UCLouvain

FACULTE DES SCIENCES

Faculté des sciences

Reinforcement Learning algorithms applied to constrained Portfolio Allocation

Author: **Pierre-Alexandre LOUIS**
Supervisor: **Pierre ARS**
Reader: **Donatien HAINAUT**
Academic year 2023-2024
Master [120] en sciences actuarielles

Abstract

While machine learning has been employed in actuarial science for a long time, many advanced techniques, including Reinforcement Learning (RL), remain underexplored in this field. In contrast, the financial sector has successfully integrated reinforcement learning for tasks such as stock trading and portfolio management, demonstrating its potential in optimising decision-making processes. This thesis investigates the application of RL to constrained portfolio allocation driven by insurer liabilities. The aim is to improve decision-making and risk management by leveraging reinforcement learning capabilities to handle various market environments and risk aversions. The research begins by detailing RL algorithms and their underlying concepts. It then applies these methods to unconstrained and constrained portfolio optimisation and proposes a RL-based methodology for managing insurance products with target returns. This approach is tested in scenarios of interest rate increases to evaluate RL's adaptive capabilities. By integrating RL with actuarial science, this thesis offers a new perspective on tackling actuarial challenges. The findings contribute to the optimisation and efficiency of insurance portfolio allocations, providing a foundation for further exploration and refinement in this emerging area of research.

Acknowledgements

Thank you to my supervisor, Pierre Ars, for your guidance and you support along those two years.

I would like to thank Donatien Hainaut to read this master thesis.

Thank you to the "copains de l'atelier mémoire", I doubt I would have finished without you.

Thank you to my family for your support and your patience.

Thank you Jeremy for your help and your commentaries.

Lastly, I would like to thank my girlfriend so much. You have been a pillar and inspiration. Thank you for your moral supports, your compassion, your listening, your patience. I know I took a long time for finish this work but you were always there.

Disclaimer

This document includes correction from an AI language model known as ChatGPT. It was solely used as an aid for writing, specifically for correcting spelling and grammar and style improvements. At no point was it was used to generate content or conduct research. The ideas, analysis, and conclusions presented in this work are entirely the author's own. All other sources used in this thesis are cited and acknowledged.

Contents

Chapter 1

Introduction

Actuaries have known machine learning since the 1980s [Denuit et al., 2007], it has gradually become a valuable tool in actuarial science. Actuaries use these techniques for various applications, such as pricing premiums in non life insurance, Asset Liability Management (ALM) or to predict development triangles of the Chain-Ladder methods [Blier-Wong et al., 2020; Grize et al., 2020].

Advances in computing power have allowed the theoretical development of machine learning methods, easing their application in finance and insurance [Dixon et al., 2020]. Research in the realm of insurance has demonstrated the various applications of machine learning, including fraud detection, claims processing, and modeling of mortality and longevity risks [Roy and George, 2017; Aslam et al., 2022].

Despite the growing prominence of machine learning and its success in research applications, its adoption in practical actuarial work remains somewhat limited. One challenge is the strict rules and regulations that actuaries operate within. Actuarial models must be transparent and easily interpretable, ensuring that stakeholders, regulators, and policyholders can fully understand and trust the results [Zappa et al., 2023]. However, many machine learning models are inherently complex, which makes it challenging to provide clear and comprehensible explanations for their predictions [Lorentzen and Mayer, 2020].

Another challenge is the data scarcity in the insurance industry. Insurance companies manage sensitive and confidential data, making it difficult to obtain large, high-quality datasets for training complex machine learning models. Machine learning algorithms often rely on substantial amounts of data to achieve optimal performance, which presents a significant obstacle for actuaries when applying such methods. However, various strategies can address this issue. Methods such as data augmentation and the generation of synthetic data can effectively expand the available dataset for training machine learning models. Additionally, transfer learning, which involves fine-tuning pre-trained models to specific tasks, can help leverage existing models even when actuarial data is limited [Kraus and Feuerriegel, 2017]. Despite the promise of these strategies, it is crucial to ensure that they are implemented in a manner that respects data privacy and maintains the quality of the data. These considerations are key to ensuring the reliable and ethical application of machine learning within the actuarial field.

At the same time, the use of machine learning has expanded in finance. Quantitative analysts now apply it for stock-trading, portfolio allocation and option pricing. Such applications have improved decision-making techniques in diverse financial landscapes, particularly in situations where traders have sparse knowledge about market conditions and competitor actions [Hambly et al., 2021]. Contrary to insurers, the fiancial sector uses a broader range of machine learning techniques. Machine learning methods can generally be categorized into three main groups, as shown in figure 1.1: supervised learning, which includes approaches like generalized linear models, decision trees, and neural networks; unsupervised learning, which includes methods like k-nearest neighbors; and the relatively unexplored domain of reinforcement learning [Kubat et al., 1998]. Reinforcement Learning (RL) is a unique approach that differs significantly from the two other methods. It has gained popularity in various fields such as computer vision and game learning but remains less explored in the actuarial world [Krasheninnikova et al., 2019]. In this thesis, we detail the concept of reinforcement learning and investigate how it can be applied to the portfolio allocation of an insurance company.

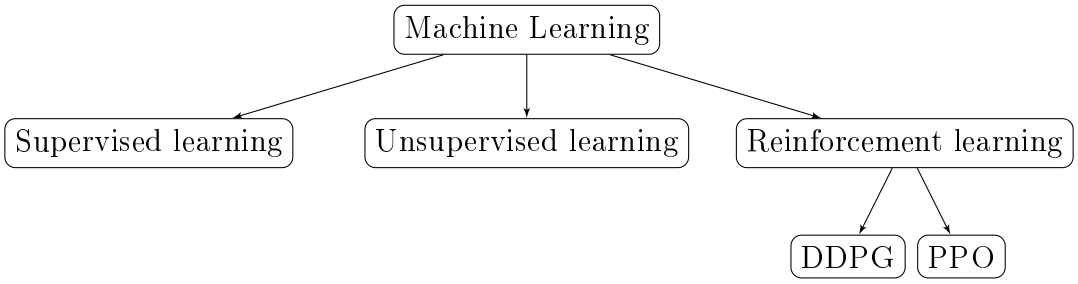

Figure 1.1: Classification of machine learning methods as described in the manuscript, with a focus on some reinforcement learning algortihms used in this work.

This master thesis aims to provide valuable insights into the feasibility, effectiveness, and potential benefits of reinforcement learning in enhancing decision-making processes and risk management within the insurance industry. Our focus is specifically on constrained portfolio allocation, driven by the liabilities of an insurer. Contrary to classic Asset Liability Management (ALM), where the goal is to harmonize both assets and liabilities cash flows, our approach concentrates mainly on the fine-tuning of the investment blend. The liabilities intervene as a ruin probability constraint.

Therefore, the objective of this work extends beyond mere portfolio allocation, which would focus solely on maximising returns from investments, free from commitments dictated by liabilities. The application of Reinforcement Learning (RL) in this context could offer several advantages over traditional methods. The first advantage is its ability to handle complex and non-linear relationships between assets and liabilities without the need for intricate quantitative models. Another advantage is its capacity to adapt to changing market conditions and optimise portfolio allocation in real time. Additionally, reinforcement learning can facilitate more robust stress-testing and scenario analysis, thereby improving the insurer's resilience to financial shocks and unexpected market movements.

In the following chapters, we present the different reinforcement learning algorithms that we use throughout this master thesis. We also present the different data that we use to train these algorithms. We start by explaining the underlying mathematics of reinforcement

learning. We first apply the reinforcement learning method to the financial domain to search for the optimal portfolio allocation without constraints. Lastly, we propose a methodology for managing an insurance product with a target return utilizing reinforcement learning. This approach is applied within the context of a sudden interest rate increase to observe the adaptive behavior of reinforcement learning algorithms. It is well understood that asset market values are inherently correlated with interest rates. An increase in interest rates typically impacts the prices of equities and bonds. Consequently, an interest rate hike should prompt asset reallocation decisions from fund managers [Hermans et al., 2023].

Chapter 2

Context for reinforcement learning and machine learning

This chapter details the mathematical background underlying Reinforcement Learning (RL). Section 2.1 presents the different elements necessary for constructing the formalism of RL. It introduces the Markov Decision Process (MDP) and explains how to solve it within an environment with a discrete number of states. A simple example of wealth allocation is provided to illustrate the concept. Section 2.2 extends the concept to continuous states or to environments with a numerous of states. This section also introduces the use of neural networks in RL. This process is referred to as Deep Reinforcement Learning (DRL). Section 2.3 depicts the Reinforcement Learning algorithm used in the asset allocation, known as Proximal Policy optimisation (PPO).

2.1 Reinforcement learning

This section details the framework of RL. It explores the key components of the RL framework, its features and characteristics that differ from other machine learning methods. It also introduces the MDP and the Bellman equations that are the mathematical basis of the RL framework. Finally, it presents the different approaches to solve the Bellman equations.

2.1.1 Differences with other machine learning approaches

In contrary to supervised and unsupervised machine learning methods, the data used in RL is not instantly accessible in its complete form. As highlighted by [Plaat, 2022], RL models gain knowledge from constant interaction with their environment. Data items are processed one after another, leading to a dataset that grows and changes over time. The central objective in RL is to find an optimal policy, a function that provides the most effective action for each conceivable state the system might be in. As the RL model interacts with its environment, it gradually refines its understanding, improving its decision-making capabilities over time. This ongoing learning process makes RL well-suited for dynamic, evolving environments where strategic adaptability is key.

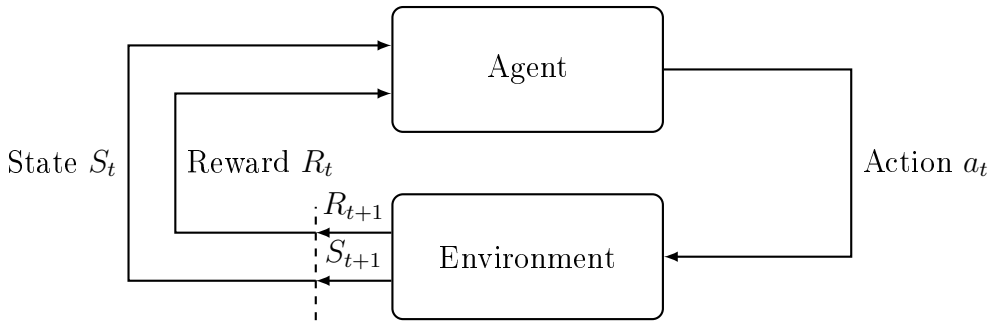

Figure 2.1: Illustration of the interaction cycle between the agent and the environment in a RL process. The agent receives a state S_t from the environment and takes an action a_t. The environment, in turn, provides the agent with the next state S_{t+1} and a reward R_{t+1}. This continuous cycle forms the basis for the learning mechanism in RL. The cycle begins with a initial state S_0 and results to an initial action a_0 by the agent.

In the context of RL, there is an interaction between two entities (see Fig. 2.1). The first one is the agent, which learns from the process occuring in the system. The second is the environment, representing the world around. The environment provides a feedback, called "reward", to the agent about its actions. The reward can be positive or negative. For example, in a simple financial world setting, we could consider the investor as the agent making decisions about buying or selling assets, and the market as the environment. The feedback from the environment to the agent comes in the form of the portfolio value after the asset transaction. The ultimate objective in RL is to select the most beneficial actions for every state, actions that would maximise the long-term accumulated expected reward. This selection of best actions is known as the optimal policy.

As previously mentioned, the environment in RL provides information on the value or 'goodness' of a state through the reward mechanism. This way, RL offers a level of guidance about the optimal action to select, but it is important to note that this information is often partial or incomplete. This characteristic distinguishes RL from other types of machine learning like supervised and unsupervised learning, which typically operate on complete datasets and provide more comprehensive information for decision-making.

2.1.2 Markov Decision Process

The current section describes the Markov Decision Process (MDP) framework and the section 2.1.3 presents an example of MDP in the context of finance. As stated in section 2.1.1, in Reinforcement Learning the goal of the agent, also known as the learner, is to maximise the highest long-term reward. A simple approximation is to assume that the environment can be fully observable and behaves as a first-order MDP [Dixon et al., 2019]. The concept of MDP allows a formalisation of the problems introduced by the Reinforcement Learning, where the objective is to select an action sequence over a specific period of time to optimise an objective function. As the RL agent keeps interacting with the environment, it continues to improve its understanding of the MDP. Each action and the subsequent reward helps the agent to adjust its policy in order to optimise the action sequence further and align it with the objective

function. The problem is called optimal control problem, which is directly derived from the systems and control theory [Sutton and Barto, 2018]. In the context of this master thesis, the control inputs are strategic decisions made by insurers, and the performance measure to optimise could be the long-term profitability of the insurance portfolio.

In the cycle of the figure 2.1, the environment is represented by a state $S_t \in \mathrm{S}$. The agent, i.e. the learner, picks an action $A_t \in \mathcal{A}(s)$ at every time step t. After the agent performs an action, it receives a numerical reward $R_{t+1} \in \mathcal{R}$. The environment goes to a new state S_{t+1}. In the notation before, S_t,A_t and R_t denotes random variables. In the next part of this chapter s, a and r refer to realisations of those random variables. Looping on the cycle of the figure 2.1 returns a sequence of random variables :

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

A finite MDP is defined by a discrete set of time steps $t_0, \ldots, t_n \in T$ and a tuple

$$\{\mathcal{S}, \mathcal{A}, p(s'|s,a), \mathcal{R}\}.$$

Here, $\mathcal{S}$ denotes the space that contains all observable states $\mathcal{S}_t$. This space can either be discrete or continuous. Similarly, $\mathcal{A}$ is the space that contains all possible actions. $\mathcal{A}(s)$ is the set of all possible action where the state is s.

For all realisations $s, s' \in \mathcal{S}$, rewards $r \in \mathcal{R}$ and $a \in \mathcal{A}(s)$, the dynamics of the MDP is given by the function

$$p(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]. \tag{2.1}$$

$p(s', r|s, a)$ represents the conditional transition probabilities, which denote the likelihood that the next state, S_{t+1}, and the reward, $\mathcal{R}_{t+1}$, are obtained from the state S_t by using the action A_t. This function domain is $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$.

As $p(s', r|s, a)$ is a probability distribution, it must satisfy the following property:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}(s).$$

The latter equation is the total probability. The conditional probability $p(s'|s, a)$, called the state-transition probabilities, can be obtained by summing all joint probabilities $p(s', r|s, a)$ for different rewards:

$$p(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, a_t = a] = \sum_{r \in \mathcal{R}} p(s', r|s, a). \tag{2.2}$$

Note that for simple cases, the reward r can be deterministic when reaching the state s'. So, the probability $p(s'|s, a) = p(s', r(s')|s, a)$. In a MDP, the probability given by p fully describes the environment dynamics.

So, each possible value for S_{t+1} and R_{t+1}, depends solely on S_t and A_{t+1}. The state should encompass details regarding all past interactions between the agent and the environment that hold significance for future outcomes. When this condition is met, the state is considered to possess the Markov Property. In this master thesis, we assume the Markov Property holds for all environments.

The expected reward $r(s,a)$ can be computed from a pair of state-action as $r : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$; it gives the expected value of the reward received when the system is in the state $S_t = s$ and when the action $a_t = a$ and when the next state is performed:

$$\begin{aligned} r(s,a) &= \mathbb{E}[R_{t+1}|S_t = s, a_t = a] \\ &= \sum_{r\in\mathcal{R}}\sum_{s'\in\mathcal{S}} r \;\; p(s', r|s, a) \\ &= \sum_{r\in\mathcal{R}}\sum_{s'\in\mathcal{S}} r \;\; p[S_{t+1} = s', R_{t+1} = r|S_t = s, a_t = a] \end{aligned} \tag{2.3}$$

The same reward can also be computed given a triplet of current state-action-next state $r(s,a,s')$, the function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is:

$$\begin{aligned} r(s,a,s') &= \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] \\ &= \sum_{r\in\mathcal{R}} r\frac{p(s', r|s, a)}{p(s'|s, a)} \\ &= \sum_{r\in\mathcal{R}} r\frac{p[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]}{p[S_{t+1} = s'|S_t = s, A_t = a]} \end{aligned} \tag{2.4}$$

For finite MDP, the unconditional expectation could also be:

$$\mathbb{E}[R_{t+1}] = \sum_{r\in\mathcal{R}} r \;\; p(s', r|s, a)$$

If the reward depends solely on s',i.e. $r(s,a,s') = k\forall a \in \mathcal{A}, \forall sin\mathcal{S}$, we can define $r(s') = r(s,a,s')$, the equation above can be rewritten as:

$$\begin{aligned} \mathbb{E}[R_{t+1}] &= \sum_{s'\in\mathcal{S}} r(s')p(s'|s, a) \\ &= \sum_{s'\in\mathcal{S}} r(s') \;\; \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] \end{aligned} \tag{2.5}$$

The last component of the MDP framework is the discount factor, γ which is a number between 0 and 1. This factor is applied to discount the value of future rewards. Thus, the total cumulative reward, given by the sum of all single-step rewards, is given by:

$$R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots \tag{2.6}$$

The discount factor controls the impact of futures far rewards over the rewards of the next few states. From an actuarial perspective, the discount factor can be likened to an actualisation factor.

The present value of all future reward at time t is given by G_t. Tt is called the discount factor:

$$G_t = \sum_{i=0}^{T-t-1} \gamma^i R_{t+i+1}. \tag{2.7}$$

How the agent chooses its actions depends on the current state of the environment and its experience. The agent has a probability of selected any action. The description of the probabilities is called the agent's policy, and they can be represented by a function π that maps a state S to a probability distribution over the possible actions:

$$\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A}(s))$$

where $p(A)$ could be discrete or continuous.

As mentionned earlier, the goal is to maximise the total cumulative reward, thereby optimising the decision-making process. It is done by selecting the best policy $\pi(a|s)$. It exists two measures of the quality of a policy:

- the value function, $v^\pi(s)$, which represents the expected total reward from following a policy π given a state s,
- the action-value function $q_\pi(s, a)$, which evaluated the expected future rewards from a specific state s when the first action a is taken.

Both approaches are related as shown in equation (2.10). $v^\pi(s)$ is calculated using the equation:

$$v_\pi(s) = \mathbb{E}^\pi \left[\sum_{i=0}^{T-t-1} G_t | S_t = s \right] \tag{2.8}$$

where R_{t+i} is a future reward at time $t+1$, and T is the planning time horizon; for finite MDP, $T < \infty$, for infinite MDP, $T = \infty$. The conditional expectation $\mathbb{E}^\pi[\cdot|S_t = s]$ is the expectation of the future rewards if the agent follows the policy π, the current time step is t. $\mathbb{E}^\pi[\cdot|S_t = s_0]$ is the optimal policy at origin. The state-value function $v_\pi(s)$ is the conditional expectation of the total cumulative reward.

Note that $v_\pi(s)$ depends on both the state s and the policy π, but is homogeneous in time, hence the notation in $v_\pi(s)$.

The second measure quoted above is the action-value function $q_\pi(s, a)$:

$$q_\pi(s, a) = \mathbb{E}^\pi \left[\sum_{i=0}^{T-t-1} G_t | S_t = s, a_t = a \right] \tag{2.9}$$

where a can be arbitrary. However, if a is given by the policy $\pi(a|s)$, then one gets:

$$v_\pi(s) = \mathbb{E}^\pi \big[q_\pi(s, a) \big] = \sum_{a \in \mathrm{A}} \pi(a|s) q_\pi(s, a) \tag{2.10}$$

Both functions depend on the conditional expectation of the future returns G_t of the states. One fundamental property of the value and action-value functions is recursivity. Indeed, the value function can be expressed as a recursive relationship:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}^\pi[G_t|S_t = s] \\ &= \mathbb{E}^\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}^\pi[R_{t+1}|S_t = s] + \gamma \mathbb{E}^\pi[G_{t+1}|S_t = s]. \end{aligned}$$

The first term is:

$$\mathbb{E}^\pi[R_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) r(s, a, s').$$

The second term of the equations can be rewritten:

$$\begin{aligned} \mathbb{E}^\pi[G_{t+1}|S_t = s] &= \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) \mathbb{E}^\pi[G_{t+1}|S_t = s, S_{t+1} = s'] \\ &= \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) \mathbb{E}^\pi[G_{t+1}|S_{t+1} = s'] \\ &= \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) v_\pi(s'). \end{aligned}$$

Finally, the recursive formula of the value function is:

$$v_\pi(s) = \sum_a \pi(a|s) \Bigg(\sum_{s'} p(s'|s, a) \Big(r(s, a, s') + \gamma v_\pi(s') \Big) \Bigg) \tag{2.11}$$

with $a \in \mathcal{A}(s)$ and $s' \in \mathcal{S}$.

The action-value function can also be expressed as a recursive relationship:

$$q_\pi(s, a) = \sum_{s'} p(s'|s, a) \Big(r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \Big) \tag{2.12}$$

where $s' \in \mathcal{S}$ and $a' \in \mathcal{A}(s')$.

The equations 2.11 and 2.12 are called the Bellman equations. There is one for state value and one for the action-value function.

The figure 2.2 presents the diagram of the step for a MDP. The black nodes represent action and the white node are states. The system is in state s. It must evaluate different actions a, a_1, a_2. The selection of the action a_1 is given by $\pi(a_1|s)$. From each action, the system goes to a new state s' according to p that maps a future state s' given a and s. The reward r is given by the environment and is obtained by gong from s to s' while executing a. The value of the state s is given by the equation 2.11 and the value of the action a is given by the equation 2.12.

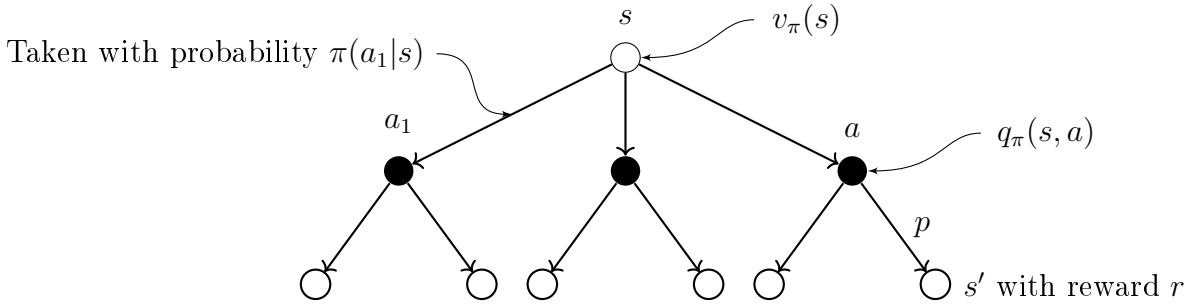

Figure 2.2: Diagram illustrating the relationships between a state s and other states. Where a, a_1 are possible actions from state s with $\pi(a_1|s)$ as the probability of taking action a_1 under the policy π, $v_\pi(s)$ is he value of the state s under the policy π, $q_\pi(s,a)$ is the action-value function, representing the expected cumulative reward starting from s, taking action a, and then following policy π, p is the transition probability, which is the probability of reaching a new state s' from state s after taking action a and r is the immediate reward received after taking action a in state s and transitioning to the new state s'.

Optimal policy

While comparing two policies, a policy π_1 is said to be better than π_2, if for every possible state $s \in \mathcal{S}$, $v_{\pi_1} \geq v_{\pi_2}$. Consequently, an optimal policy $\pi_\star$ is a policy such that

$$v_{\pi_\star}(s) \geq v_\pi(s) \quad \forall \pi, \forall s \in \mathcal{S}. \tag{2.13}$$

Thus, the optimal value function is given by:

$$v_*(s) = v_{\pi_\star}(s) = \max_\pi v_\pi(s).$$

The same optimal policy $\pi_\star$ can be defined for the action-value function. Then, the optimal action-value function is defined as:

$$q_*(s,a) = q_{\pi_\star}(s,a) = \max_\pi q_\pi(s,a).$$

The optimal policy $\pi_\star$ is the policy that maximises the action-value function $q_*(s,a)$ for all states s and actions a.

Introducing the optimal policy into the Bellman equations (2.11) and (2.12) gives:

$$\begin{aligned} v_\star(s) &= \max_{a \in \mathcal{A}} q_{\pi_\star}(s,a) \\ &= \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s')). \end{aligned} \tag{2.14}$$

and

$$q_\star(s,a) = \sum_{s',r} p(s',r|s,a)\big(r + \gamma \max_{a'} q_*(s',a')\big). \tag{2.15}$$

In the equations above, $p(s',r|s,a)$ is the probability of obtaining the reward r and the next state s' when taking the action a in the state s.

Note that $\sum_r p(s',r|s,a)r = p(s'|s,a)r(s,a,s')$. For the other terms of the sum, $\gamma v_*(s')$ and $\gamma \max_{a'} q_*(s',a')$, do not depend on r. So, $\sum_r p(s',r|s,a) = p(s'|s,a)$. The two equations right above are the Bellman optimality equations.

If a system returns the observations of the total returns G_t obtained at each state, one is able to estimate both functions. For a finite MDP model, with K discrete states, to obtain the value function or the state-value function, one could simulate or observe the sequence of actions, rewards and states obtained starting from each state of the K possibilities using the policy. The value of the functions is the average of the value obtains during the simulations. Those methods are called Monte Carlo methods [Sutton and Barto, 2018].

Dynamic Programming

Another method to deduce the state-value or the value function is to rely on their recursiveness. Dynamic programming methods are an approach to compute optimal policies given a perfect MDP model. It means a model where a computer can iterate over all possible actions and states [Sutton and Barto, 2018]. Those methods are based on the Bellman equations (2.11) and (2.12).

In this case, the optimal policy can be computed recursively. The initial state v_0 can be chosen arbitrary. The successive states are computed using the Bellman equations:

$$v_{k+1}(s) = \max_{a\in\mathcal{A}} \sum_{s',r} p(s',r|s,a)(r + \gamma v_k(s')). \tag{2.16}$$

for all $s \in S$. The algorithm stops when the difference between v_{k+1} and v_k is smaller than a given threshold. The serie v_k converges Indeed, the sequence v_k can be shown in general to converge to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π. This algorithm is called iterative policy evaluation. The similar methdology using q_π is called value iteration.

2.1.3 Markov Decision Process formulation to select a very simple investment strategy

The following section describes an investment strategy within a simplified, theoretical financial environment, and formulated as a Markov Decision Process. The problem is solved with the policy iteration algorithm in equation (2.16).

Consider an investor who aims to maximise their wealth over the next 12 months. At the beginning of each month, the investor must allocate their entire portfolio into one of three options:

- A safe investment,
- A risky investment,
- Holding cash.

Depending on the chosen investment option, the investor will either receive or lose a certain amount of money. If the investor's wealth reaches zero, it remains at zero for the remainder of the simulation. The objective of the strategy is to select the optimal investment option at each month in order to maximise the investor's wealth at the end of the 12-month period.

The key components of our MDP are as follows:

States: The state space consists of pairs (w, m), where w represents the discretized wealth level and m represents the month. Specifically, the states are defined as:

$$S = \{(w, m) \mid w \in \{0, 100, 200, \ldots, 1500\},\ m \in \{0, 1, 2, \ldots, 12\}\}.$$

The discretization of wealth values is performed within a range from 0 to a firm upper limit of 1500, progressing in steps of 100. Similarly, the month values are segmented from 0 through to 12.

Actions: The available actions at each state $s = (w, m)$ are:

$$A = \{\text{Invest safe}, \text{Invest risky}, \text{Hold cash}\}.$$

Given the selection of the action, the possible wealth changes are defined in Table 2.1.

	Action	**Wealth Changes**	**Probabilities**
1	Invest Safe	{-100, 100, 200}	{0.4, 0.4, 0.2}
2	Invest Risky	{-400, 100, 300}	{0.1, 0.5, 0.3}
3	Hold Cash	{0}	{1.0}

Table 2.1: Overview of investment strategies and associated probabilities available each month.

Transition Function: The transition function $p(s', a, s)$ specifies the probability of moving from state s to state s' given action a. It can be described as follows:

- If $w = 0$, i.e., the current wealth is zero, future wealth remains at zero:

$$p((0, m + 1), a, (0, m)) = 1$$

- For other states, taking action a results in wealth changes defined in Table 2.1:

$$\Delta w \in \texttt{changes}[a]$$

The new states are:

$$s' = (w + \Delta w, m + 1)$$

Probabilities associated with these transitions are given by:

$$P(\Delta w) \in \texttt{proba\_actions}[a]$$

For the "Invest Safe" action we obtain:

$$\begin{cases} p\big((w - 100w, m + 1), \text{Invest Safe}, (w, m)\big) & = 0.4 \\ p\big((w + 100w, m + 1), \text{Invest Safe}, (w, m)\big) & = 0.4 \\ p\big((w + 200w, m + 1), \text{Invest Safe}, (w, m)\big) & = 0.2 \end{cases}$$

These states are then mapped to the closest valid wealth states to have a limited discrete number of states:

$$(w', m') = \texttt{closest\_state}(w + \Delta w, m + 1)$$

Reward Function: The reward function $r(s', a, s)$ is defined as the change in wealth between the current state $s = (w, m)$ and the subsequent state $s' = (w', m+1)$. Mathematically, it is expressed as:

$$r((w', m + 1), a, (w, m + 1)) = \begin{cases} 0 & \text{if } w = 0 \\ w' - w & \text{otherwise} \end{cases}$$

Discount Factor: The discount factor γ is applied to future rewards to account for the time value of money. In this case, $\gamma = 0.9$.

Objective: The objective of the MDP is to maximise the expected cumulative reward over the planning horizon of 12 months. The value function $v(s)$ for a state s is defined as:

$$v(s) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_{t+1}, a_t, s_t) \mid s_0 = s\right]$$

where s_t and a_t represent the state and action at time t, respectively.

Figure 2.3 illustrates the outcomes of the dynamic programming algorithm applied to the MDP presented above. The model generally opts to maximise future profits by frequently investing in the more volatile asset, the "risky asset." When initial wealth approaches the upper limit, the agent tends to avoid the risky asset for the first few months. While the risky asset offers a higher average expected return, it also carries the potential to reduce wealth to zero before the end of the year. Furthermore, the maximal value of the discrete space caps the return. It is important to note that the reward structure indicates that the investor is not indifferent to risk. Employing a different utility function by altering the reward would yield different outcomes.

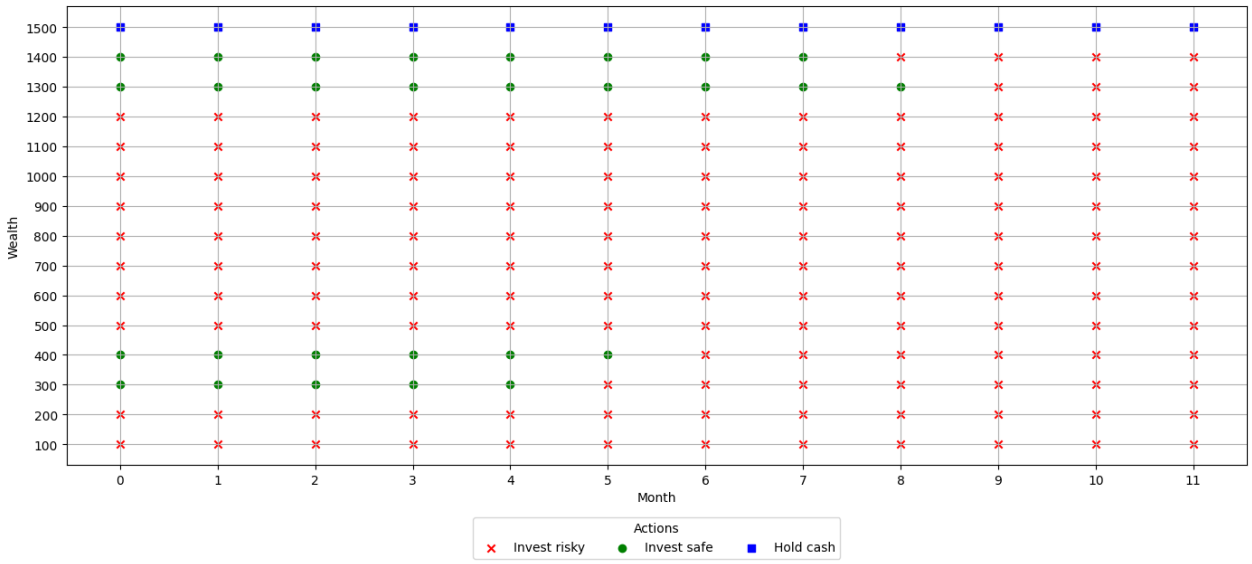

Figure 2.3: Action with best expected future return for each state of wealth and month. The wealth values are discretized from 0 to 1500 in 100 increments, and months range from 0 to 12. Month 0 means initial action taking before the start of the simulation. Actions of the 11th month are applied for the 12th month of the simulation. Consequently, the month 12 is not represented on the plot because actions taken at this point have no impact.

The example presents a theoritical case where all possibles combination of states and actions are computable. However when the space for actions or spaces becomes larger, it is no longer possible to solve the Bellman equation with Dynamic Programming. In this case, a function approximates the true value of $q_t(x, a)$. In the algorithm, the approximation replaces the Q-value. In policy gradient, one does not look at the best combination of state action but wants to optimise $v_t(x)$, i.e., to select the state whose expectation is the highest.

2.2 Deep reinforcement learning

So far, our assumption was that at each point in time, we are dealing with a finite set of possible states that we store in matrices. However, this approach becomes unbearable when we have an very large of infinite number of possible states. To address this issue, we can use function approximation. The idea is to approximate the Q-value function by a function of the state and action. These approximation can be with linear or non-linear functions.

Some choices for basis functions in a linear architecture could include B-spline, polynomial, or trigonometric functions [Sutton and Barto, 2018]. These are convenient for theoretical cases because they are easy to solve. However, in real-world applications involving high-dimensional data, highly non-linear data, or both, the identification of an appropriate basis function can become a more complex task. To overcome this limitation, function approximation is employed. The central notion is to approximate the action-value function, commonly known as the Q-value function, using a function that depends on both the state and action. These approximations can leverage either linear or non-linear functional forms.

In the linear case, the approximated action-value function, $q_\pi(s_t, a_t)$, takes the following form:

$$q_\pi(s_t, a_t) = \boldsymbol{W}_t^\top \boldsymbol{\Psi}(s_t, a_t) = \sum_{k=1}^{K} W_{tk} \Psi_k(s_t, a_t), \tag{2.17}$$

where $\boldsymbol{W}_t$ represents a weight vector and $\boldsymbol{\Psi}(s_t, a_t)$ denotes a set of basis functions, which are predetermined functions that map the state and action space into a finite-dimensional space. These basis functions are defined over the Cartesian product of the state and action spaces, $\mathcal{S} \times \mathcal{A}$, indexed by $k = 1, \ldots, K$. The weight vector $\boldsymbol{W}_t$ is updated iteratively at each time step t to minimize some measure of error in the Q-value approximation, typically through algorithms such as temporal difference learning or policy gradient methods.

optimisation of the weight vector is commonly performed using gradient descent or one of its variants. During each iteration, the weights are adjusted in the direction that reduces the expected difference between the approximated and the true Q-values.

Expanding on the optimisation of the weight vector requires introducing the objective function used in the context of RL. In most cases, the objective is to minimize the mean squared error between the predicted Q-values from the function approximator and the true Q-values, which we often estimate through various forms of bootstrapped targets.

The true value of the Q-function, $q^*(s, a)$, is the expected return when taking action a in state s and thereafter following an optimal policy. Since the true Q-value is initially unknown in a RL setting, it is commonly approximated using the Bellman equation, which describes the relationship between the current and subsequent Q-values.

Let's define the components of the optimisation:

Objective Function: The goal is to find weights $\boldsymbol{W}$ that minimize the error between the approximated Q-values and the true Q-values. A typical objective function, $J(\boldsymbol{W})$, used for optimisation is the mean squared error (MSE):

$$J(\boldsymbol{W}) = \mathbb{E}\left[\left(Q_{target}(s, a) - \hat{q}(s, a; \boldsymbol{W})\right)^2\right], \tag{2.18}$$

where $\hat{q}(s, a; \boldsymbol{W})$ represents the approximated Q-value, which is a function of state s, action a, and weights $\boldsymbol{W}$. $Q_{target}(s, a)$ is the target (or "true") Q-value, which is often computed using a bootstrapping method like temporal difference learning.

Temporal Difference Target: The target for the Q-values is generated by using the Bellman equation for TD learning:

$$Q_{target}(s, a) = r + \gamma \max_{a'} \hat{q}(s', a'; \boldsymbol{W}), \tag{2.19}$$

where r is the reward received after executing action a in state s, s' is the new state after the action is taken, and γ is the discount factor. The maximum is taken over all possible actions in the new state s'.

Gradient Descent: To minimize the objective function $J(\boldsymbol{W})$, gradient descent is employed:

$$\boldsymbol{W}t + 1 = \boldsymbol{W}t - \alpha \nabla_{\boldsymbol{W}} J(\boldsymbol{W}_t), \tag{2.20}$$

where α is the learning rate, and $\nabla_{\boldsymbol{W}} J(\boldsymbol{W}_t)$ is the gradient of $J(\boldsymbol{W})$ with respect to the weight vector $\boldsymbol{W}_t$ at iteration t. This gradient is an estimate and is computed based on the current approximation of the Q-value function and the target Q-value.

This optimisation proceeds iteratively, updating the weight vector at each time step to reduce the error between the approximated and target Q-values. Since the true Q-value is unknown, we use the TD target as an estimate to guide the learning process. This approach is known as TD learning with function approximation. Through this process, the policy and the Q-value function approximation are iteratively improved, with the objective of converging toward the optimal policy and Q-value function.

A very simple version of the algorithm with linear weights returns the following.

$$\begin{aligned}
\Delta w &= \alpha(r + \gamma \max_{a'} \hat{Q}(s',\, a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}))\nabla_w \hat{Q}(s, a; \mathbf{w}) \\
\Delta w &= \alpha(r + \gamma \hat{V}^{\pi}(s'; w) - \hat{V}^{\pi}(s; w))\nabla_w \hat{V}^{\pi}(s; w) \\
&= \alpha(r + \gamma \hat{V}^{\pi}(s'; w) - \hat{V}^{\pi}(s; w))x(s) \\
&= \alpha(r + \gamma x(s')^T w - x(s)^T w(s))x(s)
\end{aligned}$$

2.2.1 Neural networks

Rather than relying on handcrafted functions or those created by algorithms, generally defined as parameter-based functional transformations of the original data, we can turn to universal function approximation methods like trees or Neural Networks. Deep Reinforcement Learning (DRL), a subset of these methods, employs multi-layer neural networks to portray value and/or policy functions (see Fig. 2.4). This is especially handy when the action-value function is highly non-linear and no clear choice for a set of basis functions can be readily identified. The present work deals with a non-linear function approximation. In this case, the v is approximated by a neural network. The neural network is trained to minimize the loss function between the true Q-value and the approximated Q-value.

In this master thesis our focus will be on the use of neural networks to approximate the Q-value. The neural network is trained to minimize the loss function which represents the difference between the actual and approximated Q-values. As mentionned above, such process is called Deep Reinforcement Learning.

2.3 Proximal Policy optimisation

Proximal Policy optimisation (PPO) is an advanced RL algorithm that has become popular due to its stability and efficiency [Schulman et al., 2017]. At its core, PPO seeks to make

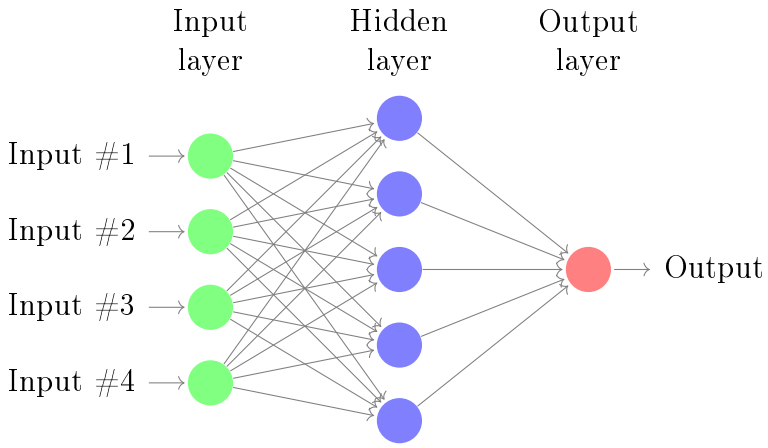

Figure 2.4: Example of a multi-layer neural network with one hidden layer (in blue).

incremental policy updates, avoiding drastic changes that could disrupt the learning process. By optimising a clipped version of the objective function, PPO ensures that the magnitude of the updates are kept controlled. In the family of the mahchine learning method, see figure 1.1, it is a RL method.

Taking a brief detour at the world of video games first: PPO has made significant strides. Picture a game where an AI-controlled character is tasked with navigating a complex maze. Instead of wildly erratic strategies that often lead to failure, PPO ensures that the character undergoes incremental learning steps. This means that the character, over time, learns to master the maze by taking consistent and logical paths without constantly bumping into walls or ending up trapped. As a result, the character's learning process becomes smoother and more predictable, enhancing the overall gaming experience [Shao et al., 2019].

Transitioning to the actuarial world, let's consider a situation where an insurance company aims to optimise their portfolio by predicting policyholder behaviors, such as when they might lapse a policy or make a claim. With traditional methods, these predictions could vary wildly based on new data, making it hard for the company to strategize effectively. With PPO, actuaries can model these behaviors with more consistency. The algorithm ensures that predictions don't swing too drastically with each new piece of data, allowing for more stable and reliable forecasting. For instance, in creating a model to predict policy lapses, PPO can help in adjusting the model's predictions in light of new data, but without making such significant jumps that would render previous strategies obsolete. This consistent approach allows insurance companies to make more informed and balanced decisions, benefiting both their operations and their clients.

Moreover, PPO incorporates the concept of entropy to maintain a balance between exploration and exploitation during the learning process [Schulman et al., 2017]. Entropy, in this context, acts as a measure of uncertainty or randomness in the policy's decision-making, encouraging the model to continue exploring different strategies rather than converging too quickly on potentially suboptimal solutions. For example, when predicting policy lapses, the introduction of entropy in PPO allows the model to explore a wider range of scenarios and behaviors. This means that, while the model remains focused on providing consistent and

reliable predictions, it also retains the flexibility to adapt to new data patterns and uncover novel insights that might have been overlooked by more deterministic approaches. This balance is crucial for actuaries, as it enables them to develop strategies that are not only robust and stable but also responsive to the dynamic nature of financial markets and policyholder behavior.

A crucial element of PPO is its objective function, which combines multiple components to guide policy updates effectively. The objective function is given by:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathrm{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)], \tag{2.21}$$

where $L_t^{\mathrm{CLIP}}(\theta)$ is the clipped surrogate objective, $L_t^{\mathrm{VF}}(\theta)$ represents the value function loss, and $S[\pi_\theta](s_t)$ corresponds to the entropy bonus term. The coefficients c_1 and c_2 are hyperparameters that control the trade-off between these terms. PPO is an on-policy algorithm. It is an actor-critic algorithm. The actor is a stochastic policy. The critic is a value function. The critic is updated using the Bellman equation. The actor is updated using the critic.

The clipped surrogate objective $L_t^{\mathrm{CLIP}}(\theta)$ ensures that the policy updates remain stable by preventing overly large updates that could destabilize the learning process. The value function loss $L_t^{\mathrm{VF}}(\theta)$ aids in reducing the discrepancy between the predicted and actual returns, thereby improving the accuracy of value estimates. Finally, the entropy term $S[\pi_\theta](s_t)$ encourages exploration by penalizing overly deterministic policies, thus maintaining a healthy level of randomness in action selection. The algorithm 1 describes the PPO-Clip algorithm.

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of estimating the advantage) based on the current value function v_{ϕ_k}.
6: Update the policy by maximising the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

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

typically via some gradient descent algorithm.
8: **end for**

PPO has already beend used in the literature for portfolio optimisation [Sood et al., 2023], [Yang et al., 2020] and [Hambly et al., 2021]. When evaluating the policy, the agents seeks to find the minimum of the loss function (2.21). In the latter chapter, when finding the best policy to optimise the portfolio return, the algorithm used PPO to search for it.

Chapter 3

Asset and Liability Management model

Insurance companies and pensions funds share a common task, with different degrees of complexity which are Asset and Liability Management (ALM). In this context, the institutions manage both their assets and liabilities aiming to ensure long-term financial stability and effectively handle risks. As part of this responsibility, insurers assess the timing and the nature of their obligations on the liability side of their balance sheet. These obligations may include pension payments, debts repayments, insurance claims or other contractual liabilities. Furthermore, ALM is generally applied over a long period of time. Due to computing limitations, this presentation shows the evolution of the liability over a one-year horizon. It and must take into account the regulatory constraints or the solvency requirements. The mathematical models behind ALM were developped in the early 2000's and have since evolved to accommodate the multifaceted nature of this task. For a complete description of the portfolios modeling of life insurance policies, we refer to Gerstner et al. [2008]. This resource details the mathematical approaches used to balance assets and liabilities over time, taking into account a diverse range of potential future obligations that insurance companies and pension funds may face.

Despite the growing interest in Reinforcement Learning (RL) in Finance, there are only a few published papers that apply it to ALM. Fontoura et al. [2019] first applied the RL tools to ALM, they solve an ALM problem modeled as a Markov Decision Process, avoiding scenario discretization and showing promising results with the use of continuous state spaces. Abrate et al. [2021] added more complexity by introducing a stochastic model that merges portfolio returns with liabilities from insurance products, proposing a risk-adjusted optimisation problem to maximise a company's capital over a specific timeframe. Finally, Wekwete et al. [2023] focuse on the duration matching : the optimal selection of assets such that the flux of future cashflows on the assets side matches the future cashflows on the liabilities side regarless of the evolution of the yield curve.

The following sections introduce the asset and liability management model considered in this study. It is worth noting that this is not pure ALM, rather, it is more akin to portfolio allocation under specific constraints as the objective is not to select assets to match the cashflow patterns of the liabilities but to optimise the return given the insurer must honour its policyholders. The company seeks to optimise its assets' allocation up to a time horizon

T. The timespan from the start of the simulation t_0 and $t_0 + T$ is discretized into timesteps of size dt. At each step, the insurance portfolio has outgoing cashflow that it must meet. In the present model, the outgoing cashflow represents policyholders reaching the product maturity, passing away, or exercising their surrender options. These expected cashflows form a guaranteed product with a bonus, shaping a part of the company's liabilities. The insurer may have other products whose premiums are invested into other buckets.

3.1 Assets

In this model, the total assets at time t are represented:

$$A_t = \sum_{i=1}^{N} A_{t,j},$$

where $A_{t,j}$ denotes the amount invested in the asset j at time t.

The fund manager has to allocate the underwritten premium among J assets. The strategic decision for the insurance company revolves around determining the optimal allocation of its assets across. The goal is to ensure the fulfillment of its financial obligations and maximise overall returns.

This allocation strategy is an optimisation problem with specific constraints. The problem fits nicely within the scope of Reinforcement Learning , which specializes in solving such problems. Reinforcement Learning algorithms can be used to determine the optimal asset allocation strategy, taking into account the potential rewards (maximised returns) and constraints (financial obligations). RL posesses the ability to learn from past experiences and adjust strategies iteratively based on feedback received from the environment. This adaptive learning capability is interesting in this context where market conditions evolve over time. Additionally, Reinforcement Learning frameworks are adept at handling uncertainty and non-linear relationships inherent in financial markets. Finally, the continuous nature of financial markets offer almost continuous fedback to the RL agent [Théate and Ernst, 2021].

The model takes into account fund prices, which are derived from market values denoted as $f_{MV,i}$. Hence, equity prices are subject to the volatility and dynamism of the real market. Each asset j, has a price at time t. The value of the asset invested in i denoted by:

$$A_{t,j} = w_{t,j} \times f_{MV,j}(t) \qquad \forall j \in \mathcal{J} \tag{3.1}$$

This equation emphasizes that equity prices aren't fixed, but vary as the market changes.

At each timestep, the model should return the optimal weights $\mathbf{w_{*t}} = (w_{t,1}, \ldots, w_{t,J})$ such that:

$$\mathbf{w_{*t}} = \arg\max_{\mathbf{w_t}} A_t = \arg\max_{\mathbf{w_t}} w_{t,j} * f_{MV,j}(t)$$

Asset prices can either be simulated as a Brownian motion, or another stochastic process, or based on historical data. This work used the latter approach as it avoids hypotheses on

the market dynamics and as it yields more interpretable result since the asset are based on existing assets.

3.2 Liabilities

The liabilities refer to a guaranteed insurance product that provides specific cashflows to policyholders upon the occurrence of certain events, such as death, surrender, or the completion of a specified investment period of maturity M. This product offers multiple features, including surrender penalties and benefits that vary depending on the policyholder's actions and the duration for which the investment is held. Upon death or surrender before maturity, the policyholder receives the guaranteed capital plus accrued returns. If the policy is held until maturity, the policyholder is entitled to receive the guaranteed capital, the accrued returns, and an additional bonus.

At any time t, there are Y originating years, representing the years when policyholders entered the product, and D different ages. The number of policyholders having entered the product in year y at age d is denoted at time t as ϕ_t^{dy}. The life insurance product guarantees a return on capital at a fixed rate i. Upon entry, the policyholder pays an initial cost C_{in}, which is invested in a fund managed by the insurance company. The payout is expressed as:

$$C_{\text{out}} = \begin{cases} C_{\text{in}} \times (1+i)^{t-t_0} & \text{if } t \text{ is the time of death} \\ C_{\text{in}} \times (1+i)^{t-t_0} \times \alpha & \text{if the policyholder surrenders at time } t \text{ and before } T \\ C_{\text{in}} \times ((1+i)^M + \text{bonus}) & \text{if the policyholder survives until } T \end{cases}$$

where α is a *penalty* ratio of the initial cost to be deducted in case of early surrender, t_0 is the entry time and T is the maturity time, and $M = T - t_0$. The bonus is only paid if the final portfolio value exceeds the guaranteed return, ensuring that policyholders receive a minimum guaranteed return with the potential for additional earnings if the portfolio performs well:

$$\text{bonus} = \max\left(\left\{\frac{A_T}{A_0} - (1+i)^M\right\} \times \beta, 0\right)$$

where $\frac{A_T}{A_{t_0}}$ is the return of the asset between T and t_0, and β is the bonus rate. The return $\frac{A_T}{A_{t_0}}$ is computed without substracting outgoing cashflows between t_0 and T. The policyholder should not be impacted by earlier outflows to other affiliates. Nonetheless, no bonus is given if the portfolio value reaches 0. The bonus rate β is a percentage. This framework ensures that policyholders do not receive entirely the additional return in the case of survival. The idea behind this lies into parts. Firstly, the fee provides a safeguard against outgoing cashflows prior to the policyholder's survival. Specifically, it mitigates the risk of death and surrender cashflows depleting the portfolio during periods when asset values are declining. Secondly, the fee allows the insurer to realize a modest profit.

The outgoing cashflows at time t are computed recursively starting from t_0. At t_0, the product can already be alive for some years. The different cashflows are computed according

to algorithm 2. In the procedure, two possible Δt, period between two computation of liabilities values, are accepted : a year or a day. If Δt is a year, once a contract reaches the maturity, it is liquidated. If Δt is a day, if a contract reaches a maturity it exits the portfolio during it's following year with a uniform probability. It simulates the continuous arrival of contract during the starting year. Indeed, a contract created on the 31th of December in year Y only reaches maturity on the 31th of December of the year $Y + M$.

Algorithm 2 Evolution of Policyholders and cashflows in the Portfolio

for each starting age d and each insertion year y **do**
  Initialize $\phi_0^{y,d} \leftarrow$ initial number of policyholders at age d at the start of year y
  for each time t until T with Δt the time spent between t and $t-1$ **do**
    Initialize $alive_t^{y,d} \leftarrow \phi_0^{y,d}$
    if $alive_{t-1}^{y,d} = 0$ **then**
      $dead_t^{y,d} = surrenders_t^{y,d} = exiters_t^{y,d} = alive_t^{y,d} = 0$
    else
      $q_{\Delta t} \leftarrow$ get mortality rate for period Δt
      $s_{\Delta t} \leftarrow$ get surrender rate for period Δt
      $dead_t^{y,d} \leftarrow alive_{t-1}^{y,d} \times q_{\Delta t}$
      $surrenders_t^{y,d} \leftarrow (alive_{t-1}^{y,d} - dead_t^{y,d}) \times s_{\Delta t}$
      death benefits$_t^{y,d} \leftarrow dead_t^{y,d} \times C_{in} \times (1+i)^{t-t_0}$
      surrender benefits$_t^{y,d} \leftarrow surrenders_t^{y,d} \times C_{in} \times (1+i)^{t-t_0} \times \alpha$
      if $t > t_0 + M$ **then**
        if Δt is a year **then**
          $exiters_t^{y,d} \leftarrow alive_{t-1}^{y,d} - dead_t^{y,d} - surrenders_t^{y,d}$
        else ▷ Δt is a day
          alive before exit $\leftarrow \left(alive_{t-1}^{y,d} - dead_t^{y,d} - surrenders_t^{y,d}\right)$
          $exiters_t^{y,d} \leftarrow$ alive before exit$\times$ daily exit probability
        end if
      else
        $exiters_t^{y,d} \leftarrow 0$
      end if
      $alive_t^{y,d} \leftarrow alive_{t-1}^{y,d} - dead_t^{y,d} - surrenders_t^{y,d} - exiters_t^{y,d}$
      exit benefits$_t^{y,d} \leftarrow exiters_t^{y,d} \times C_{in} \times \left((1+i)^{(t_0+M)} + \text{bonus}\right)$
    end if
    Update $\phi_t^{y,d} \leftarrow alive_t^{y,d}$
  end for
end for

Hence, at time t, the total outgoing cashflows is:

$$L_t = \sum_{y \in Y} \sum_{d \in D} \text{death benefits}_t^{y,d} + \text{surrender benefits}_t^{y,d} + \text{exit benefits}_t^{y,d}. \tag{3.2}$$

3.3 Evolution of the assets and liabilities

This subsection discusses the evolution of assets and liabilities over time. To maintain solvency, the value of the assets must always exceed the outgoing cashflows at each time step. The asset value evolves according to the following equation:

$$A_t^{\text{before cashflows}} = \sum_{j \in J} w_{t,j} \times f_{MV,j}(t) \tag{3.3}$$

$$r^{CF} = \frac{A_t^{\text{before cashflows}}}{A_{t-1}^{\text{before cashflows}}} - 1 \tag{3.4}$$

$$A_t = A_t^{\text{before cashflows}} - L_t \tag{3.5}$$

$$r^{RL} = \frac{A_t}{A_{t-1}} - 1 \tag{3.6}$$

where r^{CF} is the return of the assets without taking into account the cashflows, it is used to compute the return of the bonus and r^{RL} is the return of the assets taking into account the cashflows, it is used to compute the real return of the assets. In the latter chapter, Reinforcement Learning methods are based on this return. Note that there is no bonus if $A_t^{\text{before cashflows}} - L_t < 0$. If the asset value decreases to zero or less, the simulation stops because of insolvency. If the insolvency is avoided, the simulation will continue until it reaches the defined horizon at time T.

3.4 Backtesting

The chapter 5 gives an application of the Reinforcement Learning to allocation a portfolio. More traditional methods backtest the machine learning methods. This first one is based on the static asset allocation with the maximisation of the mean variance, the second is a simple equally weighted portfolio.

Algorithm 3 presents the process for calculating the portfolio allocation with Mean-Variance optimisation [Cornuéjols et al., 2018]. In the procedure, r refers to the return and not that reward.

The buy and hold strategy, conversely, involves purchasing the index at the initial time, t_0, and selling only to satisfy outgoing cashflows. By comparing the performance of RL with these more traditional strategies, this research aims to highlight the potential benefits and improvements offered by the use of Reinforcement Learning in the management of asset and liability portfolios.

Algorithm 3 Markowitz Mean-Variance Portfolio optimisation for Sharpe Ratio

Input:

Asset returns r such that $r_j, \text{ for } j = 1, 2, \ldots, J$

Covariance matrix Σ such that $\Sigma_{j,k} = \text{Cov}(r_j, r_k), \text{ for } j, k = 1, 2, \ldots, J$

Risk-free rate r_f, in this case the interest rate of the liabilities

Find w that maximises the Sharpe ratio: $w = \arg\max \left(\dfrac{r^T w - r_f}{(w^T \Sigma w)^{1/2}} \right)$

Output:

Optimal asset weights $w_j, \text{ for } i = 1, 2, \ldots, J$

Chapter 4

Implementation of the ALM in the MDP setup

Reinforcement Learning (RL) has been applied in finance in different contexts [Hambly et al., 2021]. Significant progress has been made in areas such as options pricing and in optimal execution. Reinforcement Learning has also been applied to portfolio allocation [Sood et al., 2023]. In this context, the manager of a portfolio seeks to optimise some objective function by trading assets. One can maximise its pure return or balance the return with risk measures taking risk into account. With a portfolio allocation strategy, one expects higher return over a long period of time than while keeping a unique asset. Yang et al. [2020] proposes an ensemble of different Deep RL algortihms to solve the portfolio allocation problem. In their approach, the goal of the reward function is to maximise the portfolio return. They measure the efficiency of their algorithm by comparing the Sharpe ratio over time with respect to the "buy and hold" strategy and the mean-variance model [Markowitz, 1952]. The Sharpe ratio, in this case, provides a measure of the return achieved per unit of risk taken, allowing for a fair comparison of the strategies. By leveraging deep reinforcement learning algorithms, they aim to create a portfolio allocation strategy that outperforms traditional methods while balancing risk and return.

This chapter details the implementation of the portfolio allocation with liabilities model within the context of RL, based on the theoretical framework presented in Chapters 2.1 and 3. Firstly, the chapter describes how the problem can be represented as a MDP. This involves describing the agent and the environment, and the algorirthm considered. Then it presents the algorithm used for the applications in chapter 5.

4.1 Agent

In our study, the agents are directly taken from the Stable Baselines 3 implementations [Raffin et al., 2021], which offer a broad choice of reinforcement learning agents. The framework notably offers an implementation of the PPO algortihm where the loss function as well as other parameters are customable. The study of the hypperparameters depends on the dataset. PPO was explained in 2.3.

4.2 Environment

The overall environment in which our model operates is characterized by several components: termination condition, the observation space, action space, reward function.

4.2.1 Initial and terminal states

At the initial step, all the assets are invested in cash. A state is terminal either if it reaches the last trading day withint the dataset or if the portfolio value is below the liabilities. In the latter case, the model is considered to be in ruin and the episode is terminated on the ruin date.

4.2.2 Observation and action spaces

The observation space describes all relevant information that the model can perceive. For our specific use case, this primarily includes the prices of the assets. For each trading day, the model has access to the open prices,highest price of the day, lowest price of the day, the close price as well as the trading volume of each asset. Furthermore, the model can also invest in cash. The model can access the prices for the past 50 trading days. On top of asset data, the environment knows the state of liabilities, the outgoing cashflow, the number of shares of each asset hold.

Regarding the action space, it is continuous. The actions are the weight of the portfolio in each asset and in cash as well. Since the sum of the weights of the portfolio must be equal to 1, the sotfmax normalisation is applied to the actions returned by the agent.

4.2.3 Reward function

This section introduces several reward functions applied on the model: the differential return, the log return, the differential Sharpe Ratio, and the differential Downside Deviation Ratio. Each method has its own way of taking risk into account. A key hypothesis driving our investigation is that modifications in risk perspectives can have pronounced effects on asset stability within our portfolio. For example, one method of considering risk might lead to predominant allocation towards more risky but with better average assets than the average. Conversely, another risk perspective might favor assets with other characteristics. Such allocation decisions greatly determine the portfolio's resilience against market fluctuations and its potential for consistent returns. The next section gives setails on each reward function. In this section, the notation R_t means the reward at time t, it is not a return. On top of the reward below, there is a penalty if the portfolio value is below the liabilities to increase the impact of not reaching the last day of the dataset.

Differential return

For this method, the reward is computed at the return of the asset before outgoing cashflows during the time step. The differential return follow the same formula as equation (3.6):

$$R_t = \frac{A_t}{A_{t-1}} - 1$$

where R_t is the return of the step and A_t is the portfolio value at time t. This approach aims to maximise the return without consideration for the risk of the position.

Log return

The Log Return offers a straightforward way to measure the performance of a portfolio over time. Specifically, the reward at time t is determined by the logarithmic rate of return, which looks at the change in the portfolio's value from time $t-1$ to t. Mathematically, this is given by:

$$R_t = \ln\left(\frac{A_t}{A_{t-1}}\right)$$

where R_t is the return of the step and A_t is the portfolio value at time t.

It tends to prioritize higher returns, which can be beneficial in certain investment contexts. However, a significant drawback is its disregard for risk similarly to the differential return. While it indicates how well a portfolio is performing, it doesn't account for the risk associated with achieving that performance.

Differential sharpe ratio

Evaluating the effectiveness of a portfolio allocation strategy can be complex. An effective strategy should not only aim for high returns, but also efficiently manage the associated risks. The equilibrium between these two objectives depends on the investor's profile and risk tolerance.While maximising returns intuitively appears to be the primary goal, it is a necessary but not sufficient condition for a successful portfolio strategy. A more comprehensive objective is to maximise the Sharpe ratio, a performance measure widely used in finance and portfolio management [Sood et al., 2023].

The Sharpe ratio is particularly suited for assessing portfolio performance because it considers both the returns generated and the risk associated with the portfolio strategy. Following Moody and Saffell [2001], the Sharpe ratio is defined as follows:

$$S_t = \frac{\text{Average}(r_t)}{\text{Standard deviation}(t)} = \frac{A_t}{\sqrt{\frac{t}{t-1}\left(B_t - A_t^2\right)}}, \tag{4.1}$$

where:

$$A_t = \frac{1}{t}\sum_{i=1}^{t} r_i, \tag{4.2}$$

$$B_t = \frac{1}{t}\sum_{i=1}^{t} r_i^2. \tag{4.3}$$

The Sharpe ratio can serve as a basis for designing a reward function in reinforcement learning. It helps to optimise not just for returns, but for returns that are risk-adjusted. The reinforcement learning agent learns to make decisions that lead to higher risk-adjusted returns rather than simply higher raw returns. However, directly using the Sharpe ratio as reward leads to Bias.

From the equation (4.1), we can derive the differential sharpe ratio and use it as the reward as follows. A and B can be recursively estimated as exponential moving averages of the returns and standard deviation of returns on time scale η^{-1}.

$$A_t = A_{t-1} + \eta\Delta A_t,$$
$$B_t = B_{t-1} + \eta\Delta B_t,$$

$$\Delta A_t = r_t - A_{t-1},$$
$$\Delta B_t = r_t^2 - B_{t-1}$$

where η is the adaptation rate.

The Sharpe ratio D_t can be computed using the first order taylor approximation:

$$S_t \approx S_{t-1} + \eta D_t\Big|_{\eta=0} + O(\eta^2). \tag{4.4}$$

where D_t is the differential Sharpe raio. Its values is::

$$D_t \equiv \frac{\partial S_t}{\partial \eta} = \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{(B_{t-1} - A_{t-1}^2)^{3/2}}. \tag{4.5}$$

Finally, the reward is se to be the differential Sharpe ratio: $R_t = D_t$

A key feature of the Sharpe ratio is its equal treatment of both upward and downward variations in returns. It promotes strategies that aim for consistent returns over those that might have high returns but with high volatility. However, a fundamental assumption of the Sharpe ratio is that returns are normally distributed. This poses a limitation because financial returns, in reality, might not always follow a normal distribution [Biglova et al., 2009]. Specifically, there can be instances where returns are skewed away from the mean with an asymmetric distribution. This asymmetry could lead to situations where potential risks, especially those associated with outliers or extreme events, are not fully captured by the Sharpe ratio.

In our context, where assets are modeled deterministically based on historical market data, there are no assumptions made about the underlying nature of the assets or their evolution.

Differential Sortino ratio

Another method to consider risks is to consider risk-adjusted returns. Instead of just looking at raw gains, they consider the volatility and potential downsides. This allows for a better comparison between investments, ensuring that risks are justified by the returns. In other words, it is a method that ensures that the level of risks taken are proportionate to the returns expected, aligning with investors' varying preferences for price fluctuations [Moody and Saffell, 2001].

The Downside Deviation (DD) is a risk-adjusted return function that measures the volatility of returns below a certain threshold, typically zero or a minimum acceptable return. This would encourage the reinforcement learning agent to avoid strategies that have frequent or significant losses. Following Moody and Saffell [2001], the downside deviation is defined as:

$$\mathrm{DD}_t = \left(\frac{1}{t} \sum_{i=1}^{t} \min\{r_i, 0\}^2 \right)^{\frac{1}{2}}. \tag{4.6}$$

The downside deviation ratio is a special case of the Sortino ratio [Sortino and Price, 1994]. In the Sortino ratio, the downside deviation is defined with respect to a target return i. In the present case, the target return is the interest rate offered to policyholders. The ratio is:

$$\mathrm{Sortino}_T = \frac{\mathrm{Average}(r_t)}{\left(\frac{1}{T} \sum_{t=1}^{T} \min\{r_t, i\}^2 \right)^{\frac{1}{2}}}. \tag{4.7}$$

We use it as a measure of risk to define an utility function called the differential Sortino ratio (DSR) [Acero et al., 2024]:

$$\mathrm{DSR}_t = \frac{\mathrm{Average}(r_t)}{\mathrm{Sortino}_t} = \frac{A_t}{\mathrm{Sortino}_t}. \tag{4.8}$$

where A_t is defined by equation (4.2).

The Sortino Ratio provides a relevant measure for distributions where negative returns are not symmetrical to positive returns because it only considers the downside volatility. Similarly to the differential Sharpe ratio from equation (4.5), we describe the exponential moving averages for both returns and the Sortino ratio:

$$\begin{aligned} A_t &= A_{t-1} + \eta(r_t - A_{t-1}) \\ \mathrm{DD}_t^2 &= \mathrm{DD}_{t-1}^2 + \eta(\min\{r_t, i\})^2 - \mathrm{DD}_{t-1}^2. \end{aligned}$$

Finally the differential Sortino Ratio D_t is given by:

$$D_t \equiv \frac{d\mathrm{DSR}_t}{d\eta}$$

$$D_t = \begin{cases} \dfrac{r_t - \frac{1}{2}A_{t-1}}{\mathrm{DD}_{t-1}} & \text{if } r_t > 0 \\ \dfrac{\mathrm{DD}_{t-1}^2 \cdot \left(r_t - \frac{1}{2}A_{t-1}\right) - \frac{1}{2}A_{t-1}r_t^2}{\mathrm{DD}_{t-1}^3} & \text{if } r_t \leq 0 \end{cases}$$

Finally, the reward is set to be the differential Sortino Ratio: $R_t = D_t$.

Note that the differential downside deviation ratio is not defined for $t = 1$.

Approach

The algorithm 4 describes how one can implement the evolution of the assets and liabitlies with a generic reinforcement learning algorithm dictating the asset allocation.

Algorithm 4 Asset and Liability Management (ALM) Algorithm

Initialization:
Initialize portfolio weights: $w_1, w_2, \ldots, w_n$
Initialize liability amounts: $L_1, L_2, \ldots, L_t$
Initialize asset values: $A_0, A_1, \ldots, A_t$
Initialize cashflows: $C_1, C_2, \ldots, C_t$
for $t = 1$ to T **do**
    Asset Rebalancing:
    Calculate portfolio return: $r_t^{CF} = \dfrac{A_t^{\text{before cashflows}}}{A_{t-1}^{\text{before cashflows}}} - 1$
    Adjust asset allocation with Reinforcement Learning for each asset j in J
    Liability Management:
    Compute evolution of liability L_t based on the mortality table
    Cashflow Management:
    Compute outgoing total cashflows and expected bonus
    if then$A_t^{\text{before cashflows}} -$ cashflow without bonus < 0
        Stop and apply penalty
    end if
    Give bonus if applicable
    $A_t = A_t^{\text{before cashflows}} -$ cashflows
    $r_t^{RL} = \dfrac{A_t}{A_{t-1}} - 1$
    Update Asset and Liability Values:
    Calculate new asset value: $A_t = A_t + NCF_t$
    Calculate new liability value: $L_t = L_t + C_t$
end for

Chapter 5

Portfolio allocation

This chapter presents the results of the implementation of the Asset Liability Management (ALM) model in the Markov Decision Process setup, those results are discussed in details in Chapter 6 Our objective with this study is to model a part of the asset portfolio of an insurance company and its associated liabiliaties, particularly in a context of sudden rise of the yield curve. The insurer in our scenario has a liabilities portfolio with several life insurance products. Specifically, we suppose a life insurance product with a guaranteed interest rate. The initial premium are invested on the market by the insurer. Building upon the work of Gerstner et al. [2008], we assume that the insurer also offers a bonus mechanism. This bonus is guaranteed if the insurer's return exceeds a predefined threshold. Additionally, surrenders of insurance policies are considered in our model, with associated penalties factored in. The challenge is to effectively manage the invested premium in an asset portfolio to meet the demands of the liabilities. Reinforcement Learning (RL) algorithms assist in identifying an optimal strategy for portfolio allocation under constraints.

On the asset side, the insurer has the possibility to invest in different funds invested in bonds or equities or to keep cash. The insurer's task is to manage its portfolio to maximise its return while also ensuring sufficient liquidity to match its liabilities. This often requires selling off assets when needed. To model the behavior of the agents involved, we use the Python library FinRL [Liu et al., 2021]. FinRL proposes different implementation of major reinforcement learning algorithms within a financial context. As a framework, it is built atop Stable Baselines 3 [Raffin et al., 2021], an open-source initiative that provides RL algorithms and agents. Stable Baselines 3 has already been used in finance. As example, Leung et al. [2023] proposes a portolfio recommendation system to customer according to their preferences.

The FinRL library provides a variety of environments to choose from. Our starting point is the portfolio allocation environment [Costa and Costa, 2023] as our goal is an extension of this topic. The orignal method of the library takes assets prices as inputs and outputs the portfolio allocation. The reward of their agents is centered solely on tracking the portfolio's return over time. From this, we adapt the reward function of the Reinforcement Learning approach, which can be a function of either the portfolio's return or a specific risk measure such as the Sharpe ratio. Depending on the choice of the use, one can select a function

penalizing the volatility of the portfolio.

We use the existing FinRL framework as a starting point. Therefore, creating new agents from scratch is not required. We then adjust the system that gives rewards to the agents, to reflect the simulated liabilities we are dealing with. In this study, we utilize historical market data from various European stock exchanges to train our agent. The process begins by sourcing publicly available data, which ensures the transparency and reproducibility of our research. We gather publicly available datasets, encompassing a diverse range of assets class to create a realistic and comprehensive financial environment for the algorithm. This data is not altered or pre-processed. The training procedure involves iterating over the historical data. By doing so, the technique learns from actual past market conditions, gaining insights on price movements to determine the optimal portfolio allocation.

5.1 Illustration on a simple case

This section illustrates the returned asset allocation on a theoritical case. In this case, we consider a market with different assets, each with a different return and volatility as weel as keeping cash. The assets are either a continuously increasing asset or a continuously decreasing asset up to a date. Depending on the reward function, the agent allocates its portfolio. The evolution of the assets is in the figure 5.1. The agent wishes to allocate its initial amount into the assets to maximise its reward. In the example, there is no liabilities.

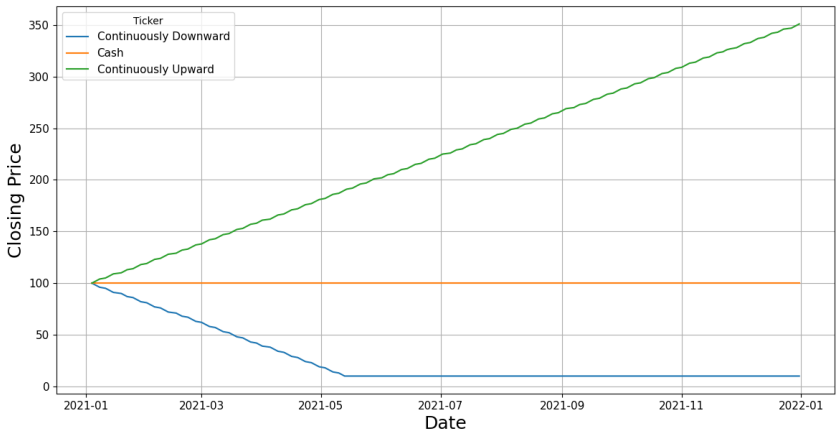

Figure 5.1: Evolution of the prices for the two assets as well as cash. The start date has been arbitrary chosen. The first asset is a continuously increasing asset and the third one is a continuously decreasing asset. Asset evolutions have been manuelly generated. The start date is informative. The initial value of both assets is 100.

Figure 5.2 shows the evolution of the portfolio value over time, divided by the initial value of the portfolio, with the return presented at each time step. The Reinforcement Learning methods focused on maximizing the return, either the differential return directly or the logarithm of the return, offer a better return overall, as expected. Both methods converge

towards the same policy: they always invest in the continuously upward asset. In contrast, MVO and equal-weighted select a mix between the assets. MVO selects a fixed combination between cash and the continuously upward asset.

Finally, the last two Reinforcement Learning methods, showed in figure 5.2, which focus on maximizing a differential risk function, are lacking in reward. The differential Sharpe ratio starts to converge to a mix of the upward asset. The Differential Sortino ratio performs badly due to the construction of the assets. The continuously upward asset is never decreasing, by construction. As such, the downside deviation at time t is always zero. To avoid errors in computation, if the downside deviation at time $t-1$ is zero, the differential Sortino ratio is set to zero. The agent is not trained to select the upward asset. In a real-world example, this does not occur since negative returns are possible. The policies of these methods are in appendix in figures A3.1 and A3.2.

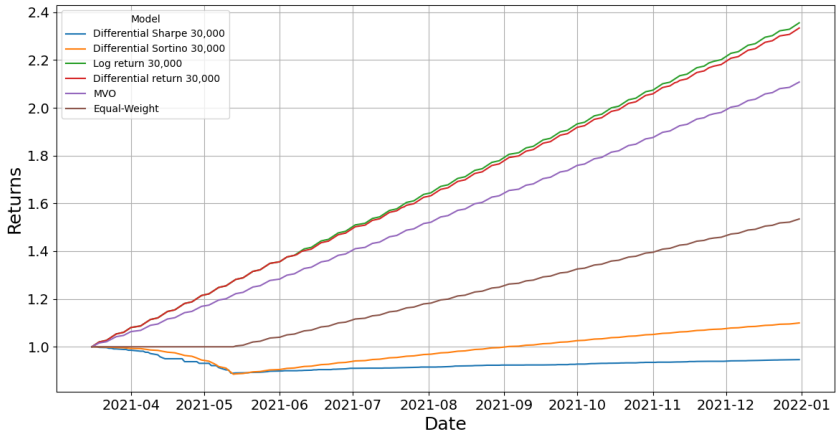

Figure 5.2: Evolution of returns for different strategies in the theoretical case. The start date is illustrative. The strategies utilize differential return, Log Return, and differential Sortino and Sharpe ratios as reward metrics. All agents were trained using the same hyperparameters over 30 000 time steps (see Appendix, Table A1.1 for details). Additionally, two other strategies are included: the Minimum Variance portfolio, where weights are optimised to maximize the Sharpe ratio during the training period, and an equal-weighted approach, assigning the same weight to each asset, including cash.

5.2 Dataset for the portfolio allocation

To see the impact of a rise of the yield curve on the portfolio of an insurance company, the methodology has been applied on a dataset of several Exchange Traded Funds (ETFs). The training period is from 2020 to 2022 whereas the test period is over 2023. As shown in the 5.3, the yield curve has been rising over the training period, espcially during 2022. Over the test perdio, the curves remains stable. The figures shows the EIOPA yield curve for several quarter over the training and test periods.

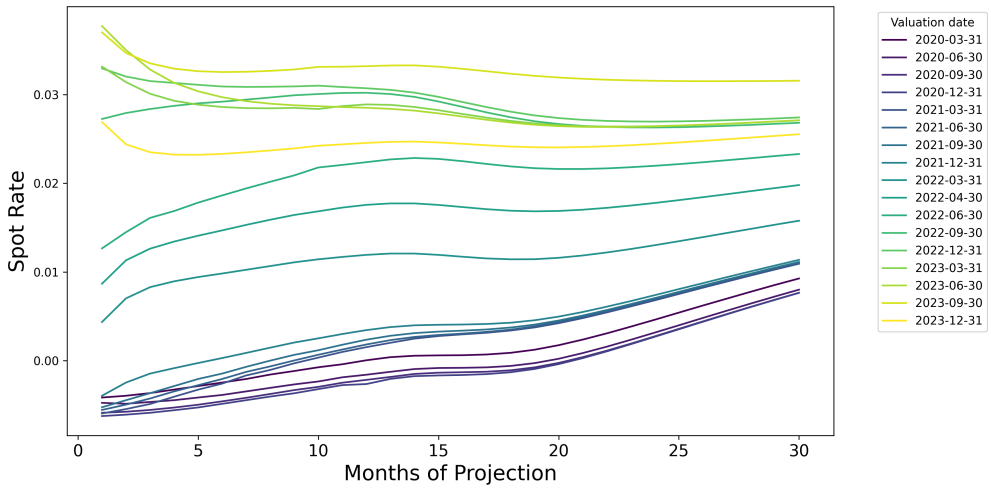

Figure 5.3: EIOPA yield curves without VA from 2019-01-02 to 2021-12-31. The older the date, the darker the color. The yield curve on the top of the plot means that spot rates are higher. The sudden rise occurs during 2022.

The evolution of the available assets to invest in on the training period is shown in the figure 5.4 whereas their evolution on the test period is shown in the figure 5.5. The price ratio is given in the figure 5.6 for the training set and in the figure 5.7 for the test set.

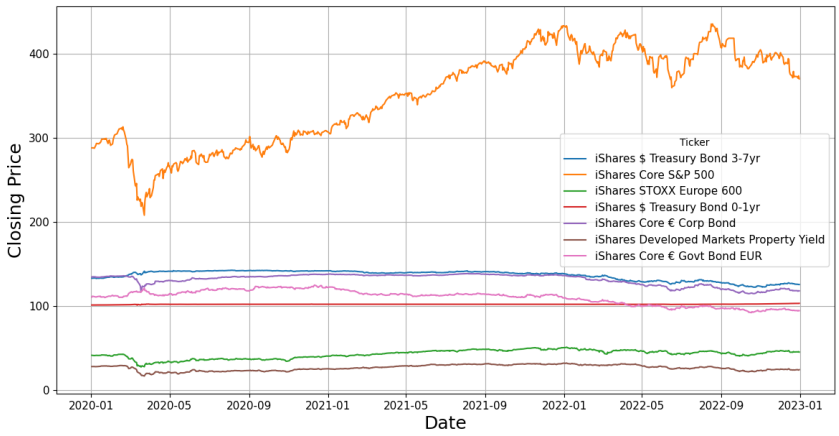

Figure 5.4: Evolution of the prices of the assets in the training set considering the outgoing cashflows of the liabilities.The start date is 2023-03-16 and the end date is 2022-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

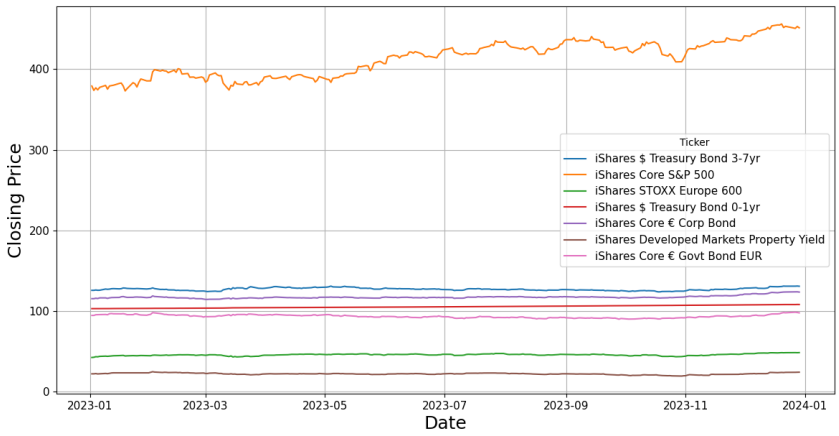

Figure 5.5: Evolution of the prices of the assets in the test set considering the outgoing cashflows of the liabilities.The start date is 2023-03-01 and the end date is 2023-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

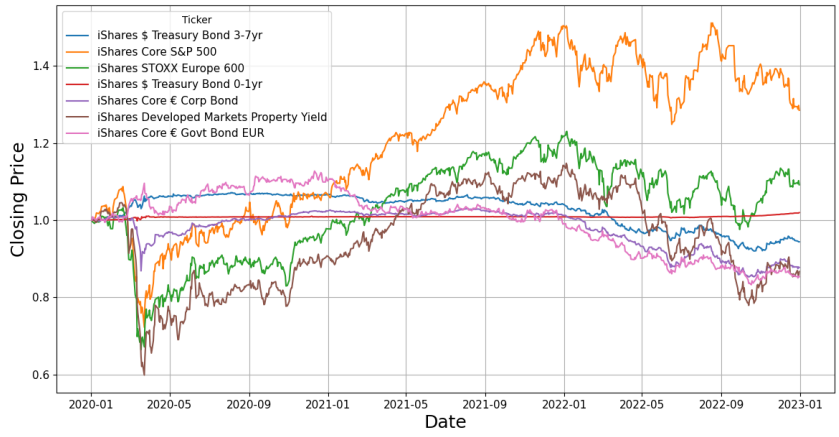

Figure 5.6: Evolution of the prices of the assets in the training set. The start date is 2023-03-16 and the end date is 2022-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

5.3 Portfolio allocation without liabilities

5.3.1 Selection for hyperparameters

This section focuses on the model sensitivity to hyperparameters. The methodology and the results is explained below. The hyperparameters selections holds for the portfolio optimisa-

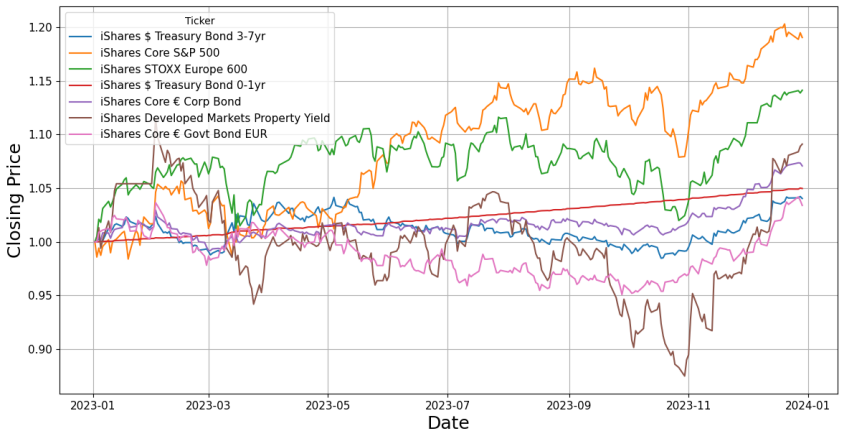

Figure 5.7: Evolution of the price ratio relative to the asset price on the initial date within the test set. The start date is 2023-03-01 and the end date is 2023-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

tion with and without liabilities constraints. The objective is to find a good balance between performance on the training and on the test while avoiding overfitting.

Different runs with several hyperparameters have been launched on the same dataset. The runs propose differ over three hyperparameters: maximum number of training steps, learning rate, and entropy coefficient. They play a crucial role in shaping the behavior and performance of the Proximal Policy Optimisation (PPO) algorithm.

The maximum number of training steps directly influences the convergence behavior of the algorithm. A higher value for this hyperparameter can allow the algorithm to explore the action space more comprehensively, potentially leading to improved performance. However, excessively large values may result in overfitting or training inefficiency, necessitating a balance to be struck to achieve optimal convergence within a reasonable timeframe.

The learning rate governs the magnitude of parameter updates during training, impacting the speed and stability of convergence. A well-tuned learning rate is essential for efficient learning without oscillations or divergence. A too high learning rate may cause the algorithm to overshoot optimal solutions, while a too low learning rate can slow down convergence or lead to suboptimal solutions. Fine-tuning this hyperparameter is important to strike a balance between rapid learning and stability.

The entropy coefficient in PPO serves as a regularization term that encourages exploration by promoting a diverse range of actions. By balancing exploitation and exploration, the entropy coefficient influences the risk-taking behavior of the algorithm. A higher entropy coefficient can lead to more exploration, potentially discovering superior strategies, while a lower coefficient may result in exploitation of known policies. Adjusting this hyperparameter allows for control over the level of risk-taking within the portfolio allocation framework, with

implications for both short-term performance and long-term strategy optimisation.

The figures 5.8 and 5.9 shows the result of runs with the hyperparameters using the Log Return reward. As results are comparable for the differential return and the Log Return rewards, figures A2.1 and A2.2 showing the result of runs with the hyperparameters using the differential return reward are in appendix. The values for other hyperparameters than number of steps, learning rate and entropy coefficients are shown in the table A1.1.

Comparing the different results, the number of steps has no impact in the test selection. Indeed, the run the training step is 1,000,000 in blue in figure 5.8 does not generate a better return than the others methods. As such, in the next sections, the number of step is fixed to 400,000.

Using the red and green curves on figure 5.8 where the hyperparmeters are the same except for the learning rate (1e-3 vs 5e-4), it shows that the decreasing the learning rate improves the return. The same observation is made on figure A2.1 with the orange and green curves (respectively 5e-4 vs 1e-3 learning rate). In the next section, a learning samller than 5e-4 is considered.

Finally, the entropy coefficient has a relative impact on the return up to a certain treshold. Indeed, in the figure 5.8, up to an entropy coefficient of 2e-2, the difference is not significant as the returns are not correlated with the entropy coefficient. The purple curve with an entropy coefficient of 2e-2 obtains a returns twice bigger than the other runs. However on the test set, see figure 5.9, this return performs badly. In the next section, the entropy coefficient is fixed to 5e-3.

5.3.2 Comparison with classic methods

This section compares the results of the Reinforcement Learning (RL) algorithms with classic methods. The classic methods are the Minimum-Variance Optimisation, where weights are optimised to maximise the Sharpe ratio during the training period, and an equal-weighted approach, assigning the same weight to each asset, including cash. The results are shown in the figure 5.10 for the training period and in the figure 5.11 for the test period. The actions taken by the agents for the Log Return reward are shown in the figure 5.13 for the training period and in the figure 5.12 for the test period. On the training set, the RL methods based on the maximization of return between trading days outperform both classic methods. However, on the test set, the results inverse : MVO and equal-weight portfolio returns a better result than Reinforcement Learnings approach.

The table 5.1 explains that the MVO fully invests in the index following the S&P 500. On the contrary as shown in figures 5.13 and 5.12, the Reinforcement Learning approach invest mainly into the fund "iShares Developed Markets Property Yield", which invest into real estate compagnies. On the figure 5.6, the asset price for "iShares Developed Markets Property Yield" is increasing between March 2020 and the beginning of March 2022. Since the Reinfrocement Learning method evaluates the allocation based on the last 50 training days, the training days before early March 2020 are not used to calibrate the algorithm. Hence, this asset is in increase over the majority of the training period. After March 2022, the price

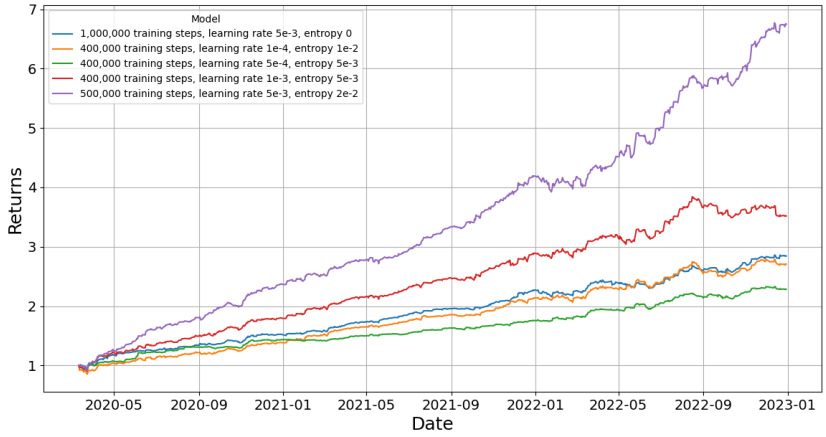

Figure 5.8: Evolution of the returns for sets of parameters using the log return as reward over the training set. The hyparparamters are the learning rate and the entropy coefficient of the PPO loss function as well as the number of timesteps.The start date is 2020-03-11 and the end date is 2022-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

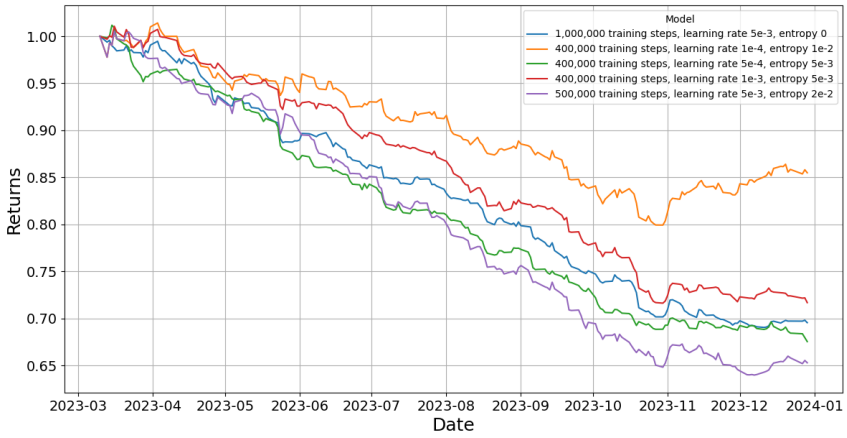

Figure 5.9: Evolution of the returns for sets of parameters using the Log return as reward over the training set. The hyparparamters are the learning rate and the entropy coefficient of the PPO loss function as well as the number of timesteps.The start date is 2023-03-16 and the end date is 2023-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

of the product is decreasing. Since the Reinforcement Learning approach is adaptative it adajust its allocation in response to changing market conditions over time.

However, the market is quite different on the test set. The strategy of the Reinforcement Learning approach is not adapted to the new market conditions. The MVO and equal-weighted approach are still returning a profit. The "iShares Core S&P 500" product is still increasing over the period. The market in overall is also slightly increasing.

Asset Name	**MVO Weights**	**Equal-Weight**
iShares $ Treasury Bond 3-7yr	-0.000	0.125
iShares Core S&P 500 USD	1.000	0.125
iShares STOXX Europe 600 (DE) EUR	-0.000	0.125
iShares $ Treasury Bond 0-1yr	-0.000	0.125
iShares Core € Corp Bond EUR	0.000	0.125
iShares Developed Markets Property Yield USD	0.001	0.125
iShares Core € Govt Bond EUR	0.000	0.125
Cash	-0.000	0.125

Table 5.1: Asset Weights for the Mean-Variance Optimisation and Equal-Weight method. The MVO has been calibrated on the dataset from 2020-01-01 until 2023-01-01 by selecting the portfolio with the highest Sharpe ratio over the whole period. Weights are kept constant over the period. The same weight without recalibration are reused for the test set.

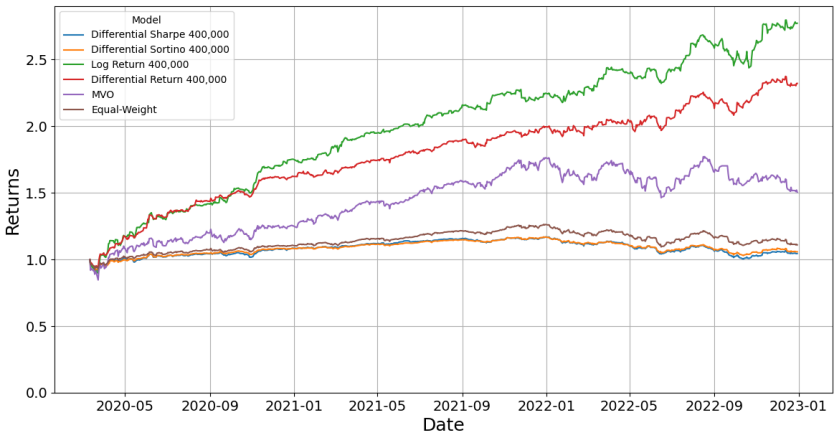

Figure 5.10: Evolution of the returns for different methods over the training set. The start date is 2020-03-11 and the end date is 2022-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

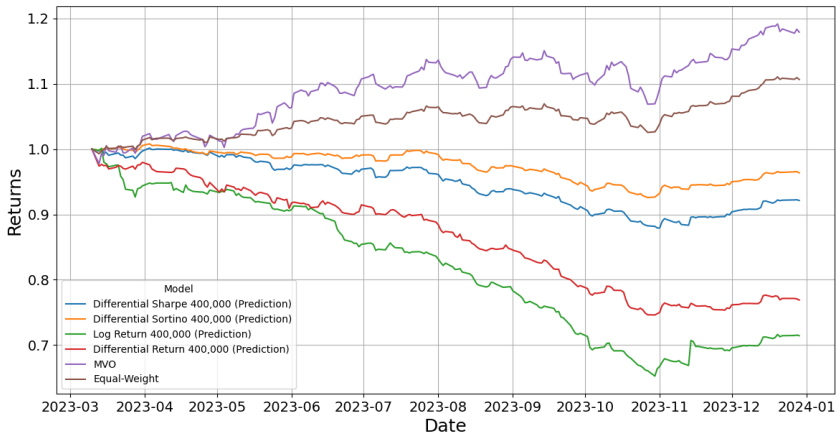

Figure 5.11: Evolution of the returns for different methods over the test set. The start date is 2023-03-11 and the end date is 2023-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

5.4 Portfolio allocation with liabilities

This section details the evolution of the portfolio allocation with liabilities. The results are shown in the figures 5.14 for the training period and in the figure 5.15 for the test period. The actions taken by the agents for the Log Return reward are shown in the figure 5.16 for the training period and in the figure 5.17 for the test period.

As expected, the asset repartition is quite different while taking liabilities into account. The reinforcement learning with Log Return reward favours "iShares STOXX Europe 600", which is sligthly less volatile. The impact of the liabilities is clearly visible on the return on the portfolio. No method reaches an accumaltive return greater than 1.5 at the end of the training period. Similarly to the case without liabilities, the returns observed on the test set is inversly correlated with the returns on the training set. The asset repartition of the MVO method has not been updated for this scenario.

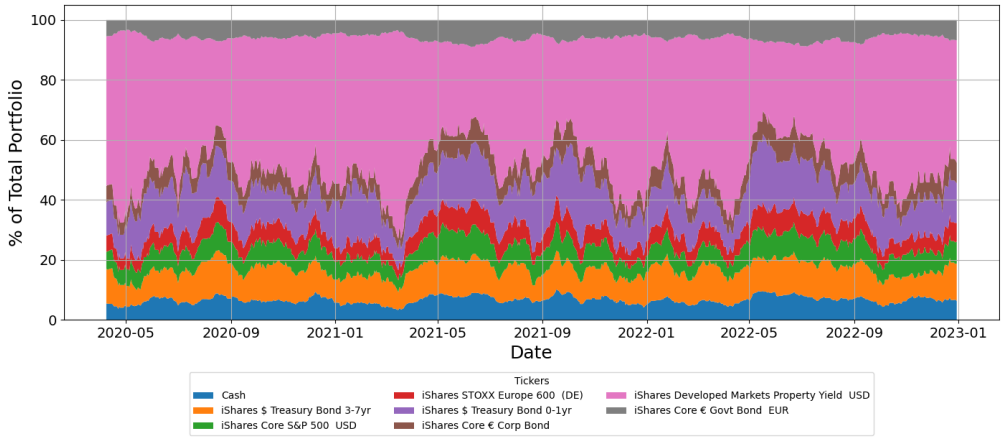

Figure 5.12: Evolution of the portfolio repartition over time with the method Log Return 400,000 for the training set.The asset repartitiion is smoothed over 20 trading days to ease visualisation.The start date is 2020-03-11 and the end date is 2022-12-31.

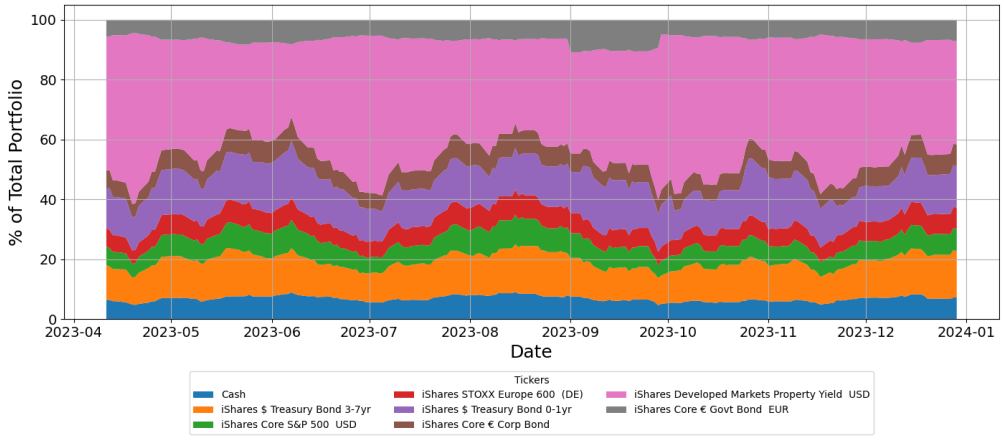

Figure 5.13: Evolution of the portfolio repartition over time with the method Log Return 400,000 (Prediction) for the training set.The asset repartitiion is smoothed over 20 trading days to ease visualisation.The start date is 2023-03-16 and the end date is 2023-12-31.

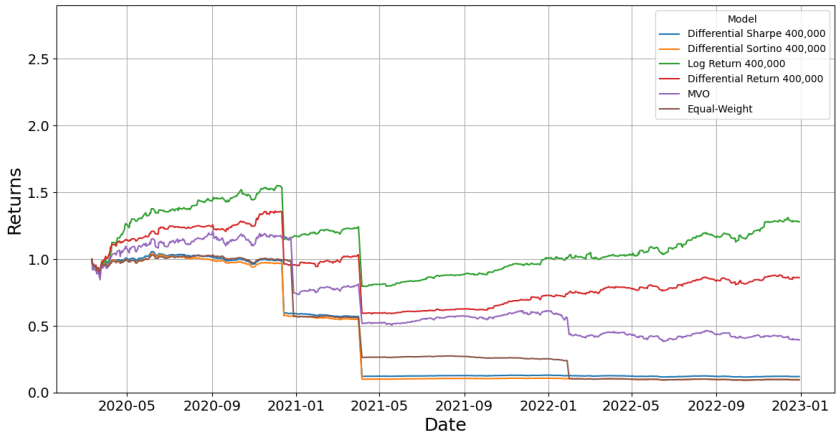

Figure 5.14: Evolution of the returns for different methods over the training set considering the outgoing cashflows of the liabilities.. The start date is 2020-03-11 and the end date is 2022-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

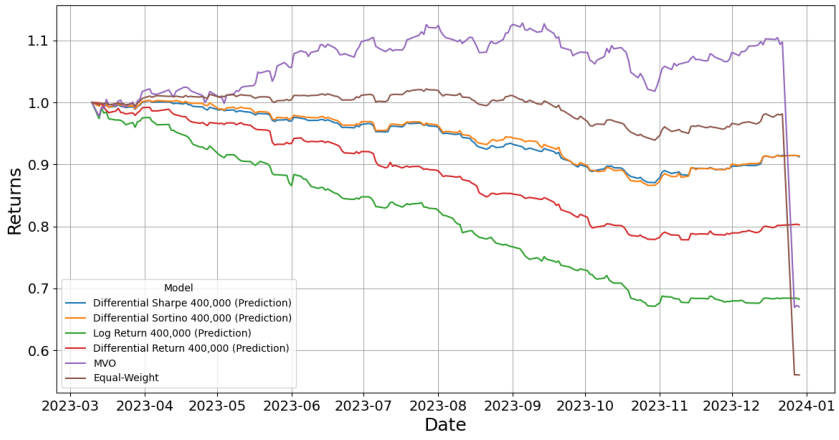

Figure 5.15: Evolution of the returns for different methods over the test set considering the outgoing cashflows of the liabilities. The start date is 2023-03-11 and the end date is 2023-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

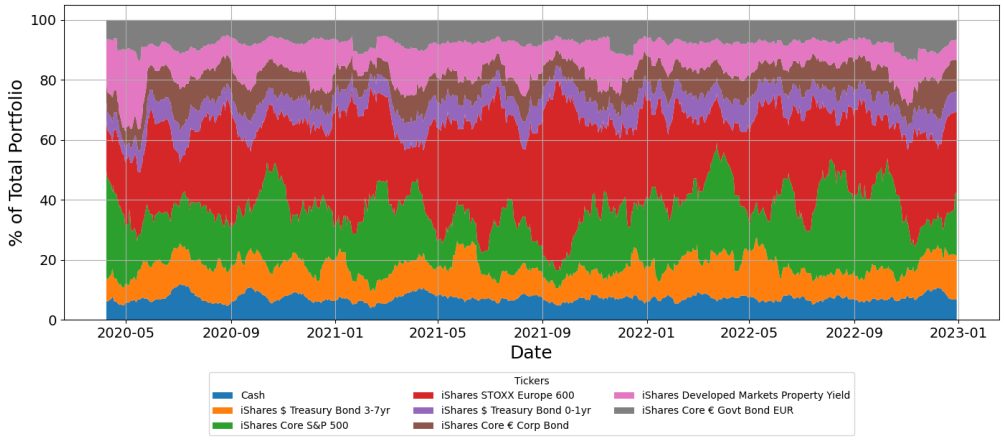

Figure 5.16: Evolution of the portfolio repartition over time with the method Log Return 400,000 for the training set.The asset repartitiion is smoothed over 20 trading days to ease visualisation.The start date is 2020-03-11 and the end date is 2022-12-31.

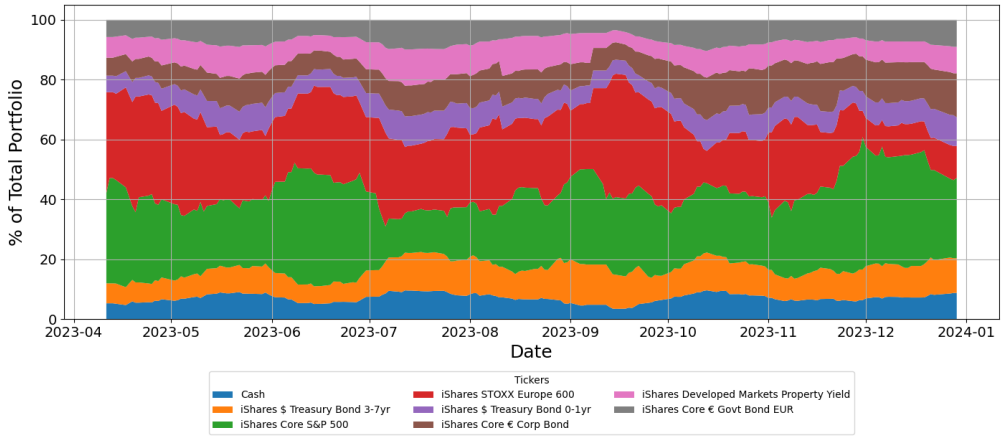

Figure 5.17: Evolution of the portfolio repartition over time with the method Log Return 400,000 (Prediction) for the training set.The asset repartitiion is smoothed over 20 trading days to ease visualisation.The start date is 2023-03-16 and the end date is 2023-12-31.

Chapter 6

Discussion

While the Reinforcement Learning (RL) methods show promises, the current computational limitations and model constraints inhibited their full potential. Further exploration and resources are necessary to optimise these methods and achieve superiority over traditional MVO. Integrating stock correlation metrics and experimenting with diverse datasets are promising steps toward achieving this goal.

Figures 5.10 and 5.14 demonstrate that while RL methods do converge, they exhibit signs of overfitting. This overfitting is manifested in the evolving nature of actions over time, indicating the model's sensitivity to fluctuations in the training environment. This sensitivity is further confirmed by the evolving actions as market conditions shift, as depicted in Figure 5.13. This implies that while the RL model can adapt to the specific patterns in the training data, in this case, it struggles to generalize to new or unseen market conditions. The evolving nature of the actions suggests that the model may be overly tuned to the specific scenarios it has encountered during training, which could reduce its effectiveness in real-world applications where market dynamics are constantly changing. This potential overfitting underscores the need for careful validation and potentially more robust regularization techniques to ensure the model's performance remains stable and reliable across a broader range of market conditions. The differential Sortino ratio, orange curves in Figures 5.10 and 5.14, has rarely been used for portfolio optimisation [Hambly et al., 2021]. It offers a more subtle risk measure than the Sharpe ratio since it penalizes downside returns more than positive returns. As such, it may be seen as a utility function [Moody and Saffell, 2001]. Finally, the modeling of liabilities from a model with real mortality directly links expected outgoing cash flows to the asset allocation. Failing to serve the liabilities leads to ruin that stops the iteration. This impacts the agent's policy as it can no longer earned reward in the following steps, similarly to the example in Section 2.1.3.

Abrate et al. [2021] also studied portfolio optimisation for an insurance company. In contrast to this work, they modeled different asset classes such as equities and bonds. Their objective was to maximise risk-adjusted returns. They also modeled their liabilities using mortality and surrender probabilities. They applied their algorithm to a smaller asset portfolio than that used in this master's thesis. The value of the liabilities did not directly affect their reward. Wekwete et al. [2023] focused on ALM; however, their methodology differed

from the present work. They concentrated on cash flow matching between assets and liabilities and modeled the liabilities from an aggregated perspective. In this master's thesis, the liabilities were constructed based on real mortality tables, with the characteristics of the insurance product directly integrated into the model. Regarding the assets, they focused on bonds, whereas this research used funds invested in diverse asset classes. The latter methodology was more versatile. Indeed, market values of bonds could be deduced from their characteristics, allowing an asset based on those to be created and integrated into a portfolio.

For insurance companies, the capacity to manage assets and liabilities is important. Good ALM management leads to a reduction in risk and diminishes the required capital of the insurer. The RL approach allows for a more dynamic and responsive allocation strategy, adapting to market changes in real-time. This could potentially lead to more frequent or more market-adjusted asset allocations, resulting in better risk management and higher returns.

Compared to existing literature, our experiments revealed certain limitations primarily due to insufficient computing power. Some parts of the literature lack details about the hyperparameters used to solve their RL problem. For others, such as Sood et al. [2023] or Park et al. [2024], they use a number of time steps multiple times greater than in this research, with bigger neural networks. Due to limited hardware and long computation times, this work reduces the number of time steps, curtails exploration, and utilizes smaller neural network architectures. These limitations can ultimately affect the model's generalizability and robustness, making it less effective in real-world applications.

Despite the positive outcomes mentioned earlier, several limitations must be acknowledged. The RL model's performance is contingent on the quality and quantity of training data. In real-world scenarios, data may be incomplete or noisy, which could impact the model's effectiveness. Additionally, the model's complexity and computational intensity might pose practical challenges for implementation within existing financial systems.

While our model incorporates liability constraints, it does so in a relatively simplified manner. Future research could explore more complex and realistic liability structures, including stochastic liabilities that reflect real-world uncertainties more accurately. Furthermore, liabilities act as a ruin probability and have a limited impact on the reward. A more refined model could incorporate liabilities more explicitly into the environment, potentially leading to more robust and accurate results. To expedite the process, we employed GPUs, yet the computational constraints remained a significant bottleneck. This limitation became evident as our RL models were unable to surpass the performance of the MVO approach on the test set. Specifically, the Sharpe and Sortino ratios remained stagnant, suggesting a need for further iterations and optimisation. Interestingly, the performance of our RL models was still quite akin to that of an equal-weighted portfolio, pointing to potential areas for methodological refinement. Currently, the model cannot handle irregular dates, i.e., having dates with multiple off days between consecutive market days. Consequently, this leads to a huge drop in the portfolio value, as shown in Figure 5.14. However, this could also be seen as an approximation to a global reach to the maturity of a part of the portfolio. If the initial entry date is the same for all policyholders entering in the same year, such drops in the portfolio value are expected.

Several avenues for improvement are identified. One significant enhancement could be the integration of stock correlation metrics, as demonstrated by Jang and Seong [2023]. By incorporating relationships between assets, we could potentially capture more informative features that benefit the RL training process. This approach might mitigate overfitting and lead to more robust and generalizable policies. Another potential improvement is the use of different datasets. Expanding the dataset can provide a broader and more varied training environment, enhancing the model's adaptability and performance. Utilizing datasets from various sectors or including international markets might offer additional insights and opportunities for model refinement. To find the best strategy, the model iterates over historic asset prices and adjusts the portfolio weights accordingly. To improve the generality of the model, noise could be added to the dataset as done by Théate and Ernst [2021] or by adding a small noise before updating the policy, as in Plappert et al. [2018]. Remember that the policy is generally updated at each time step. At each update, a small noise on the new policy may be applied. It would increase exploration by the model. Introducing constraints on the asset allocation, as insurers generally aim to contain their asset allocation within certain bounds, could further enhance the model.

While the limitations of this study are notable, the exploration of RL methods in constrained portfolio allocation still offers meaningful insights. The challenges encountered, such as computational constraints and data quality, point to important areas that warrant further investigation. These constraints do not diminish the study's contributions; instead, they highlight the ongoing need for refinement and optimisation in applying RL to portfolio management. Though the potential of RL in this field is promising, it remains to be fully realized. The contributions of this research, though modest, provide a useful basis for future studies, which may build upon these initial insights to develop more robust and adaptable RL models in financial applications.

Chapter 7

Conclusion

This master thesis explores an application of reinforcement learning algorithms to constrained portfolio allocation, particularly within the context of insurance companies. The research aimed to investigate the feasibility, effectiveness, and potential advantages of using RL for optimising portfolio allocations while considering the liabilities of an insurer. Throughout the study, several key insights were gained, and while the results demonstrate promise, they also highlight significant challenges and limitations.

This work presents the basic of reinforcement learning. It introduces is concepts as well as it formalism, notably by detailing the concept of Markov Decision Process. The study also gives an example of fully descrivable MDP in the context of finance. It then presents how to model assets and liabilities following the concepts presented at the beginning of this master thesis. Notably, it shows then compares different rewards for portfolio optimisation problem.

The theoric model is then apply on a example with asset prices for historical market data. The RL model is applied with and without considering the liabilities of the insurer. No mathematical hypothesis was posed on the evolution of the data, the machine learning models were model-free. The models are compared with traditional portfolio optimisation methods, such as Mean-Variance Optimisation. The results show that the methods can outperform classical methods. However, their generalisation to new data is limited, it relies heavily on the training data. The results are nuanced and explained. Finally, the thesis is compared with other state of the art RL methods for similar tasks. The limitations and possible future research topics are discussed.

The research uncovered some obstructions for RL to be effectively implemented in real-world actuarial contexts. The challenges include the computational intensity of RL algorithms, the risk of overfitting, and the sensitivity of models to data scarcity and quality. These topics are reinforced by the constraints imposed by the need for transparent, interpretable, and regulatory-compliant models in the actuarial domain.

These limitations underscore the need for future research to focus on optimising RL algorithms for practical use in constrained portfolio allocation. Future research could investigate dataset model taking account of relations between the assets or increase the liabilities complexity by using an open portfolio with new policyholders during the simulations.

With room for improvement, this thesis seeks to contribute to the growing body of literature on applying reinforcement learning to financial and actuarial challenges. While the findings provide an initial understanding of how RL might be used for constrained portfolio allocation, the full potential of RL in this area remains to be fully explored, and the results of this study should be seen as an early step that others can build upon to develop more effective and adaptable models.

Appendices

Appendix A1

Model hyperparameters

The hyperparameters used in the illustrative example are shown in Table A1.1.

Hyperparameter	**Value**
batch_size	64
ent_coef	0.005
gae_lambda	0.95
gamma	0.99
learning_rate	0.0003
n_epochs	10
n_steps	64
num_timesteps	30000
vf_coef	0.5

Table A1.1: Hyperparameters and their corresponding values used for the policy training, where: *batch_size* is the size of the minibatches used during training; *ent_coef* is the entropy coefficient for the loss calculation; *gae_lambda* is the trade-off factor between bias and variance for the Generalized Advantage Estimator; *gamma* is the discount factor; *learning_rate* is the rate at which the model's parameters are updated; *n_epochs* is the number of passes during training; *n_steps* is the number of steps to run for each environment per update; *num_timesteps* is the total number of timesteps for training and *vf_coef* is the coefficient for the value function loss.

Appendix A2

Hyperparameters tuning

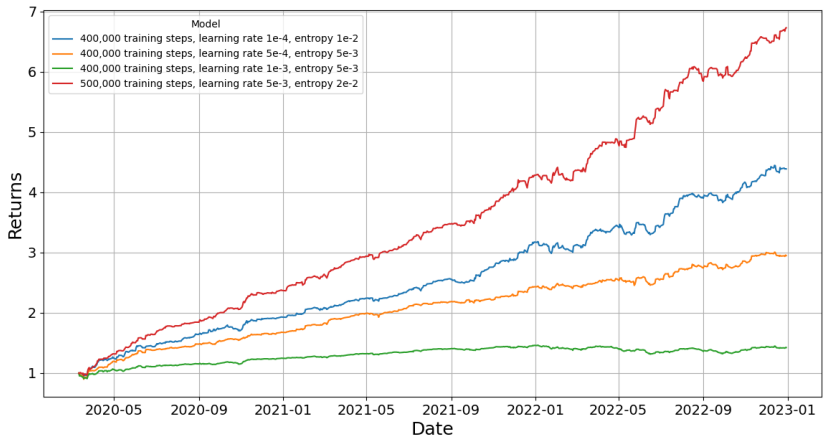

Figure A2.1: Evolution of the returns for sets of parameters using the differential return as reward over the training set. The hyparparamters are the learning rate and the entropy coefficient of the PPO loss function as well as the number of timesteps.The start date is 2020-03-11 and the end date is 2022-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

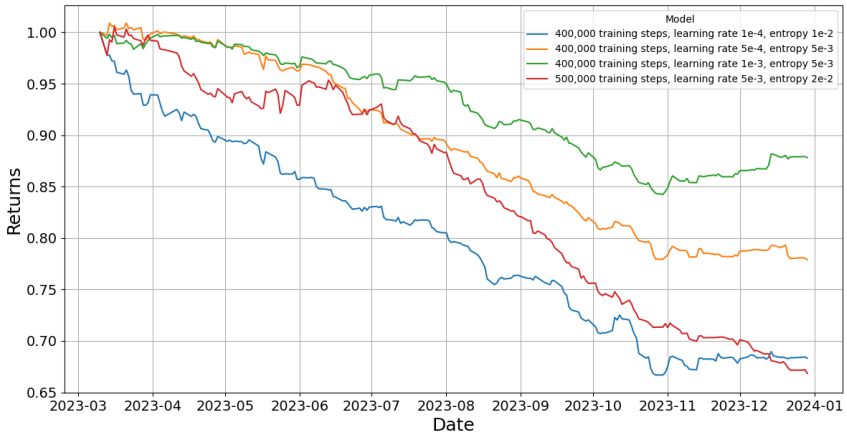

Figure A2.2: Evolution of the returns for sets of parameters using the differential return as reward over the training set. The hyparparamters are the learning rate and the entropy coefficient of the PPO loss function as well as the number of timesteps.The start date is 2023-03-16 and the end date is 2023-12-31. The assets covers wide Exchange Traded Funds (ETFs) from different sectors and regions.

Appendix A3

Actions

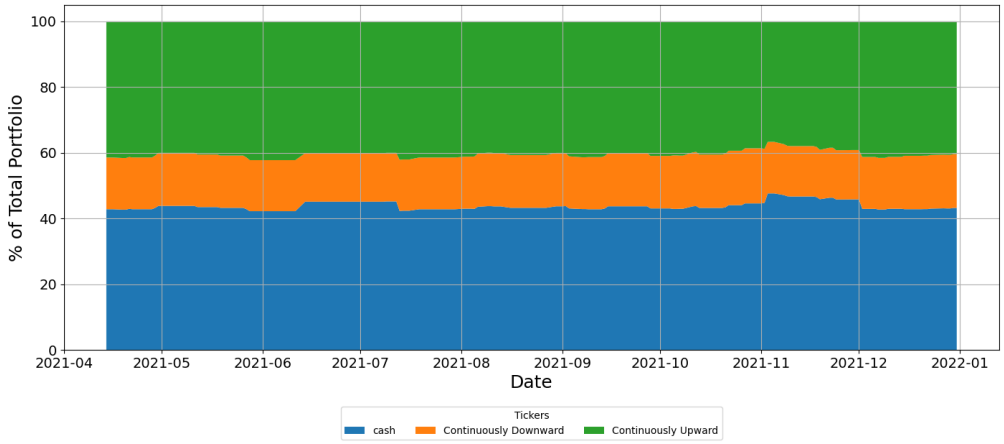

Figure A3.1: Evolution of the repartition with the method Differential Sharpe on an illustrative case. Dates are arbitrary.The asset repartitiion is smoothed over 20 trading days to ease visualisation.

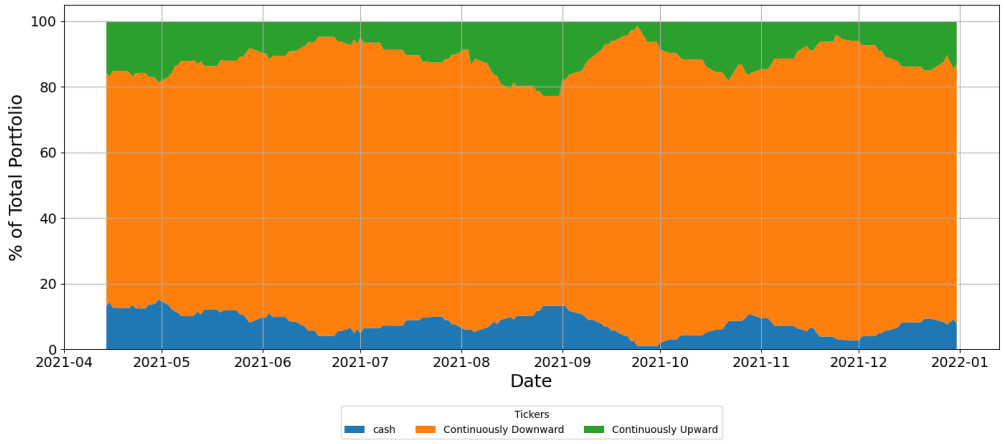

Figure A3.2: Evolution of the repartition with the method Differential Sortino on an illustrative case. Dates are arbitrary.The asset repartitiion is smoothed over 20 trading days to ease visualisation.

Bibliography

Abrate, C., Angius, A., De Francisci Morales, G., Cozzini, S., Iadanza, F., Li Puma, L., Pavanelli, S., Perotti, A., Pignataro, S., and Ronchiadin, S. (2021). Continuous-action reinforcement learning for portfolio allocation of a life insurance company. In Dong, Y., Kourtellis, N., Hammer, B., and Lozano, J. A., editors, *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track. ECML PKDD 2021*, volume 12978 of *Lecture Notes in Computer Science*, pages 237–252, Cham. Springer.

Acero, F., Zehtabi, P., Marchesotti, N., Cashmore, M., Magazzeni, D., and Veloso, M. (2024). Deep reinforcement learning and mean-variance strategies for responsible portfolio optimization. *arXiv e-prints*, page arXiv:2403.16667.

Aslam, F., Hunjra, A. I., Ftiti, Z., Louhichi, W., and Shams, T. (2022). Insurance fraud detection: Evidence from artificial intelligence and machine learning. *Research in International Business and Finance*, 62:101744.

Biglova, A., Ortobelli, S., Rachev, S., and Fabozzi, F. J. (2009). Modeling, estimation, and optimization of equity portfolios with heavy-tailed distributions. In Satchell, S., editor, *Optimizing Optimization: The Next Generation of Optimization Applications and Theory*, chapter 5, pages 117–141. Academic Press.

Blier-Wong, C., Cossette, H., Lamontagne, L., and Marceau, E. (2020). Machine learning in p&c insurance: A review for pricing and reserving. *Risks*, 9(1):4.

Cornuéjols, G., Peña, J., and Tütüncü, R. (2018). *Optimization Methods in Finance*. Cambridge University Press, second edition.

Costa, C. d. S. B. and Costa, A. H. R. (2023). POE: A General Portfolio Optimization Environment for FinRL. In *Anais do Brazilian Workshop on Artificial Intelligence in Finance (BWAIF)*, pages 132–143. SBC.

Denuit, M., Maréchal, X., Pitrebois, S., and Walhin, J.-F. (2007). *Actuarial modelling of claim counts: Risk classification, credibility and bonus-malus systems*. John Wiley & Sons.

Dixon, M. F., Halperin, I., and Bilokon, P. (2019). *Machine Learning in Finance*.

Dixon, M. F., Halperin, I., and Bilokon, P. (2020). Applications of Reinforcement Learning. pages 347–418. Springer International Publishing, Cham.

Fontoura, A., Haddad, D., and Bezerra, E. (2019). A Deep Reinforcement Learning Approach to Asset-Liability Management.

Gerstner, T., Griebel, M., Holtz, M., Goschnick, R., and Haep, M. (2008). A general asset–liability management model for the efficient simulation of portfolios of life insurance policies.

Grize, Y.-L., Fischer, W., and Lützelschwab, C. (2020). Machine learning applications in nonlife insurance. *Applied Stochastic Models in Business and Industry*, 36(4):523–537.

Hambly, B. M., Xu, R., and Yang, H. (2021). Recent Advances in Reinforcement Learning in Finance. *SSRN Electronic Journal*, (3971071).

Hermans, L., Kostka, T., and Vassallo, D. (2023). Asset Allocation and Risk Taking Under Different Interest Rate Regimes. Technical Report 4405492, Rochester, NY.

Jang, J. and Seong, N. (2023). Deep reinforcement learning for stock portfolio optimization by connecting with modern portfolio theory. *Expert Systems with Applications*, 218:119556.

Krasheninnikova, E., García, J., Maestre, R., and Fernández, F. (2019). Reinforcement learning for pricing strategy optimization in the insurance industry. *Engineering Applications of Artificial Intelligence*, 80:8–19.

Kraus, M. and Feuerriegel, S. (2017). Decision support from financial disclosures with deep neural networks and transfer learning. *Decision Support Systems*, 104:38–48.

Kubat, M., Bratko, I., and Michalski, R. S. (1998). A review of machine learning methods. *Machine Learning and Data Mining: Methods and Applications*, pages 3–69.

Leung, M. F., Jawaid, A., Ip, S.-W., Kwok, C.-H., and Yan, S. (2023). A portfolio recommendation system based on machine learning and big data analytics. *Data Science in Finance and Economics*, 3(2):152–165.

Liu, X.-Y., Yang, H., Gao, J., and Wang, C. (2021). FinRL: Deep reinforcement learning framework to automate trading in quantitative finance. *SSRN Electronic Journal*.

Lorentzen, C. and Mayer, M. (2020). Peeking into the black box: An actuarial case study for interpretable machine learning. *Available at SSRN 3595944*.

Markowitz, H. (1952). Portfolio Selection. *The Journal of Finance*, 7(1):77–91.

Moody, J. and Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12(4):875–889.

Park, J.-H., Kim, J.-H., and Huh, J.-H. (2024). Deep Reinforcement Learning Robots for Algorithmic Trading: Considering Stock Market Conditions and U.S. Interest Rates. *IEEE Access*, 12:20705–20725.

Plaat, A. (2022). *Deep Reinforcement Learning*. Springer.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2018). Parameter Space Noise for Exploration. Technical report, arXiv. arXiv:1706.01905 [cs, stat] type: article.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Roy, R. and George, K. T. (2017). Detecting insurance claims fraud using machine learning techniques. In *2017 International Conference on Circuit ,Power and Computing Technologies (ICCPCT)*, pages 1–6.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms.

Shao, K., Tang, Z., Zhu, Y., Li, N., and Zhao, D. (2019). A Survey of Deep Reinforcement Learning in Video Games.

Sood, S., Papasotiriou, K., Vaiciulis, M., and Balch, T. (2023). Deep Reinforcement Learning for Optimal Portfolio Allocation: A Comparative Study with Mean-Variance Optimization. *FinPlan*, 2023(2023):21.

Sortino, F. A. and Price, L. N. (1994). Performance Measurement in a Downside Risk Framework. *The Journal of Investing*, 3(3):59–64.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction, 2nd ed.* Reinforcement learning: An introduction, 2nd ed. The MIT Press, Cambridge, MA, US.

Théate, T. and Ernst, D. (2021). An application of deep reinforcement learning to algorithmic trading. *Expert Systems with Applications*, 173:114632.

Wekwete, T., Kufakunesu, R., and van Zyl, G. (2023). Application of Deep Reinforcement Learning in Asset Liability Management. (4417872).

Yang, H., Liu, X.-Y., Zhong, S., and Walid, A. (2020). Deep reinforcement learning for automated stock trading: An ensemble strategy.

Zappa, D., Clemente, G. P., Della Corte, F., and Savelli, N. (2023). Editorial on the special issue on insurance: complexity, risks and its connection with social sciences. *Quality & Quantity*, pages 1–6.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Faculté des sciences
Place des Sciences, 2 bte L6.06.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/sc