CEFET/RJ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO

A DEEP REINFORCEMENT LEARNING APPROACH TO ASSET-LIABILITY MANAGEMENT

Alan Rodrigues Fontoura

Dissertation submitted to the Graduate Program of the Federal Center for Technological Education of Rio de Janeiro, CEFET/RJ, as partial fulfillment of the requirements for the degree of master.

Advisor: Eduardo Bezerra, D.Sc.
Co-advisor: Diego Haddad, D.Sc.

Rio de Janeiro,
July 2020

A DEEP REINFORCEMENT LEARNING APPROACH

TO ASSET-LIABILITY MANAGEMENT

Dissertation submitted to the Graduate Program of the Federal Center for Technological Education of Rio de Janeiro, CEFET/RJ, as partial fulfillment of the requirements for the degree of master.

Alan Rodrigues Fontoura

Examining jury:

President, Eduardo Bezerra, D.Sc. (CEFET/RJ) (Advisor)

Diego Barreto Haddad, D.Sc. (CEFET/RJ) (Co-Advisor)

Laura Silva de Assis, D.Sc. (CEFET/RJ)

Aline Marins Paes Carvalho, D.Sc. (IC/UFF)

Rio de Janeiro,

July 2020

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

F684 Fontoura, Alan Rodrigues
A deep reinforcement learning approach to asset-liability management / Alan Rodrigues Fontoura — 2020.
78f : il. (algumas color.) , enc.

Dissertação (Mestrado) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca , 2020.
Bibliografia : f. 75-78
Orientador: Eduardo Bezerra
Coorientador: Diego Haddad

1. Aprendizado de máquina. 2. Algorítimos. 3. Redes neurais. 4. Administração financeira. 5. Investimentos. I. Bezerra, Eduardo (Orient.). II. Haddad, Diego. (Coorient.) Título.

CDD 006.3

Elaborada pela bibliotecária Tania Mello – CRB/7 nº 5507/04

DEDICATION

To my mother Tania, my past

To my wife Marcelli, my present

To my daugther Manuela, my future

The loves of my life

ACKNOWLEDGMENTS

I would like to thank all the professors I had at CEFET/RJ. I would like to specially thank professors Jorge Soares and Kele Belloze, for the great reception I had when I first joined this institution, and professors Eduardo Bezerra and Diego Haddad, for being excellent advisors.

I want to thank my superiors and co-workers at both Petros and Petrobras for all the understanding and help they have given me.

ABSTRACT

A Deep Reinforcement Learning Approach to Asset-Liability Management

Asset-Liability Management (ALM) is a technique to optimize investment portfolios, considering a future flow of liabilities. Its stochastic nature and multi-period decision structure favors its modeling as a Markov Decision Process (MDP). Reinforcement Learning is a state-of-the-art group of algorithms for MDP solving, and with its recent performance boost provided by deep neural networks, problems with long time horizons can be handled in just a few hours. In this work, an ALM problem is addressed with an algorithm known as Deep Deterministic Policy Gradient. Opposed to most of the other literature approaches, this model does not use scenario discretization, which is a significant contribution to ALM study. Experimental results show that the Reinforcement Learning framework is well fitted to solve this kind of problem, and has the additional benefit of using continuous state spaces.

Keywords: ALM; Deep Reinforcement Learning; Deep Deterministic Policy Gradient.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

ALM	Asset-Liability Management
DDPG	Deep Deterministic Policy Gradient
DPG	Deterministic Policy Gradient
DR	Discount Rate
DRL	Deep Reinforcement Learning
FV	Future Value
MDP	Markov Decision Process
MSP	Multistage Stochastic Programming
MVO	Mean-Variance Optimization
PV	Present Value
RL	Reinforcement Learning

CONTENTS

1- Introduction

1.1- Contextualization

One of the most studied issues in finance is asset management. Usually seen as a portfolio optimization problem, there are two conventional approaches to it: trying either (1) to minimize the portfolio's chosen risk measure given an expected return, or (2) to maximize expected return, while not surpassing a certain risk level [Markowitz, 1952].

Asset-Liability Management (ALM), on the other way, is a more complicated matter. In such a context, the investor's goal is to fulfill a series of obligations, which may or may not be stochastic. These obligations (or liabilities) usually are correlated to one or more of the assets available to the decision-maker. In this scenario, one cannot just aim to optimize the risk-return relation: investments have to match (or preferably, outperform) liabilities, respecting their due dates.

Most obvious uses for ALM strategies lie in the financial sector. In banking, ALM arises as a tool for interest rate and liquidity management (short and long term) [Dermine, 2008]. Its use is crucial for insurance companies to handle uncertainty related to the total premium received and debts due to indemnity and pension funds, which need to balance their investments to guarantee participants' present and future retirements. Nonetheless, any individual who saves money for a future debt (an individual who wants to pay for his children's university) may benefit from these techniques [Rosen and Zenios, 2008].

The structure of an ALM problem is simple: at each time step, there is an *asset*, which is the total amount to be invested, and a set of available investments. An allocation of the asset is chosen among all (or part) of the available investments, and after one time step, incomes are added to the original amount. The *liability* of a given time step is paid (subtracted from the current asset), and a new allocation happens. This process is repeated until there are no more assets (hence generating a *deficit*) or liabilities (and there is a *surplus*).

The above structure can be modeled as a Markov Decision Process (MDP) [Sutton and Barto, 2018]. In such a process, there is a current state s_t, which is observed by

an agent. At each time step, this agent selects an action a_t, based on a *policy*[1] $\pi(a_t|s_t)$ and the environment transits to a new state s_{t+1}, according to a *transition probability* $p(s_{t+1}|s_t, a_t)$. Based on this state-action-next state tuple, the agent receives a reward r_t, provided by the environment. Then, a new current state s_{t+1} is observed, a new action a_{t+1} is selected, a state transition occurs again, and a reward is given. This process is repeated, until a terminal condition is met.

Reinforcement Learning (RL) comprises a group of machine learning algorithms appropriate for MDP solving [Sutton and Barto, 2018]. Its core idea is to find the policy $\pi(s_t|a_t)$ that maximizes the expected value of the sum of rewards, along the whole process.

In this work, a Reinforcement Learning algorithm is used for the first time to solve different instances of an Asset-Liability Management problem. In order to do so, an ALM environment is modeled as a Markov Decision Process. Several debt flows have been simulated, with different lengths and asset/liability ratios. Results are compared to those of more basic investment strategies, and it is shown that such a framework is well suited for this kind of problem, becoming a reasonable option to be considered against more popular methods, such as those based on Multistage Stochastic Programming (MSP).

1.2- Motivation

As mentioned earlier, the study of asset-liability management problems is of paramount importance for all sorts of companies, especially in the financial sector. In the particular case of closed pension funds, an erratic strategy can lead thousands of people to lose their retirement benefits, with catastrophic consequences. For insurance plans, bad ALM decisions lead companies to fail to fulfill their obligations, leaving their customers uncovered in practice.

RL algorithms have never been used to address asset-liability management problems, to the best of the author's knowledge. A major contribution of this study is the use of continuous state spaces, instead of discrete scenario trees, as in most common MSP approach [Cariño et al., 1994; Hilli et al., 2007; Duarte et al., 2017; de Oliveira et al., 2017; Consigli et al., 2018]. This is possible due to recent advances in Deep Reinforcement

[1] Such policy can be stochastic or deterministic

Learning (DRL) algorithms, which combine RL techniques with the power of deep neural networks. Besides, RL enables handling larger state spaces and longer time horizons more efficiently, as we show in the computational experiments section.

1.3- Goals

The main question that this dissertation tries to answer is: are reinforcement learning algorithms suitable for solving asset-liability management problems? The author's hypothesis is that this type of algorithm generates adequate results when the problem is modeled properly.

The major goal of this work is to analyze how DRL algorithms perform and behave at an ALM problem. In order to do so, an environment capable of interaction with such algorithms was developed.

As minor goals, we can enlist:

- Properly model asset-liability management as a Markov Decision Process, so that it can be solved with a reinforcement learning algorithm;
- Compare the performance of the chosen algorithm, known as Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015], with two other strategies;
- Conduct an experimental analysis (through simulations) of the algorithm's behavior for different liability lengths and values.

1.4- Methodology

As previously stated in Section 1.1, the structure of an ALM problem can be easily modeled as a Markov Decision Process. So, the first thing done was to map ALM elements into MDP ones. This mapping is thoroughly detailed in Chapter 5.

With the MDP properly set, the Reinforcement Learning literature has been researched to find algorithms suited for such modeling. The fact that both state and action

spaces are continuous reduce the set of available algorithms to those who use policy gradient techniques. Among these, those known as actor-critic algorithms are state-of-the-art nowadays.

After analyzing this reduced set of algorithms, one of them seemed the most suited for the addressed problem: *DDPG*, which is detailed in Section 3.4.2. It is a deterministic algorithm, as its name suggests.

The next step was to write the environment's code effectively. The Python library known as OpenAI Gym [Brockman et al., 2016] has been used to such an end. It has tens of environments already modeled, and a set of tools for creating new, customized ones. Special attention was given to the choice of the reward function, which is done in this step. Technical details can be found in Chapter 6.

Fifteen instances for the problem were generated, split into three groups: every group has a time horizon, and within each of them, five different liability flows, to simulate increasing levels of difficulty. The higher the liabilities, the harder it is to fulfill all its obligations.

Then, computational experiments were performed. To do so, an educational Python package called SpinningUp, developed by the OpenAI group was used, with a few minor changes. Details of the experiments are shown in Chapter 6.

Results found were presented and analyzed in Chapter 7. Two basic investment strategies are compared to the proposed approach, using the very same test environment: the first using a uniform allocation, splitting total assets evenly trough all available investments; and the second using a technique known as *Mean-Variance Optimization (MVO)*, as proposed by Markowitz [1952] in his modern portfolio theory.

A uniform strategy for result comparison is a technique used in other works, such as a recent Master's dissertation [de Almeida, 2016], while the mean-variance optimization approach has been the market standard for simple portfolio optimization (not considering liability flows).

1.5- Organization of Chapters

The remaining of this work is structured as follows. Chapters 2 and 3 describe, respectively, the fundamentals of Asset-Liability Management and Reinforcement Learning. Chapter 4 lists ALM's most common solving techniques. Chapter 5 details the proposed model. Chapter 6 describes the conducted computational experiments, their results, their analysis, and brings comparisons of given results with other techniques. Finally, the conclusions, as well as some ideas for future research, are presented in Chapter 7.

2- Fundamentals of Asset-Liability Management

According to the Society of Actuaries ALM Principles Task Force [Luckner et al., 2003], the definition of asset-liability management is as follows:

> *"Asset-Liability Management is the ongoing process of formulating, implementing, monitoring, and revising strategies related to assets and liabilities to achieve financial objectives, for a given set of risk tolerances and constraints"*.

In other words, asset-liability management is a technique used to continuously optimize investment portfolios, trying to match investment returns with a given flow of future liabilities, while respecting a policy of risks and regulatory constraints.

Section 2.1 summarizes a few basic ALM concepts, Section 2.2 gives a brief description of usual portfolio optimization, while Section 2.3 compares it to general ALM idea, and Section 2.4 presents how asset and liability dynamics behave.

2.1- Basic Concepts

An *asset* is a total amount one has already invested (like bonds, real estate, or equity) or yet to be invested (cash). A *liability flow* is nothing more than a series of debts that must be fulfilled at their proper due dates, like installments of a loan, a mortgage, or yearly tuition. *Available investments* are the options one has to invest, which may be restricted by the local market, regulatory constraints, etc. Based on these concepts, we define the liability's present value, the discount rate, and the solvency rate.

2.1.1 Present Value and Discount Rate

As money can (and should) be invested, a debt in the future is different from a debt today. For example, if someone has to pay $ 1,060 within a year, but can invest their

money for a 6% return, he will need only $ 1,000 today. In this example, the original debt, which will be paid in a year, is called the *Future Value (FV)*, and the amount you need today is the *Present Value (PV)* of the debt (see Equation 1). To calculate one from the other, one has to know (or estimate) the investment's return: this is the *Discount Rate (DR)*. If the debt is more than one time period ahead, the future value has to be discounted once for every period.

$$PV = \frac{FV}{(1+DR)^t} \tag{1}$$

This operation is known as *discounting a flow to its present value*. The present value of a liability flow L_{PV} is the sum of all its single debts, discounted to their present values, at a proper rate.

$$L_{PV} = \frac{L_1}{(1+DR)} + \frac{L_2}{(1+DR)^2} + \ldots + \frac{L_T}{(1+DR)^T} = \sum_{t=1}^{T} \frac{L_t}{(1+DR)^t}, \tag{2}$$

where L_{PV} represents the liability's present value, L_t is liability's installment at time t, and T is the amount of installments.

2.1.2 Solvency Rate

The ratio between total assets and liability's present value is known as *solvency rate*. In order to keep the risk-return relation of an ALM problem well balanced, it is desired to keep it always close to 1. A low solvency rate is associated with high deficit probabilities, while a high value suggests the fund is incurring in too much risk.

2.2- Usual Portfolio Optimization

The modern portfolio theory states how a risk-averse investor can optimize a portfolio by maximizing its expected return, given a maximum level of a chosen risk measure (usually, portfolio volatility) [Markowitz, 1952]. According to this theory, higher returns lead to higher risks, and it is possible to find an efficient frontier (Fig. 1) of optimized

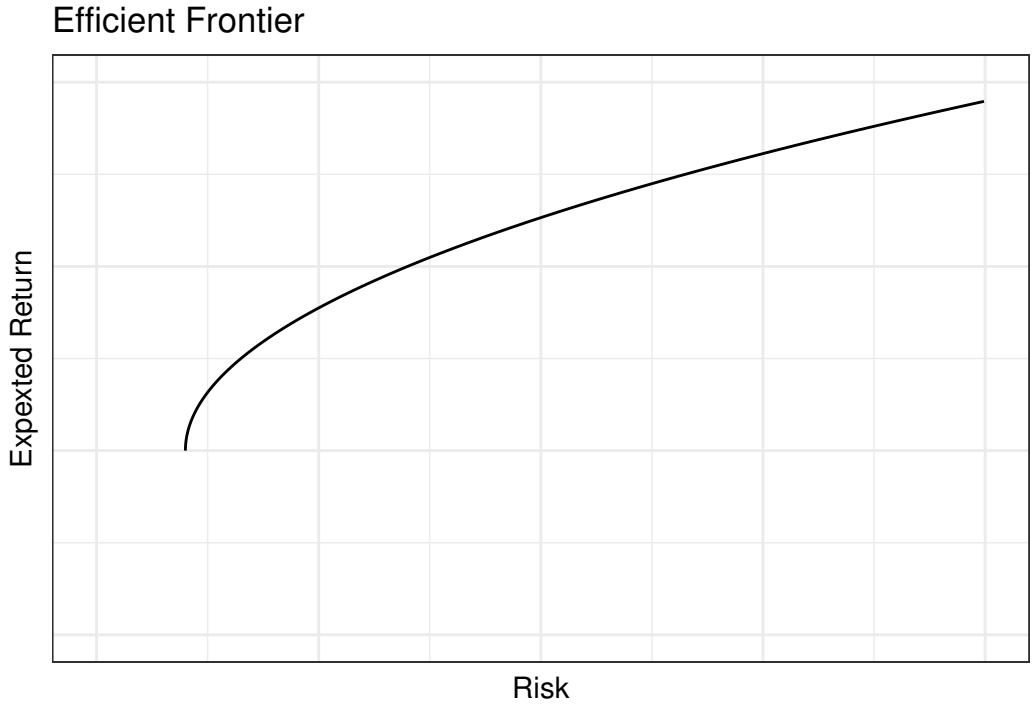

Figure 1 – Efficient frontier, as idealized by Harry Markowitz.

portfolios for all possible expected returns and their respective risk measures. Each point on that frontier represents the minimum-variance portfolio for the given expected return.

This approach has been extensively studied by the academy and widely used by the market ([Dhrymes, 2017; Kaplan, 2017; Aouni et al., 2018], among many others). However, it is not very useful in the presence of debt flows, since it is short-sighted (optimizing only for the present moment), and does not consider risk factors for the liability. In such a situation, an optimization model that takes into account the expected returns on investments, risk measures, and liability attributes together is needed [Aro and Pennanen, 2017].

2.3- ALM vs. Portfolio Optimization

To illustrate the difference between an ALM problem and a standard portfolio optimization, imagine the following situation: a person has $ 1.000 to invest, and has to pay $ 1.080 after two years. He chooses a high-risk investment, which has a 10% loss in

the first year (bringing him down to $ 900), and a 20% profit in the second (getting him back to $ 1.080), and the debt can be honored. On the other hand, if he had to pay $ 540 in each year (so, having a liability flow, instead of a single payment), the first year would bring him down to $ 900 - $ 540 = $ 360, and the 20% profit of the second year would apply only upon this difference, leaving him with $ 432, and so, unable to honor his debt.

The context presented above is where ALM comes in hand. Its main goal is to find an optimal investment strategy that considers assets and liability flows together. The above example considers stochasticity only in asset returns, but liabilities are usually highly unpredictable, and somehow correlated with available investments.

In other words, portfolio optimization is concerned with expected returns and volatility for *the whole period*, since a loss in a given time step can be recovered in the future. ALM, on the other hand, considers returns, volatility and debts for every single time step, because any loss can have a huge impact in future returns.

2.4- Asset and Liability Dynamics

ALM dynamics are pretty simple, as represented in Figure 2. The investor has an initial asset amount, which will be allocated among n available investments. After a certain time period, the return of these investments is added to the original amount. When the time is due, an obligation must be paid, which is that period's liability. This amount is subtracted from the total assets, and the process is repeated with the assets left.

To further clarify liability stochasticity and its interaction with asset returns, let us consider an example scenario: a closed pension fund has gathered one billion monetary units through the contributions made by its participants. This current amount, A_t, has to be invested, and will later be used to pay their retirements. The fund administration should pay each retired participant until his/her death. To do so, they estimate how much has to be paid each year, and until when. This estimation is based on the number of living pensioners, their life expectancy, and how much they receive per year.

For example, suppose in a given year, fund's administration expects to have 500 pensioners alive, with an average year income of 100,000 monetary units: so, this period's liability L_t is estimated in 50 millions. Chosen asset allocation had a 4% return r_t; so, total

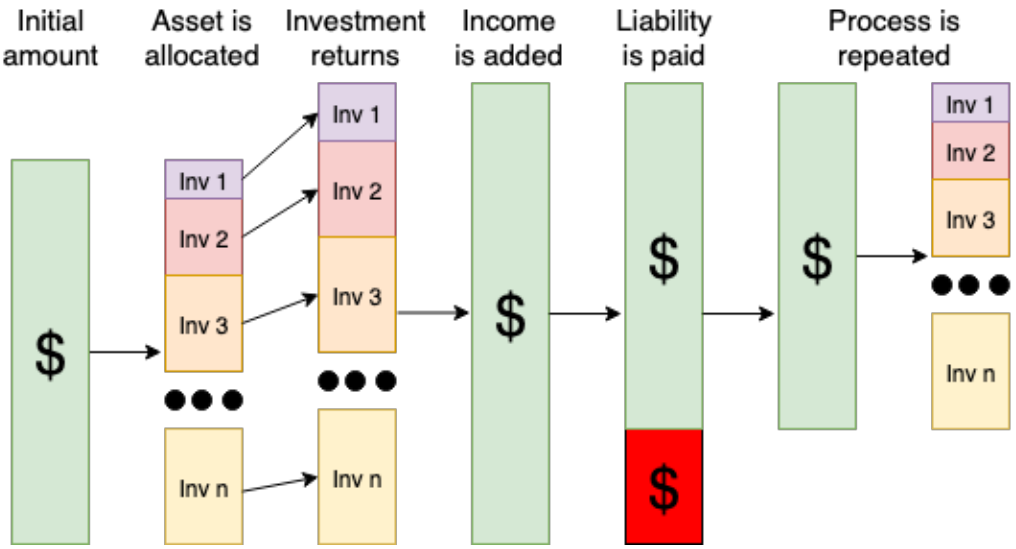

Figure 2 – ALM Dynamics.

amount at next year is calculated by updating the current amount by the year's return, and then subtracting current liability (see Equation 3).

$$A_{t+1} = A_t \cdot (1 + r_t) - L_t = 1,000,000,000 \cdot (1 + 0.04) - 50,000,000 = 990,000,000 \quad (3)$$

In the following year, sixty more participants shall retire, and the actuarial table indicates ten will perish. So, estimated debt for the following year, L_{t+1}, will be 55 million. These yearly estimates will grow up to a certain point when the amount of living retired participants reaches its peak, and then slowly shrink to zero, as they cease living. This liability behavior tends to be similar to a chi-squared density function, like the one seen in Figure 3, which is pretty typical for closed pension funds, but not a general rule in ALM problems.

In the above scenario, a risky investment strategy could lead to losses that would let lots of pensioners without their incomes. On the other way, a too conservative strategy may not make enough money to honor fund's debts. Thus, the choice of an adequate investment strategy is of paramount importance to the fund's success. This choice must simultaneously consider total assets, available investments, and liability flow (Section 2.1).

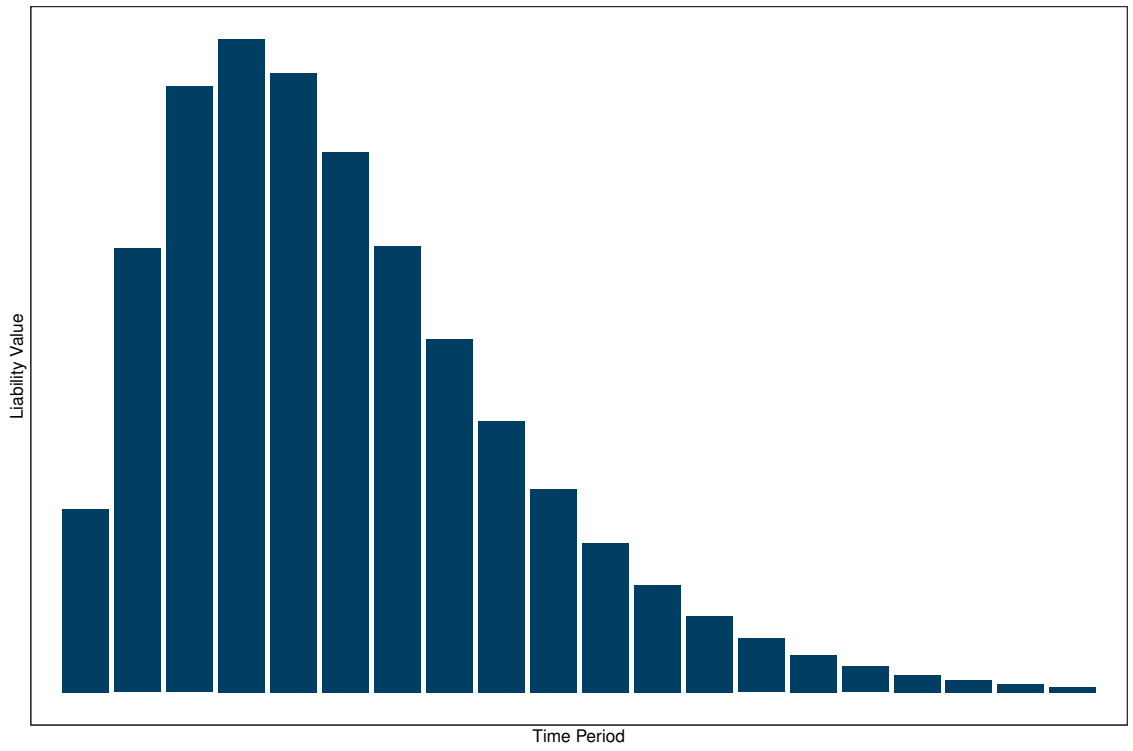

Figure 3 – Chi-squared shaped liability example.

3- Fundamentals of Reinforcement Learning

In machine learning, there are three major groups of algorithms: supervised learning algorithms try to find patterns in labeled data; unsupervised learning does the same in unlabeled data; and reinforcement learning, which seeks to find a *policy* that maximizes a given metric, i.e., to define what action should be taken in a particular situation to maximize the probability of something desired to happen.

Reinforcement learning is the group of interest in this work. The goal is to train an algorithm that, at every time step, given a total asset, a liability flow, and a set of available investments, returns an allocation that maximizes the probability of all future debts to be paid.

In order to do so, first thing to be done is model the problem at hand as a *Markov Decision Process*, which is described in Section 3.1. In Section 3.2, basic concepts of the reinforcement learning framework are described. *Policy gradient* methods, which are better suited for problems with continuous variables, are shown in Section 3.3, and their natural evolution, *actor-critic* algorithms, are described in Section 3.4.

3.1- Markov Decision Process

A Markov Decision Process is defined as “a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states” [Sutton and Barto, 2018]. The elements of such formalization are:

- a *state space* $\mathcal{S}$;
- an *action space* $\mathcal{A}$;
- an initial state $s_1 \in \mathcal{S}$ with density $p_1(s_1)$;
- a *reward function* $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$;

- a *transition distribution function* with density $p(s_{t+1}|s_t, a_t)$, which must satisfy the first-order Markov property: $p(s_{t+1}|s_1, a_1, \ldots, s_t, a_t) = p(s_{t+1}|s_t, a_t)$; and
- a *terminal condition* that, when reached, ends the execution.

A *policy function* $\pi_\theta : \mathcal{S} \to \mathcal{A}$ is defined, to select the actions from a given state. This policy might be stochastic (defined by $\pi_\theta(a_t|s_t)$) or deterministic (defined by $a_t = \mu_\theta(s_t)$). In both cases, $\theta \in \mathbb{R}^n$ is the parameter vector of the function.

3.2- Basic Concepts

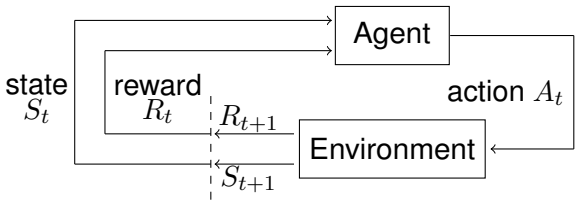

Figure 4 – Block diagram of the RL framework [Sutton and Barto, 2018].

Reinforcement learning algorithms work through repeated interactions between the *agent* (which represents the decision maker) and an *environment* (which simulates the problem at hand). The agent observes a state s_t, and through its current policy, takes an action a_t. Based on this state-action pair, the environment returns a new state s_{t+1}, and a reward signal r_{t+1}. The process is repeated until a terminal state is reached. The sequence $s_0, a_0, r_1, s_1, a_1, r_2, \ldots$ generated trough these interactions is called a *trajectory* and is denoted by τ.

3.2.1 Exploration vs Exploitation

A fundamental concept is that of the ”conflict” between exploration and exploitation [Sutton and Barto, 2018]. If an agent always tries to take the best possible known action, it will fail to explore the solution space. So, there may be an action that seems to be sub-optimal now but is a better choice in the long run.

On the other hand, if the agent is always exploring new possibilities, it will not exploit what it has learned, hence failing to converge to a satisfying solution. So, every algorithm tries to balance when it must explore new possibilities, and when it exploits what is already known.

3.2.2 Important Equations

The *return* of a trajectory τ, known as $G(\tau)$, or *cumulative discounted reward*, is defined as the sum of its discounted rewards. Usually, a *discount factor*[1] $\gamma \in [0, 1]$ is used, to avoid problems with infinite trajectories, and to enforce the fact that current reward is more important than future ones[2]. It can be described as Equation 4.

$$G(\tau) = \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots + \gamma^T r_T = \sum_{t=1}^{T} \gamma^t r_t \tag{4}$$

The main goal of a reinforcement learning algorithm is to find a policy that maximizes return's expectation $\mathbb{E}(G(\tau))$.

Another important concept is that of *value functions*. The value function of policy π and state s_t, $V^\pi(s_t)$, represents the expected cumulative discounted reward of state s_t, given that policy π is followed by the agent. That is: if one is in state s_t, and follows policy π, what is the expected return? Note that, in mathematical terms, $V^\pi(s_t)$ can be written as Equation 5.

$$V^\pi(s_t) = \mathbb{E}_\pi(G_t|s_t) \tag{5}$$

A similar concept is the *action-value function* of state s_t, action a_t and policy π, $Q^\pi(s_t, a_t)$. The difference from the value function comes from the fact that the agent takes an arbitrary action a_t in its first step, and follows policy π afterward. Or: if one is in state s_t, takes action a_t now, and follows policy π from state s_{t+1} till the end, what is the expected return? In mathematical terms, Q^π can be described as in Equation 6.

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi(G_t|s_t, a_t) \tag{6}$$

[1] Although concepts are similar, this is *not* the rate used to discount a liability to its present value.

[2] Although not usual, discount factors can be equal to 0 (if the current reward is just as important as future ones) or 1 (if only current reward is important).

At last, the *advantage function* represents how much one can expect as an excess return if the agent takes arbitrary action a_t now instead of following policy π. It is used to measure how much the expected return will increase (or decrease) if we change the action that policy π suggests for an arbitrary action. It is given by the difference between action-value function and value function. Equation 7 calculates the advantage function.

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \tag{7}$$

3.2.3 Main Solution Methods

The goal of a reinforcement learning algorithm is to find the policy that maximizes the expectation of cumulative discounted reward, given an initial state s_1. There are several ways of doing this, which can be divided in three major groups:

1. **Tabular Methods**: better suited for discrete state and action spaces (or spaces small enough to be discretized and represented as a table). Approximate value functions are represented as arrays or tables, and their optimal values are calculated through repeated iterations of *Bellman equations*, as shown in equations 8 and 9.

$$V^{\pi}(s_{t+1}) = r_t + \gamma V^{\pi}(s_t), \tag{8}$$

$$Q^{\pi}(s_{t+1}, a_{t+1}) = r_t + \gamma Q^{\pi}(s_t, a_t). \tag{9}$$

   The optimal policy is implicit: at any state, the action taken is the one that maximizes V^{π} or Q^{π}. An example of a tabular algorithm is Q-Learning [Watkins and Dayan, 1992].

2. **Policy Optimization**: the policy is explicitly given as a probability distribution $\pi_{\theta}(a_t|s_t)$ (stochastic policy) or as a function $a_t = \mu_{\theta}(s_t)$ (deterministic policy). This approach is suited to continuous state spaces, and both continuous and discrete action spaces. One of the most known algorithms in this group is REINFORCE [Williams, 1992].

3. **Actor-Critic Algorithms**: this is the most used framework nowadays, combining

the previous two. Algorithms in this group present two components: an *actor*, which works as the agent, just like in policy gradient algorithms, and a *critic*, which estimates the action-value function $Q^{\pi}(s, a)$, used to update the actor's parameters. Asynchronous Advantage Actor-Critic (A3C) is a good example of these algorithms [Mnih et al., 2016].

The algorithm used in this work, Deep Deterministic Policy Gradient, belongs to the third group. In order to understand how it works, we first need to understand the Policy Gradient Theorem.

3.3- Policy Gradient Theorem

The first version of a policy gradient algorithm is known as REINFORCE [Williams, 1992]. Like any other reinforcement learning algorithm, it tries to solve problems modeled as Markov Decision Processes, as defined in Section 3.1. According to MDP's elements, the algorithm takes as inputs:

- an initial state probability distribution $p(s_0)$;
- a trajectory τ as a sequence of states and actions, beginning at an initial, random state s_0, and ending when a certain terminal condition is met:
  $\tau = s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T$;
- a reward function, which maps every *state - action - next state* tuple into a real value;
  $R : S \times A \times S \to \mathbb{R}$
- a stochastic policy $\pi_{\theta}(a_t|s_t)$, with a probability distribution of a_t conditioned on s_t, with parameter vector θ;
- a transition probability function $p(s_{t+1}|s_t, a_t)$, which is the probability distribution for s_{t+1}, given s_t and a_t;

According to these definitions, the probability distribution of a trajectory τ is given by the product of the probabilities for each of its events, like in Equation 10 [Sutton and

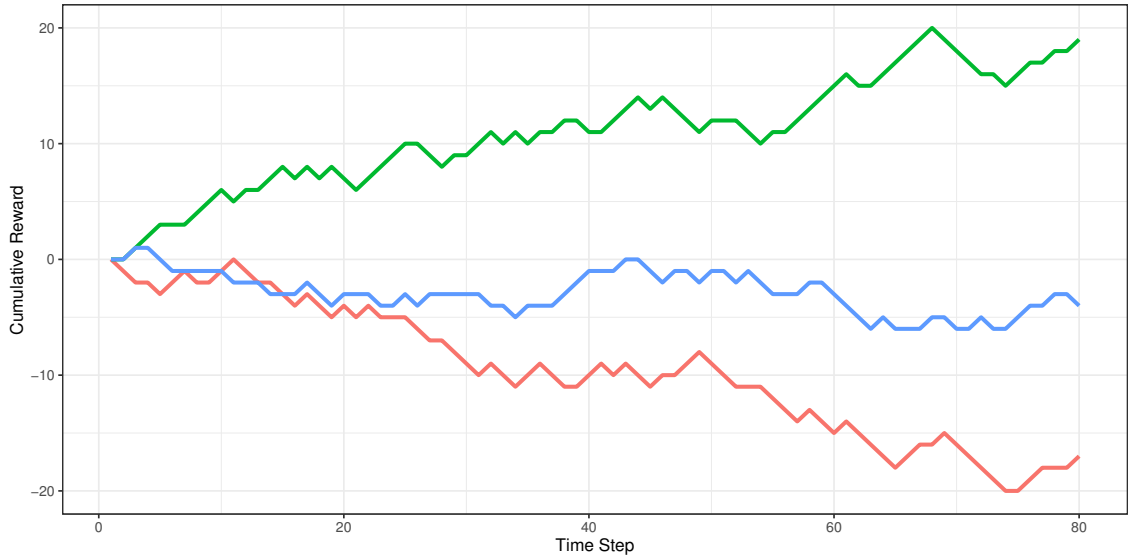

Figure 5 – Representation of Policy Gradient Trajectories.

Barto, 2018].

$$p_\theta(\tau) = p(s_0) \cdot \pi_\theta(a_0|s_0) \cdot p(s_1|s_0, a_0) \cdot \ldots \cdot \pi_\theta(a_{T-1}|s_{T-1}) \cdot p(s_T|s_{T-1}, a_{T-1}). \tag{10}$$

The goal of a policy gradient algorithm is to find a parameter vector θ^*, for the policy $\pi_\theta(a_t|s_t)$, which maximizes the expected cumulative discounted reward, expressed in Equation 11.

$$\theta^* = \arg \max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t \gamma^t r_t(s_t, a_t) \right]. \tag{11}$$

In Figure 5, we can see a graphical representation of the intuition behind this theorem[3]. The image shows three random trajectories. All of them start at the same initial state s_0, but as they are stochastic, they have different outcomes. The red one has a low return associated; the blue one has a better return, while the green one is the best. Based on this, the algorithm will try to improve its parameters to maximize the probability of a better trajectory and minimize others.

[3]Figure based on policy gradients lecture on Berkeley's Deep RL course.

3.3.1 Direct Policy Differentiation

In order to estimate parameter vector θ^*, the policy gradient makes use of the *gradient ascent* algorithm. It randomly initializes θ, and then, at every iteration, updates it in the direction of cost function's gradient $\nabla_\theta J(\theta_t)$ by a small, arbitrary learning rate α:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \cdot \nabla_\theta J(\theta_t) \tag{12}$$

The cost function that depicts the expected return (Equation 11) should be maximized. So, we need to find its gradient concerning to θ. Since it is an expectation, it can be rewritten as an integral, which suits our needs:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [G(\tau)] = \int p_\theta(\tau) \cdot Q^\pi(s, a) \; d\tau \tag{13}$$

so that its gradient w.r.t. θ is:

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) \cdot Q^\pi(s, a) \; d\tau \tag{14}$$

In order to handle this integral, term $\nabla_\theta p_\theta(\tau)$ will be replaced, using the following identity:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \cdot \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = p_\theta(\tau) \; \nabla_\theta \; \log p_\theta(\tau) \tag{15}$$

so that the original integral now becomes:

$$\nabla_\theta J(\theta) = \int p_\theta(\tau) \nabla_\theta \; \log p_\theta(\tau) \cdot Q^\pi(s, a) \; d\tau = \mathbb{E}_{\tau \sim p_\theta(\tau)} [\nabla_\theta \; \log p_\theta(\tau) \cdot Q^\pi(s, a)] \tag{16}$$

The problem now is to find the gradient of $\log p_\theta(\tau)$, since $G(\tau)$ does not depend

on θ. Using equaton 10, we have:

$$
\begin{aligned}
p_\theta(\tau) &= p(s_0) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \qquad (17)\\
\Rightarrow \log\ p_\theta(\tau) &= \log\ p(s_0) + \sum_{t=1}^{T} \log\ \pi_\theta(a_t|s_t) + \log\ p(s_{t+1}|s_t, a_t)\\
\Rightarrow \nabla_\theta\ \log\ p_\theta(\tau) &= \nabla_\theta \left[\log\ p(s_0) + \sum_{t=1}^{T} \log\ \pi_\theta(a_t|s_t) + \log\ p(s_{t+1}|s_t, a_t) \right]\\
\Rightarrow \nabla_\theta\ \log\ p_\theta(\tau) &= \sum_{t=1}^{T} \nabla_\theta\ \log\ \pi_\theta(a_t|s_t)
\end{aligned}
$$

The distribution of the initial state and the transition probability distribution do not depend on θ, so, their gradients are zero. This let us with:

$$
\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta\ log\ p_\theta(a_t|s_t) \right) \cdot Q^\pi(s, a) \right] \qquad (18)
$$

Equation 18 has become known as the *policy gradient theorem* [Williams, 1992].

3.3.2 REINFORCE

In the same work in which Policy Gradient Theorem was developed, the algorithm known as REINFORCE, or Monte Carlo Policy gradient, was also proposed [Williams, 1992]. This algorithm defines a stochastic policy $\pi_\theta(a_t|s_t)$ (usually Gaussian for continuous action spaces, or a softmax for discrete ones), and arbitrarily initializes a parameter vector θ. Then, for each episode, it samples a full trajectory through an agent/environment interaction, and updates θ at every step, as can be seen in Algorithm 1.

Algorithm 1 – REINFORCE

Data: θ, N, T
Initialize θ arbitrarily;
for episode $\leftarrow 1$ to N **do**
  for $t \leftarrow 1$ to T **do**
    $\theta \leftarrow \theta + \alpha\ \nabla_\theta\ log\ \pi_\theta(s_t, a_t) \cdot G_\tau$;
  end
end

Despite its substantial theoretical importance, this algorithm is not very useful in practice, since the estimation of $Q^{\pi}(s_t, a_t)$ (Equation 6) with a single rollout implies in extremely high variance, leading to convergence issues.

3.4- Actor-Critic Methods

The main idea behind actor-critic algorithms is to use the policy gradient framework, described in Section 3.3, with a *second* function approximator to estimate $Q^{\pi}(s, a)$, so, avoiding the convergence issues of REINFORCE. It is like having an actor (the agent) making decisions, and a critic (the action-value function estimator) telling it how right (or wrong) these decisions are. At every iteration, both actor and critic parameters are updated.

Decisions made by the actor can be either *stochastic* or *deterministic*. In the first case, the action actually taken is randomly chosen according to a probability density function $\pi_{\theta}(a_t|s_t)$ (as described in Section 3.3), while in the second, the action is the output of a function: $a_t = \mu_{\theta}(s_t)$. In order to use this option, it's mandatory to understand the *deterministic policy gradient* theorem.

The policy gradient version, which is used in this work, is the *Deterministic Policy Gradient (DPG)*, detailed in Section 3.4.1. The specific algorithm used is the *Deep Deterministic Policy Gradient* is presented in Section 3.4.2.

3.4.1 Deterministic Policy Gradient

In 2014, the deterministic policy gradient theorem was presented [Silver et al., 2014]. Until then, it was believed that a deterministic policy gradient did not exist in a model-free algorithm, but they proved that it exists, and is the expected gradient of the action-value function.

The authors demonstrate that, in continuous action spaces, when using a deterministic policy $a_t = \mu_{\theta}(s_t)$, policy parameters should be moved in the direction of the gradient

of Q with respect to θ, and applying a simple chain rule to the action.

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] \tag{19}$$

The deterministic policy gradient theorem, together with a few other ideas (explained in Section 3.4.2), form the basis for the algorithm known as *Deep Deterministic Policy Gradient*.

3.4.2 Deep Deterministic Policy Gradient

Until 2013, it was believed that non-linear function approximators could not be used to estimate value or action-value functions, due to several problems [Lillicrap et al., 2015]. One of them is the dependency between sequential states; another is the change in data distribution as the algorithm is updated.

To address these problems, an altered version of the Q-learning algorithm [Watkins and Dayan, 1992] was proposed in [Mnih et al., 2013]. An *experience replay buffer* was introduced, where transition tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ are stored and randomly sampled, to minimize correlations. They also introduced a *target Q network* to provide consistent targets for the main network's training. This auxiliary network is used to calculate Q-values one step ahead, and use them to find targets for the Q-values of the current step, using one of the *Bellman equations*:

$$Q(s_t, a_t) = r_t + Q(s_{t=1}, a_{t+1}). \tag{20}$$

These targets are then used to train the main Q network. The resulting algorithm reaches desired training stability but works only with discrete action spaces.

These ideas have been adapted to an actor-critic framework, with a deterministic policy gradient agent [Lillicrap et al., 2015], along with batch normalization [Ioffe and Szegedy, 2015]. The resulting algorithm is known as *Deep Deterministic Policy Gradient* (DDPG).

DDPG makes use of a deep neural network, $\mu(s|\theta^\mu)$, as the actor, and another one, $Q(s, a|\theta^Q)$, as the critic. After randomly initializing the parameters of both networks, it

creates copies of them, which are called *target actor* μ' and *target critic* Q', which have a role identical to the auxiliary Q network [Mnih et al., 2013]. These networks are used to set targets for critic updates, and are not trained as the main ones. Instead, their parameters are updated slowly, in the direction of actor and critic learned parameters. This four networks setup improves training stability, and reduces convergence issues. Besides that, as the algorithm is deterministic, random noise is applied to every action during training to ensure the exploration of the solution space.

The pseudocode can be seen in Algorithm 2, and Figure 6 shows its flow chart. It works as follows:

- Actor and Critic networks are initialized with random weights (lines 1-2)
- Target Actor and Target Critic are initialized, with the same weights of main Actor and Critic (lines 3-4)
- Replay Buffer is initialized, through n iterations of actor and environment (line 5)
- The main algorithm loop begins (lines 6-21), and runs for an arbitrary number M of episodes.
- In each iteration, an initial state s_1 is observed (line 8) and then an inner loop begins (lines 9-20):
  - For every time step t, action a_t is selected, based on deterministic policy $\mu(s_t|\theta^\mu)$ and random noise $\mathcal{N}_t$ (line 10)
  - Based on a_t, observe r_t and s_{t+1} (lines 11-12)
  - Store s_t, a_t, r_t and s_{t+1} in R (line 13)
  - A random minibatch is sampled from replay buffer R, to avoid correlation between sequential agent/environment interactions (line 14)
    - s_{t+1} is used as input for target actor, to generate a_{t+1}
    - With s_{t+1} and a_{t+1} as inputs, target critic calculates $Q(s_{t+1}, a_{t+1})$
  - According to the Bellman equation, vector of target values y is calculated, adding r_{t+1} with $Q(s_{t+1}, a_{t+1})$ (line 15)
  - Vector y is used as target labels, to train the main critic network, as a simple supervised learning problem (line 16)

– With this newly updated critic, actor network is updated, trough Deterministic Policy Gradient theorem (line 17)

– Target critic and target actor are updated, according to new main critic and actor parameters, and a learning rate, τ (lines 18-19)

– Iterate until the end of the episode

Algorithm 2 – Deep Deterministic Policy Gradient

Data: $\theta^Q, \theta^\mu, M, N, T$
1 Initialize critic network $Q(s, a|\theta^Q)$ with weights θ^Q
2 Initialize actor $\mu(s|\theta^\mu)$ with weights θ^μ
3 $\theta^{Q'} \leftarrow \theta^Q$
4 $\theta^{\mu'} \leftarrow \theta^\mu$
5 Initialize replay buffer R
6 **for** episode $\leftarrow 1$ to M **do**
7 Initialize random process $\mathcal{N}$
8 Receive initial observation state s_1
9 **for** $t \leftarrow 1$ to T **do**
10 Select action $a_t \leftarrow \mu(s_t|\theta^\mu) + \mathcal{N}_t$
11 Execute selected action
12 Observe reward and next state
13 Store $(s_t, a_t, r_{t+1}, s_{t+1})$ tuple in R
14 Sample a random N-size minibatch from R
15 $y_i \leftarrow r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
16 $\theta^Q \leftarrow \arg\min_{\theta^Q} \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
17 $\nabla_{\theta^\mu} \mu|_{s_i} \approx \frac{1}{N} \sum_i \nabla_a Q(\cdot)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$
18 $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
19 $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
20 **end**
21 **end**

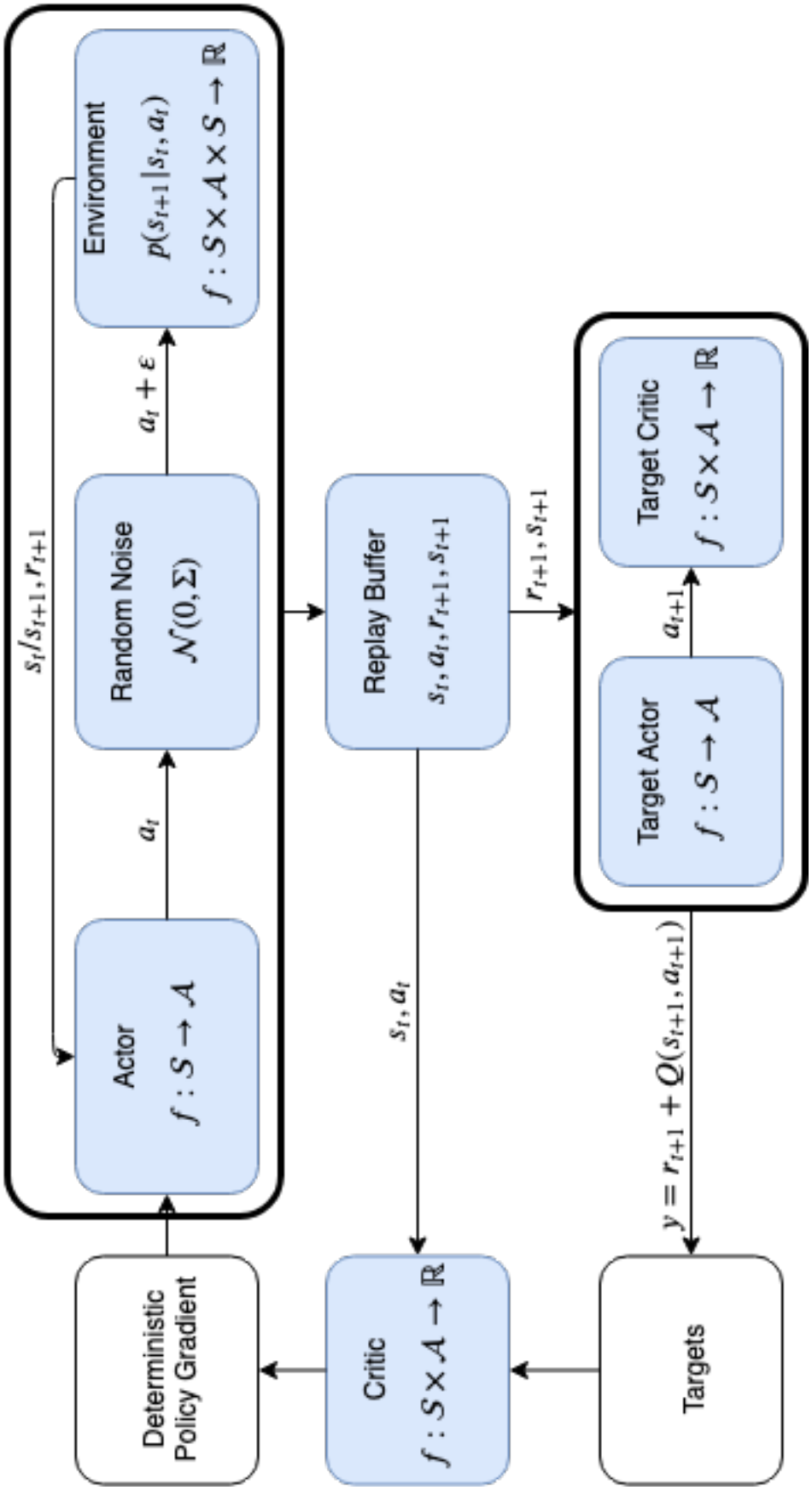

Figure 6 – Deep Deterministic Policy Gradient.

4- Related Work

In the *Handbook of Asset and Liability Management* [Rosen and Zenios, 2008], ALM solution methods are grouped along two axes: time (single or multiple periods) and risk factors (static or stochastic). Single period solutions are short-sighted, making decisions only in the present moment, while multiple period ones take future steps into account. On the risk factor axis, static solutions consider that uncertainty in the model will remain constant through time, as opposed to stochastic models.

Single period static models usually try to match assets and liabilities *durations*[1], a financial technique known as *immunization*. In doing so, the portfolio is secured against small changes in interest rates, but not against major ones. Single period stochastic models are similar, but take risk factor's volatility into account. The most known and used technique in this group is the mean-variance optimization [Markowitz, 1952].

Multiple-periods static models are similar to their single period counterparts but incorporate portfolio rebalancing. One of the very few examples of this group is found in the *Stochastic Optimization Models in Finance* [Fama, 1975]. Moreover, due to ALM problems' very nature, multiple-periods stochastic models are the most common approach. This work will focus on them from now on.

4.1- Usual Solution Methods

There are very few analytical solutions to an ALM problem, and all of them on limited cases [Gulpinar and Pachamanova, 2013]. Most solutions in literature are numerical and fall under three categories[2]: dynamic programming, simulation-based, and the most used, stochastic programming, which focuses on finding an optimal investment policy over a set of simulated scenarios.

Although being the most used technique, scenario simulation is still challenging.

[1] An average of the asset's maturities, weighted by their net present values

[2] Actually, Gulpinar and Pachamanova [2013] proposed a fourth one, based on robust optimization.

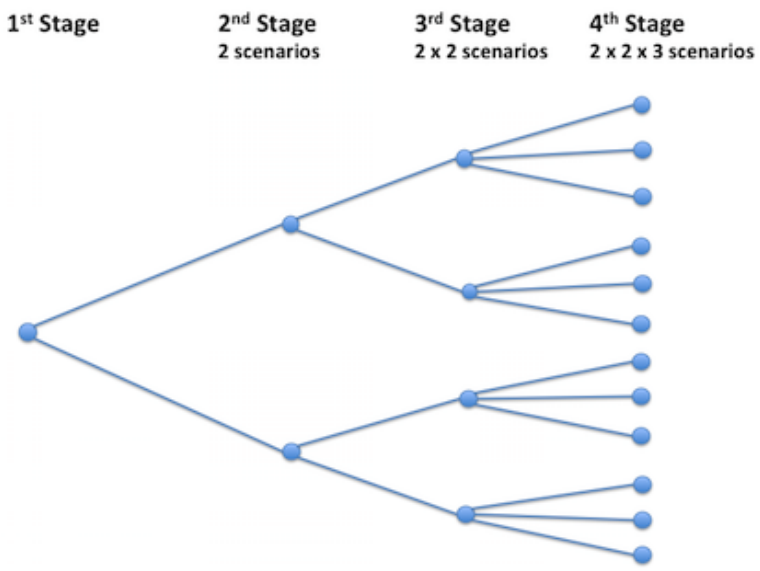

Figure 7 – An example scenario tree, with 4 stages and 12 scenarios.

The multi-period nature of an ALM problem makes the number of scenarios needed for a good solution to increase exponentially for longer time lengths, increasing the problem's dimension and its difficulty level. Therefore, most works use state-space approximations and problem relaxations to generate their scenario trees, described in Section 4.1.1. Section 4.1.2 elucidates the concept of an investment policy, and a few MSP models present in the literature are exposed in Section 4.1.3.

4.1.1 Scenario Trees

A scenario tree is generated from a root node (associated with an initial observation). For each possible outcome of the problem's random variables, a child node is attached to the tree [Defourny et al., 2011]. In a multistage schema, this branching process is repeated for every child node, until a terminal state is reached. This tree generation, as described, is impossible when we are dealing with continuous random variables, and so, there is the need to discretize them. The *structure* of the tree is a sequence of integers that indicate how many children a node in a given stage has. In Figure 7, we can see a 2-2-3 structure. The product of these numbers gives us the total number of scenarios in the last stage (12 in the example).

4.1.2 Investment Policies

It has been demonstrated that the optimal investment strategy depends mainly on the investment policy, which determines how much should be allocated in the major investment groups (stocks, bonds, or cash) [Brinson et al., 1986]. Individual securities selection, market timing, and costs (the other elements analyzed) had a minimal impact on the pension funds examined. Therefore, most ALM models try to find the optimal investment policy, given a particular set of regulatory constraints, using major asset groups, instead of individual assets.

4.1.3 Multistage Stochastic Programming

Many models address portfolio optimization, with many different approaches, but the first to incorporate liabilities was created for a Japanese insurance company [Cariño et al., 1994]. It has become known as the Russel-Yasuda Kasai model, named after the Frank Russel Company and the Yasuda Fire and Marine Insurance Co. The model utilizes multistage stochastic programming to define an optimal investment strategy, incorporating a multi-period approach, as well as risk measures and regulatory constraints. It uses a scenario generator module, where the decision-maker can define the premises to be used, like correlation and temporal dependency among assets, depth of the scenario tree, and the number of nodes in each layer. Technical issues for the discrete scenario generation, the mathematical approach used, comparisons of algorithms, and a comparison of the multistage stochastic programming with the previous mean-variance technique used are provided in a posterior work about the problem [Cariño et al., 1998]. According to the authors, the model's major goal is to generate a high-income return, while maximizing long term wealth. A real-life application of the model returned US$ 79 million of extra income along its first two years of use, 1991 and 1992.

In the particular case of pension funds, there are several approaches in the literature. One of them proposed a stochastic programming model for a Finnish pension insurance company, which pays particular attention to the model's uncertainty factors:

investment returns, cash flows, and technical, regulatory reserves [Hilli et al., 2007]. This approach utilizes a stochastic model for assets and liabilities, based on infinite sample spaces [Koivu et al., 2003]. Such an approach leads to infinite-dimensional optimization problems that are not tractable, being numerically solved trough a discretization process. According to the authors, this process is fully automated and hidden from the user [Pennanen and Koivu, 2002]. The model was tested against a static fixed-mix strategy[3], and performed well.

A Brazilian case has been studied, and the model proposed is a multistage stochastic programming algorithm with a new method for measuring and controlling equilibrium risk, e.g., long term solvency risk, or the risk of not paying all benefits until the end of the plan (becoming insolvent) [Valladão and Veiga, 2008]. The model considers five classes of assets, loans (in case of cash shortages), transaction costs, regulatory constraints, market liquidity, and asset inventory. These considerations make the model very realistic. Chosen scenario tree has 5 stages of one, one, three, five, and ten years length, for a total of a 20 years time horizon optimization, with a 10-6-6-4-4 structure (5,760 scenarios), and was based on "Adjusted Random Sampling" [Kouwenberg, 2001]. Instead of measuring insolvency risk at the end of each scenario as other methods do, with a regulatory discount rate, a bootstrapping of the portfolio's returns is proposed, to find a better discount rate estimate.

A multistage stochastic programming model proposed for a Dutch pension fund focused on short-term risk, controlling it using Integrated Chance Constraints [Klein Haneveld, 1986], which is a kind of constraint based on probability [Haneveld et al., 2010]. A good example is Equation 21.

$$\mathbb{P}(F_{t+1} \geq \alpha | (s_t)) \geq \gamma_t \tag{21}$$

The above equation states that the probability of solvency rate at next period of time F_{t+1} being greater than or equal to a certain α level (e.g., 105%), given current state s_t, must be greater than a certain required reliability level (e.g., $\gamma_t = 0.95$). Scenario tree used a 10-10-10 structure for a total 1,000 equally probable scenarios and three periods of time.

[3] A strategy where a fixed asset proportion is defined, and the portfolio is rebalanced at every time step to keep it constant. Such strategy, although not realistic, is a good benchmark.

4.2- More recent approaches

Recently, a few other approaches were proposed to solve ALM problems.

4.2.1 Robust Optimization

As stated at the beginning of Chapter 4, a new approach has been proposed to ALM problems: robust optimization, which takes a worst-case approach to optimization models [Gulpinar and Pachamanova, 2013]. It assumes that the model's uncertain data belongs to an uncertainty set and looks for the optimal solution for the worst values in this set. The risk level of the model can be controlled by varying shape and size of uncertainty's set. Although it has no need to use scenario trees, this approach still has time span problems.

4.2.2 Genetic Algorithms

Genetic algorithms are population-based algorithms, based on the evolution theory by Darwin. They are initialized with a set of feasible solutions for a given problem (an initial *population*). These solutions are evaluated using a fitness function. The best are selected, while the worst are discarded (simulating the "survival of the fittest"). Remaining solutions suffer a variation process (a "mutation") and crossovers, and the new solutions go through the evaluation process again, restarting the cycle. This is repeated until a specific stopping criterion is met.

Genetic algorithms have been used in several portfolio optimization problems [Rao et al., 2010; Zhang and Zhang, 2009; Huang, 2012], but recently, an ALM approach has been proposed [de Almeida, 2016].

Table 1 – Summary of Related Work

Author	State Spaces	Algorithm	Number of Investments	Single/Multi Period	Time Steps
MVO, by Markowitz [1952]	Continuous	Quadratic Programming	Many	Single	1 at a time
Hilli et al. [2007]	Discrete	MSP	3	Multi	5
Valladão and Veiga [2008]	Discrete	MSP	5	Multi	20 years in 5 steps
Haneveld et al. [2010]	Discrete	MSP	3	Multi	3
Gulpinar and Pachamanova [2013]	Discrete	Robust Optimization	3	Multi	3
de Almeida [2016]	Continuous	Genetic Algorithm	3	Multi	40
Fontoura et al. [2019]	Continuous	DDPG	6	Multi	Up to 80

4.3- Overall Conclusion

Multistage stochastic programming is the most common approach to ALM problems. This approach relies on scenario tree generation and variable discretization, which leads to several issues when dealing with too many variables or long time spans since the number of necessary scenarios grows exponentially. Other approaches, like robust optimization and genetic algorithms, are still studied in an ALM context, and this work presents one more option to be considered.

Departing from all the approaches described above, our proposed model considers a continuous state space and can handle virtually infinite periods. This naturally fits the structure of the random variables we are dealing with in our ALM problem.

Table 1 presents a comparison of the main features of each discussed algorithm.

5- Deep Reinforcement Learning Model for ALM

The main proposal in this dissertation is to solve an ALM problem with the selected reinforcement learning algorithm. To fulfill this purpose, the first thing to be done is to model the addressed problem as a Markov Decision Process, as stated in Section 3.1. For this, ALM elements were enlisted on Section 5.1, and then, mapped to MDP elements, in Section 5.2. The initial state of the problem is described in Section 5.3, and its transition model, in Section 5.4. Section 5.5 details the chosen reward function.

5.1- ALM Elements

The elements of an ALM problem are:

- *Asset*: the total amount of money available for investments at any given time. Its value at any observed time is a non-negative deterministic scalar, represented by A_t. At any future moment, it depends on investment's returns and liabilities, and so, is stochastic.
- *Liabilities*: a flow of future debts that have to be paid in their proper due dates. It can be deterministic, which is more unusual. Alternatively, it can have one or more stochasticity sources, like variable interest, insurance indemnities, or pension fund's payments. At any given time t, it is represented by $L_t = [L_1, L_2, \ldots, L_T]$.
- *Time horizon*: time until all debts are paid, represented by a discrete integer, from 1 to T. It can be deterministic (a loan), stochastic (lifelong pension fund), or even infinite (insurance company portfolio).
- *Available investments*: at any given time, this is the set of all investments that can be picked by the decision-maker. Each investment has its features, like expected return, risk level, and liquidity. Correlations among available investments, and between them and liabilities are of paramount importance.

- *Allocation*: how to split the assets among available investments. It is represented by an n-sized vector $a_t = [a_{1,t}, a_{2,t}, \ldots, a_{n,t}]$, where n is the total number of available investments, with $a_{i,t} \geq 0 \; \forall i \in 1, \ldots, n$ and $\sum_{i=1}^{n} a_{i,t} = 1$[1].

5.2- ALM Mapping to MDP

The elements of a Markov decision process (see Section 3.1), and their ALM mappings, are:

- *State*: at any time, a state will be composed of current assets and the liability flow until the end of the time horizon; so, it is a vector of size $T + 1$.
- *Action*: this is the investment's allocation made by the agent. It is a continuous n-sized vector.
- *Transition model*: the new asset will be given by current amount of money, updated by investment's returns, and subtracted from current liability's first element, as in Equation 3. New liability is current liability flow, without its first element (which has just been paid), and updated by a chosen index (usually, inflation). New assets and new liability form the next state.
- *Reward function*: several options can be used, like rewarding excessive assets at the end of the time horizon (surplus) or total periods with debts paid (longevity).

5.3- Initial State

The primary inputs of the problem are an initial total asset, a liability flow, and a data frame with the historical returns of available assets. An index used to update the liability, such as inflation, is added to the set of available assets, in order to be simulated together, respecting their correlations.

[1] The $a_{i,t} \geq 0$ constraint is a regulatory restriction typical of pension funds. Other problem setups, like insurance companies or banking may or may not have a similar constraint.

5.4- Transition Model

At every step, state s_t, which contains A_t and L_t, is observed. The input of the model will be the ratio L_t/A_t, which is obviously a T-sized vector. The agent, based on this input and in its deterministic policy μ_θ, selects an action $a_t = [a_{1,t}, a_{2,t}, \ldots, a_{n,t}]$, where $\sum_{i=1}^{n} a_{i,t} = 1$ (Equation 22). This vector a_t is multiplied by current assets A_t, and returned to the environment as an allocation.

$$a_t = \mu_\theta \left(\frac{L_t}{A_t} \right) \tag{22}$$

The environment then simulates current returns I_t for all available investments and liability's update index U_t, considering their correlations. There are several ways to do this simulation. In this work, it is made by a multivariate random normal, with mean vector and covariance matrix given by available asset's time series inputs, and so, returning an n-sized vector.

New total asset, A_{t+1}, is given by current total asset plus investments incomes, minus current step's liability, as stated in Equation 3. The incomes are the inner product of the action vector and return vector, added to 1.

New liability flow is given by current liability, with its first element removed (representing current debt's payment), and a 0 appended to its end (to keep T size constant). This new vector is then multiplied by chosen update index U_t. In mathematical terms:

$$\begin{aligned} I_t &\sim N(\mu, \Sigma) \\ A_{t+1} &= A_t \cdot \left(a_t \times (1 + I_t)' \right) - L_t[0] \\ L_{t+1} &= L_t[1:].\text{append}(0) \cdot U_t \end{aligned} \tag{23}$$

The identities seen in Equation 23 represent the transition distribution. As every variable at the next step relies solely on variables at the current step (and simulated results), it satisfies first-order Markov property.

5.5- Reward Function

In this work, a binary reward function is used. It returns 1 whenever new asset A_{t+1} is greater than 0, meaning that current liability was successfully paid, and 0 otherwise (Equation 24). As the algorithm tries to maximize the expectation of cumulative discounted rewards, this choices guarantees this model will try to pay as many debts as possible, and will not incur in unnecessary risks trying to accumulate more money than needed.

$$r_{t+1} = \begin{cases} 1 & \text{if } A_{t+1} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

5.6- Terminal State

A terminal state is reached whenever one of two conditions occur: (1) new asset A_{t+1} is equal to or lower than 0, meaning that we have run out of assets, or (2) total sum of liability's vector is equal to 0, meaning there are no more debts to be paid (Equation 25).

$$\text{Terminal} = \begin{cases} \text{True} & \text{if } (A_{t+1} \leq 0) \cup (\sum L_i = 0) \\ \text{False} & \text{otherwise} \end{cases} \tag{25}$$

5.7- Episode

An episode is defined as a single simulation rollout, from its initial to terminal state. We consider an episode *successful* whenever it reaches its terminal state due to the second condition (liability sum equal to 0), which means all debts were paid. As a result, the total sum of rewards is equal to maximum episode length T.

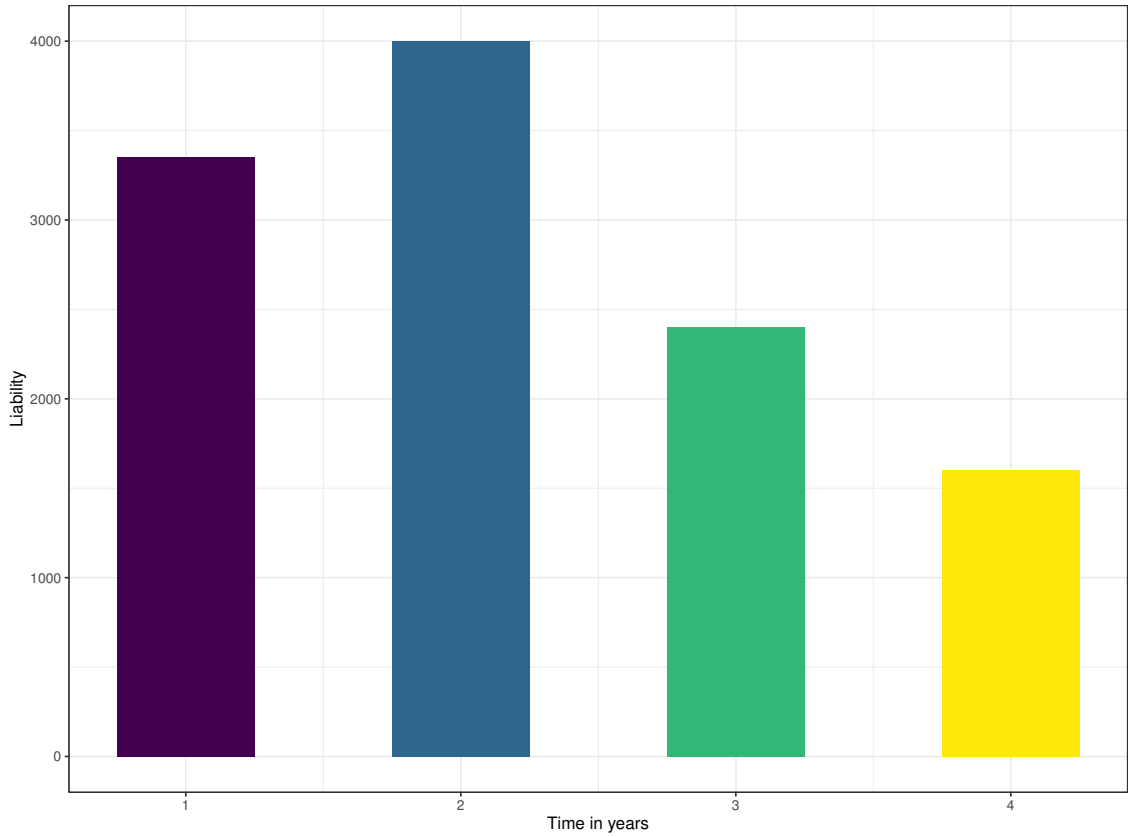

Figure 8 – Estimated liability flow for toy example.

5.8- Toy Example

To further clarify the proposal, we provide a toy example: suppose someone wants to pay for her children's graduation. The tuition fee is proportional to the number of classes taken in each year ($ 800 for each class), and the first payment is a year from now. According to her plans, the first year's debt will be $ 3,350 ($ 150 as an initial fee, plus four classes taken), and three next years will cost her $ 4,000, $ 2,400 and $ 1,600, for a total of a 4-years *time horizon*. These values are updated every year by the inflation index and can be seen in Figure 8. This is the *liability flow*.

So far, she has $ 10,000 in savings (her *initial assets*), which is less than the total $ 11,350 of her total future debts, but she believes she can invest her money for an average return of inflation plus 6% each year. Using this value as a discount rate, she calculates

her *liability's present value*, given by L_{PV}, with Equation 2, and finds the following result:

$$L_{PV} = \frac{3,350}{(1+0.06)} + \frac{4,000}{(1+0.06)^2} + \frac{2,400}{(1+0.06)^3} + \frac{1,600}{(1+0.06)^4} = 10,002.80 \quad (26)$$

Her *solvency rate* is pretty close to 1 (0.9997), which is a good sign. If she takes the right decisions, it is highly probable she will be able to honor her debts.

She has three *available investments* to chose: a pre-determined fixed-income bond, which returns a net 10% a year (BOND1); an inflation-based bond, returning inflation plus 5% a year (BOND2); and the stock market, with a 15% a year expected return, but with high volatility (STOCKS). Besides that, expected inflation for the next few years is 4.5%, with low volatility.

The initial asset and the liability flow compose the *initial state*. This state is observed by the parent, who takes the place of the *agent*. Available investments, their expected returns, and their volatilities are part of the *environment*.

$$s_0 = [10000, 3350, 4000, 2400, 1600].$$

In first year, the agent decides to use the following *allocation*: $ 2,000 in BOND1, $ 6,000 in BOND2, and $ 2,000 STOCKS. She knows the return from the inflation-based bond is less than she needs but thinks she can profit enough from other investments to cover the difference.

At the end of the year, BOND1 returned 10%, to a total amount of $ 2,200. Inflation was just 3% in the period, so, BOND2 returned 8% to a net value of $ 6,480. On the other hand, the stock market had a bad year, returning only 6%, for a total $ 2,120. Her updated asset is now $ 10,800, and after paying her first liability, she will have $ 7.450 to invest in her next year (new total asset). Her new liability flow is the old one, without its first element, and updated by a 3% inflation. So, it is now $ 4,120, $ 2,472 and $ 1,648. As the debt was paid, and she still has money at hand, the *reward* of this period is 1, and *terminal condition* is *False*, since debts are not over.

$$s_1 = [7450, 4120, 2472, 1648, 0]$$

In the second year, the agent believes the stock market will perform better, and then decides to allocate $ 1,450 in BOND1, $ 3,000 in BOND2 and $ 3,000 in STOCKS.

Inflation has a high peek, going to 6% in that year, but the stocks market returns a net 15%, proving his strategy was right. She now has a 10% return from BOND1, updating its value to $ 1,595, an 11% return from BOND2, going to $ 3,330, and STOCKS go up to $ 3,450. The total asset is now $ 8,375. The current debt of $ 4.120 is paid, and asset for the next year is $ 4,255. Liability flow is updated by 6%, and is now $ 2,620.32 and $ 1,746.88. The period's reward is 1 again, as well as terminal condition, which is *False*.

$$s_2 = [4255, 2620.32, 1746.88, 0, 0]$$

The agent now realizes that the present value of this new liability, when discounted by 5% is $ 4,080.15, which means that, thanks to adopted strategy, she can now allocate all her assets in BOND2, which no matter the inflation index, will return more than needed to pay liabilities. Her children's education is covered.

But, as investment returns are stochastic, the agent might not be so lucky. If in the first year, inflation were 5% and the stock market returned nothing, she would have the same $ 2,200 for BOND1 (as it is not subject to market volatility), but now BOND2 would have a net value of $ 6,600 while STOCKS would be stuck at $ 2,000. She would have **the same $ 10,800** updated asset, and of course, the same $ 7,450 after paying first liability. But now, updated liabilities would be $ 4,200, $ 2,520 and $ 1,680. Reward and terminal condition would still be 1 and *False*, respectively, but the new state now is:

$$s_1 = [7450, 4200, 2520, 1680, 0]$$

She can now follow the very same belief she had, and then again allocate $ 1,450 in BOND1 and $ 3,000 in both BOND2 and STOCKS. But her bad luck is not gone, and inflation went up, to 7%, while the stocks market returned 8%, which is better than the last year, but still under the expected return. Her investment's updated values are now $ 1,595, $ 3,360 and $ 3,240, adding to $ 8,195. New asset, after liability is paid, is now $ 3,995, and updated liabilities are $ 2,696.40 and $ 1,797.60. Reward and terminal condition are 1 and *False* again, while new state is:

$$s_2 = [3995, 2696.40, 1797.60, 0, 0]$$

Liability's present value, for a 5% discount rate, is now $ 4,198.48, which is greater than the current asset. She cannot allocate all her assets in BOND2, like the last example.

Actually, she has to take more risk, as her solvency rate is below 1. She then decides to allocate $ 495 in BOND1, $ 1,500 in BOND2 and $ 2,000 in STOCKS. But, in this year, both inflation and stocks returned only 5%, making her investments worth of $ 544.50, $ 1,650 and $ 2,100, for a total amount of $ 4,294.50. After paying current liability, she is left with only $ 1,598.10, while the updated liability is $ 1,887.48.

$$s_3 = [1598.10, 1887.48, 0, 0, 0]$$

She will have to allocate all her assets in STOCKS, and expect it to return over 18% this year, or else, she'll have no money to pay next year's liability, leading to a reward of 0. Anyhow, next year's terminal condition will be *True*, either due to no more liabilities, if she pays it all, or no more assets, if she fails to pay it.

6- Experiments

In order to validate our proposed model, several computer simulations were devised. Details of the code implementation for the proposed algorithm and the custom environment are described in Section 6.1. The simulated data generation process is detailed in Section 6.2, describing the inputs of the environment, and a data frame containing available investment's time series. Experimental settings of the algorithm are in Section 6.3. The outputs of the training sessions can be seen in Section 6.4, while Section 6.5 shows some simulations with trained policies.

6.1- Code

The code used in the experiments is composed of two parts: the algorithm itself, and the custom environment. For the algorithm, an educational Python package known as Spinning Up has been used, with a few minor changes. The ALM environment has been created from scratch, using tools provided by the gym package.

6.1.1 Spinning Up

The OpenAI group [1] is an AI research and deployment company, according to themselves. They have developed many artificial intelligence-related resources in several fields, like deep learning, generative adversarial networks, transfer learning, and reinforcement learning.

One of these resources is Spinning Up [2], which is a Python package aimed at the study of various reinforcement learning algorithms. The version used in this work

[1] https://openai.com/

[2] https://spinningup.openai.com/en/latest/

supports six different algorithms (DDPG among them), with two implementations for each one: TensorFlow (our choice) or PyTorch. The following minor changes have been made to its code:

- The output activation function for both actor and target-actor networks has been changed, from *tanh* to *softmax*, since it has the exact output shape the problems needs (normalized, non-negative vector)
- The random noise added to the action in the training phase has been changed to a multiplicative one. A multivariate normal, with mean vector 1, and an identity covariance matrix multiplied by the noise parameter τ, is multiplied by the action. The resultant vector is then normalized (since the total sum of the action vector has to be 1).

SpinningUp package has been chosen because it is, according to their own site, *"an educational resource produced by OpenAI that makes it easier to learn about deep reinforcement learning"*[3]. It has tools which help the analysis of the whole training process, and not just the results, like other packages.

6.1.2 Gym

Gym [Brockman et al., 2016], similar to Spinning Up, is an OpenAI python package, which provides tens of ready-to-use reinforcement learning environments and an extensive toolkit to create custom ones. The creation of a new environment requires:

- *Input data* for the environment, which will define: (1) the problem's initial state (initial asset and liability vector, described in Section 6.2.1) and; (2) its transition model (available investment's time series, detailed in Section 6.2.2).
- *Definitions* for action and state space's domains.
- *A 'step' method*, used for interactions between agent and environment. This method is responsible for providing an observation to the agent, receiving its action, and

[3] `https://spinningup.openai.com/en/latest/user/introduction.html`

calculating new asset and liabilities, as well as returning the reward, and a Boolean value indicating if the agent reached a terminal state or not. This method is further detailed in Figure 9.

- *A 'reset' method*, used to return the environment to its initial state after an episode reaches a terminal condition.

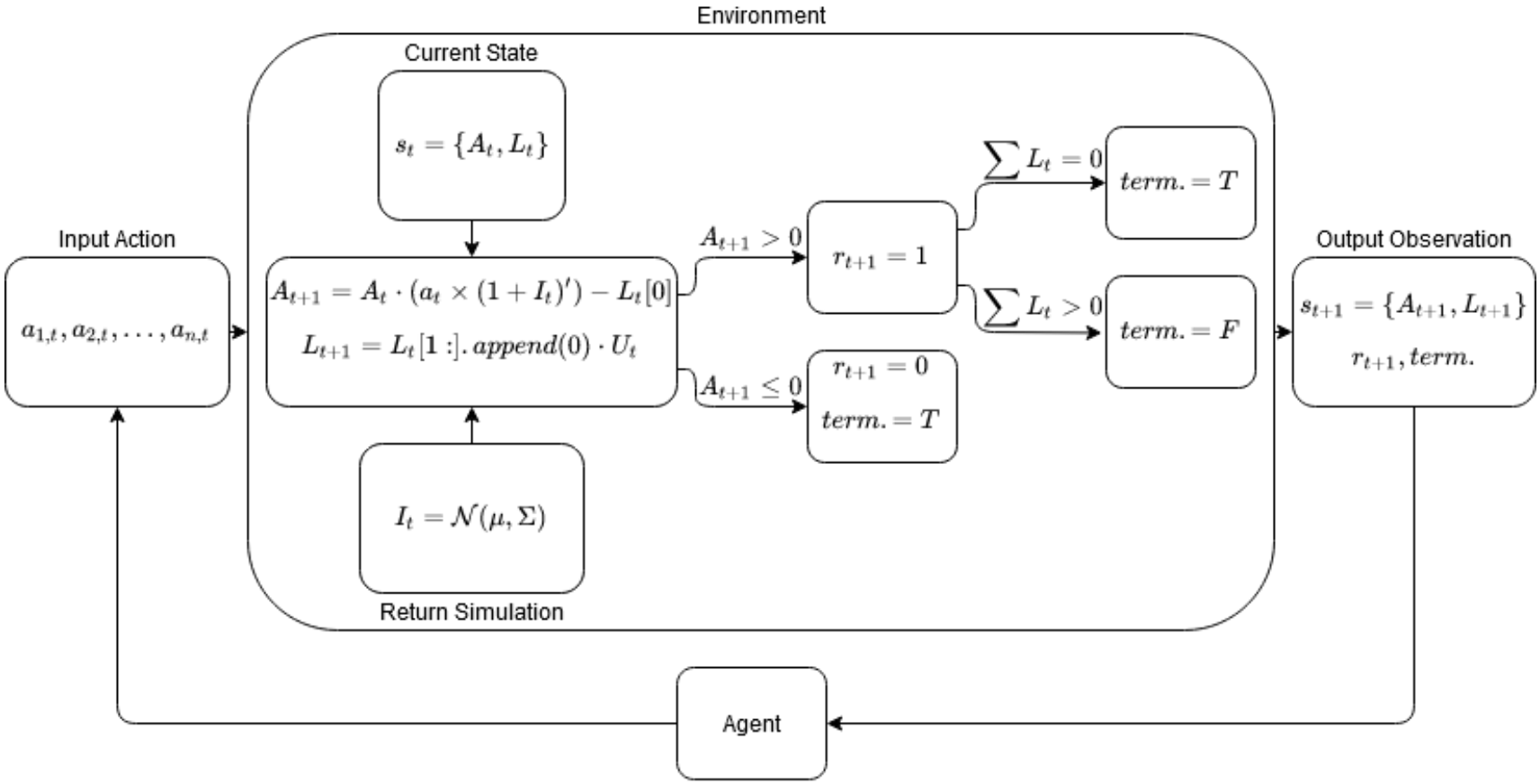

Figure 9 – Step method of the custom environment.

6.2- Simulated Data

Financial data usually is sensitive. Institutions do not share how they estimate their future liabilities, data about their customers and pensioners, how much money they have, among other confidential information. Due to this (necessary) secrecy, simulated data had to be used in this work, for both assets and liabilities. This data is described in Section 6.2.1. On the other hand, indexes yearly returns are easy to find on the internet, and the time series we used are real ones (source: `https://www3.bcb.gov.br/sgspub/localizarseries/localizarSeries.do?method=prepararTelaLocalizarSeries`)

6.2.1 Assets and Liabilities

Fifteen asset/liability setups have been created. In all of them, there is an initial asset of $ 1,000,000. For the liabilities, fifteen flows have been simulated: three groups, with 30, 50, and 80 years time horizons, and within each group, discount rates of 6.0%, 6.5%, 7.0%, 7.5%, and 8.0% were used[4]. In all setups, liability's present value is also $ 1,000,000, to match initial asset, making the initial solvency rate always 1.

Flow's shapes are similar to the chi-squared distribution, as can be seen in Figure 10, to simulate the behavior of a pension fund that has not reached its maturity yet. Debts are smaller now than they will be in the future (when more participants will retire), and after that, they will slowly shrink to zero. The chosen shape is arbitrary, and other options could be used.

6.2.2 Available Investments

Six investments are available to the decision-maker, each one representing a major asset group: BRL-USD exchange rate, iBovespa (stocks market), IRF-M (predetermined

[4]The higher the discount rates, the greater is the performance investments need to provide.

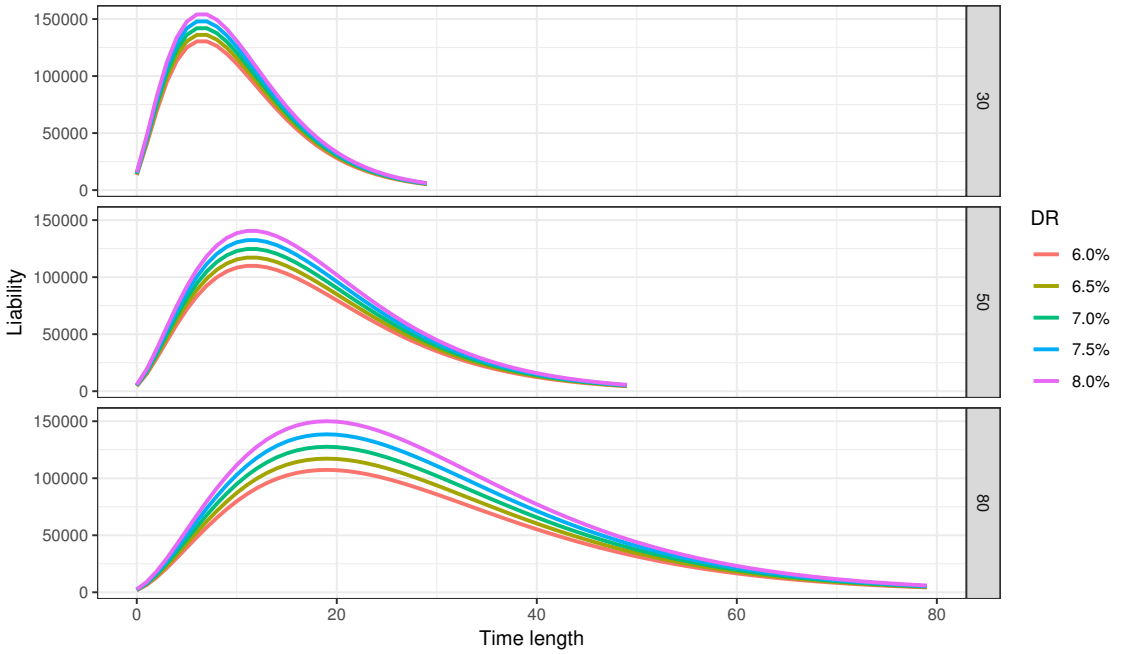

Figure 10 – Liabilities modeled as proportionate to chi-squared distributions. (a) over 30 years; (b) over 50 years; (c) over 80 years.

fixed income), IMA-S (SELIC based fixed income), IMA-B 5 (short term inflation based fixed income) and IMA-B 5+ (long term inflation based fixed income). Besides these indexes, IPCA (inflation index) has been added to update liabilities. Yearly returns from 2005 to 2018 have been used, since not all series are available for year 2004 and before.

6.3- Experimental Settings

The actor is a fully-connected deep neural network, with input layer the size of the respective liability's length T (30, 50, or 80). Two hidden layers have 400 and 300 neurons each (both with ReLU activation). The output follows a softmax pattern, with size $n = 6$, according to the number of available investments. The critic has a similar structure but with input size $n + T$ (i.e., 6 plus 30, 50 or 80), and a scalar output, with identity activation. Target actor and critic networks, by design, follow the same configuration of their counterparts.

Action's noise parameter introduced in training, to enforce exploration of the state space, is a random multivariate normal vector multiplied by the actor's output, as stated

in Section 6.1.1. An additive parameter would lead to negative values, which is out of the action's domain. It is distributed as a multivariate normal, with mean vector one, and covariance $\tau = 0.01$ times an identity matrix (higher values would cause too much disturbance, making the training more erratic). The resulting action is then normalized. Although more testing is needed, we suspect that the relatively high variance observed during training is mainly due to this parameter.

Reward's discount factor γ was set to 0.99; the actor and critic's learning rate are 10^{-3}; ρ parameter for the target network's updates is 0.995; batch size is 100. Replay buffer maximum size is 10^6, and its first ten thousand step's actions are completely random, following a Dirichlet distribution with all parameters equal to 1 (a multivariate version of a standard uniform distribution, which was used in the original code). In each simulation, 500 epochs have been run, with a total of five thousand steps per epoch. All the other hyperparameters, except actions and state's sizes and noise distribution, are the Spinning Up default values.

Training sessions were run in a personal computer, equipped with an Intel Core i7 7600 processor and 16Gb of RAM, running Ubuntu 18.04 and Python 3.6. A single training took about four hours to complete, and five concurrent training sessions took up to fifteen hours. The complete code and result logs can be found at `https://github.com/afontoura/DRLA2ALM`.

6.4- Outputs

Figures 11, 12 and 13 show average episode return evolution for 30, 50 and 80 years groups, respectively. It is important to emphasize these returns are calculated with the training policies; that is, the noise parameter is included in the actions taken. This fact by itself raises the average return's volatility.

Note that it is easy to see (Figure 11) that all 30 years of simulations converged with no problems, and three did it at pretty early epochs.. Besides having a very similar behavior, the 50 years group (12) had some unstable periods, particularly for liabilities with 6.5% and 8.0% discount rates, although it was not enough to compromise final results.

As expected, the 80 years group is very unstable when compared to the previous

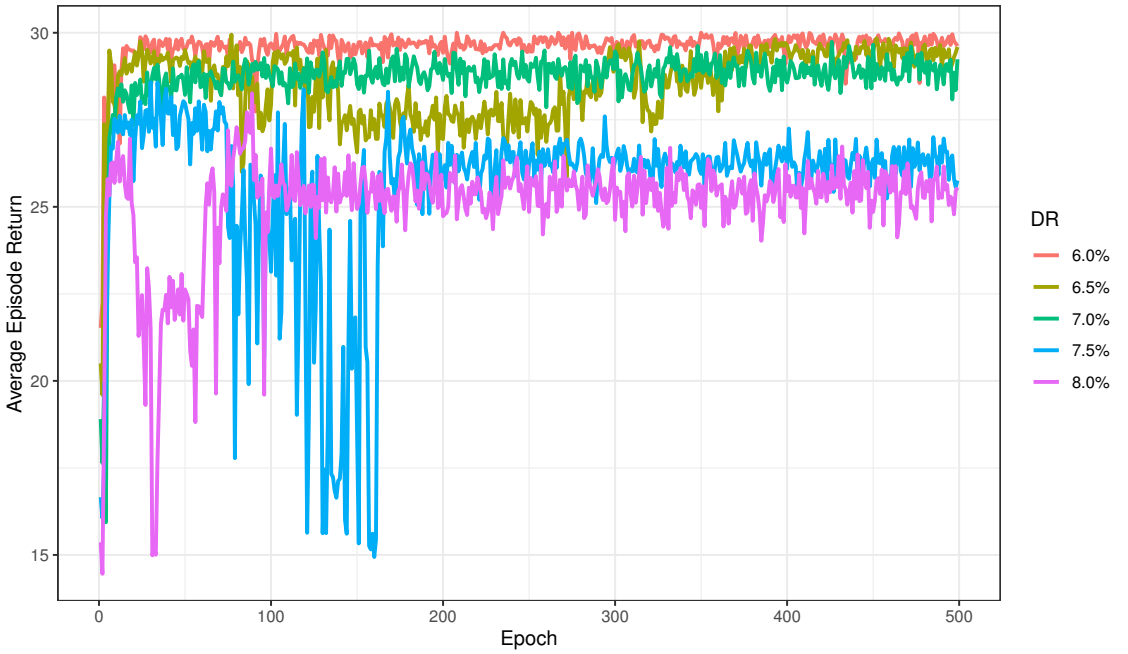

Figure 11 – Average episode return for 30-years simulations.

two (13). While liabilities with 6.0%, 6.5%, and 7.0% discount rates converged pretty fast (with a small instability for the second one), simulations with 7.5% and 8.0% had more unstable periods, and they probably should benefit from a more extended period of training.

6.5- Policy Testing

After training sessions were complete, a thousand simulations were run with each policy. In these simulations, the policy is deterministic - the noise parameter is no longer used. Results are presented in Table 2. In each line, we have the percentile of simulations with maximum return, the minimum return, average return among all simulations, and average return among simulations that did not reach maximum return.

Percentile of maximum return is over 90% in 8 out of 15 scenarios. The worst result is 52.6%, which is still pretty good, considering an 80 years time horizon and 7.5% discount rate. Interestingly, in all time horizon groups, the simulation with 7.5% discount rate performed worse than the one with 8.0%. This behavior is also found in average total return, except for a 30-years horizon.

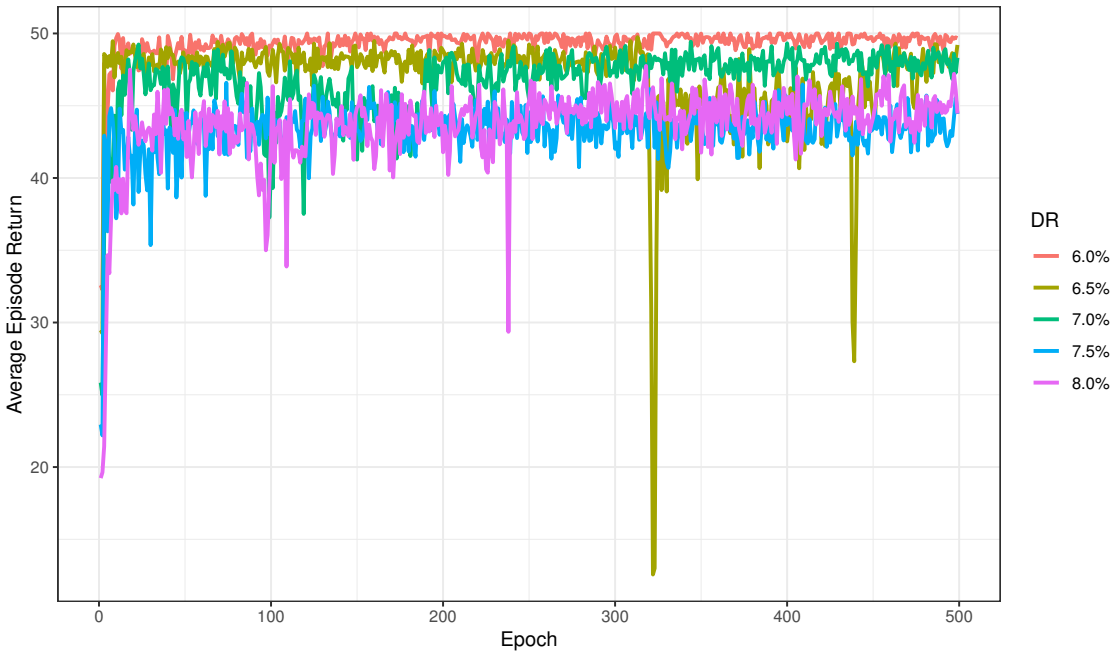

Figure 12 – Average episode return for 50-years simulations.

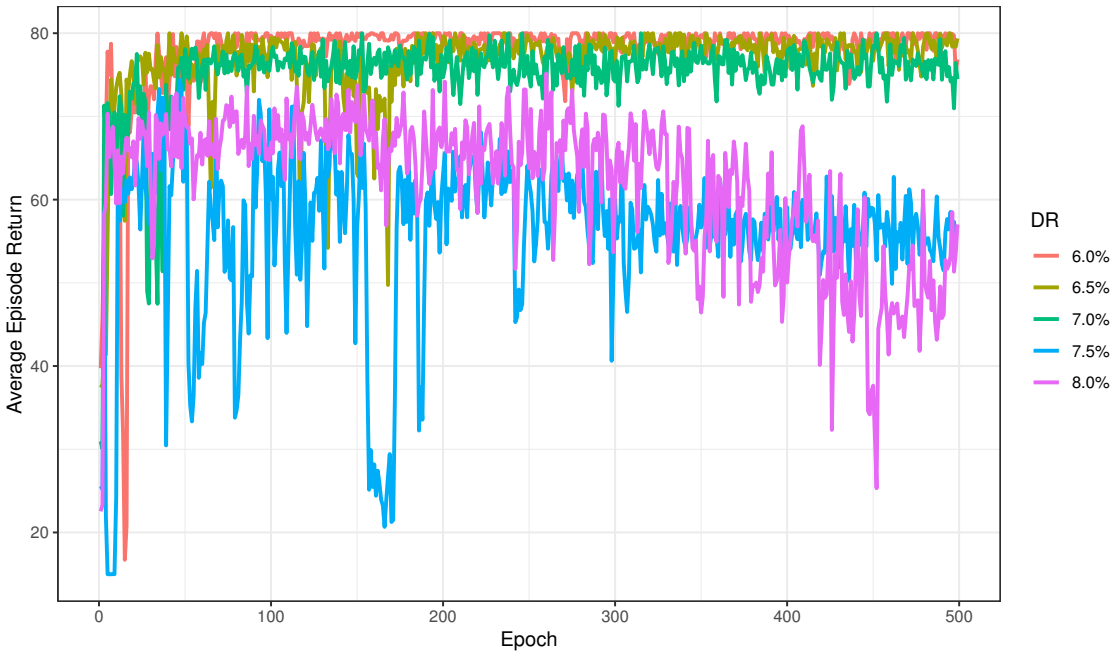

Figure 13 – Average episode return for 80-years simulations.

Table 2 – Simulation results.

Horizon	Discount Rate	% Max. Return	Min. Return	Avg. (Total)	Avg. (Not Max.)
30	6.0%	96.6%	11	29.654	19.824
	6.5%	93.9%	11	29.551	22.639
	7.0%	93.4%	9	29.199	17.864
	7.5%	67.7%	12	26.241	18.362
	8.0%	71.2%	7	25.408	14.056
50	6.0%	99.0%	25	49.815	31.500
	6.5%	93.3%	16	48.530	28.060
	7.0%	92.4%	12	48.166	25.868
	7.5%	77.7%	11	43.370	20.269
	8.0%	84.7%	6	45.118	18.092
80	6.0%	84.2%	24	76.611	58.551
	6.5%	94.6%	16	78.113	45.056
	7.0%	94.2%	17	77.278	33.069
	7.5%	52.6%	17	57.056	31.595
	8.0%	70.1%	12	64.832	29.271

Figure 14 shows the distribution of non-maximal simulations. The height of each column is the number of simulations with a given return. We can see that, for the three first discount rate levels, they are pretty flat (except for the 80-years / 6.0% discount rate). In the other two, they concentrate on the worst reward levels.

One of the most critical decisions for a reinforcement learning algorithm when modeling a problem is how to set its reward function since it defines what will be optimized. Present work maximizes the number of periods in which the decision-maker will be able to honor his debts. According to this principle, results are considered very good.

6.6- Comparisons

To better analyze the performance of the proposed model, its results were compared to those of two other models: a straightforward uniform portfolio, where assets were distributed equally among all available investments, and another one, using Mean-Variance Optimization.

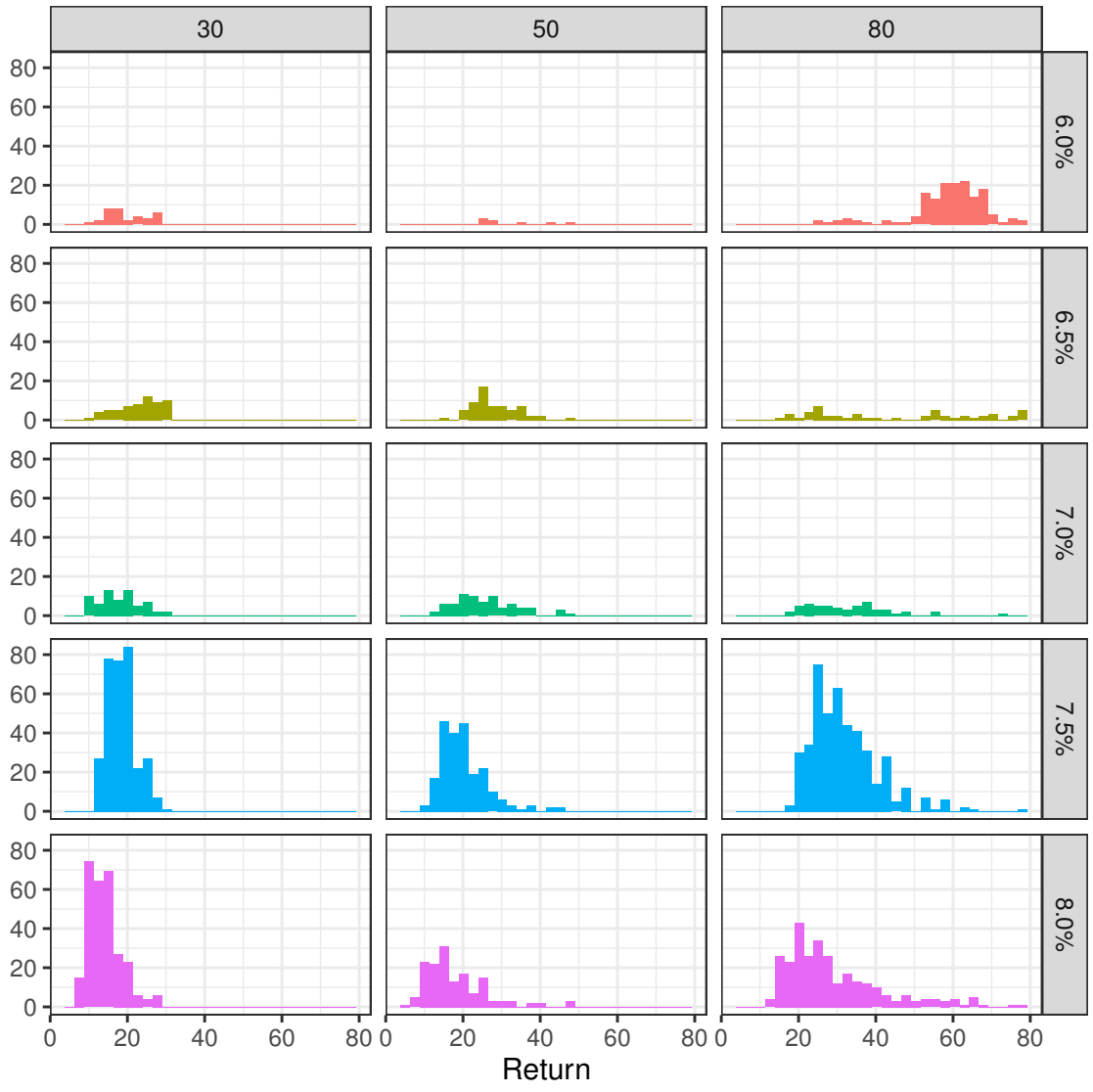

Figure 14 – Non-maximal simulations.

Table 3 – Mean-Variance Optimization Portfolios.

DR	Câmbio	Bovespa	IRF-M	IMA-S	IMA-B 5	IMA-B 5+	$\mathbb{E}(R)$	Vol.
6.00%	7.21%	2.52%	6.60%	48.60%	35.07%	0.00%	11.87%	2.73%
6.50%	2.59%	0.00%	3.75%	43.09%	50.58%	0.00%	12.39%	2.98%
7.00%	0.05%	0.00%	4.97%	27.08%	67.90%	0.00%	12.92%	3.40%
7.50%	0.00%	0.00%	11.92%	0.00%	87.28%	0.80%	13.45%	4.05%
8.00%	0.00%	0.00%	0.00%	0.00%	81.20%	18.80%	13.98%	5.39%

6.6.1 Selected Portfolios

The first portfolio, as mentioned before, has approximately 16.67% of total assets in each of the six available investments. Its expected return and volatility are 10.90% and 2.78%, respectively.

For the MVO model, five different portfolios have been created, one for each discount rate used in the liabilities. This was done because, for higher levels of debt (the greater discount rates), we need portfolios with higher expected returns, therefore, greater risk. The expected returns used to find these portfolios were the average IPCA[5] of the period - 5.536% - plus the discount rate (from 6.0% to 8.0%). After defining these expected returns, investment weights were calculated by minimizing the resulting portfolio variance, trough a simple quadratic programming formulation:

$$\begin{aligned} & \min_{w} \frac{1}{2} w^T \cdot \Sigma \cdot w \\ \text{subject to: } & \mu^T \times w = \mathbb{E}(R) \\ & w_i \geq 0, \ \forall i \in 1, \cdots, n \\ & \sum_{i=1}^{n} w_i = 1 \end{aligned} \tag{27}$$

where w is the vector with the portfolio weights, μ and Σ are the mean vector and the covariance matrix of the investment's historical returns respectively, and $\mathbb{E}(R)$ is the resulting portfolio's expected return. The resulting portfolios can be seen in Table 3.

The uniform portfolio and the five MVO portfolios were exposed to the same simulation scenario used by the proposed model, and the results were as follows:

[5]The chosen update index

Table 4 – Uniform portfolio results.

Horizon	Discount Rate	% Max. Return	Min. Return	Avg. (Total)	Avg. (Not Max.)
30	6.0%	62.5%	18	28.210	25.227
	6.5%	51.6%	16	27.290	24.401
	7.0%	37.5%	17	26.171	23.874
	7.5%	23.3%	16	24.615	22.979
	8.0%	14.3%	15	23.238	22.110
50	6.0%	62.7%	19	43.510	32.601
	6.5%	44.2%	19	39.714	31.566
	7.0%	28.7%	18	35.650	29.874
	7.5%	15.3%	16	31.113	27.701
	8.0%	08.3%	16	27.186	25.121
80	6.0%	61.0%	20	62.668	35.559
	6.5%	43.1%	18	53.826	34.000
	7.0%	26.2%	16	43.386	30.388
	7.5%	16.8%	16	37.158	28.507
	8.0%	08.0%	16	30.771	26.490

6.6.2 Uniform Portfolio Results

Table 4 displays the simulation results for the uniform portfolio. In all simulation setups, the percentage of simulations with the maximum return is clearly below those in the reinforcement learning proposed model, as expected. A uniform portfolio is a very naive approach, serving only as a basic benchmark.

Figure 15 displays the distributions of all non-maximal simulations of this portfolio. All the histograms show a similar pattern: a peak of simulations in early steps (usually, between 20 and 30), and a slow decay after that. This concentration of total returns in early periods of time happens because this portfolio is not dynamic. Therefore, it does not change its allocation strategy according to its results, meaning that when it has bad returns in early years, it won't take more risk, trying to recover its losses.

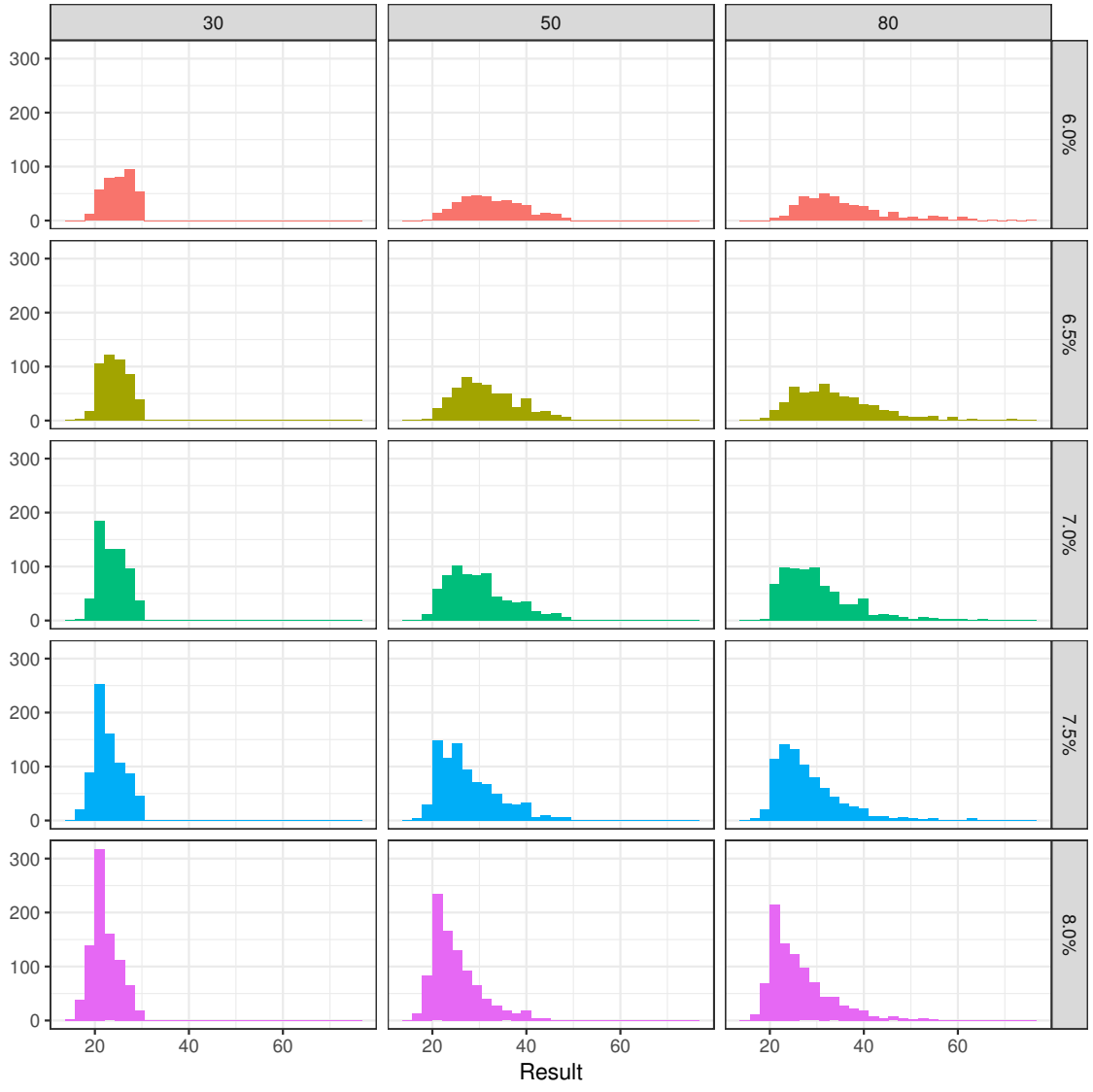

Figure 15 – Non-maximal simulations - Uniform portfolio.

Table 5 – MVO portfolio results.

Horizon	Discount Rate	% Max. Return	Min. Return	Avg. (Total)	Avg. (Not Max.)
30	6.0%	76.9%	22	29.366	27.255
	6.5%	72.4%	21	29.135	26.866
	7.0%	70.1%	21	28.919	26.385
	7.5%	63.8%	20	28.438	25.685
	8.0%	60.4%	15	27.823	24.503
50	6.0%	68.4%	27	46.379	38.541
	6.5%	67.3%	22	45.754	37.015
	7.0%	65.8%	24	44.917	35.137
	7.5%	64.3%	22	44.106	33.490
	8.0%	57.0%	19	41.683	30.658
80	6.0%	67.9%	26	67.914	42.349
	6.5%	66.3%	25	66.842	40.955
	7.0%	61.3%	23	63.644	37.736
	7.5%	58.3%	21	61.575	35.815
	8.0%	53.9%	20	57.750	31.735

6.6.3 MVO Portfolio Results

In Table 5, we can see MVO portfolio's results. Percentage of simulations with maximum returns are better than those from the uniform portfolio, but still below the results of the reinforcement learning model. For a long time, this approach has been used to solve ALM problems (and is still used nowadays for simpler portfolio optimization). So, an average-to-good performance was expected.

Figure 16 shows that non-maximal simulations' behavior is similar to those of the uniform portfolio, but their peaks shift to the right (between 30 and 40).

6.6.4 Results Comparison

For a cleaner comparison of the results, an analysis of each one of the metrics is presented.

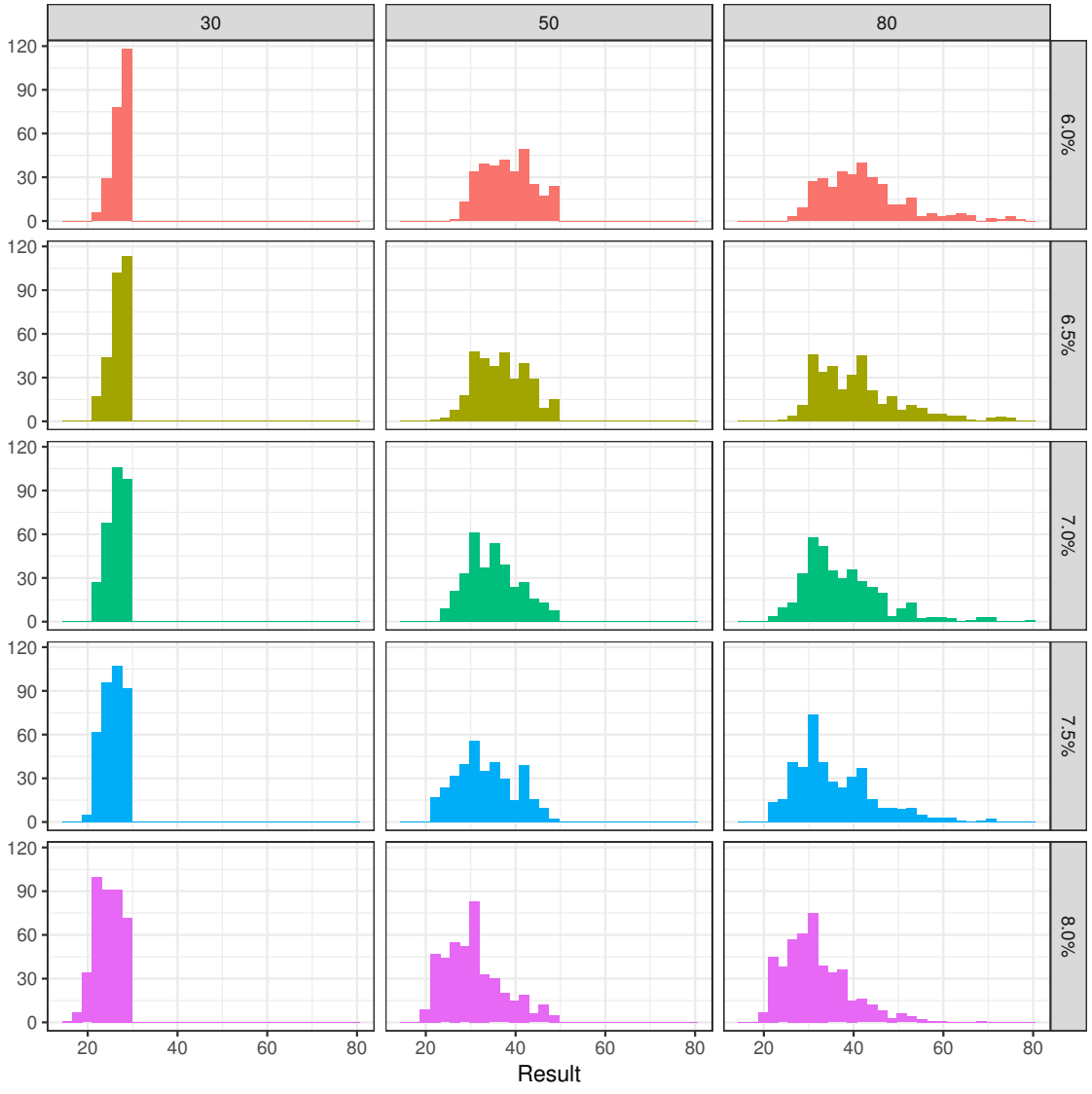

Figure 16 – Non-maximal simulations - MVO portfolio.

Table 6 – Percentage of maximum returns.

Horizon	Discount Rate	Uniform	MVO	DDPG
30	6.0%	62.5%	76.9%	96.6%
	6.5%	51.6%	72.4%	93.9%
	7.0%	37.5%	70.1%	93.4%
	7.5%	23.3%	63.8%	67.7%
	8.0%	14.3%	60.4%	71.2%
50	6.0%	62.7%	68.4%	99.0%
	6.5%	44.2%	67.3%	93.3%
	7.0%	28.7%	65.8%	92.4%
	7.5%	15.3%	64.3%	77.7%
	8.0%	08.3%	57.0%	84.7%
80	6.0%	61.0%	67.9%	84.2%
	6.5%	43.1%	66.3%	94.6%
	7.0%	26.2%	61.3%	94.2%
	7.5%	16.8%	58.3%	52.6%
	8.0%	08.0%	53.9%	70.1%

Percentage of maximum returns

Table 6 shows, side by side, the main metric of the reinforcement learning proposed model: the percentage of maximum returns. It is considered the primary metric because, as stated in Section 5.5, the reward function of this algorithm was constructed to maximize the number of steps in which the liability was paid. Therefore, more simulations with maximum return mean better training of the algorithm.

According to the metric of the percentage of maximum returns, It is pretty obvious as noted in Table 6 that DDPG had an outstanding performance when compared to the other models, beating them in 14 out of 15 simulation setups. The best result from the other two algorithms is a 76.9% of the MVO portfolio, for 30 years time horizon and 6.0% discount rate. Nothing less than 11 results from DDPG are above this number, including seven above 90.0%,

Table 7 – Minimum returns.

Horizon	Discount Rate	Uniform	MVO	DDPG
30	6.0%	18	22	11
	6.5%	16	21	11
	7.0%	17	21	9
	7.5%	16	20	12
	8.0%	15	15	7
50	6.0%	19	27	25
	6.5%	19	22	16
	7.0%	18	24	12
	7.5%	16	22	11
	8.0%	16	19	6
80	6.0%	20	26	24
	6.5%	18	25	16
	7.0%	16	23	17
	7.5%	16	21	17
	8.0%	16	20	12

Minimum returns

The minimum return are showed in Table 7. This metric is better for the MVO technique since it is the best in all 15 setups (one tie with the uniform portfolio).

As DDPG is a dynamic allocation algorithm, when the first few time steps have bad returns, it will try to reallocate the assets, incurring in more risk, in order to improve the expected return. Sometimes, this increase in risk will pay off, and the simulation will reach maximum return, or something close to it. Nevertheless, when the additional risk does not increase the returns, the algorithm will try to increase the risk more and more, until it runs out of assets. This behavior is the reason why, when investments go wrong, they go *really* bad.

MVO, on the other hand, is a static allocation model, which will not change its strategy, no matter the results. So, the percentage of maximum returns is lower because it will not correct its course when the first year's returns are low, and its assets will not go below 0 as fast as the dynamic DDPG model.

Table 8 – Average returns - Total.

Horizon	Discount Rate	Uniform	MVO	DDPG
30	6.0%	28.210	29.366	29.654
	6.5%	27.290	29.135	29.551
	7.0%	26.171	28.919	29.199
	7.5%	24.615	28.438	26.241
	8.0%	23.238	27.823	25.408
50	6.0%	43.510	46.379	48.815
	6.5%	39.714	45.754	48.530
	7.0%	35.650	44.917	48.166
	7.5%	31.113	44.106	43.370
	8.0%	27.186	41.683	45.118
80	6.0%	62.668	67.914	76.611
	6.5%	53.826	66.842	78.113
	7.0%	43.386	63.644	77.278
	7.5%	37.158	61.575	57.056
	8.0%	30.771	57.75	64.832

Average returns - Total

Table 8 show the average returns total. Note that DDPG beats MVO in 11 out of 15 setups, which is expected, since DDPG has the greatest amount of simulations with maximum return, which increases its total average. The four setups where MVO beats DDPG are among the five worse DDPG setups (the only exception is the 80 years, 8.0% discount rate setup).

Average returns - Non Maximal

At last, the average returns of non maximal simulations are shown in Table 9. Just as the minimum return metric, the MVO model has a good performance when it comes to simulations that not reach maximum return. As mentioned in Section 6.6.4, non-maximal simulations of DDPG model are worse than the others, due to its dynamic allocation nature. This phenomenon is reflected in this metric as well.

Table 9 – Average returns - Non Maximal.

Horizon	Discount Rate	Uniform	MVO	DDPG
30	6.0%	25.227	27.255	19.824
	6.5%	24.401	26.866	22.639
	7.0%	23.874	26.385	17.864
	7.5%	22.979	25.685	18.362
	8.0%	22.110	24.503	14.056
50	6.0%	32.601	38.541	31.500
	6.5%	31.566	37.015	28.060
	7.0%	29.874	35.137	25.868
	7.5%	27.701	33.490	20.269
	8.0%	25.121	30.658	18.092
80	6.0%	35.559	42.349	58.551
	6.5%	34.000	40.955	45.056
	7.0%	30.388	37.736	33.069
	7.5%	28.507	35.815	31.595
	8.0%	26.490	31.735	29.271

6.6.5 Discussion

The chosen reinforcement learning model (DDPG) had a good performance, beating the two benchmark algorithms in the metrics it was designed to excel: to pay as many years of liabilities as possible. Its dynamic allocation nature, as opposed to the others, makes it perform worse when it has bad returns in the first few years, although its overall performance is worth it.

7- Conclusions

An ALM problem consists in optimizing a portfolio in order to pay a future flow of debt. It is highly stochastic, since the return on investments and the values and duration of the liability are not known in advance. The number of investments available to the decision maker and the total amount of time periods are of paramount importance, since too many variables can make the problem intractable. Thanks to this, most common approaches to this problem involve scenario discretization, which is not good.

7.1- Retrospective Analysis

In this work, common concepts of ALM have been introduced, and fifteen variations of a basic formulation have been solved, using an algorithm known as Deep Deterministic Policy Gradient.

An implementation of the DDPG algorithm provided by the OpenAI SpinningUp package was used, with some minor changes, as well as a customized environment created especially for this work, using the OpenAI Gym package.

The results of the proposed algorithm were then compared with the results of more simple approaches present in literature. Such comparisons corroborate the hypothesis stated in Section 1.3, showing that the reinforcement learning framework is a reliable alternative, worth of research, and with a significant advantage: there is no need to use scenario discretization.

7.2- Contributions

The feature of being able to work with continuous state spaces in the ALM task is one of the main contributions of this dissertation. The very modeling of an ALM problem

as a Markov Decision Process to be solved with a Deep RL algorithm is another, since, to the best of author's knowledge, this has never been done before.

7.3- Future Work

This present work is just a first approach at Deep Reinforcement Learning for Asset-Liability Management problems, and for sure, there is plenty of room for improvement.

For the environment, the next natural step would be simulating stochastic liabilities, in values or time horizons, which would require more in-depth research in actuarial modeling. Scenarios other than a pension fund could be used, which could lead to a problem with an infinite time horizon (like an insurance portfolio, for example).

Another possible future work is to investigate changes in the reward function detailed in Section 5.5 for handling short term liquidity. One possibility is to label one of the available investments as a 'highly liquid investment'. If its allocation is less than the current liability, the agent is penalized. This should be used to simulate the need to have cash available to pay current debts, which is more realistic.

Another idea related to the reward is not to use a binary function. An option is to use surpluses and deficits as rewards and penalties or use just deficits as penalties, to avoid unnecessary risks related to rewarding a surplus.

Introducing reallocation costs and adding regulatory constraints, such as solvency rates and asset limitations, would bring the model closer to reality, which would undoubtedly be a good improvement.

A stochastic policy gradient algorithm can be used to estimate investment reallocation bandwidths. In practice, the decision-maker cannot reallocate his investments every day; therefore, a confidence interval for investments is better than a point estimate.

A routine to find an optimal set of hyperparameters is needed since no fine-tuning has been made in this work, and such parameters have a significant impact on the outcome.

The whole RL literature has plenty of other algorithms that can address this problem and should be tested.

7.4- Derivative Work

A partial version of this work has been submitted to the Brazilian Conference on Intelligent Systems 2019, as a paper entitled 'A Deep Reinforcement Learning Approach to Asset-Liability Management'. The paper has been accepted after a double-blind peer review and presented on October 16, 2019: [Fontoura et al., 2019]. It can be found in `https://www.researchgate.net/profile/Alan_Fontoura/research`.

Bibliography

Aouni, B., Doumpos, M., Pérez-Gladish, B., and Steuer, R. E. (2018). On the increasing importance of multiple criteria decision aid methods for portfolio selection. Journal of the Operational Research Society, 69(10):1525–1542.

Aro, H. and Pennanen, T. (2017). Liability-Driven Investment in Longevity Risk Management, pages 121–136. Springer International Publishing, Cham.

Brinson, G. P., Hood, L. R., and Beebower, G. L. (1986). Determinants of portfolio performance. Financial Analysts Journal - FINANC ANAL J, 42:39–44.

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

Cariño, D. R., Kent, T., Myers, D. H., Stacy, C., Sylvanus, M., Turner, A. L., Watanabe, K., and Ziemba, W. (1994). The russell-yasuda kasai model: An asset-liability model for a japanese insurance company using multistage stochastic programming. Interfaces, 24:29–49.

Cariño, R., Myers, D. H., and Ziemba, W. (1998). Concepts, technical issues, and uses of the russell-yasuda kasai financial planning model. Operations Research, 46:450–462.

Consigli, G., Moriggia, V., Benincasa, E., Landoni, G., Petronio, F., Vitali, S., di Tria, M., Skoric, M., and Uristani, A. (2018). Optimal Multistage Defined-Benefit Pension Fund Management, pages 267–296. Springer International Publishing, Cham.

de Almeida, J. F. C. R. (2016). Genetic algorithms applied to asset & liability management. Master's thesis, NOVA Information Management School.

de Oliveira, A. D., Filomena, T. P., Perlin, M. S., Lejeune, M., and de Macedo, G. R. (2017). A multistage stochastic programming asset-liability management model: an application to the brazilian pension fund industry. Optimization and Engineering.

Defourny, B., Ernst, D., and Wehenkel, L. (2011). Multistage stochastic programming: A scenario tree based approach to planning under uncertainty. LE, Sucar, EF, Morales,

and J., Hoey (Eds.), Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions. Hershey, Pennsylvania, USA: Information Science Publishing.

Dermine, J. (2008). Alm in banking. In Handbook of Asset and Liability Management - Set. North Holland.

Dhrymes, P. J. (2017). Portfolio Theory: Origins, Markowitz and CAPM Based Selection, pages 39–48. Springer International Publishing, Cham.

Duarte, T. B., Valladão, D. M., and Álvaro Veiga (2017). Asset liability management for open pension schemes using multistage stochastic programming under solvency-ii-based regulatory constraints. Insurance: Mathematics and Economics, 77:177 – 188.

Fama, E. F. (1975). Multiperiod Consumption-Investment Decisions. In Stochastic Optimization Models in Finance. World Scientific Pub Co Inc.

Fontoura, A., Haddad, D., and Bezerra, E. (2019). A deep reinforcement learning approach to asset-liability management. In 2019 8th Brazilian Conference on Intelligent Systems (BRACIS), pages 216–221.

Gulpinar, N. and Pachamanova, D. (2013). A robust optimization approach to asset liability management under time-varying investment opportunities. Journal of Banking & Finance, 37:2031.

Haneveld, W. K. K., Streutker, M. H., and van der Vlerk, M. (2010). An alm model for pension funds using integrated chance constraints. Annals OR, 177:47–62.

Hilli, P., Koivu, M., Pennanen, T., and Ranne, A. (2007). A stochastic programming model for asset liability management of a Finnish pension company. Annals of Operations Research.

Huang, C. F. (2012). A hybrid stock selection model using genetic algorithms and support vector regression. Applied Soft Computing Journal.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML.

Kaplan, P. D. (2017). From markowitz 1.0 to markowitz 2.0 with a detour to postmodern portfolio theory and back. The Journal of Investing, 26(1):122–130.

Klein Haneveld, W. K., editor (1986). Duality in Stochastic Linear and Dynamic Programming. Springer-Verlag, Berlin, Heidelberg.

Koivu, M., Pennanen, T., and Ranne, A. (2003). Modeling assets and liabilities of a finnish pension insurance company: a veqc approach. Scandinavian Actuarial Journal - SCAND ACTUAR J, 2005:1–1.

Kouwenberg, R. (2001). Scenario generation and stochastic programming models for asset liability management. European Journal of Operational Research, 134(2):279 – 292. Financial Modelling.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. CoRR, abs/1509.02971.

Luckner, W. R., Backus, J. E., Benedetti, S., Bergman, D., Cox, S. H., Feldblum, S., Gilbert, C. L., Liu, X. L., Lui, V. Y., Mohrenweiser, J. A., Overgard, W. H., Pendersen, H. W., Rudolph, M. J., Shiu, E. S., and Smith Jr., P. L. (2003). Professional Actuarial Specialty Guide: Asset-Liability Management. Society of Actuaries.

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

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M. F. and Weinberger, K. Q., editors, Proceedings of The 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pages 1928–1937, New York, New York, USA. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. CoRR, abs/1312.5602.

Pennanen, T. and Koivu, M. (2002). Integration quadratures in discretization of stochastic programs. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Institut für Mathematik.

Rao, Z., Conroy, P., Yeowell, A., De Rosa, S., and Williams, W. (2010). Asset management optimisation using genetic algorithms and whole life, cost benefit analysis. In Integrating Water Systems - Proceedings of the 10th International on Computing and Control for the Water Industry, CCWI 2009.

Rosen, D. and Zenios, S. A. (2008). Enterprise-Wide Asset and Liability Management: Issues, Institutions, and Models. In Handbook of Asset and Liability Management - Set. North Holland.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. A. (2014). Deterministic policy gradient algorithms. In ICML.

Sutton, R. S. and Barto, A. G. (2018). Reinforcement Learning: An Introduction. The MIT Press, second edition.

Valladão, D. and Veiga, A. (2008). Optimum allocation and risk measure in an asset liability management model for a pension fund via multistage stochastic programming and bootstrap. EngOpt 2008 - International Conference on Engineering Optimization.

Watkins, C. J. C. H. and Dayan, P. (1992). Technical note: Q-learning. Machine Learning, 8:279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3):229–256.

Zhang, X. L. and Zhang, K. C. (2009). Using genetic algorithm to solve a new multi-period stochastic optimization model. Journal of Computational and Applied Mathematics.