Continuous-Action Reinforcement Learning for Portfolio Allocation of a Life Insurance Company

Carlo Abrate[1], Alessio Angius[1], Gianmarco De Francisci Morales[1], Stefano Cozzini[2], Francesca Iadanza[2], Laura Li Puma[3], Simone Pavanelli[2], Alan Perotti[1(✉)], Stefano Pignataro[2], and Silvia Ronchiadin[3]

[1] ISI Foundation, Turin, Italy
alan.perotti@isi.it
[2] Intesa Sanpaolo Vita, Turin, Italy
[3] Intesa Sanpaolo Innovation Center, Turin, Italy

Abstract. The asset management of an insurance company is more complex than traditional portfolio management due to the presence of obligations that the insurance company must fulfill toward the clients. These obligations, commonly referred to as *liabilities*, are payments whose magnitude and occurrence are a byproduct of insurance contracts with the clients, and of portfolio performances.

In particular, while clients must be refunded in case of adverse events, such as car accidents or death, they also contribute to a common financial portfolio to earn annual returns. Customer withdrawals might increase whenever these returns are too low or, in the presence of an annual minimum guaranteed, the company might have to integrate the difference. Hence, in this context, any investment strategy cannot omit the interdependency between financial assets and liabilities.

To deal with this problem, we present a stochastic model that combines portfolio returns with the liabilities generated by the insurance products offered by the company. Furthermore, we propose a risk-adjusted optimization problem to maximize the capital of the company over a pre-determined time horizon.

Since traditional financial tools are inadequate for such a setting, we develop the model as a Markov Decision Process. In this way, we can use Reinforcement Learning algorithms to solve the underlying optimization problem. Finally, we provide experiments that show how the optimal asset allocation can be found by training an agent with the algorithm Deep Deterministic Policy Gradient.

Keywords: Reinforcement learning · Portfolio allocation

1 Introduction

Portfolio management is a core activity in finance, whereby an entity, such as a fund manager or an insurance company, oversees the investments of its clients

Y. Dong et al. (Eds.): ECML PKDD 2021, LNAI 12978, pp. 237–252, 2021.
https://doi.org/10.1007/978-3-030-86514-6_15

to meet some agreed-upon financial objectives. In the context of an insurance company (henceforth simply referred to as 'company'), the clients not only contribute premiums to a common fund to buy assets, but also acquire the right to be paid in case of certain events (e.g., death in the case of a life insurance policy). Therefore, the company has to manage not only the assets, but also the liabilities deriving from the insurance. This combination of asset-liability management, and their inter-dependency, is one of the reasons why the insurance case is more complex than traditional portfolio management.

In this paper, we consider the problem of a company that handles insurance products for its clients, and wishes to optimize the risk-adjusted returns of the investment portfolio, while at the same time ensuring that its future liabilities are covered despite possible market fluctuations. These liabilities can be stochastic, and are usually correlated to some of the assets available to the company. In this scenario, one cannot just optimize for the risk-adjusted return, rather the investment portfolio has also to match the liabilities, and in particular their due dates. Finally, in a life insurance setting, the time horizon of the problem is relatively long (e.g., 30 years), and the portfolio gets rebalanced sporadically.

Commonly used financial tools for asset allocation such as Modern Portfolio Theory (MPT) [17] are inadequate for the considered setting. First, Markowitz's theory does not take into account liabilities and the future negative cash flows they generate. Second, it assumes a single decision point where the portfolio is optimized. While the methodology can be repeatedly applied at each decision point, it fails to take account for the path dependency of the problem: previous choices affect later ones. For instance, the decision to buy a risky asset early on in the lifetime of the fund might affect the ability to face negative cash flows later on, and thus inform a more conservative strategy. Clearly, an optimal strategy needs to take into account the whole decision space of the problem, i.e., the whole *sequence* of decisions (asset allocations) that lead to the final outcome.

Given the stochastic nature of markets and the multi-period decision nature of the problem, it is only natural to use a Markov Decision Process (MDP) as a model. An MDP is an extension of a Markov chain (a stochastic model of a sequence of events) that allows for account possible actions so that the stochastic outcomes are partly under the control of a 'decision maker'. The system moves in discrete steps from a state s to a new state s' according to some transition probability $P_a(s, s')$, which also depends on the action a taken. The transition generates a reward $R_a(s, s')$, and the goal is to find an optimal *policy*, i.e., a (stochastic) mapping of states to actions, such that the expected reward is maximized.

While there are several possible ways to solve an MDP, such as linear and dynamic programming [5], for large systems Reinforcement Learning (RL) is the de-facto standard tool to tackle the problem [26].

The contributions of this paper can be summarized as follows:

- We describe, formalize, and implement a realistic model of the asset-liability management process for an insurance company as a Markov Decision Process. The action space of the model is particularly challenging to explore, as each action can be sampled from a continuous $k-1$ simplex (where k is the number of available assets).

- We adapt a well-known algorithm for deep reinforcement learning for continuous action spaces (DDPG [14]) to our problem. To do so, we employ several techniques that are necessary for a quick and stable convergence: a warm-up stage to pre-train the critic network, a modification to the exploration policy to maintain important structural constraints of the problem, and a careful crafting of the reward function to implement domain-specific, parametric asset allocation constraints.
- We show experimentally that our solution is able to outperform a traditional mean-variance optimization baseline computed via Monte-Carlo sampling.

2 Problem Definition

This section provides a detailed description of the inner workings of a life-insurance company. We begin with a description of the mathematical model underlying the financial evolution of the company. Then, we provide a brief description of the implementation in terms of components and their interaction. Finally, we formalize the optimization problem. The Appendix includes further details about the components described in this Section.

The problem consists in optimizing the investments of an insurance company in order to maximize the profits. On the one side, the company manages a segregated fund that handles a portfolio of *assets* of different nature (equity, bonds, cash). On the other side, the company sells insurance products that differ from each other in their client characterization (in terms of age, behavioral properties such as the probability to pay premiums), and the percentage of the profits owed to the client from the returns generated by the segregated fund during the year. Irrespective of the profits generated by the portfolio, policies usually stipulate a guaranteed minimum return on investment for the clients. This minimum, referred to as *Minimum Annual Yield* (MAY) and denoted with κ, is particularly important because the insurance company must integrate the amount whenever the returns of the segregated fund are not able to meet the MAY. Conversely, the insurance company is allowed to take part of the *Surplus* (SP), by retaining a fixed spread over the surplus, or a fraction of it.

The profit of the company consists of the residual surplus once all the cash flows of the policies have been paid off. These payments, referred to as *liabilities*, are a consequence of several factors such as: insurance claims, and integration to reach the MAY. Liabilities depend on the type of insurance policy, but are also connected to the profits generated during the year for the client. For instance, the probability of client withdrawal may be affected by the amount of profits generated by the fund. To cover the liabilities, every insurance product is associated with *reserves*, which represent the value of the outstanding liabilities. Unused reserves contribute to the profits of the company.

2.1 Formalization

Our goal is to optimize the asset-liability management of the fund given an *economic scenario*, over a finite time horizon in $[0, T]$ divided into discrete slots

of one year. This scenario is a stochastic process which describes the financial market, based on existing models whose parameters are calibrated by using historical data. The model which generates the scenario is a black box from the point of view of the optimization, and can only be queried to generate a new realization from the process. Each realization from the process provides the information necessary to characterize the financial assets along the considered time interval. These random variables describe *Key Financial Indicators*(KFI) such as equity indexes, interest rates, and market spreads. Therefore:

Definition 1. *An* Economy *is a realization of KFIs from the stochastic process* $\mathcal{E}$ *which defines the economic scenario.*

We assume a set of financial asset classes, denoted with $\mathcal{C}$ and indexed from 1 to $|\mathcal{C}|$, that can be exchanged during the considered time horizon. Assets of each class are created at every time unit by combining a set of basic properties specific of the asset class with their corresponding KFIs. The creation of an asset corresponds to the definition of the minimal set of terms that allow for its accounting.

Definition 2. *An asset is a tuple composed of seven terms* $Y = \langle c, t^0, p^0, t^p, m, r, \overline{\chi} \rangle$*:*

- $c \in \mathcal{C}$ *the class of the asset;*
- $t^0 \in [-\infty, T]$ *the issue time (can be arbitrarily back in time);*
- $p^0 \in \mathbb{R}$ *the issue price, the market price of the asset at the moment of creation;*
- $t^p \in [-\infty, T]$ *the purchase time (can be arbitrarily back in time);*
- $m \in [0, M]$ *the remaining maturity of the asset;*
- $r \in \mathbb{R}$ *the redeem value of the asset;*
- $\overline{\chi} \in \mathbb{R}^M$ *a vector of coupons paid every year by the asset up to maturity (maximum maturity* M*).*

The accounting of any single asset Y at a given time t requires four basic functions: market value $f_{MV}(t, Y)$, book value $f_{BV}(t, Y)$, cash-flow $f_{CF}(t, Y)$, and generated income $f_{GI}(t, Y)$. The collection of assets owned by the fund at time t is the *portfolio*.

Definition 3. *The portfolio is a multiset* $\mathcal{P}(t) = \{Y_1 : n_1, Y_2 : n_2, \dots\}$ *where every* Y_i *is an asset that has not been sold yet and has* $t_i^p \leq t < t_i^p + m_i$ *(purchased before* t *and not expired yet), and* $n_i \in \mathbb{R}$ *is its nominal amount, i.e., how many units of that asset the portfolio contains.*

The accounting functions listed above apply to the portfolio as the sum of the function applied to each asset weighted by its nominal amount. To disambiguate the notation, we use the letter g to denote the functions applied to the portfolio while we keep the letter f for the functions applied to a single asset. As an example, the market value applied to the portfolio corresponds to $g_{MV}(t)$.

The contribution of a single asset class c to the portfolio value is calculated as follows:

$$A_c(t) = \frac{\sum_{Y_i \in \mathcal{P}(t) | c_i = c} n_i \cdot f_{MV}(t, Y_i)}{g_{MV}(t)}. \tag{1}$$

Definition 4. *We define the asset allocation at time t as a vector* $\overline{A}(t) = \langle A_1(t), A_2(t), \ldots, A_{|\mathcal{C}|}(t)\rangle$.

The portfolio is modified by means of selling and buying functions that take in input the current portfolio $\mathcal{P}(t)$ and a target asset allocation $\overline{X}_t = \langle X_1, X_2, \ldots, X_{|\mathcal{C}|}\rangle$.

Selling is performed first in order to free resources to buy new assets. Selling is guided by a projection function $g_{sell}(\mathcal{P}(t), \overline{X}_t)$ that returns in output a multiset $S = \{Y_1 : s_1, Y_2 : s_2, \ldots\}$ which contain the nominal amount of each asset in the portfolio that has to be sold in order to move the asset allocation toward $\overline{X}$. Thus, after the selling actions, the nominal amount of every asset $Y_i \in \mathcal{P}(t)$ is equal to $n_i - s_i$. The purchase of new assets is done in a similar way by using a projection function $g_{buy}(\mathcal{P}(t), \overline{X}_t)$ that provides a multiset of new assets $\{\tilde{Y}_1 : n_1^b, \tilde{Y}_2 : n_1^b, \ldots, \tilde{Y}_k : n_k^b\}$ bought from the market where n_i^b is the nominal amount of the asset to be added to the portfolio.

Putting all together, we can derive the portfolio at the next time step as:

$$\mathcal{P}(t+1) = (\mathcal{P}(t) \setminus g_{sell}(\mathcal{P}(t), \overline{X}_t)) \cup g_{buy}(\mathcal{P}(t), \overline{X}_t). \tag{2}$$

In order to complete the functions necessary to describe the segregated fund, let us define the capital gain of the portfolio at time t as follows:

$$g_{CG}(t, S) = \sum_{Y_i \in \mathcal{P}(t)} s_i \cdot (f_{MV}(t, Y_i) - f_{BV}(t, Y_i)), \tag{3}$$

and the portfolio return as $g_{PR}(t, S) = \frac{g_{GI}(t) + g_{CG}(t,S)}{1/2(g_{BV}(t-1) + g_{BV}(t))}$. The insurance company has to face liabilities in the form of insurance claims due to deaths and client withdrawal from the contract (surrender). Each insurance product guarantees different benefits to the clients. Hence, its liabilities affect the profits of the company differently from those of another product. For this reason, we assume that the ith insurance product is completely described in terms of the negative cash flow generated by the product.

Definition 5. *The i-th insurance product is a function $q_{NF}(\mathcal{Z}_i, \mathcal{R}(t))$ which determines the negative cash flow generated by the product as function of a set of parameters $\mathcal{Z}_i$ and the set of portfolio returns $\mathcal{R}(t)$ for time $t \in [1, T]$.*

The market value of the portfolio is monitored yearly and adjusted every time it moves outside a certain range in comparison with a projection of the (discounted) liabilities in the future, denoted by $q_{DL}(t)$.

Adjustments are capital injections/ejections that corresponds to loans. Let $g_{CI}(t)$ be the function that determines the amount of cash that is paid or earned at time t by applying the interest rate ϕ_t^{inj} to the open loans plus an additional penalty ϵ for cash injections. Finally, we can define the return on capital at time t as $g_R(t) = \frac{g_{CA}(t) - g_{CA}(t-1)}{g_{CA}(t-1)}$ where $g_{CA}(t) = g_{MV}(t) - q_{DL}(t) - (g_{MV}(0) - q_{DL}(0)) + g_{IJ}(t)$ is the fund capital gain, net of the overall discounted liabilities and the total injection $g_{IJ}(t)$ which corresponds to of the sum all the capital adjustments (injections and ejections).

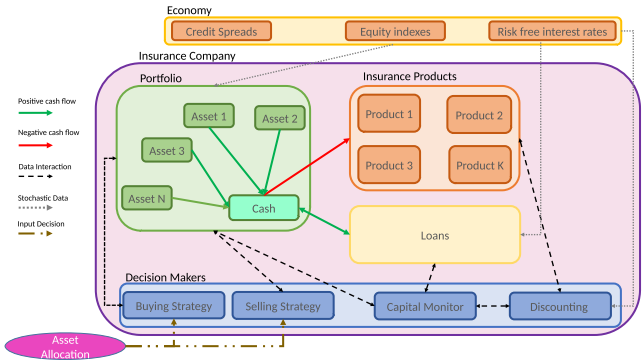


Fig. 1. Diagram describing the main components and interactions of the Insurance Company Model.

2.2 Implementation Details

Figure 1 provides a graphical description of the model, its components, and their interactions. Components have been realized as black boxes, so that they can be implemented with the desired level of detail and substituted without affecting the soundness of the model as a whole.

At the top of the figure, we observe the *Economy* which provides three different classes of KFIs. The current implementation is based on a combination of Cox-Ingersoll-Ross models [3] but the underlying process can be changed transparently.

At the bottom we find the block *Decision Maker* which is composed of: *Buying* and *Selling strategy* which correspond to the the functions $g_{sell}(\mathcal{P}(t), \overline{X})$, and $g_{buy}(\mathcal{P}(t), \overline{X})$, respectively; the *Discounting* which perform the projection of the liabilities in the future and discounts them according to a discount curve given by the Economy; finally, the component named *Capital Monitor* implements capital injection/ejection mechanism.

The component labelled as *Loans* manages the state of the loans and computes the costs at every time unit. Costs are computed by using an interest curve taken from the Economy. *Insurance Products* is a collection of insurance products where each entry stores the state of the reserves and calculates the negative cash flow generated by the product. Finally, the component *Portfolio* contains all the assets that have been bought and have not reached their maturity. The asset referring to *Cash* is unique and always present because it interacts with other components.

To exemplify the temporal dynamic of the interactions, Algorithm 1 shows the pseudo-code of the routine required to move the Company one year forward in the future. This function constitutes the cornerstone for building the environment of our RL framework.

The first observation is that the time update does not occur at the end of the function but in the middle. This is because, in principle, this routine describes

Algorithm 1. Step forward in the evolution of the segregated fund in time

```
function STEP(X̄)
    costLoans = loans.g_CI(t)
    discounting = disc.q_DL(t)insuranceProds
    inj = capitalMonitor.verify(portfolio.g_MV(t), discounting)
    if inj ≠ 0 then
        loans.insert(inj)
    sells ← g_sell(portfolio, X̄)
    new ← g_buy(portfolio, X̄)
    portfolio = (portfolio\sells) ∪ new
    returns = returns ∪ portfolio.getReturn(sells)
    t = t + 1
    ncashflow = insuranceProds.q_NF(returns)
    pcashflow = portfolio.g_CF(t)
    portfolio.updateCash(pcashflow, ncashflow, costLoans)
    return portfolio.g_CA(t)
```

what happens between the end of the current year and the beginning of the next. The operations performed at the end of the year are: the computation of the costs of the loans which can be either positive or negative and will be subtracted from the cash in the next year; the capital injection/ejection if needed; finally, the selling and buying operations as well as the computation of the capital gain of the year. Then, the time counter is increased and the cash is updated by considering negative and positive cash flow together with the costs of the loans. Finally, the routine ends by returning the current capital. Let us remark that the capital does not change only because of the cash flow, but also as a consequence of the changes of the KFIs that are embedded in the assets composing the portfolio.

2.3 Optimization Problem

Our goal is to optimize the average final return on capital, adjusted for its volatility. Specifically, we measure volatility as the standard deviation of the return on capital over the time horizon, given an asset allocation strategy and an economic scenario.

The volatility provides an estimation of the yearly oscillations of the returns within each simulation run. The idea behind its use is to penalize portfolios that lead to large oscillations of the returns during the considered time interval, which are a hindrance to the payment of the liabilities. Let $\mu = \frac{1}{T}\sum_{t=1}^{T} g_R(t)$ be the average of the return within the same realization, and let $\sigma = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(g_R(t)-\mu)^2}$ be the standard deviation. The objective function can be written as follows:

$$\underset{\overline{X}_0,\ldots,\overline{X}_{T-1}}{\operatorname{argmax}} = \mathop{\mathbb{E}}_{\mathcal{E}}[\mu - \lambda \cdot \sigma] \tag{4}$$

where $\overline{X}_0, \ldots, \overline{X}_{T-1}$ are the asset allocations at any point in time and λ is a *risk aversion* factor representing the weight of the volatility over the return on capital, and the expectation is over the possible realizations of the economy $\mathcal{E}$.

We define the problem in such a way that the objective function in Eq. (4) can be guided by two different classes of constraints. The first class (*Type 1*) is necessary to maintain the problem sound from a theoretical point of view:

$X_{t,i} \geq 0 \ \forall t, i; \quad \left|\overline{X}_t\right| = 1 \quad \forall t$, which verifies that each target asset allocation is long-only and properly defined on a simplex. The second class (*Type 2*) includes constraints depending on external parameters that are used to restrict the domain of the asset allocation; for example, we might want to set boundaries for the allocation of a subset of the asset classes (e.g., no more than 60% allocation on all bonds). By denoting the subset with $\mathcal{Q} \subset \mathcal{C}$, we can formalize this type of constraint as $\check{\beta} \leq \sum_{q \in \mathcal{Q}} A_c(t)q \leq \hat{\beta}, \ \forall t$.

3 Solution

We use DDPG [14] as a starting point for the implementation of our Reinforcement Learning agent. DDPG, or Deep Deterministic Policy Gradient, is an actor-critic, off-policy, model-free algorithm based on deterministic policy gradient, and that can operate over continuous action spaces. DDPG belong to the set of actor-critic agents, whose high-level architecture is depicted in Fig. 2.

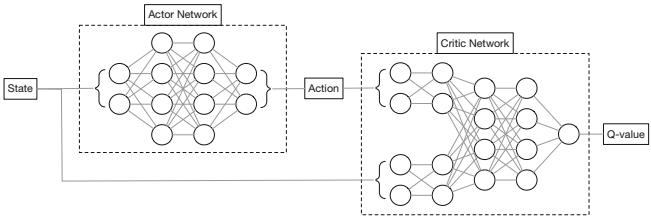


Fig. 2. Actor-Critic agent architecture

The *critic* network learns to approximate the temporally discounted cumulative reward of an action on a given state, exploiting the Bellman equation as in Q-learning. The *actor* network, given a state, learns to produce actions that maximize the Q-value estimated by the critic. It is worth observing that the actor receives no direct feedback from the environment: the back-propagated error used to train the actor flows through the critic first. During the experimental phase, which will be described in the next section, we observed that, thanks to the off-policy property of the algorithm, pre-training the critic on randomly sampled actions (and the respective environment-generated rewards) had a strong positive impact on the performance of the RL agent. We therefore systematically perform a critic *warm-up* phase before undergoing the standard actor-critic training loop.

3.1 Structural and Parametric Constraints

In our specific settings, actions are asset allocations, and are therefore modeled as a point on a simplex. Within our Reinforcement Learning agent, the actions are produced by the actor, and ensuring that these actions are on a simplex

can be easily achieved by setting the last actor activation function as a softmax. We call these requirements **structural constraints**. However, DDPG implements RL-exploration by means of a *perturbation policy* that adds to the action noise produced by an Ornstein-Uhlenbeck process [6]. Clearly, a noisy action would likely violate the structural constraints, thus producing non-admissible actions. In order to maintain action admissibility, we modified the standard DDPG approach by moving the perturbation upstream with respect to the activation function. We have adopted the recently-proposed *parameter perturbation* approach, where in order to perform explorative actions we add noise to the actor weights [23]. By doing so we are sure to produce exploratory actions that satisfy the structural constraints, as the action is produced by the final softmax activation function of the actor. The weights are then reverted to their previous values before proceeding with the training.

Our specific setting might impose additional, domain-related constraints, such as upper or lower bounds on specific assets, for instance: *the Equity asset shall not surpass 20% of the total asset allocation*. Since these values vary from one scenario to another, we have implemented them in a parametric fashion, where the threshold values are read from external configuration files, and we call them **parametric constraints**. Unlike the structural constraints, there is no straightforward way to design an actor network so that all proposed actions are compliant with the parametric constraints. Instead of structurally preventing the actor from expressing actions that violate the parametric constraints, we elected to teach the agent, as a whole, that such actions are undesirable. We have therefore added a regularization term to the environment reward that penalizes the action by an amount proportional to the excess threshold violation, by using a hinge loss function. With this approach, the actor can quickly learn that the simplest way to obtain higher rewards is to propose admissible actions. At the same time, this approach allows for high flexibility, since the parametric constraints are set in the environment, and thus decoupled from the agent architecture.

4 Experimental Evaluation

Before presenting the results obtained by using the reinforcement learning framework to solve the asset allocation problem, we describe those settings that are shared by all the experiments presented in this section. We assume an initial asset allocation composed of cash only. The initial amount of cash is equal to 1050, while the reserves amount to 1000, which implies an initial capital of 50. The interest rate on loans is set to the interest rate of the "Italian BTP" bond with one year maturity, and the penalty ϵ is set to 2%. Similarly, the discounting interest rate is set to the 30% of "Italian BTP" bonds. A single insurance product is considered. The product guarantees a minimum yield of 0.5% per annum, and uses a uniform distribution over time of the payments for surrender or death.

In order to have a baseline to compare our RL framework to, we evaluate it on a simplified scenario on which the traditional Markowitz/Black-Litterman [1,17] approach can be applied. In particular, we consider a scenario in which:

- a single asset allocation is decided at time zero;
- rebalancing aims only to replenish negative amounts of cash by selling the other assets "pro quota" at every time $t > 0$.

These assumptions lead to a "fire and forget" scenario in which a single decision taken at the beginning determines the overall quality of the investments. Hence, $\overline{X}_0$ is the only decisional variable, and the entries of asset allocations $\overline{X}_{t>0}$ are determined directly the from state of the portfolio according to the formula:

$$X_{t,i} = \begin{cases} max(0, A_i(t)) & i = \text{Cash} \\ \frac{A_i(t)}{max(0, A_C(t)) + \sum_{i \neq C} A_i(t)} & \text{otherwise.} \end{cases} \tag{5}$$

The use of a single decisional variable allows the comparison of the policy found by the agent with the results obtained by performing a gird search on the action space combined with Monte-Carlo sampling. The grid search is performed by exploring the action space simplex with a fixed step size in $\{0.20, 0.10, 0.05, 0.02, 0.01\}$. Each action is evaluated by averaging the obtained reward over 500 realizations of the economy. To avoid stochastic effects from affecting the comparison, these realizations are drawn in advance and fixed for all the actions of the search, and are used in round-robin during the training of the agent. The number of realization is sufficiently large that the probability of an agent exploring all them on a given small section of the action space is negligible.

4.1 Three Assets Scenario.

In the first experiment we focus on a scenario with three assets: cash, equity, and bond. We include a parametric constraint that sets 0.17 as an upper bound for the equity asset, and set the λ risk-aversion coefficient to 0.2. In order to create a controlled experimental environment, we run a set of simulations with 0.01 grid step – corresponding in this scenario to 5151 simulations. From this fine-grained set of experiments we can obviously extract coarser subsets by increasing the grid step size, as shown in Table 1.

We use the coarsest grid (step = 0.20) for the warm-up phase of the Critic, while keeping the more fine-grained best actions and rewards aside in order to use them to evaluate the Actor's performance during and after training. The warm-up phase is a standard fully supervised learning task, and we report the Critic loss (computed as mean absolute percentage error) during training in Fig. 3a. We then store the pre-trained critic weights and re-load them in subsequent experiments.

The core learning task for all our experiments is the training of our custom DDPG agent, and this process involves several hyper-parameters. These

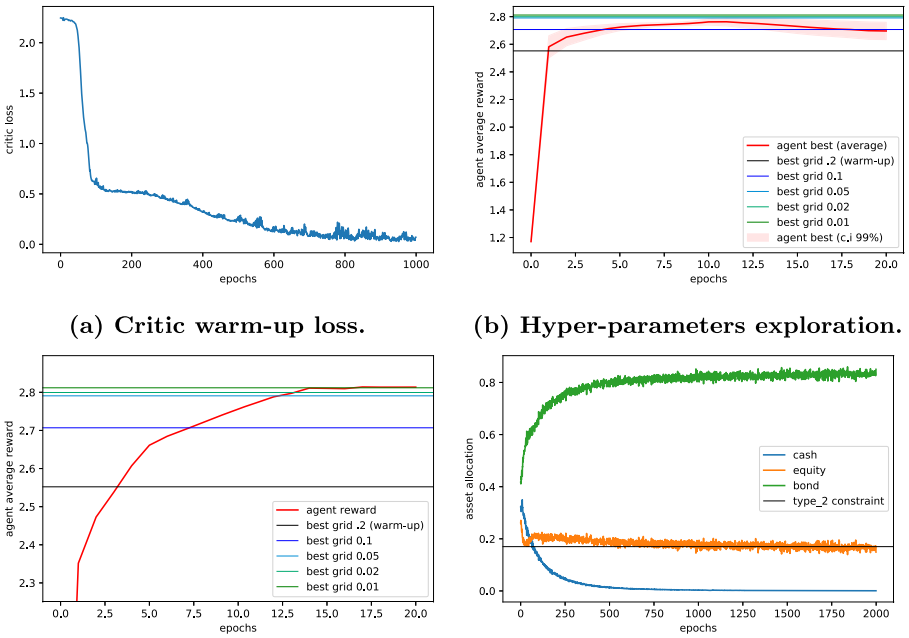
Table 1. Parameters and results for the grid-search based simulations: respectively, step size for the grid, number of different actions explored, best action found with the given grid, corresponding average reward estimated on the 500 fixed realizations of the economy.

Step	# actions	Best action	Best reward
0.20	21	[0.0, 0.20, 0.80]	2.552
0.10	66	[0.0, 0.10, 0.90]	2.707
0.05	231	[0.0, 0.15, 0.85]	2.790
0.02	1326	[0.0, 0.16, 0.84]	2.799
0.01	5151	[0.0, 0.17, 0.83]	2.811

include structural details for the Actor neural network (number of neurons per layer, weight initialization parameters), training details (learning rate and decay for both the Actor and Critic), memory buffer parameters (capacity, batch size), and a noise parameter governing the weight perturbation process used for exploration. We therefore carried out a hyper-parameter optimization, where we observed that our system is rather sensitive to hyperparameter setting, with the Actor prone to converge on one-hot actions – most commonly assigning everything to bonds. The first hyper-parameter search rounds were used to define an 'admissibility subspace' of hyper-parameters that did not cause the agent to spiral into such states, while subsequent iterations (such as the one visualized in Fig. 3b) allowed to progressively approximate the known optimal scores. To obtain these results we trained a batch of 32 agents with different configurations of hyper-parameters and, every 100 iterations, measured their average score on the set of 500 pre-computed realizations. In Fig. 3b we show the cross-agent average score and, as shaded area, its 99% confidence interval; we also show the known best scores for grids with increasing granularity (as reported in Table 1) as horizontal lines, with the black line corresponding to the .20 step used for the Critic warm-up.

Figure 3c shows the learning curve of the optimal agent, able to match and even surpass the best known action, corresponding to the 0.01-step grid. Figure 3d also reports the actions played by the optimal Actor during the training phase. It clearly shows that the agent learns to assign the Equity asset (which gives the highest reward) to the highest possible value that would not incur a penalty (horizontal black line, corresponding to the set parametric constraint of .17).

We remark that we use this three assets scenario as a sandbox where it is still feasible to exhaustively explore the action space with non-trivial grid steps in order to compute the best action and reward; with an increasing number of assets this procedure quickly becomes computationally too expensive, as the number of actions to explore grows exponentially.



(a) Critic warm-up loss. **(b) Hyper-parameters exploration.**

(c) Selected agent's training score. **(d) Selected agent's training actions.**

Fig. 3. Training agents on a three assets scenario.

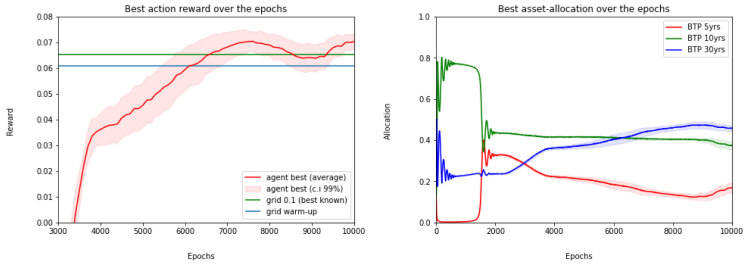
4.2 Six Assets Scenario

The second experiment aims to show that a near optimal solution can still be found when actions have larger dimensionality. In particular, we test the case in which the portfolio is composed only of cash, and Italian BTP Bonds with 3, 5, 10, 20, and 30 years tenors. No constraints have been considered; hence, any portion of the action space might contain candidates for the optimum. Furthermore, the risk-aversion factor λ has been set to a high value of 4 in order to avoid that the optimal solution comprises solely of the most profitable and most risky asset, i.e., the 30-years bond. In this setting, we perform 15 trainings of the agent by using only 21 actions for the warm-up. This number of actions corresponds to an exhaustive search on a grid with step size equal to 0.5. Only two actions used for the warm-up were able to provide a positive reward. The largest average reward included in the warm-up was equal to 0.0607 and was obtained by investing equally in BTP with 5 and 30 years tenors.

All the experiments provided an improvement from the initial warm-up from a minimum of 3.9% (reward 0.0631) to a maximum of 21.7% (0.0739). Figure 4 provides a summary of the experiments by showing the evolution of the best action found by the agent, both in terms of asset allocation[1] and reward over

[1] The other three assets are omitted as they go to zero very quickly.

the training epochs. In order to provide a further comparison, we provide also the best action found with an exhaustive search performed with a step size 0.1. In this setting, the testing of a single action for all the 500 economies requires around one minute, and the grid contains 3003 actions; hence, whole computation required more than 2 days. In spite of this, the obtained reward is still quite far from the best found by the agent, whose training requires only two hours. The rationale behind this gap can be explained by observing the evolution of the best asset allocation in Fig. 4b where we can notice that a near optimal solution can be found only at precision below 1%. In particular, the best reward found was generated by an action representing an asset allocation where only BTP at 5,10, and 30 years had non-zero weights equal to 0.19, 0.361, and 0.449, respectively.



(a) Average Reward **(b) Asset Allocation**

Fig. 4. Training agents on a six assets scenario

5 Related Work

The excellent results obtained in games [18,25] and robotics [13,22] have put the spotlight on the ability of RL to find near optimal solutions in large multi-stage, high-dimensional problems. And as such, they have drawn the attention of the financial sector since modern portfolio theory deals with similar settings.

Modern portfolio theory, initiated by Nobel-prize-winner Markowitz [17] and improved by Black and Litterman [1], consists in finding the optimal financial allocation over a single time horizon by using mean-variance asset allocation models. These models heavily rely on Markov processes to characterize the stochastic nature of the economy. Hence, they naturally suggests the coupling of Markov Decision Process (MDP) with Reinforcement Learning (RL) as a framework to solve these problems [26]. It is thus not surprising that the literature on RL methods for asset allocation problems is growing year by year [24].

For example, Wang and Zhou [27] present a framework, called exploratory-mean-variance (EMV), for continuous portfolio selection (action) in continuous time and continuous wealth (state) spaces. Q-learning methods are also common. Halperin [8] provides an example of a data-driven and model-free methods for

optimal pricing and hedging of options with RL by constructing a risk-adjusted MDP for a discrete-time version of the classical Black-Scholes-Merton model. Nevmyvaka et al. [21] use Q-learning for large scale optimal order execution.

Direct policy search for portfolio allocation is instead presented by Moody et al. [19,20], and a tree-search approach that integrates the advantages in solving continuous action bandit problem with sample-based rollout methods is introduced by Mansley et al. [15]. The algorithm, named Hierarchical Optimistic Optimization applied to Tree (HOOT), adaptively partitions the action space, thus enabling it to avoid the pitfalls encountered in algorithms that use a fixed action discretization.

De Asis et al. [4] explore fixed-horizon temporal difference (TD) reinforcement learning algorithms for a new kind of value function that predicts the sum of rewards over a fixed number of future time steps. Jangmin et al. [9] perform dynamic asset allocation with a reinforcement-learning framework which uses the temporal information from both stock recommendations and the ratio of the stock fund over the asset. Buhler et al. [2] tackle option pricing and hedging by using deep RL methods. [10] explains how to handle When the model available to the agent is estimated from data, Since the seventies, portfolio theory has been extended in order to consider liabilities. Notable examples are *Asset-Liability Management* (ALM) and *dedicated portfolio theory* models [12]. These models were considered intractable before it was suggested that they can be handled with an underlying Markovian structure and deep learning techniques [2,11]. In this direction, Fontoura et al. [7] consider the optimization of investment portfolios where investments have to match (or outperform) a future flow of liabilities within a time constraint. They address an ALM problem with a variation of Deep Deterministic Policy Gradient algorithm (DDPG). In spite of the fast growing literature, only one work [7], takes in consideration a multi-stage setting that takes into account both asset-allocation and liabilities by still allowing the use of off-the-shelf RL methods. However, liabilities are far from the level of detail of those presented in the current work, since their description is limited to simple phenomena such as inflation. To the best of our knowledge, our work is the first RL framework able to describe a strong correlation between asset allocation and liabilities.

6 Conclusions

This paper presented a framework for the asset management of a life insurance company which differs from traditional portfolio management due to the strong dependency between the profits of the portfolio and the liabilities generated by the obligations toward the clients. The framework has been developed as a Reinforcement Learning environment by maintaining flexibility in many aspects of the problem. The most important are: (*i*) not being bound to any specific set of assets; (*ii*) having user-defined buying/selling strategies; (*iii*) the modeling of the liabilities directly from the parameters of the insurance products that generate them; (*iv*) general strategies for capital control and leverage.

We defined a risk-adjusted optimization problem to maximize the capital over a finite time horizon by choosing the asset-allocations at possibly any time unit. We validated the framework by means of a set of experiments performed on a simplified scenario where a single asset allocation must be chosen at time zero. Despite the smaller setting, experiments demonstrate how fast the problem grows in complexity by pointing at RL as the only viable solution for the problem.

Testing our proposed framework in a proper multi-stage setting is the future work with the highest priority, although proper baselines for this case need to be devised. However, the generality of the framework suggests many other problems. For example, the compounding effect on the capital is currently not addressed but should be taken into account as well as the definition of different measures for the control of the risk. In addition, while our framework defines the objective function in line with modern portfolio theory for comparison purposes, the literature on risk-adjusted MDPs [16] might provide a more robust grounding for our portfolio allocation problem. On the experimental side, since the results so far have shown marked oscillations when the agent is close to a near optimal policy, early stopping strategies should be explored. Finally, the design of more advanced buying and selling strategies is an orthogonal but nevertheless interesting future direction.

Acknowledgements. The research was conducted under a cooperative agreement between ISI Foundation, Intesa Sanpaolo Innovation Center, and Intesa Sanpaolo Vita. The authors would like to thank Lauretta Filangieri, Antonino Galatà, Giuseppe Loforese, Pietro Materozzi and Luigi Ruggerone for their useful comments.

References

1. Black, F., Litterman, R.: Global portfolio optimization. Financ. Anal. J. **48**(5), 28–43 (1992)
2. Buhler, H., Gonon, L., Teichmann, J., Wood, B.: Deep hedging (2018)
3. Cox, J.C., Ingersoll, J.E., Ross, S.A.: A theory of the term structure of interest rates. Econometrica **53**(2), 385–407 (1985). ISSN 00129682, 14680262
4. De Asis, K., Chan, A., Pitis, S., Sutton, R.S., Graves, D.: Fixed-horizon temporal difference methods for stable reinforcement learning. arXiv preprint arXiv:1909.03906 (2019)
5. Denardo, E.V.: On linear programming in a Markov decision problem. Manag. Sci. **16**(5), 281–288 (1970)
6. Doob, J.L.: The Brownian movement and stochastic equations. Ann. Math. 351–369 (1942)
7. Fontoura, A., Haddad, D., Bezerra, E.: A deep reinforcement learning approach to asset-liability management. In: 2019 8th Brazilian Conference on Intelligent Systems (BRACIS), pp. 216–221. IEEE (2019)
8. Halperin, I.: QLBS: Q-learner in the Black-Scholes(-Merton) worlds. arXiv preprint arXiv:1712.04609 (2017)
9. Jangmin, O., Lee, J., Lee, J.W., Zhang, B.T.: Adaptive stock trading with dynamic asset allocation using reinforcement learning. Inf. Sci. **176**(15), 2121–2147 (2006)

10. Jiang, N., Kulesza, A., Singh, S., Lewis, R.: The dependence of effective planning horizon on model accuracy. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Richland, SC, pp. 1181–1189. International Foundation for Autonomous Agents and Multiagent Systems (2015). ISBN 9781450334136
11. Krabichler, T., Teichmann, J.: Deep replication of a runoff portfolio (2020)
12. Leibowitz, M., Fabozzi, F.J., Sharpe, W.: Investing: The Collected Works of Martin L. Leibowitz. Probus Professional Pub (1992)
13. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. J. Mach. Learn. Res. **17**(39), 1–40 (2016)
14. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
15. Mansley, C., Weinstein, A., Littman, M.: Sample-based planning for continuous action Markov decision processes. In: Twenty-First International Conference on Automated Planning and Scheduling (2011)
16. Marcus, S.I., Fernández-Gaucherand, E., Hernández-Hernandez, D., Coraluppi, S., Fard, P.: Risk sensitive Markov decision processes. In: Byrnes, C.I., Datta, B.N., Martin, C.F., Gilliam, D.S. (eds.) Systems and Control in the Twenty-First Century. PSCT, vol. 22, pp. 263–279. Springer, Heidelberg (1997). https://doi.org/10.1007/978-1-4612-4120-1_14
17. Markowitz, H.: Portfolio selection. J. Financ. **7**(1), 77–91 (1952)
18. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). ISSN 00280836
19. Moody, J., Saffell, M.: Learning to trade via direct reinforcement. IEEE Trans. Neural Netw. **12**(4), 875–89 (2001)
20. Moody, J., Wu, L., Liao, Y., Saffell, M.: Performance functions and reinforcement learning for trading systems and portfolios. J. Forecast. **17**(5–6), 441–470 (1998)
21. Nevmyvaka, Y., Feng, Y., Kearns, M.: Reinforcement learning for optimized trade execution. In: Proceedings of the 23rd International Conference on Machine Learning, ICML 2006, pp. 673–680. Association for Computing Machinery, New York (2006)
22. Peters, J., Vijayakumar, S., Schaal, S.: Reinforcement learning for humanoid robotics. In: IEEE-RAS International Conference on Humanoid Robots (Humanoids2003), Karlsruhe, Germany, 29–30 September (2003). CLMC
23. Plappert, M., et al.: Parameter space noise for exploration. arXiv preprint arXiv:1706.01905 (2017)
24. de Prado, M.L.: Advances in Financial Machine Learning, 1st edn. Wiley, Hoboken (2018)
25. Silver, D., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–503 (2016)
26. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)
27. Wang, H., Zhou, X.Y.: Continuous-time mean-variance portfolio selection: a reinforcement learning framework. Math. Financ. **30**(4), 1273–1308 (2020)