historical prices. For instance, bonds and equities tend to have opposite price movements [8]. By incorporating a bond price indicator

8. This is not always true. For instance, in the 2020 sell-off due to the COVID-19 pandemic, both equities and bonds prices fell sharply.

into the decoupled equities model, the agent could potentially achieve a better performance.

The algorithm for training the RRL using the matrix form is presented in Algorithm 3.

Algorithm 3 Recurrent Reinforcement Learning - Matrix version (RRLM)

Inputs: N_d subset of assets, states $s_t \in \mathbb{R}^{N \times M}$, prices p_t, learning rate α
Initialize: Policy parameters Θ_0, initial allocation weights $\mathbf{w}_0$
Returns: Optimal weights Θ^*
while Θ has not converged or maximum number of epochs has not been reached **do**
  $G = 0$
  for $t = 1, ..., T$ **do**
    get state s_t
    get asset allocation from policy according to Eq. 5.15
    evaluate allocation $J_t = J(\mathcal{R}_t)$
    accumulate the performance value $G \longleftarrow J_t$
  end for
  update policy parameters Θ according to equations 4.11 and 4.12
end while
$\Theta_d^* = \Theta$

The decoupled methodology is presented in Algorithm 4.

Algorithm 4 Recurrent Reinforcement Learning - Decoupled version (RRLD)

Inputs: N assets, states $s_t \in \mathbb{R}^{N \times M}$, prices p_t, learning rate α
Initialize: List of asset classes or financial sectors L
Returns: Final allocations $\mathbf{w}^*$, trained decoupled weights Θ_d^*
for $i = 1, ..., |L|$ **do**
  current_class = L[i]
  train RRL model according to Algorithm 3 with N_d = current_class
  store optimum asset class weight Θ_d^* and final allocations $\mathbf{w}_d$
end for
train RRL model with all assets according to Algorithm 3
store weights $\mathbf{w}_{all}$
normalize individual allocations $\mathbf{w}_d$ using the proportions from $\mathbf{w}_{all}$
$\mathbf{w}^*$ = normalized allocations

The original RRL proposed by Moody et al. [88, 90, 91] uses an online version to optimize the trading decisions. However, since the computational cost is low compared to deep learning techniques, we employ batch gradient ascent as the optimization method.

5.8 Summary

In the first section of this chapter, we formalized our financial market environment in mathematical terms. Using this methodology, we enable RL agents to interact with the environment and optimize a target portfolio. The agents' actions are defined in terms of the available assets, and the states are derived from market information and past actions. In this framework, we have flexibility to define the agent's goal by specifying different reward functions. Subsequently, we identified the most appropriate RL methodology to apply to the asset allocation problem. We conducted preliminary experiments to validate the baseline agents.

We proposed two extensions to the RRL model, and showed that having different policy parameters for each asset can improve the performance. More detailed and complete experiments are presented in the next chapter. These allow us to more fully evaluate the value added by these proposed extensions.

Chapter 6

Results

In this chapter, we presents the results for the classical portfolio and reinforcement learning agents described in previous chapters using real market data.

6.1 Data Description

In this work, we use financial data from 26 different assets that can be divided into three different classes:

— **Equities -** Country-level stock indexes from 12 major developed economies: Australia (S&P/ASX 200), Canada (S&P/TSX Composite), France (CAC 40), Italy (Milano Italia Borsa), Spain (IBEX 35), Germany (DAX), Switzerland (Swiss Market), Japan (Nikkei 225), Sweden (OMX Stockholm 30), Norway (OBX), United Kingdom (UK 100) and United States (S&P 500). These indexes represent the performance of the largest public traded companies listed in their respective stock exchanges.

— **Bonds -** Sovereign 10-year bonds from the same 12 countries. These assets are debt obligations that are issued by national governments to support spending. They are considered safer than equities as their volatility is lower, and the default risk of a sovereign bond from a developed economy is very low.

— **Others (Alternative investments) -** The last two assets are gold and commodities (represented by the S&P GSCI index), aggregated into the "others" class. Despite being in the same class, they are quite different. Gold is considered a safe haven asset [10] [1], while commodities are considered a good investment in times of an optimistic economic outlook.

The data consists in monthly prices of the assets presented above from January of 1980 to January of 2020.

1. Safe havens are financial assets that investors turn to to protect themselves or even profit from periods of market turmoil.

6.2 Classic Portfolio Optimization

Both Mean-variance and Black-Litterman portfolios depend on estimating the covariance matrix and the expected returns. The Black-Litterman portfolio uses a Bayesian approach to form new estimations of these quantities. On the other hand, the standard mean-variance portfolio does not provide any specific technique on how to estimate them. Coming up with good estimators for these quantities can be challenging, and is currently the main limitation of these models. Their optimization procedure is robust with strong mathematical guarantees, but estimation errors can have significant impact on the model's performance.

In the following two subsections we present two alternative methods to estimate returns and the covariance matrix.

6.2.1 Expected Return

The simplest and most intuitive approach to estimate expected returns is to calculate the average of historical returns, which can be defined for an asset i over a period T as:

$$\bar{R}_i^s = \frac{1}{T}\sum_{i=1}^{T} R_{it}^s \tag{6.1}$$

where N denotes the total number of assets and $1 \leq i \leq N, 1 \leq t \leq T$.

One of the limitations of the simple historical returns estimate is that it assigns equal importance for all returns. However, the most recent information is often more relevant that older information. As an alternative to simple historical returns, we can use exponentially-weighted average (EWA) returns, which gives higher weight to more recent data. We can define the EWA as:

$$EWA_t = \frac{\sum_{i=0}^{t} k_i R_{t-i}^s}{\sum_{i=0}^{t} k_i}\,, \tag{6.2}$$

where $k_i = (1 - \frac{2}{s+1})^i$, and $s \geq 1$ is the term that controls the decay of the series. Therefore, the further away the observation is compared to most recent period, the lower the weight it receives.

6.2.2 Covariance Estimation

Similar to the sample mean, the sample covariance matrix is the simplest method to estimate the covariance. Besides being easy to compute, it is also an unbiased estimator. However, this estimation can present significant errors if the number of assets is large compared to the number of observations [61].

One alternative is using shrinkage to reduce estimation errors. The key idea of this technique is combining the estimated sample covariance with a structured estimator F. The estimator becomes:

$$\hat{\Sigma} = \delta F + (1 - \delta)S\,, \tag{6.3}$$

where δ is the shrinkage constant,which controls the compromise between the two estimators and S is the sample covariance matrix. This process tends to pull the most extreme coefficients toward more central values, reducing estimation errors. In this thesis, we use the Ledoit-Wolf shrinkage as the shrinkage method [72].

Let s_{ij} denote the entries of the sample covariance matrix S. We can write the sample correlations (ν) between assets i and j as

$$\nu_{ij} = \frac{s_{ij}}{\sqrt{s_{ij}s_{jj}}}\,. \tag{6.4}$$

The sample correlations are:

$$\bar{\nu} = \frac{2}{(N-1)N}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\nu_{ij}\,. \tag{6.5}$$

Finally, the entries of the matrix F are:

$$f_{ii} = s_{ii} \text{ and } f_{ij} = \bar{\nu}\sqrt{s_{ii}s_{jj}}\,. \tag{6.6}$$

In order to define the shrinkage coefficient, we need to first define three components. The first element is the error of the shrinkage target $\hat{\chi}$.

$$\hat{\chi} = \sum_{i=1}^{N}\sum_{j=1}^{N}(f_{ij} - s_{ij})^2\,. \tag{6.7}$$

The other two components are estimates of the sum of asymptotic variances of the entries of the sample covariance matrix $\hat{\phi}$ and the sum of asymptotic covariances of the entries of the shrinkage target with the entries of the sample covariance matrix $\hat{\rho}$, both scaled by $\sqrt{T}$.

$$\hat{\phi} = \sum_{i}^{N}\sum_{j}^{N}\hat{\phi}_{ij} \quad \text{with} \quad \hat{\phi}_{ij} = \frac{1}{T}\sum_{t=1}^{T}\{(R_{it} - \bar{R}_i)(R_{jt} - \bar{R}_j) - s_{ij}\}^2\,, \tag{6.8}$$

$$\hat{\rho} = \sum_{i=1}^{N}\hat{\phi}_{ij} + \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}\frac{\bar{\nu}}{2}\left(\sqrt{\frac{s_{jj}}{s_{ii}}}\hat{\omega}_{ii,ij} + \sqrt{\frac{s_{ii}}{s_{jj}}}\hat{\omega}_{jj,ij}\right)\,, \tag{6.9}$$

where

$$\hat{\omega}_{ii,ij} = \frac{1}{T}\sum_{t=1}^{T}\{(R_{it} - \bar{R}_i)^2 - s_{ii}\}\{(R_{it} - \bar{R}_i)(y_{jt} - \bar{R}_j) - s_{ij}\} \tag{6.10}$$

Finally, the optimal shrinkage constant is given by:

$$\hat{\delta} = \max\left\{0, \min\left\{\frac{\hat{\kappa}}{T}, 1\right\}\right\}. \tag{6.11}$$

The new covariance estimate can then be calculated by plugging $\hat{\delta}$ back into (6.3). In order to assess if using different methods to estimate returns and covariance improves the portfolio performance, we conduct an experiment with both methodologies and compare their results.

Table 6.1: Different investor profiles, their allocations, and volatility constraints. The volatility constraint is based on the historical volatility from the in-sample period (1980-2012).

Investor Profile	**Equities Allocations (%)**	**Bonds Allocations (%)**	**Commodities and Gold Allocations (%)**	**Historical Volatility**	**Maximum Volatility Constraint**
Conservative	15%	80%	5%	4.6%	5%
Moderate	45%	40%	15%	7.5%	10%
Aggressive	65%	10%	25%	10.5%	15%

6.2.3 Parameter optimization

One way to evaluate portfolio allocation models is to considered different settings. For example, what would be the performance of a portfolio derived from a given method for a given risk level? With this in mind, we can define three "types" of investors [2], presented in the Table 6.1.

A conservative investor is more risk averse, and he/she would allocate most of the available capital to government bonds, as these are inherently safer than equities and commodities. Therefore, this investor keeps 80 % of the portfolio invested in bonds at all times, and the remaining 20% is divided into equities (15 %) and gold/commodities (5%).

A investor with moderate risk profile would be willing to experience higher volatility when compared to the conservative investor, thus, he/she would invest more capital in equities and gold/commodities. For this reason, the distribution is 45% equities, 40 % bonds and 15% commodities/gold.

Finally, the aggressive investor would be willing to support even higher levels of volatility, so the allocation percentages are set to 65% equities, 10% bonds and 25% commodities/gold. These proportions are based on [13].

We use a volatility constraint in the optimization procedure of the mean-variance and Black-Litterman portfolios, as presented in (3.19). In other words, we optimize these models using the "risk tolerance" (maximum volatility contraint) as the σ_p parameter in (3.19).

Additionally, instead of defining an upper bound for the maximum volatility allowed, we can define an additional unconstrained version that maximizes the Sharpe ratio (Eq. (3.21)). This is referred as the "unconstrained" investor as there are no impositions on the amount allocated in each asset class.

It is important to assess how the portfolio performance varies for different risk profiles. A rational investor would only accept more risk if it results in a higher expected return. However, the relative performances of mean-variance portfolios do not necessarily accord with target risk profiles. Estimating returns is challenging, so an investor might be exposed to more risk than anticipated.

In order to assess the performance of different parameter estimates, we evaluate three different portfolios for the risk profiles presented in Table 6.1. The in-sample period was set

2. Here, we aggregated gold and commodities due to how we divided the data. In practice, gold is not often seen in aggressive portfolios. Nevertheless, this does not affect the main goal of this experiment.

from January 1980 to April 2012, and the out-of-sample period from May 2012 to January 2020. The portfolios were rebalanced every month, i.e., the most recent available data was incorporated to estimate the next month's allocation. Fig. 6.1 shows the out-of-sample performances of Markowitz, Markowitz "naive" (expected return and covariance are estimated by their sample quantities), and Black-Litterman portfolios.

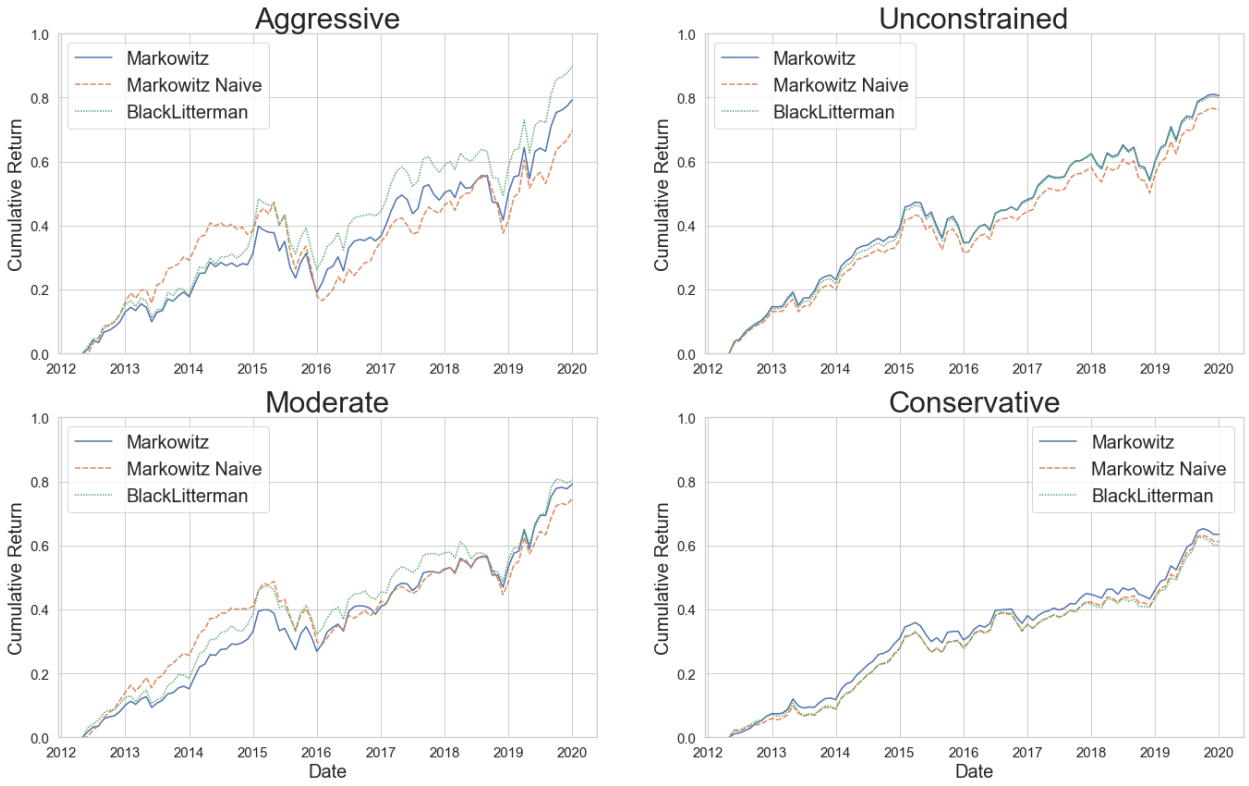

Figure 6.1: Out-of-sample performance for Markowitz portfolio with exponentially-weighted moving average and covariance shrinkage, Markowitz "naive" (sample mean and sample covariance) and Black-Litterman portfolios for 4 different risk profiles.

It is important to note that comparing cumulative return does not represent the whole picture of a portfolio performance. For instance, we would expect lower volatility for the conservative profile and, as a consequence, lower cumulative return. In order to compare their risk-adjusted returns, we evaluate the Sharpe ratio of each portfolio, presented in the Table 6.2.

There are two significant aspects to notice in the results of Table 6.2. First, the naive version of the mean-variance achieves a lower Sharpe ratio than the version with shrinkage and EWA returns. Second, the conservative portfolios have significantly higher Sharpe ratios when compared to their more aggressive counterparts. This may imply that the higher returns obtained by these portfolios do not compensate the additional risk (volatility) exposure.

Table 6.2: Sharpe ratios calculated over one year of the evaluated portfolios.

	Markowitz	**Markowitz Naive**	**Black Litterman**
aggressive	0.87	0.78	**0.92**
moderate	1.20	1.12	**1.20**
conservative	**1.53**	1.47	1.39
unconstrained	**1.27**	1.25	1.26

6.3 Reinforcement Learning Models

The RL framework outlined in the previous chapter does not depend on estimates of returns and covariance. On the other hand, there are other parameters that must be defined. These include the number of time lags that will compose the state space, the exploration parameter, the discount rate, the architecture of the neural network, and the learning rate.

As the number of possible combinations is very large, we need to limit the scope and define the parameters that can be adjusted. These parameters are: the number of lags (M) used to model the state ($\mathbf{S}_t$) and the exploration parameter (σ_e).

All the other parameters are fixed, and they were obtained by validating the RL algorithms on different OpenAI gym [3] environments [17]. More specifically, we solved two environments ("Pendulum-v0" and "LunarLanderContinuous-v2") using PPO and REINFORCE. These environments contain continuous state and action spaces, similar to our model for the environment.

For both algorithms, we use a discount factor of $\gamma = 0.99$, as the rewards obtained in the future are approximately as good as the rewards obtained in the initial time steps. This is the case since we are modelling rewards as the return of the portfolio. For the PPO, we use the recommended clipping factor $\epsilon = 0.2$ from the original paper [104].

The neural network used was the same used to validate the agents in Section 5.6. Figure 6.2 presents the architecture of the neural network used to model the actor. The input corresponds to all M time lags of the N assets plus the previous allocation, and the output layer has the same dimension as the number of assets. The final output is normalized by a softmax function.

The PPO model also employs a critic network, which has an almost identical architecture as the one Fig. 6.2. The only differences are in the output layer, which contains only one neuron that is used to calculate the advantage as defined by Eq. 2.25, and there is no softmax in the output.

The architectures of the actor and critic are very similar for simplicity, as tuning multiple neural networks can be a time-consuming task. Some researchers even propose using genetic algorithms to [75, 115] to design them.

3. OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms.

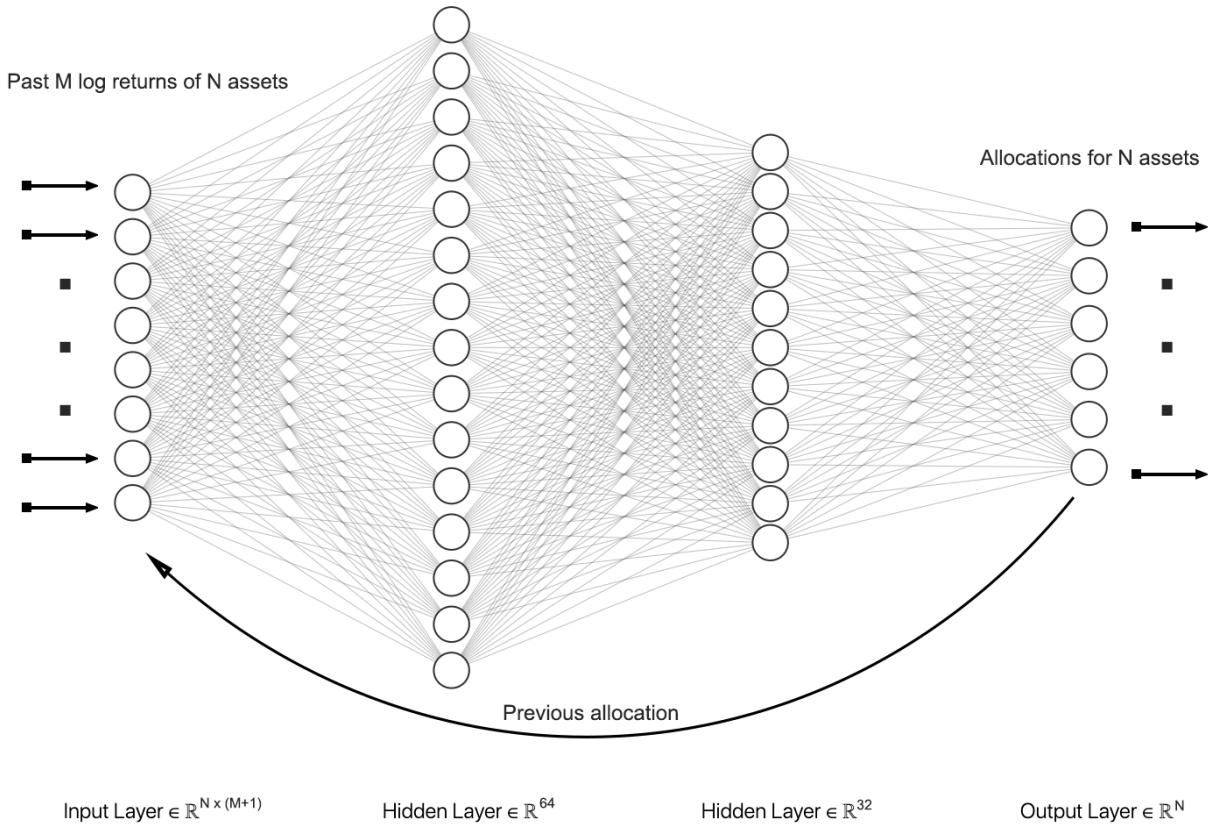

Figure 6.2: Illustration of the neural network architecture used. Not all nodes are represented for the sake of clarity.

For the purpose of selecting the optimum parameters, we employ the same experiments for both REINFORCE and PPO. These consist in training for up to 10000 epochs using 80% of the data (January 1980 to April 2012), and testing in the 20 % remaining (May 2012 to January 2020).

First, we test each model with 3 different time lag setting, $M \in \{3, 6, 12\}$ Next, we select the M which yields the best performance, and vary the exploration parameter $\sigma_e \in \{0.1, 0.25, 0.5\}$. We also tested the remaining combinations, the results can be found in the Appendix A.

The REINFORCE results are presented in Fig. 6.3. Note that the only variant that was able to beat the benchmark was the one with $M = 6$ and $\sigma_e = 0.1$.

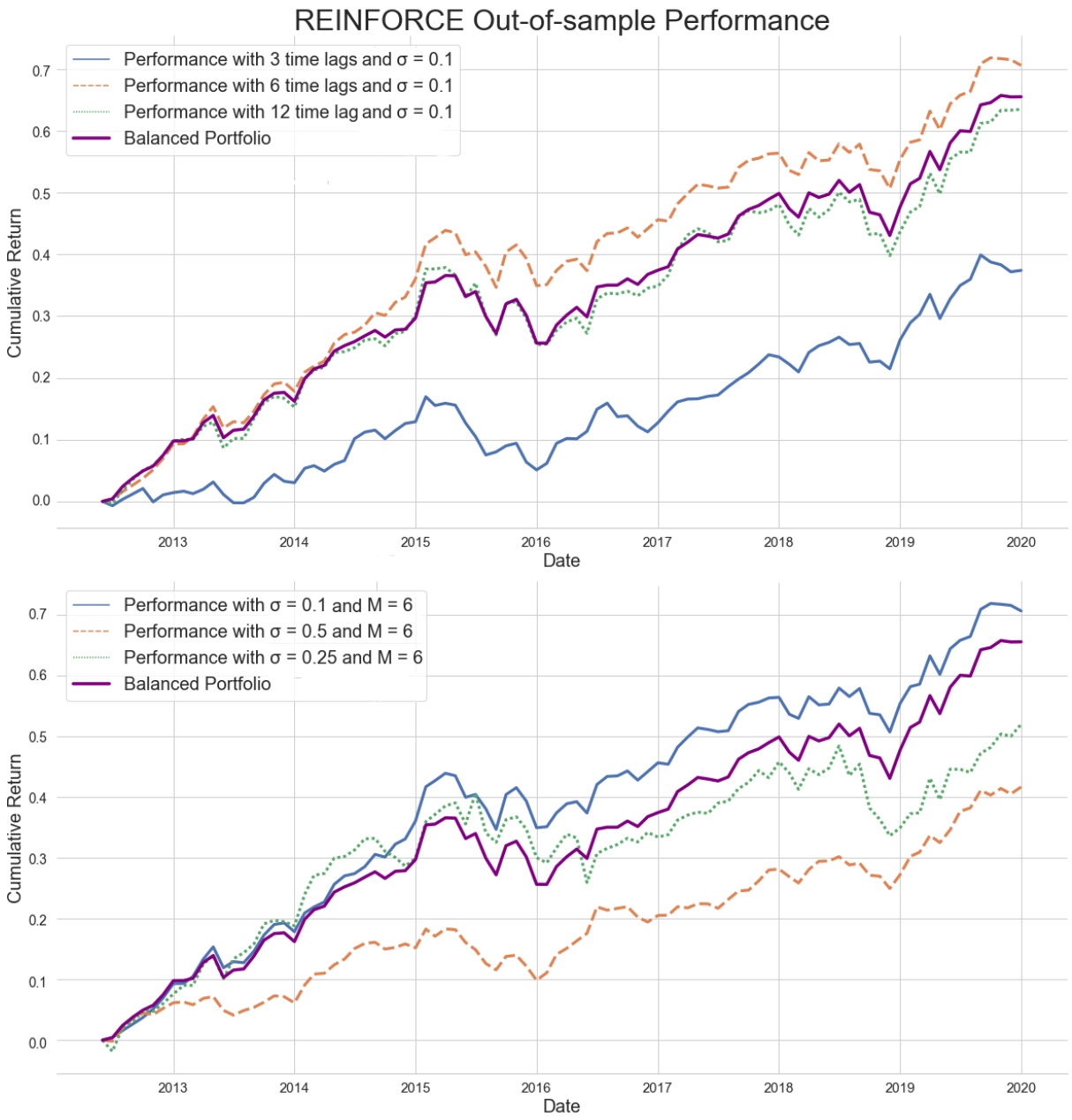

Figure 6.3: Out-of-the-sample results for the REINFORCE algorithm under different hyper-parameter configurations.

One of the main limitations of the REINFORCE is its performance variance, as small changes in the policy can result in dramatic changes in the performance. Since the number of epochs were limited to 10,000, which was when the model performance stopped improving during training, agents with higher σ_e may have trouble in converging to a local maximum.

This shortcoming may be the reason why the best performance was obtained with the lowest value of σ_e among the values considered. The results for other time lags and σ_e combinations (see Appendix A) also present similar behaviour.

On the other hand, PPO addresses this limitation by using a trust region for the policy update. As seen in Fig. 6.4, the higher σ_e allows the agent to explore different alternatives, obtaining a better performance without deviating abruptly from old policies.

Some implementations of RL models in continuous action spaces define the parameter σ_e as one additional target for their networks. In other words, they use a neural network to approximate the optimum value for σ_e. We also tried using this technique, but the results were not as good as using a fixed σ_e.

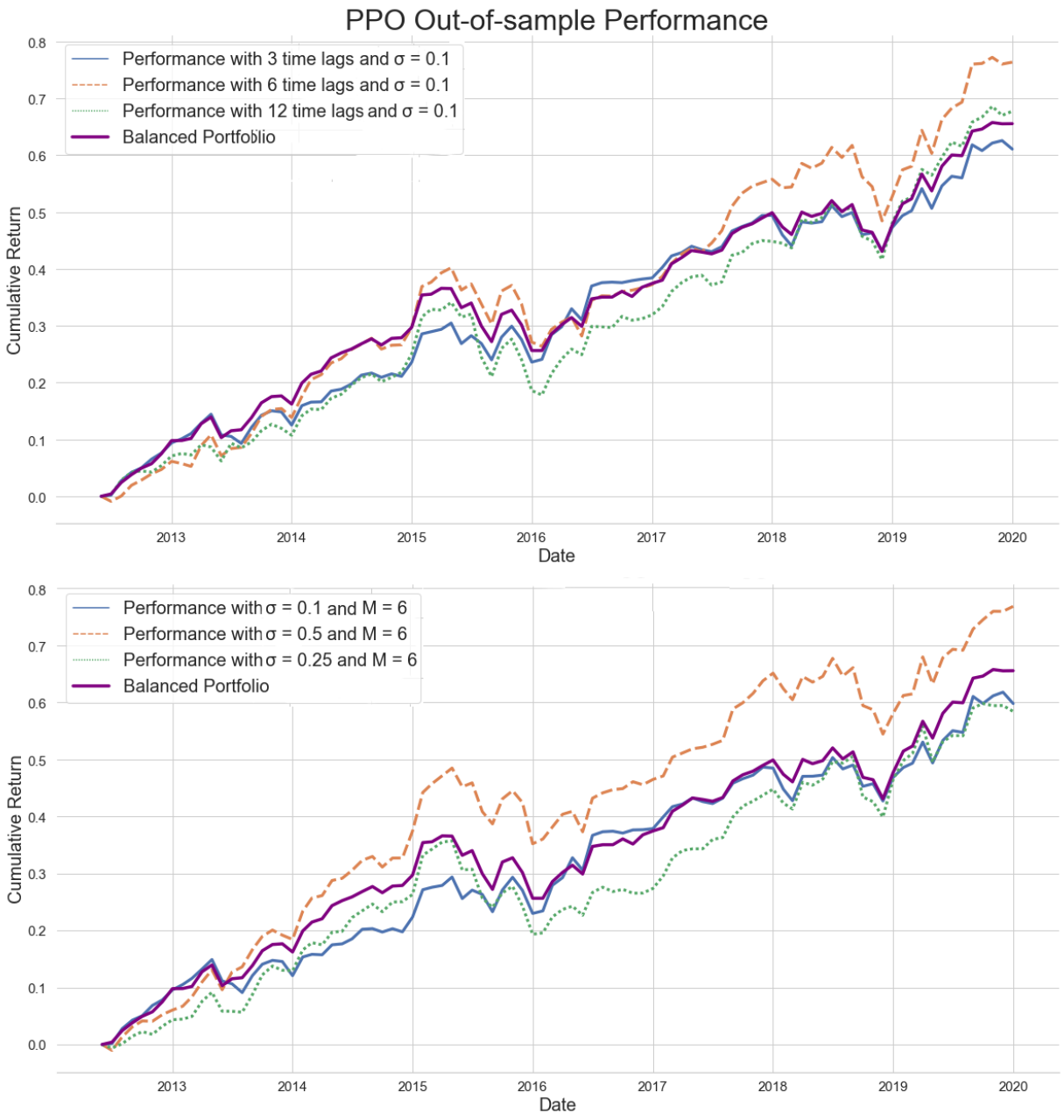

Figure 6.4: Out-of-the-sample results for the PPO algorithm under different hyperparameter configurations.

According to the results presented in Figs. 6.3 and 6.4, we select the past 6 monthly returns to compose the state s_t for both agents, $\sigma_e = 0.5$ for the PPO agent, and $\sigma_e = 0.1$ for the REINFORCE agent.

6.4 RRL

In this section, we give the final experimental details for the recurrent reinforcement learning algorithms used in this thesis.

6.4.1 Exploration vs. Exploitation

In terms of "exploration vs. exploitation" behaviour, RRL works differently when compared to PPO and REINFORCE, as it does not sample its action from a distribution. Nonetheless, it is also possible to add an exploration component by incorporating a noise variable ϵ_t in the decision function, which then becomes:

$$\mathbf{w}_t = \mathbf{f}(\boldsymbol{\theta}_t; \boldsymbol{S}_t; \epsilon_t). \tag{6.12}$$

The magnitude of the noise variance controls the trade-off between exploration of the strategy space and exploitation of a learned policy, and it can be gradually reduced over time during training.

However, according to Moody and Saffell [88], adding the noise term provides little advantage for the real financial applications considered, as the data contain significant intrinsic noise.

Different from the exploration vs exploitation trade-off, the number of time lags dynamics is the same for RRL, PPO and REINFORCE. Since $M = 6$ appears the best configuration for both PPO and REINFORCE, we also employ this setting for the RRL.

6.4.2 Other Baselines

As most of the papers in the literature do not include a comparison of their proposed model and models from other authors, we decide to include two RRL baselines which we mentioned in Chapter 4. More specifically, we evaluate the regime-switching recurrent reinforcement learning introduced in [82] (RSRRL), and the "average elitist" update scheme for recurrent reinforcement learning (RRL "elitist") proposed by [125].

According to Zhang and Maringer, the "average elitist" scheme outperformed the "multiple elitist" version. Moreover, it also has less computational cost. As suggested in the paper, we ran each elitist 100 times and initialized the weights using a Gaussian distribution with a mean of 0 and a standard deviation of 0.05. The updates were the same as those described in (4.22) and (4.23).

For the RSRRL, we employed the smooth transition version (4.20). The authors found no concrete evidence to suggest that either transition scheme is superior to the other, but we opted for the smooth version as it allows a certain amount of overlap between the two regimes. We followed the authors' parameter recommendations, using slope parameter $\gamma = 5$, threshold $c = 1$ and the volatity for the transition/indicator variable q_t.

As for all the previous models, the number of time lags M is also set to 6.

6.5 Results

In this section, we present the results for the models described in the previous chapter. The agents were tested on real financial data described in Section 6.1.

6.5.1 Experimental Setting

In order to evaluate the models we designed the following experimental setting:

— Train each model over a duration of 180 months, test in the following 100 months. Shift the training and testing periods 100 months forward.

— Train each for duration 180 months, test in the following 30 months. Shift the training and testing periods 30 months forward.

The total testing duration is 300 months for both settings. The former setting is evaluated over three periods while the latter is evaluated 10 times over the whole testing period. The intuition of testing on different time windows is to present to each agent diverse market conditions and behaviours, which are expected to be encountered in practice. The process used for evaluating the models is illustrated in Fig 6.5

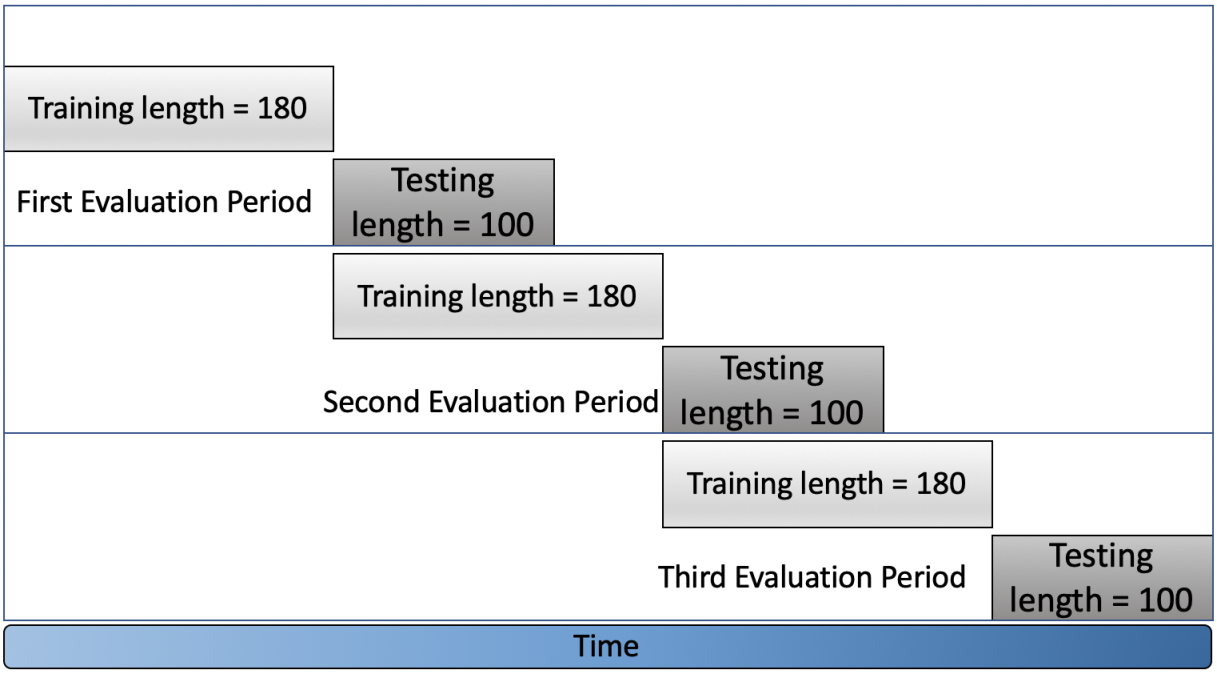

Figure 6.5: Evaluation scheme for time window of length 100.

The final performance is evaluated considering all the testing windows, which result in 300 months out-of-sample.

This is particularly important when we consider that the market is known to have different tendencies depending on the global economic outlook and other variables. For example, in

the investing world, the terms "bull" and "bear" are frequently used to describe opposing market conditions [4], i.e., the overall direction of assets [76].

As a consequence, a good model for asset allocation should be able to make good decisions for both bull and bear markets. Therefore, it is important that the agent experience diverse conditions.

Since the proposed extension to RRL has more parameters, it is also more prone to overfitting the data, especially in an application with not much data available. Thus, in order to evaluate the impact on the amount of data in the performance, we designed a simple experimental setting described in the appendix C. This involved using different amounts of training data; we observe that the performance of the proposed model RRLMD is more sensitive to a reduction in data than RRL.

6.5.2 Experimental Results

In order to evaluate agents in a proper manner, we use seven different metrics defined in Chapter 3. As the RL agents were trained during two different sliding windows of length 100 and 30, we average their performance to reduce the variability between runs of different time windows, and make the comparison more fair. The results for the agents trained using log return as reward function are presented in Table 6.3. In addition, we include the cumulative return obtained over time by all tested agents in Appendix D.

Model	Cumulative Returns	Annualized Volatility	Sharpe Ratio	Max Drawdown	Calmar Ratio	Avg. Drawdown Time (days)	Hit Ratio
RRL Matrix Decoupled	**18.045**	0.136	0.940	-0.287	**0.037**	81	**0.580**
RRL Decoupled	9.187	0.163	0.655	-0.369	0.024	112	0.560
RRL Matrix	9.952	0.149	0.722	-0.246	0.036	111	0.537
RRL [88]	5.340	0.161	0.543	-0.344	0.021	121	0.537
RSRRL [82]	7.612	0.156	0.634	-0.333	0.025	106	0.54
RRL 'Elitist' [125]	3.176	0.145	0.468	-0.333	0.017	122	0.516
PPO	5.108	0.096	0.809	-0.253	0.025	83	0.493
REINFORCE	4.802	0.075	0.985	-0.260	0.024	70	0.520
Balanced	5.484	**0.068**	**1.138**	**-0.225**	0.029	**69**	0.000
MV Unconstrained	5.736	0.080	1.003	-0.272	0.024	77	0.560
BL Unconstrained	5.922	0.081	1.001	-0.270	0.025	76	0.560

Table 6.3: Comparison of performance metrics of reinforcement learning algorithms trained using log returns as the reward function, and classical portfolio techniques. The Sharpe ratios were calculated over a period of one year.

The proposed RRL Matrix Decoupled (RRLMD) achieved the highest cumulative return, Calmar Ratio and Hit Ratio among all the models evaluated. In general terms, it means that it outperformed the benchmark (balanced portfolio) more often than the others, and the excess return obtained was not overshadowed by the slightly higher maximum drawdown.

4. A bull market is a market that is on the rise and the economic outlook is usually optimistic. A bear market exists in an economy that is receding, where most stocks are declining in value.

Moreover, there is also an improvement by considering the proposed extensions separately. The RRL Matrix and RRL Vector Decoupled outperform the standard RRL in all metrics.

Finally, the PPO and REINFORCE algorithms did not achieve impressive performance. In fact, they are outperformed by both the mean-variance and Black-Litterman portfolios.

Next, we present in Table 6.4 the results for the case when the DSR is used as the reward function.

Model	Cumulative Returns	Annualized Volatility	Sharpe Ratio	Max Drawdown	Calmar Ratio	Avg. Drawdown Time (days)	Hit Ratio
RRL Matrix Decoupled	**15.738**	0.135	0.910	-0.208	0.049	84	**0.570**
RRL Decoupled	5.436	0.101	0.795	-0.248	0.027	80	0.510
RRL Matrix	10.900	0.154	0.723	-0.237	0.035	110	0.523
RRL [88]	6.421	0.090	0.941	-0.256	0.028	82	0.533
RSRRL [82]	7.689	0.083	1.088	-0.162	0.046	69	0.513
RRL 'Elitist' [125]	5.171	0.102	0.764	-0.237	0.023	89	0.516
PPO	6.010	**0.066**	**1.215**	**-0.113**	**0.059**	65	0.493
REINFORCE	6.927	0.071	1.214	-0.207	0.035	**64**	0.550
Balanced	5.484	0.068	1.138	-0.225	0.029	69	0.000
MV Unconstrained	5.736	0.080	1.003	-0.272	0.024	77	0.560
BL Unconstrained	5.922	0.081	1.001	-0.270	0.025	76	0.560

Table 6.4: Comparison of performance metrics of reinforcement learning algorithms trained using differential Sharpe ratio as the reward function, and classical portfolio techniques. The Sharpe ratios were calculated over a period of one year.

In this configuration, the RRLMD still outperforms all the remaining agents in terms of cumulative returns and hit ratio. It is interesting to notice that the maximum drawdown is lower for all agents when compared to their result using the log return as reward function. This is consequence of the penalization of the volatility that occurs when we employ the Sharpe ratio as performance metric.

Despite the fact that penalizing volatility can also negatively affect returns, the deep reinforcement learning algorithms achieve better performance in this setting. In fact, PPO achieves the best performance for the 4 metrics related to risk and volatility, and the REINFORCE performance is only marginally worse.

However, from an investor's point-of-view, the RRLMD may still be more attractive as an asset selector. The returns obtained are significantly higher and it still outperforms the benchmark most of the times. The maximum drawdown is still comparable with REINFORCE and not too much worse than that of PPO. The results for the cumulative return over time are shown in Fig. 6.6.

One interesting aspect to notice is that using DSR as a reward function does not have a significant impact on the achieved volatility and Sharpe ratio for the RRL Matrix algorithm. In fact, the Sharpe ratio for the decoupled version using log returns is even higher than the one using DSR. We present the allocations in each asset class for the two variants in Fig. 6.7.

Note that the log version of RRL has an average allocation on equities that is higher than the Sharpe ratio variant. However, the overall distribution of asset classes throughout time for both agents is very similar. In other words, their asset class timings (when they were

more invested in a particular asset class) did not vary significantly. This implies that both agents were equally exposed to exogenous financial shocks, such as the dot-com bubble, the 2008 financial crisis, commodities crashes and the 2015 European debt crisis.

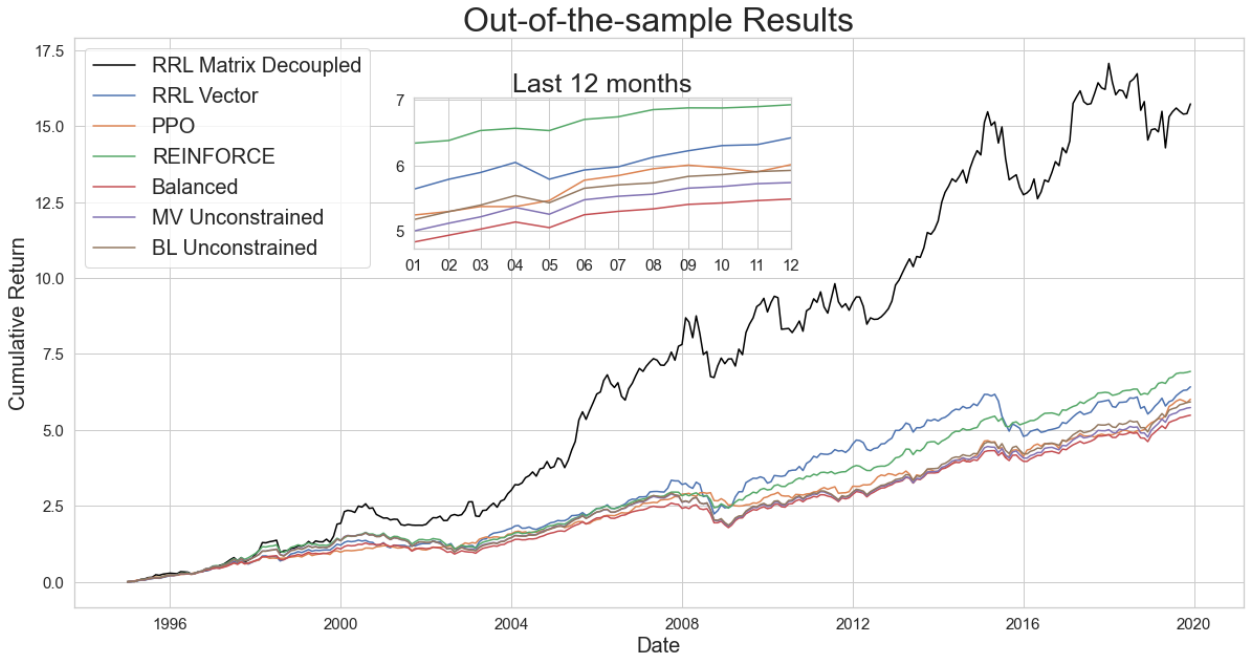

Figure 6.6: Cumulative returns of all portfolio agents. The RL algorithms were trained using the DSR. The last 12 months of all portfolios but the RRL matrix decoupled are zoomed in to facilitate the comparison.

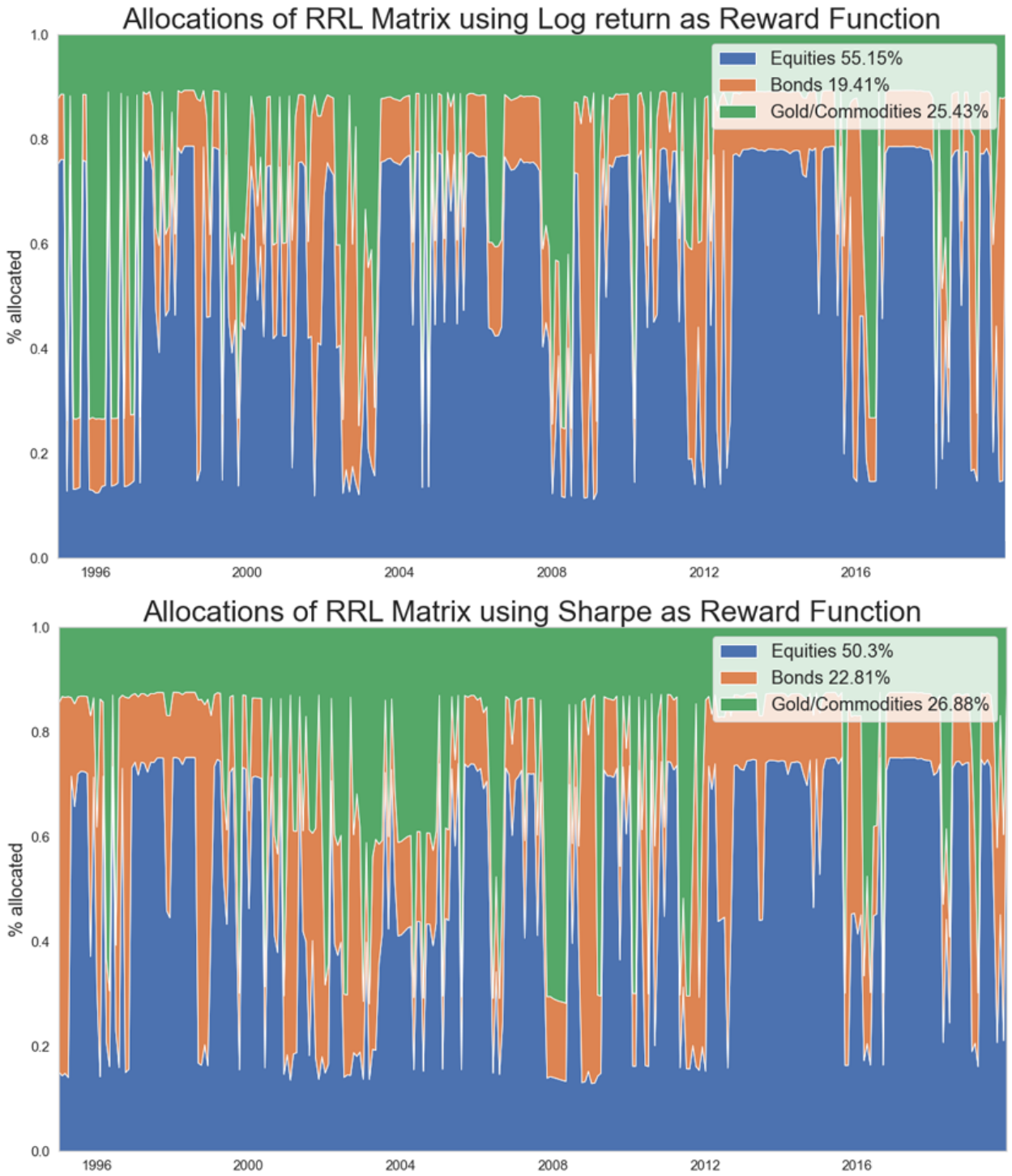

Figure 6.7: Allocations of RRL Matrix algorithm using DSR and log return as reward function. The average percentage allocated in each asset class is shown in the label.

6.5.3 Constraints Results

An allocation strategy can outperform a benchmark such as the the balanced portfolio via two different approaches. The first is to allocate more resources to riskier assets. For example, instead of allocating to bonds, allocating to equities is riskier, but potentially more rewarding, as equities tend to yield higher returns. However, this portfolio would be more exposed to high volatility.

The other way a portfolio can beat a benchmark is by selecting better assets within the same asset class. For instance, if the balanced portfolio allocates 40% in bonds equally, a better strategy may decide to give more weight to some specific bonds, while reducing the weight of the others. This approach aims to keep the percentage of each asset class equal to the reference portfolio, but allocated differently. In other words, this strategy aims to outperform the benchmark without being exposed to more risk.

In order to better understand the behaviour of the best performing RL agents in the previous experiments, we present in Table 6.5 the results of these models with constraints imposed on the amount allocated in each asset class.

Results for the moderate constraints: 45% allocated in equities, 40% in bonds and 15% in gold/commodities

Model	Cumulative Returns	Annualized Volatility	Sharpe Ratio	Max Drawdown	Calmar Ratio	Avg. Drawdown Time (days)	Hit Ratio
RRL Matrix Decoupled	**8.668**	0.080	1.178	-0.241	0.033	75	0.563
RSRRL [82]	6.829	0.073	1.179	-0.185	0.038	67	0.553
PPO	7.490	0.074	**1.203**	**-0.192**	**0.038**	66	**0.580**
REINFORCE	6.858	0.074	1.164	-0.203	0.035	66	0.583
Balanced Portfolio	5.204	**0.071**	1.075	-0.240	0.026	72	0

Results for the conservative constraints: 15% allocated in equities, 80% in bonds and 5% in gold/commodities

Model	Cumulative Returns	Annualized Volatility	Sharpe Ratio	Max Drawdown	Calmar Ratio	Avg. Drawdown Time (days)	Hit Ratio
RRL Matrix Decoupled	**6.005**	0.051	1.556	-0.067	0.099	60	0.560
RSRRL [82]	4.798	0.047	1.538	-0.071	0.084	60	0.503
PPO	5.410	0.043	1.750	-0.056	0.113	**55**	0.573
REINFORCE	4.446	**0.039**	**1.775**	-0.047	0.121	56	**0.580**
Balanced Portfolio	4.526	0.040	1.745	**-0.045**	**0.129**	**55**	0

Results for the aggressive constraints: 65% allocated in equities, 10% in bonds and 25% in gold/commodities

Model	Cumulative Returns	Annualized Volatility	Sharpe Ratio	Max Drawdown	Calmar Ratio	Avg. Drawdown Time (days)	Hit Ratio
RRL Matrix Decoupled	**10.240**	0.115	**0.907**	-0.342	**0.036**	89	**0.583**
RSRRL [82]	8.306	0.106	0.901	-0.287	0.028	83	0.547
PPO	8.740	0.109	0.892	**-0.304**	0.027	**77**	0.540
REINFORCE	8.614	0.111	0.874	-0.328	0.025	87	0.463
Balanced Portfolio	5.285	**0.105**	0.754	-0.381	0.017	93	0

Table 6.5: Comparison of performance metrics of different agents with constrained allocations

The RRLMD still achieves the highest cumulative return among all the algorithms evaluated. This implies that the agent is also capable of outperforming other models by selecting better assets within each class.

Moreover, this ability was particularly valuable in the aggressive constraints setting as the model achieved almost two times the cumulative return of the benchmark, while maintaining a higher Sharpe ratio and lower maximum drawdown.

Finally, the proposed RRLMD seems not only to be able to rotate between asset classes better than the other baselines but also to select superior assets within each class.

Chapter 7

Conclusions

This thesis focused on the application of reinforcement learning techniques in the portfolio allocation problem. In order to accomplish this task, different ideas and concepts from finance and machine learning were combined. We conclude this thesis by summarizing the contributions and providing some suggestions for future work.

7.1 Conclusions

In this thesis, we model financial markets as discrete-time stochastic dynamical systems in order to apply the reinforcement learning framework to train asset allocation agents.

We provided an overview of the foundations of reinforcement learning theory and defined three algorithms that are used as baselines: the Monte-Carlo Policy Gradient, the Proximal Policy Optimization and Recurrent Reinforcement Learning (RRL). We also included other two methods from the literature: the regime-switching RRL, and the "average elitist" scheme for RRL.

Moreover, we propose a novel extension for the RRL algorithm, called RRL Matrix Decoupled (RRLMD), which can account for each asset's particularities by defining a matrix of parameters instead of the conventional vector of weights. Furthermore, the assets are separated into different categories by employing a decoupled methodology, which allows flexibility with respect to two characteristics: the individual dynamics and risk exposure of each asset class.

In addition, we also defined and evaluated two methods of portfolio optimization, discussing their limitations and possible improvements.

The proposed RRLMD achieved the highest cumulative return among all tested models for all settings evaluated. Also, our algorithm achieved competitive results for other considered metrics.

More importantly, the model's parameters can be interpreted, which adds an extra benefit to its implementation in real applications as the lack of interpretability can make financial models infeasible. Furthermore, due to its simple architecture, hyperparameter tuning can be performed without significant computational overhead.

7.2 Future work

There are several research directions that can be further explored to improve the work presented in this thesis.

First, it would be interesting to include more indicators as input for the model. Interest rates, business confidence, unemployment and Gross domestic product (GDP) estimates are a few examples of the numerous indicators that affect financial markets.

A second possibility is evaluating the model on different assets. In particular, further insight could be acquired by considering individual stocks instead of the country-level indexes we utilized. The decoupling methodology could be employed to divide the equities by different sectors, and some sector-specific inputs could also be used to improve the model.

Another alternative is incorporating aspects already presented in the literature in the model. For instance, we could add some sort of regime-switching to the model, or employ stop losses and other risk management techniques.

There is also room for improving the reward function by considering other metrics, such as the maximum drawdown. It would be illuminting to embed the proposed model in a larger trading system, similar to the work proposed by [29].

Finally, incorporating Bayesian methods would be very useful, as the Bayesian framework provides a principled approach to deal with uncertainty, which is extremely valuable in financial markets. It also facilitates the incorporation of prior knowledge into the algorithms [42].

Bibliography

[1] Aguilar, O. and West, M. (2000). Bayesian dynamic factor models and portfolio allocation. *Journal of Business & Economic Statistics*, 18(3):338–357.

[2] Allaj, E. (2013). The black–litterman model: a consistent estimation of the parameter tau. *Financial Markets and Portfolio Management*, 27(2):217–251.

[3] Almahdi, S. and Yang, S. Y. (2019). A constrained portfolio trading system using particle swarm algorithm and recurrent reinforcement learning. *Expert Systems with Applications*, 130:145–156.

[4] Azayite, F. Z. and Achchab, S. (2016). Hybrid discriminant neural networks for bankruptcy prediction and risk scoring. *Procedia Computer Science*, 83:670–674.

[5] Bacon, C. R. (2008). *Practical portfolio performance measurement and attribution*, volume 546. John Wiley & Sons.

[6] Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., and Schieber, B. (2001). A unified approach to approximating resource allocation and scheduling. *Journal of the ACM (JACM)*, 48(5):1069–1090.

[7] Barreto, A. M., Precup, D., and Pineau, J. (2016). Practical kernel-based reinforcement learning. *The Journal of Machine Learning Research*, 17(1):2372–2441.

[8] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

[9] Bäuerle, N. and Rieder, U. (2011). *Markov decision processes with applications to finance*. Springer Science & Business Media.

[10] Baur, D. G. and McDermott, T. K. (2010). Is gold a safe haven? international evidence. *Journal of Banking & Finance*, 34(8):1886–1898.

[11] Baz, J., Granger, N., Harvey, C. R., Le Roux, N., and Rattray, S. (2015). Dissecting investment strategies in the cross section and time series. *Available at SSRN 2695101*.

[12] Bertoluzzo, F. and Corazza, M. (2012). Testing different reinforcement learning configurations for financial trading: Introduction and applications. *Procedia Economics and Finance*, 3:68–77.

[13] Bessler, W., Opfer, H., and Wolff, D. (2017). Multi-asset portfolio optimization and out-of-sample performance: an evaluation of black–litterman, mean-variance, and naïve diversification approaches. *The European Journal of Finance*, 23(1):1–30.

[14] Bhatia, A., Varakantham, P., and Kumar, A. (2019). Resource constrained deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 610–620.

[15] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[16] Black, F. and Litterman, R. (1990). Asset allocation: combining investor views with market equilibrium. *Goldman Sachs Fixed Income Research*, 115.

[17] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

[18] Buehler, H., Gonon, L., Teichmann, J., and Wood, B. (2019). Deep hedging. *Quantitative Finance*, pages 1–21.

[19] Chakraborty, S., Tomsett, R., Raghavendra, R., Harborne, D., Alzantot, M., Cerutti, F., Srivastava, M., Preece, A., Julier, S., Rao, R. M., et al. (2017). Interpretability of deep learning models: a survey of results. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1–6. IEEE.

[20] Chekhlov, A., Uryasev, S., and Zabarankin, M. (2005). Drawdown measure in portfolio optimization. *International Journal of Theoretical and Applied Finance*, 8(01):13–58.

[21] Choey, M. and Weigend, A. S. (1997). Nonlinear trading models through sharpe ratio maximization. *International Journal of Neural Systems*, 8(04):417–431.

[22] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[23] Cootner, P. H. (1967). *The random character of stock market prices*. MIT press.

[24] Csáji, B. C. and Monostori, L. (2008). Adaptive stochastic resource control: a machine learning approach. *Journal of Artificial Intelligence Research*, 32:453–486.

[25] Cura, T. (2009). Particle swarm optimization approach to portfolio optimization. *Nonlinear Analysis: Real World Applications*, 10(4):2396–2406.

[26] Dantas, S. G. and e Silva, D. G. (2018). Equity trading at the brazilian stock market using a q-learning based system. In *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 133–138. IEEE.

[27] De Faria, E., Albuquerque, M. P., Gonzalez, J., Cavalcante, J., and Albuquerque, M. P. (2009). Predicting the brazilian stock market through neural networks and adaptive exponential smoothing methods. *Expert Systems with Applications*, 36(10):12506–12509.

[28] DeMiguel, V., Garlappi, L., and Uppal, R. (2009). Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy? *The Review of Financial Studies*, 22(5):1915–1953.

[29] Dempster, M. A. and Leemans, V. (2006). An automated fx trading system using adaptive reinforcement learning. *Expert Systems with Applications*, 30(3):543–552.

[30] Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):653–664.

[31] Deng, Y., Kong, Y., Bao, F., and Dai, Q. (2015). Sparse coding-inspired optimal trading system for hft industry. *IEEE Transactions on Industrial Informatics*, 11(2):467–475.

[32] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[33] Du, X., Zhai, J., and Lv, K. (2016). Algorithm trading using q-learning and recurrent reinforcement learning. *Positions*, 1:1.

[34] Dungan, J., Frank, J., Jónsson, A., Morris, R., and Smith, D. (2002). Advances in planning and scheduling of remote sensing instruments for fleets of earth orbiting satellites. In *Earth Science Technology Conference*.

[35] Fabozzi, F. J., Kolm, P. N., Pachamanova, D. A., and Focardi, S. M. (2007). *Robust portfolio optimization and management*. John Wiley & Sons.

[36] Fama, E. F. (1965). The behavior of stock-market prices. *The Journal of Business*, 38(1):34–105.

[37] Fama, E. F. and French, K. R. (2004). The capital asset pricing model: Theory and evidence. *Journal of Economic Perspectives*, 18(3):25–46.

[38] Fischer, T. G. (2018). Reinforcement learning in financial markets-a survey. Technical report, FAU Discussion Papers in Economics.

[39] Gabrielsson, P. and Johansson, U. (2015). High-frequency equity index futures trading using recurrent reinforcement learning with candlesticks. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 734–741. IEEE.

[40] Gao, X. and Chan, L. (2000). An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization. In *Proceedings of the International Conference on Neural Information Processing*, pages 832–837.

[41] GERS, F. A., SCHMIDHUBER, J., and CUMMINS, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

[42] Ghavamzadeh, M., Mannor, S., Pineau, J., and Tamar, A. (2016). Bayesian reinforcement learning: A survey. *arXiv preprint arXiv:1609.04436*.

[43] Gold, C. (2003). Fx trading via recurrent reinforcement learning. In *2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings.*, pages 363–370. IEEE.

[44] Goldberg, L. R. and Mahmoud, O. (2017). Drawdown: from practice to theory and back again. *Mathematics and Financial Economics*, 11(3):275–297.

[45] Graesser, L. and Keng, W. L. (2019). *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. Addison-Wesley Professional.

[46] Grossman, S. J. and Zhou, Z. (1993). Optimal investment strategies for controlling drawdowns. *Mathematical finance*, 3(3):241–276.

[47] Guo, Y., Fu, X., Shi, Y., and Liu, M. (2018). Robust log-optimal strategy with reinforcement learning. *arXiv preprint arXiv:1805.00205*.

[48] Hamilton, J. D. (2010). Regime switching models. In *Macroeconometrics and time series analysis*, pages 202–209. Springer.

[49] Hanias, M., Curtis, P., and Thalassinos, J. (2007). Prediction with neural networks: the athens stock exchange price indicator. *European Journal of Economics, Finance and Administrative Sciences*, 9:21–27.

[50] Haugen, R. A. and Haugen, R. A. (2001). *Modern investment theory*, volume 5. Prentice Hall Upper Saddle River, NJ.

[51] He, G. and Litterman, R. (2002). The intuition behind black-litterman model portfolios. *Available at SSRN 334304*.

[52] Hoceini, S., Mellouk, A., and Amirat, Y. (2005). K-shortest paths q-routing: a new qos routing algorithm in telecommunication networks. In *International Conference on Networking*, pages 164–172. Springer.

[53] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[54] Hudson, R. S. and Gregoriou, A. (2015). Calculating and comparing security returns is harder than you think: A comparison between logarithmic and simple returns. *International Review of Financial Analysis*, 38:151–162.

[55] Hunter, D. R. and Lange, K. (2004). A tutorial on mm algorithms. *The American Statistician*, 58(1):30–37.

[56] Idzorek, T. (2007). A step-by-step guide to the black-litterman model: Incorporating user-specified confidence levels. In *Forecasting expected returns in the financial markets*, pages 17–38. Elsevier.

[57] Jang, J.-S. R., Sun, C.-T., and Mizutani, E. (1997). Neuro-fuzzy and soft computing-a computational approach to learning and machine intelligence [book review]. *IEEE Transactions on Automatic Control*, 42(10):1482–1484.

[58] Jangmin, O., Lee, J., Lee, J. W., and Zhang, B.-T. (2006). Adaptive stock trading with dynamic asset allocation using reinforcement learning. *Information Sciences*, 176(15):2121–2147.

[59] Jiang, Z. and Liang, J. (2017). Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent Systems Conference (IntelliSys)*, pages 905–913. IEEE.

[60] Jiang, Z., Xu, D., and Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*.

[61] Jobson, J. D. and Korkie, B. (1980). Estimation for markowitz efficient portfolios. *Journal of the American Statistical Association*, 75(371):544–554.

[62] Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274.

[63] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE.

[64] Kim, H.-J., Jo, N.-O., and Shin, K.-S. (2016). Optimization of cluster-based evolutionary undersampling for the artificial neural networks in corporate bankruptcy prediction. *Expert Systems with Applications*, 59:226–234.

[65] Kim, W. C., Kim, J. H., and Fabozzi, F. J. (2015). Shortcomings of mean-variance analysis. *Robust Equity Portfolio Management+ Website: Formulations, Implementations, and Properties Using MATLAB*, pages 22–38.

[66] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[67] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[68] Kirkpatrick II, C. D. and Dahlquist, J. A. (2010). *Technical analysis: the complete resource for financial market technicians*. FT press.

[69] Klir, G. and Folger, T. (1988). Fuzzy sets, uncertainty, and information. *Hall1988*.

[70] Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, pages 1008–1014.

[71] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

[72] Ledoit, O. and Wolf, M. (2004). A well-conditioned estimator for large-dimensional covariance matrices. *Journal of Multivariate Analysis*, 88(2):365–411.

[73] Lee, J. W., Park, J., Jangmin, O., Lee, J., and Hong, E. (2007). A multiagent approach to q-learning for daily stock trading. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(6):864–877.

[74] Lefebvre, G., Lebreton, M., Meyniel, F., Bourgeois-Gironde, S., and Palminteri, S. (2017). Behavioural and neural characterization of optimistic reinforcement learning. *Nature Human Behaviour*, 1(4):1–9.

[75] Leung, F. H.-F., Lam, H.-K., Ling, S.-H., and Tam, P. K.-S. (2003). Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural networks*, 14(1):79–88.

[76] Li, X., Li, Y., Zhan, Y., and Liu, X.-Y. (2019). Optimistic bull or pessimistic bear: adaptive deep reinforcement learning for stock portfolio allocation. *arXiv preprint arXiv:1907.01503*.

[77] Liang, Z., Chen, H., Zhu, J., Jiang, K., and Li, Y. (2018). Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*.

[78] Lipton, Z. C. (2018). The mythos of model interpretability. *Queue*, 16(3):31–57.

[79] Lo, A. W., Mamaysky, H., and Wang, J. (2000). Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. *The journal of finance*, 55(4):1705–1765.

[80] Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization.

[81] Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM.

[82] Maringer, D. and Ramtohul, T. (2012). Regime-switching recurrent reinforcement learning for investment decision making. *Computational Management Science*, 9(1):89–107.

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

[84] Meucci, A. (2009). Enhancing the black–litterman and related approaches: Views and stress-test on risk factors. *Journal of Asset Management*, 10(2):89–96.

[85] Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30.

[86] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

[87] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[88] Moody, J. and Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12(4):875–889.

[89] Moody, J. and Wu, L. (1998). *High frequency foreign exchange rates: Price behavior analysis and "true price" models*, volume 2. Chap.

[90] Moody, J., Wu, L., Liao, Y., and Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, 17(5-6):441–470.

[91] Moody, J. E. and Saffell, M. (1999). Reinforcement learning for trading. In *Advances in Neural Information Processing Systems*, pages 917–923.

[92] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*.

[93] Neuneier, R. (1996). Optimal asset allocation using adaptive dynamic programming. In *Advances in Neural Information Processing Systems*, pages 952–958.

[94] Neuneier, R. (1998). Enhancing q-learning for optimal asset allocation. In *Advances in neural information processing systems*, pages 936–942.

[95] Ormoneit, D. and Sen, Ś. (2002). Kernel-based reinforcement learning. *Machine learning*, 49(2-3):161–178.

[96] Osterland, A. (2019). As behemoth brokerage firms go zero-commission on trades, advisors are concerned. `https://www.cnbc.com/2019/11/06/as-brokerage-firms-go-to-zero-commission-on-trades-advisors-worry.html`. Accessed 08/07/20.

[97] Pal, N. R. and Bezdek, J. C. (1994). Measuring fuzzy uncertainty. *IEEE Transactions on Fuzzy Systems*, 2(2):107–118.

[98] Park, H., Sim, M. K., and Choi, D. G. (2020). An intelligent financial portfolio trading strategy using deep q-learning. *Expert Systems with Applications*, page 113573.

[99] Patel, J., Shah, S., Thakkar, P., and Kotecha, K. (2015). Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Systems with Applications*, 42(1):259–268.

[100] Petersen, B. K., Yang, J., Grathwohl, W. S., Cockrell, C., Santiago, C., An, G., and Faissol, D. M. (2018). Precision medicine as a control problem: Using simulation and deep reinforcement learning to discover adaptive, personalized multi-cytokine therapy for sepsis. *arXiv preprint arXiv:1802.10440*.

[101] Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

[102] Satchell, S. and Scowcroft, A. (2000). A demystification of the black–litterman model: Managing quantitative and traditional portfolio construction. *Journal of Asset Management*, 1(2):138–150.

[103] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

[104] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[105] Sharpe, W. F. (1966). Mutual fund performance. *The Journal of business*, 39(1):119–138.

[106] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484.

[107] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms.

[108] Srivastava, N., Mansimov, E., and Salakhudinov, R. (2015). Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning*, pages 843–852.

[109] Styan, G. P. (1973). Hadamard products and multivariate statistical analysis. *Linear Algebra and its Applications*, 6:217–240.

[110] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[111] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[112] Tan, Y., Liu, W., and Qiu, Q. (2009). Adaptive power management using reinforcement learning. In *2009 IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers*, pages 461–467. IEEE.

[113] Tan, Z., Quek, C., and Cheng, P. Y. (2011). Stock trading with cycles: A financial application of anfis and reinforcement learning. *Expert Systems with Applications*, 38(5):4741–4755.

[114] Taylor, H. M. (1975). A stopped brownian motion formula. *The Annals of Probability*, pages 234–246.

[115] Tsai, J.-T., Chou, J.-H., and Liu, T.-K. (2006). Tuning the structure and parameters of a neural network by using hybrid taguchi-genetic algorithm. *IEEE Transactions on Neural Networks*, 17(1):69–80.

[116] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

[117] Walters, C. et al. (2013). The factor tau in the black-litterman model. *Available at SSRN 1701467*.

[118] Wang, J.-B., Wang, J., Wu, Y., Wang, J.-Y., Zhu, H., Lin, M., and Wang, J. (2018). A machine learning framework for resource allocation assisted by cloud computing. *IEEE Network*, 32(2):144–151.

[119] Watkins, C. J. and Dayan, P. (1992a). Q-learning. *Machine Learning*, 8(3-4):279–292.

[120] Watkins, C. J. and Dayan, P. (1992b). Q-learning. *Machine Learning*, 8(3-4):279–292.

[121] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[122] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

[123] Williams, R. J. and Zipser, D. (1989). Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111.

[124] Yu, P., Lee, J. S., Kulyatin, I., Shi, Z., and Dasgupta, S. (2019). Model-based deep reinforcement learning for dynamic portfolio optimization. *arXiv preprint arXiv:1901.08740*.

[125] Zhang, J. and Maringer, D. (2014). Two parameter update schemes for recurrent reinforcement learning. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1449–1453. IEEE.

[126] Zhang, Z., Zohren, S., and Roberts, S. (2020). Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2(2):25–40.

Appendices

Appendix A

Experiment setting - hyperparameters PPO and REINFORCE

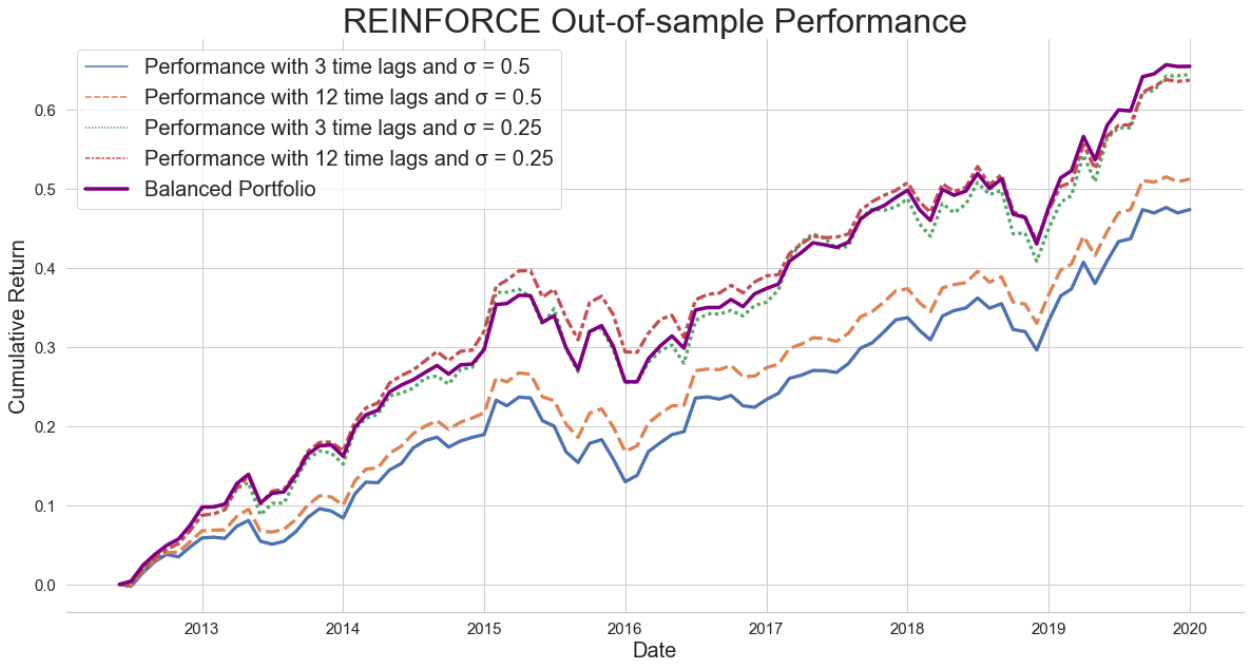

Figure A.1: Out-of-the-sample results for the REINFORCE algorithm for different combinations of σ and time lags

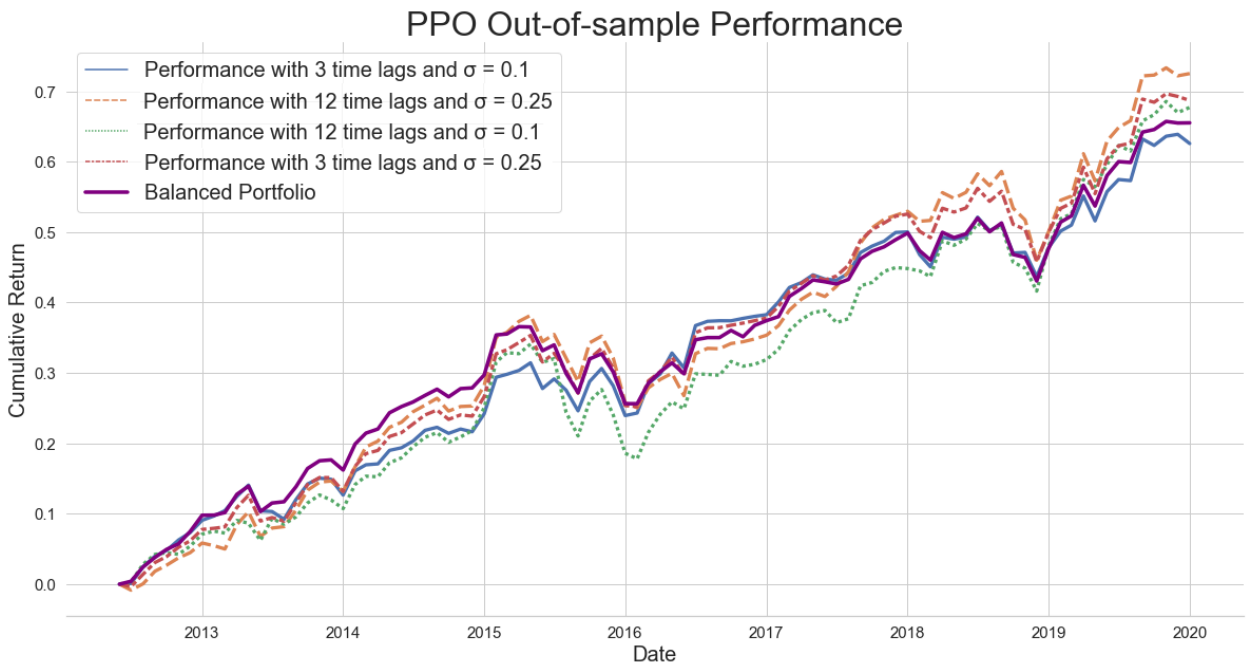

Figure A.2: Out-of-the-sample results for the PPO algorithm for different combinations of σ and time lags

Appendix B

Experiment setting - gold and commodities

The results are derived from the experiment that consisted in comparing the standard version of the RRL algorithm [88], with our proposed matrix extension.

We ran both algorithms for the same training and testing periods. The train/test split was set to 80%/20%, so both agents were trained using data from 1980 to 2012, and later tested on data from 2012 to 2019. We also restricted the asset universe to only gold and commodities. The reason is since those assets usually have quite different dynamic (gold does well in a stressed market while commodities have the opposite behaviour), it would be interesting to analyze the difference in performance.

The number of lags M to create the state $\mathbf{S}_t$ was set to 5. The balanced portfolio consisted in a portfolio with 50% allocated in gold and 50% in commodities.

Appendix C

Overfitting experiments

The standard RRL proposed by Moody [89] approximates the policy as a vector of weights. Our proposed model extends this approach by considering a more complex policy. In order to understand how this affects the out-of-the-sample performance, we trained the standard recurrent reinforcement learning (RRL) agent and the proposed RRL matrix decoupled (RRLMD) using three different training lengths.

All agents are evaluated in the same testing period, therefore the difference in performance is due to different training sizes. The results are shown in Fig. C.1.

As expected, having more training data resulted in a model that is able to generalize better. Also, it is important to note that the impact of reducing the available data to train affects more the RRLMD when compared to the standard RRL.

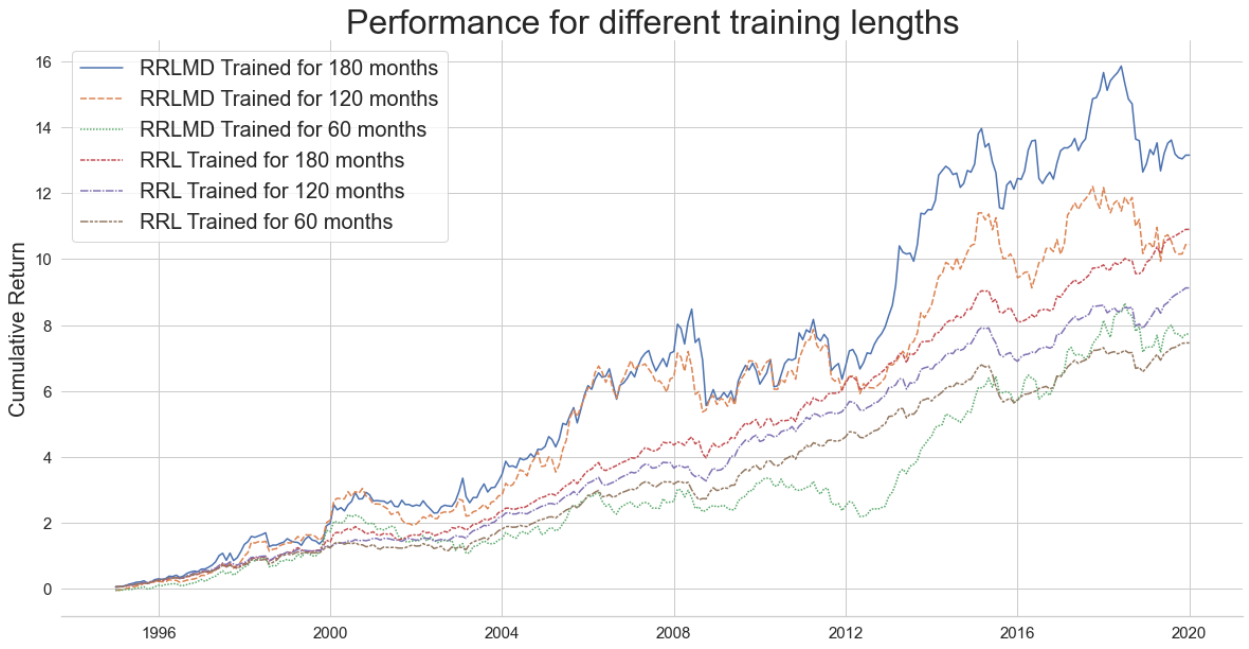

Figure C.1: Results for standard RRL and proposed RRL for different training lengths.

This implies that approximating the policy as a matrix in the RRL framework requires more data to be effective. The performances of both RRL and RRLMD are very similar using only 60 months for training.

Appendix D

Results of Experiments

We present here the figures with results of the experimental setting. Each model was evaluated on the testing window of 100 months and 30 months.

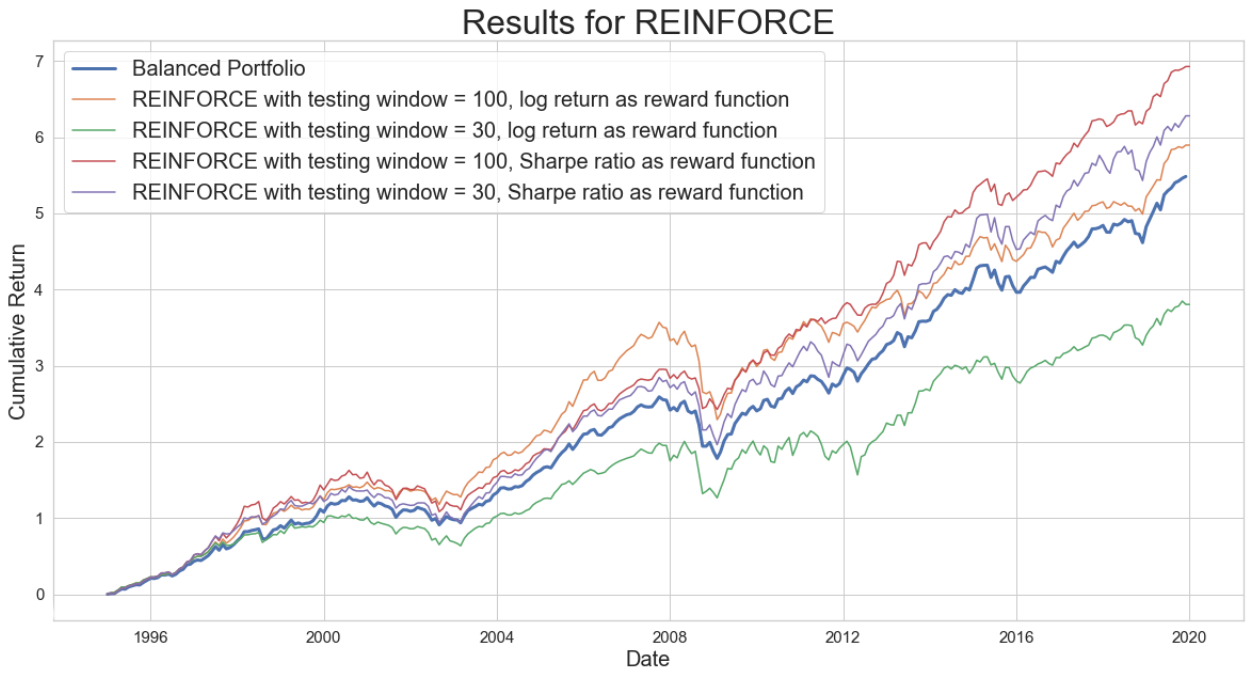

Figure D.1: Testing results for REINFORCE.

For the REINFORCE algorithm, it is clear that the Sharpe ratio agents outperformed the ones that used log return as reward function. As this algorithm is known for its variance, a risk-adjusted return metric might help the agent to be more consistent in its allocations. Moreover, the agents that were evaluated on testing windows of length = 100 achieved a better performance. As the REINFORCE suffers from high variance, there is a significant probability that the agent's performance gets worse during retraining.

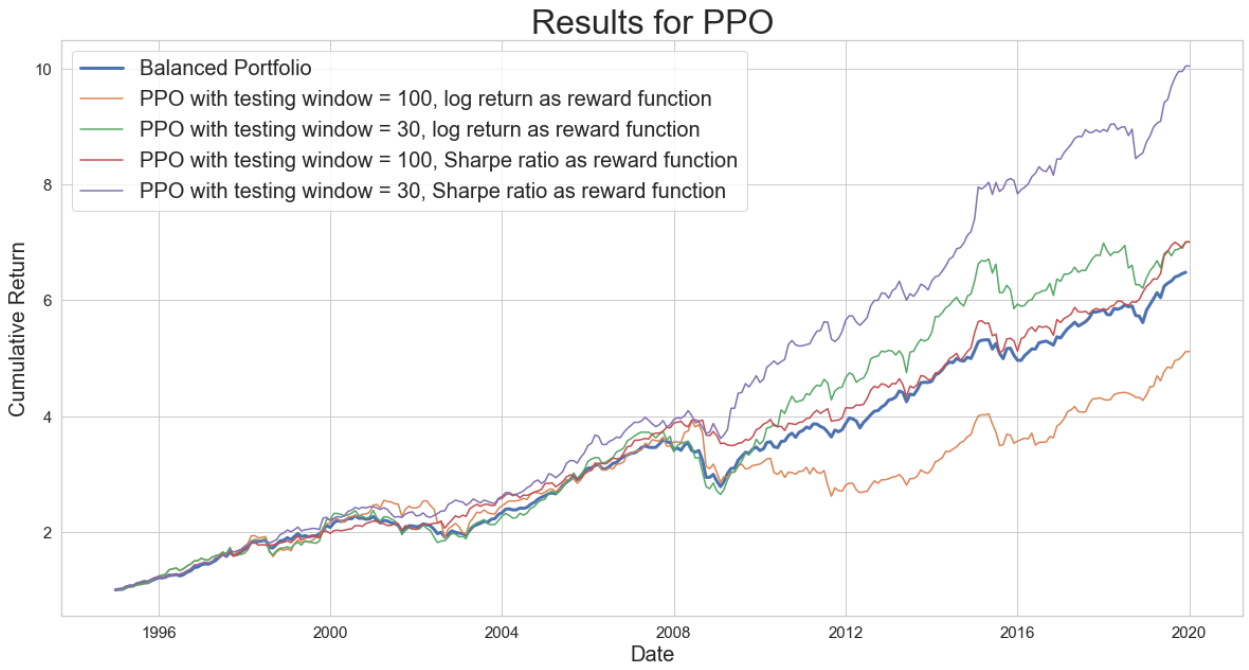

Figure D.2: Testing results for PPO.

On the other hand, the PPO agent did best with more frequent retraining. In terms of reward function, the Sharpe ratio agents also outperformed the log-returns.

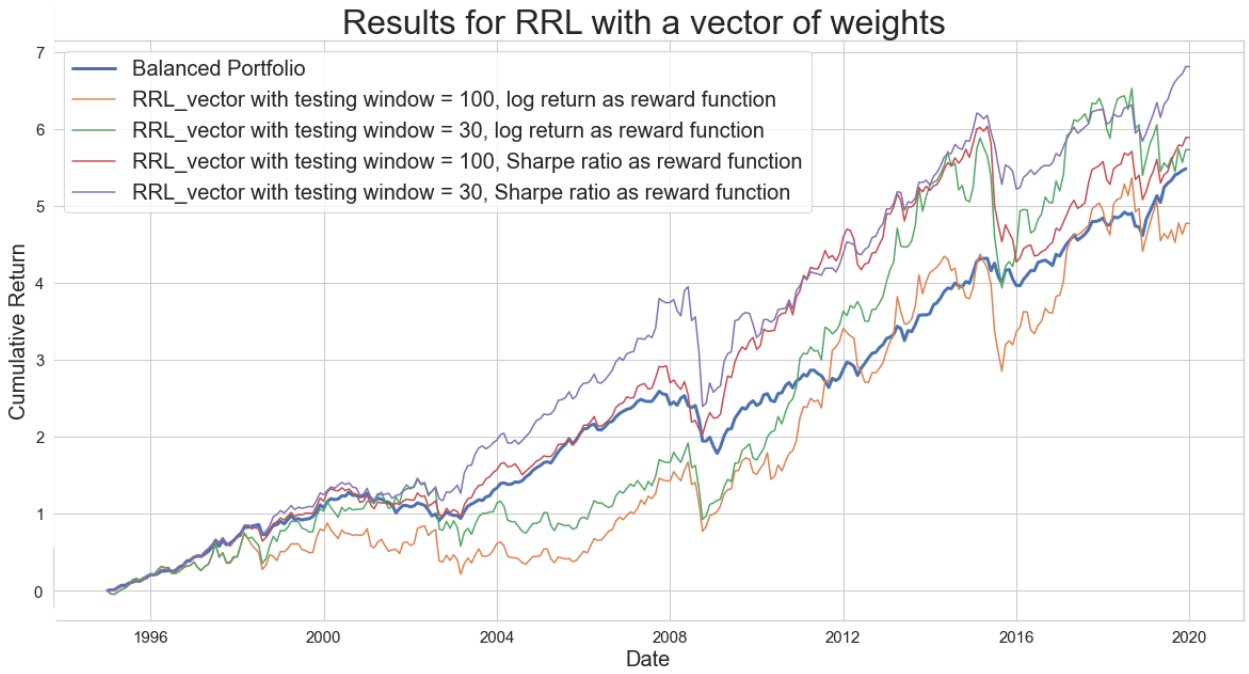

Figure D.3: Testing results for RRL on vector form.

For the standard RRL, the Sharpe ratio also appear to improve the overall performance. However, the volatility is way higher when compared to the previous algorithms.

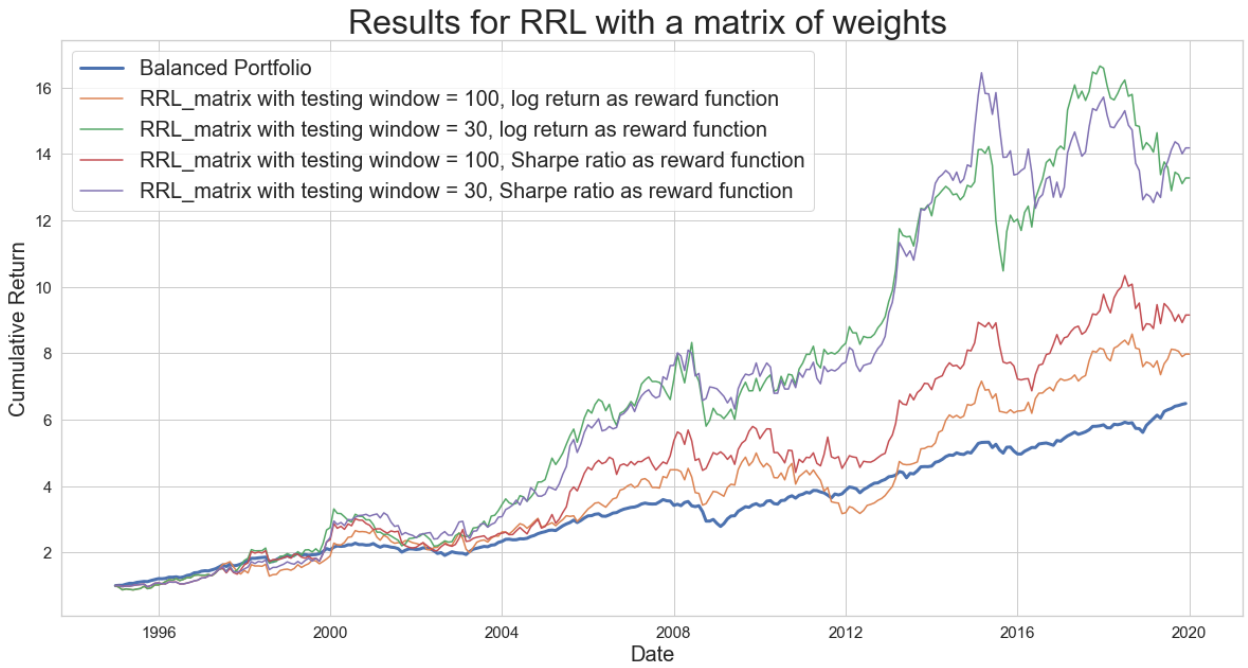

Figure D.4: Testing results for RRL on matrix form.

The proposed matrix extension outperforms the vector version of RRL in all runs. Moreover, the testing window seems to have greater influence than the reward function.

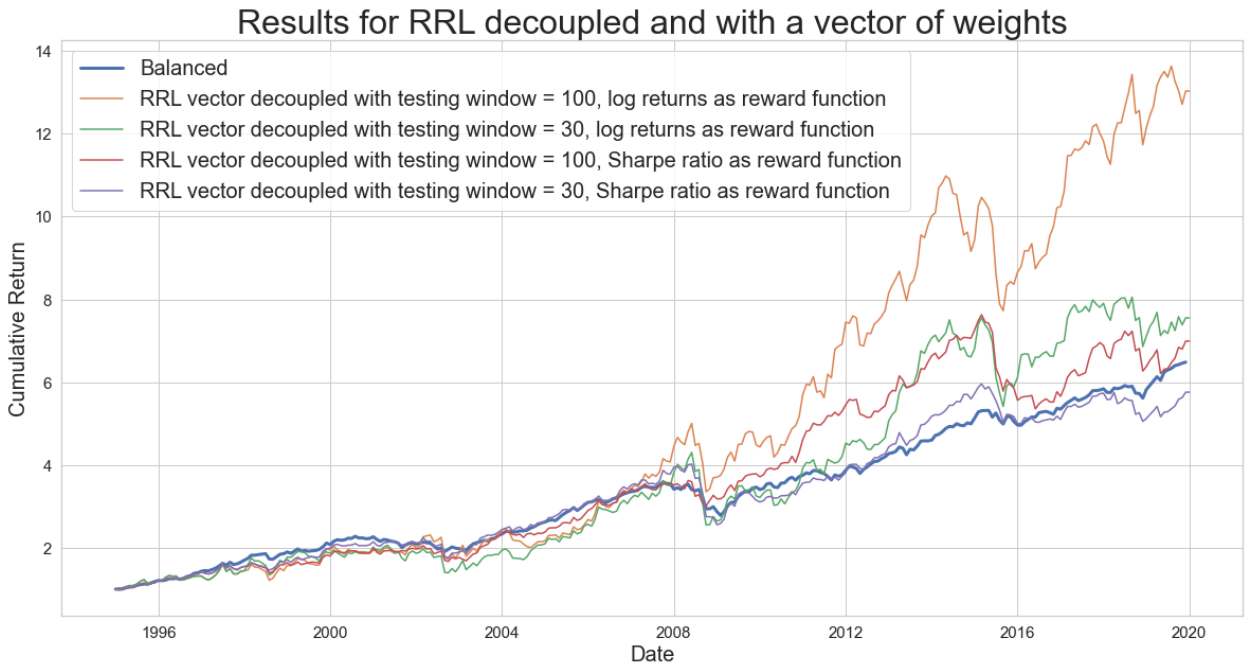

Figure D.5: Testing results for RRL on vector form using the decoupled framework.

There is a slightly improving by decoupling the RRL vector. However, the volatility is significantly higher than other models.

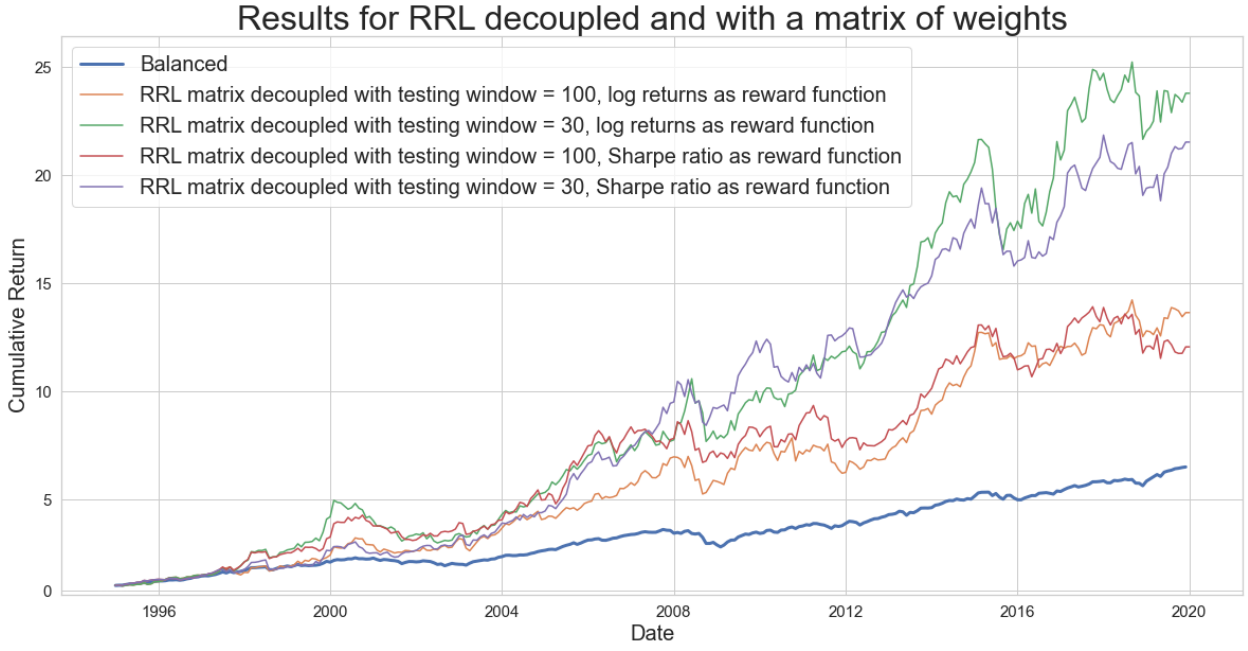

Figure D.6: Testing results for RRL on matrix form using the decoupled framework.

The decoupled RRL matrix is the best performing algorithm. The testing window appears to have more influence than the reward function.

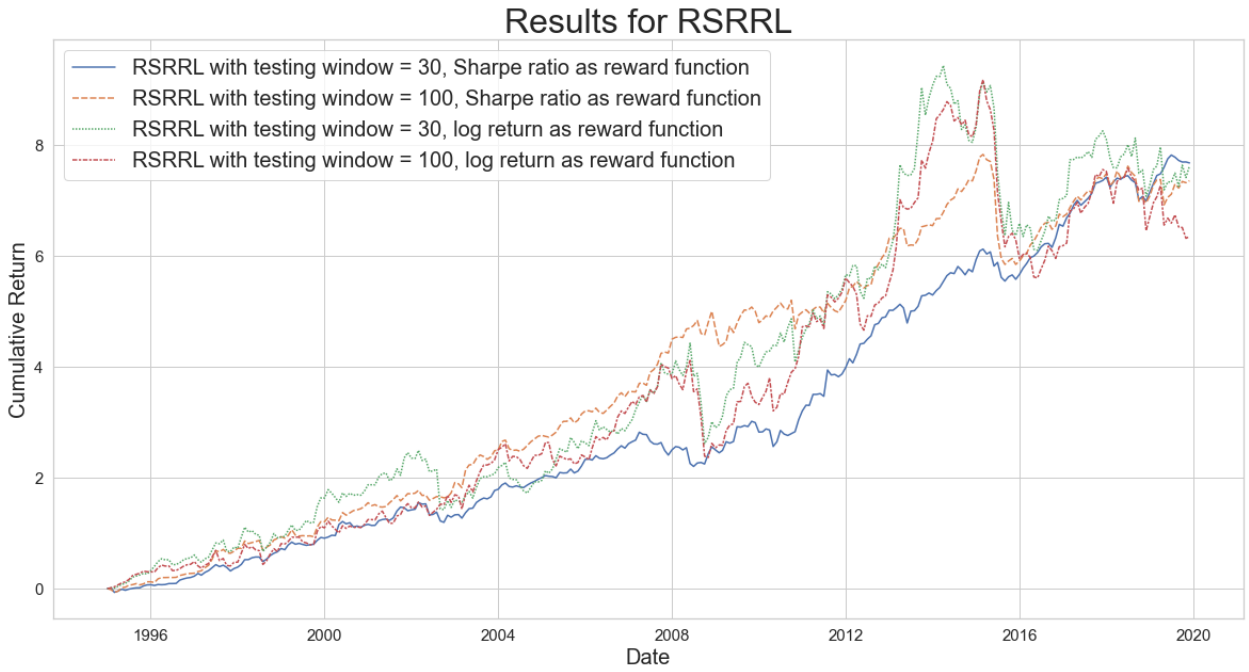

Figure D.7: Testing results for Regime Switching RRL for different reward functions

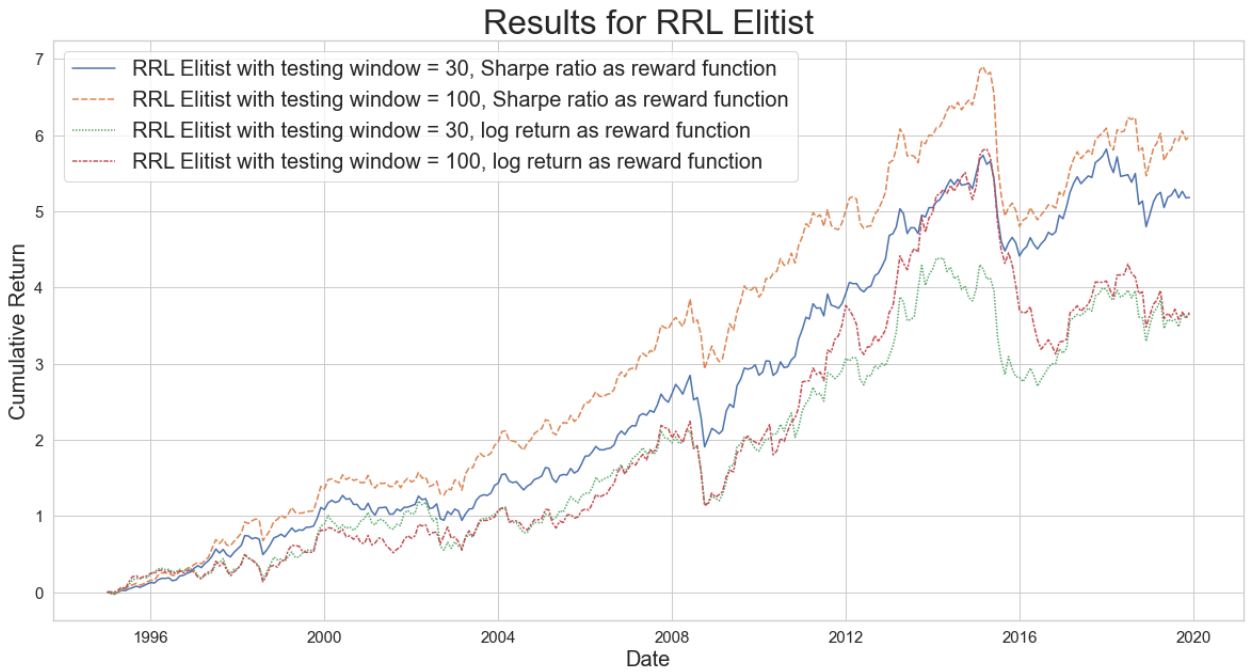

Figure D.8: Testing results for RRL 'Elitist' for different reward functions.