HEC Montréal

Goal-Based Wealth Management by Reinforcement Learning

par

Ioannis Volakakis

Sciences de la gestion
(Option ingénierie financière)

Mémoire présenté en vue de l'obtention
du grade de maîtrise ès sciences
(M. Sc.)

Codirecteur de recherche: Michel Denault
Codirecteur de dépôt: Jean-Guy Simonato

December, 2023

Résumé

L'objectif de ce mémoire est de développer et optimiser un environnement d'entraînement d'une politique pour un problème de gestion de richesse à un objectif d'investissement (GBWM) en utilisant des méthodes d'apprentissage par renforcement (RL) où le degré de réussite de l'implémentation est déterminé par sa capacité d'approcher la solution cible déterminée par l'approche de programmation dynamique (DP). Bien que le Q-Learning, une méthode de RL, fût en mesure d'approximer la politique optimale lors de projets de recherche précédents, cette approche ne fût pas en mesure d'offrir de solution complète face à des inefficacités liées aux problèmes à grandes dimensions qui surviennent lorsque l'environnement d'entraînement offre un nombre d'actions abondant à l'agent. Le tout mène à considérer des méthodes approximant la politique, qui elles permettent de prévenir des encombrements liés aux grandes dimensions tout en permettant à l'agent de naviguer dans un environnement continu, chose désirable considérant la nature du probleme d'optimisation dans un environnement de marchés financiers. Une méthode d'approximation de la politique est donc utilisée par l'entremise de l'algorithme REINFORCE. L'application est faite à l'aide de la librairie Python nommée «Pytorch» qui permet de bâtir un environnement d'entraînement de manière efficace pour le modèle. Dans un but de permettre la mise-en-oeuvre d'une méthode d'approximation de la politique dans un contexte de problème GBWM, des approches de normalisation et centrage ont été développées. D'ailleurs, la méthode de centrage s'est avérée supérieure quant à l'amélioration de la vitesse d'apprentissage du modèle. Par la suite, une approche d'approximation de la politique a été appliquée pour un problème de gestion d'actifs à plusieurs objectifs d'investissements (MGWM). L'environnement d'entraînement utilisé n'est pas parvenu à mener la politique à l'optimalité malgré des solutions qui se sont approchées des solutions cibles.

Mots-clés

Gestion d'actifs à un objectif d'investissement, programmation dynamique, apprentissage par renforcement, méthode d'approximation de la politique, gestion d'actifs à plusieurs objectifs d'investissements

Abstract

The objective of this thesis is to develop and optimize a framework for the training of a policy in the scope of a goal-based wealth management (GBWM) problem by employing Reinforcement Learning (RL) techniques. The success level of a given RL implementation is based on its ability to approach the benchmark solution established by the Dynamic Programming (DP) algorithm developed for a GBWM problem. While Q-Learning, an RL technique, was successful in approximating the optimal policy in previous research projects, it did not offer a complete solution to address the curse of dimensionality that becomes a concern when actions are abundant within the training environment. Policy approximation methods, however, offer a solution to this limitation and allow for training the policy in a continuous state-space, which is a desirable feature given the nature of the optimization problem in an environment replicating financial markets. A policy approximation method is applied through the REINFORCE algorithm, which involves using the *PyTorch* library in Python as it allows to practically construct the model's training environment. The implementation of a policy gradient method to solve a GBWM problem led to the development of frameworks to both normalize and center the model's input variables. In fact, an adjusted centering approach tailored for a GBWM problem proved to maximize the model's speed of convergence towards the optimal policy. An attempt to apply a policy approximation approach is then made for a multi-goals wealth management (MGWM) problem. The training environment used for a MGWM problem was unable to fully guide the policy towards optimality despite the fact that the resulting solutions approached their respective benchmark.

Keywords

Goal-Based Wealth Management, Dynamic Programming, Reinforcement Learning, Policy Approximation, Multi Goals Wealth Management

Contents

List of Figures

List of Tables

Acknowledgements

It is with much gratitude and enthusiasm that I embarked the journey of pursuing graduate studies at HEC, in which I had the privilege to learn from talented, passionate and caring professors. For a significant part of this journey, I had the chance to rely on the tremendous support of my thesis co-supervisors, Michel Denault and Jean-Guy Simonato, who entrusted me with working on a topic to which they have made substantial contributions. I thank them for their patience, the guidance I was provided, and for the countless encouragements I received throughout this process. The completion of this thesis was highly impacted by the support of Florian Mitjana, whom I thank for always making himself available and his willingness to pass on valuable knowledge. I would also like to thank Inovestor for their contribution to this thesis, and for the great treatment I received during the course of our partnership.

I would also like to convey my appreciation to everyone at Collège Notre-Dame, Vanier College, Concordia University and HEC Montréal who supported me, contributed to my development and nourished my interests.

I am deeply grateful to my family and friends, who placed my aspirations at the forefront of their priorities. It goes without saying that their unconditional support has helped me push through the most challenging moments.

1 Introduction

In recent years, investment management and machine learning have been two attention-grabbing fields due to prompt advancements in their own respect. These two fields have been merged and placed at the center of multiple research projects across financial and academic institutions. As technological capacities continue to expand, additional changes with strong impacts can be expected in the near and long-term future. In this thesis, reinforcement learning is used to solve a dynamic goal-based wealth management (GBWM) problem. GBWM itself represents an alternative approach to the widely known Modern Portfolio Theory (MPT), developed by Harry Markowitz (1952). Under mainstream investing methods that are for the most part founded on the basis of MPT, portfolio performance is determined by returns earned as a result of the amount of risk undertaken by an investor. By labeling risk as the standard deviation of returns, MPT seeks to maximize the mean-variance return of a given portfolio, which is synonymous with maximizing its Sharpe ratio. On the other hand, GBWM aims to maximize the likelihood of attaining a predefined goal, usually a wealth level, with the risk being falling short of this goal.

GBWM takes its roots in the work of Abraham Maslow (Parker, 2022). Maslow studied the impact of psychological and physical needs on human behaviour, where the hierarchy of one's needs is referred to as mental accounting. Two decades later, Jean L.P. Brunel examined the human tendency to structure financial planning around short- and long-term life aspirations. In 2000, Hersh Shefrin and Meir Statman (Parker, 2022) were able to build on Brunel's work by introducing the behavioral portfolio theory (BPT). In contrast to the MPT, BPT redefines risk as the probability of failing to obtain a minimum return or wealth level within a certain period. These developments motivated Das, Scheid, Statman and Markowitz (Parker, 2022) to join their efforts and put forth the concept of GBWM with the objective of solidifying the theoretical content behind investment approaches that identify the source of risk as being the probability of failing to attain a financial goal (Das et al., 2011).

Technological advances have spurred extensive efforts in leveraging Reinforce-

ment Learning (RL) techniques to improve financial decision-making. RL is an approach where an algorithm leads an individual (or an agent) to understand a given environment by interacting directly with it (Hambly et al., 2023). This approach leverages computational tools as it requires to perform numerous computations to complete optimization tasks in a scalable manner. In fact, RL algorithms have been successfully used to optimize decisions relating to the execution of securities, hedging strategies, market making and more (Hambly et al., 2023).

RL methods have the ability to learn optimal decision sequences by allowing the agent to explore the environment without any interference, consequently allowing such algorithms to gain a deep understanding of complex environments (Hambly et al., 2023). RL has been linked to GBWM in previous works by Das and Varma (2020), where a tabular Q-Learning algorithm was employed to estimate the sequence of optimal actions by the agent. Maxence Prémont (2021) improved the performance of the original Q-Learning algorithm proposed by Das and Varma (2020) by using decay functions that improved the learning process in the training phase. In addition, Prémont introduced a Policy Approximation approach to estimate the optimal policy, which ultimately led to identifying a series of limitations that will serve as buildings blocks in this thesis as the downright objective is to develop a Policy Gradient approach involving the use of an artificial neural network (ANN).

This thesis is organized as follows. Section 2 includes a literature review in which key takeaways from previous research projects on GBWM are revisited. Section 3 includes a detailed description of the GBWM multi-period optimization problem, followed by a section on the dynamic programming method to solve a multi-period GBWM problem whose solution will serve as a benchmark in analyses included in later chapters, and concludes with a numerical experiment to exemplify its contained concepts. Section 4 shifts the focus to the exploration of a new approach to solve a dynamic GBWM problem: the use of a Policy Gradient approach by leveraging the REINFORCE algorithm to train a model that is capable of learning the environment's optimal policy. In order to improve the model's learning ability, two scaling methods were used: normalization (a widely known scaling method to train neural networks) and an adjusted centering approach adapted to a muti-period GBWM

problem. When combined with a baseline and batch training, the adjusted centering application proved to be effective in supporting the model converge towards an optimal policy while also maximizing the rigor of the training phase. Finally, Section 5 applies the newly developed Policy Gradient (with REINFORCE) approach onto a multiple goals wealth management (MGWM) environment, which considers multiple goals all throughout the investment period as it optimizes the policy. The model's performance is compared to a set of benchmark solutions obtained using the dynamic programming method. While the training environment used for a MGWM problem showed an ability to improve the model's policy, it revealed limitations with respect to fully optimizing it.

2 Literature Review

The desire to optimize risk-bearing investments is a priority to all portfolio managers. In 1952, Harry Markovitz defined an optimal portfolio as one providing the highest expected return for a given level of risk. Markowitz emphasized the importance of diversifying portfolio funds to earn returns that are close to expected levels as the law of big numbers comes into play (Markowitz, 1952). Markowitz introduced the notion of an efficient frontier, which represents the set of optimal portfolios that offer the highest returns for a given risk level (Markowitz, 1952). Investors can use the efficient frontier to identify and select an optimal portfolio tailored to their risk tolerance.

In an effort to enhance the use of the Markowitz Efficient Frontier, goal-based wealth management (GBWM) was developed to be an investment approach that focuses on maximizing investors' probability of achieving a predetermined wealth level within a certain time horizon. The mean-variance efficient frontier of Markowitz is a recurring theme in GBWM as the set of actions available to investors are represented by portfolio allocations that lie on the efficient frontier, and where the definition of risk is extended to the probability of not attaining an investment goal, rather than limiting it to the standard deviation of returns (Das et al., 2018). In addition, Das et al. (2018) propose a method to identify the portfolio that maximizes the probability of attaining a financial goal under a single-period framework. This is demonstrated through a series of numerical computations.

In the scope of multi period investment environment, Das et al. (2020) built on the optimal portfolio selection framework detailed in *A New Approach to Goals-Based Wealth Management* (Das et al., 2018) by introducing a dynamic programming (DP) algorithm to optimize the probability of attaining an investment objective. The optimal policy is derived using the Bellman equation which expresses the relationship between a state-action pair and the resulting expected future returns, hence allowing the agent to select the return-maximizing action for every state in the investment environment (Das et al., 2020). In GBWM, building a wealth grid is an essential step when attempting to dynamically optimize the sequence of actions

by backpropagating all the way to the initial period (Das et al., 2020). The wealth grid is time-homogenous and subject to appropriate minimum and maximum wealth levels, where all elements are equally spaced in their log-transformed form (Denault and Simonato, 2022). Das et al. (2020) propose a wealth grid construction method that assures the initial wealth value is included in the grid by exogenously selecting a wealth grid density parameter, and then following a series of numerical steps to fill the grid with values between its extrema. The wealth grid is sensitive to adjustments to its foundational hyperparameters such as the density parameter and the number of wealth levels it contains. An additional feature to the wealth grid was introduced by Denault and Simonato (2022) to stabilize and smooth the convergence of the optimal policy. The approach consists of assuring that the investment goal G is exactly halfway amongst the elements contained in the wealth grid by shifting the grid's values. For wealth grids that exclude the portfolio's initial wealth value, Denault and Simonato (2022) propose an interpolation method. This approach involves using neighboring values within the grid to provide additional flexibility to the grid's composition.

Das and Varma (2020) opened the door to RL methods applied to the problem set forth in Das et al. (2020) by employing a tabular Q-Learning method with the objective of converging to the solutions obtained from DP. The use of RL algorithms helps mitigate the impact of the curse of dimensionality as their model-free nature enables the extrapolation of solutions from resulting forward simulations (Das and Varma, 2020). Given that the number of calculations in DP can explode when increasing the number of wealth values, time increments, and actions, the use of RL was a favorable step toward developing a scalable approach to optimize a dynamic GBWM problem (Das and Varma, 2020).

An improved algorithm based on Das and Varma's formulation of the Q-Learning algorithm was introduced by Maxence Prémont (2021) with the objective of accelerating the speed of convergence and stabilizing success rates when testing the optimal policy. The improved algorithm consists of maintaining the structure of the Q-Learning approach employed by Das and Varma (2020) with the addition of gradually reducing the learning rate throughout the training process (Prémont,

2021). Adjustments to the learning rate were made by using different versions of decay functions; the exponential and linear decay functions were tested under both the state and exponential methods (Prémont, 2021). Additionally, Prémont (2021) attempted to replicate the results of the DP and Q-Learning methods by using a policy gradient (PG) approach given the numerous advantages that can arise from employing such methods. Policy-based methods estimate the optimal policy by approximating a parametric function that is representative of the environment's dynamics and specifications (Sutton and Barto, 2018). Using a PG approach allows to move past the dependency on a wealth grid, which was a limitation in previous methods given the grid's discrete nature and time-homogeneity. PG methods can function in continuous state spaces which is desirable in GBWM as financial markets are subject to continuous states. Another advantage of using PG methods is their ability to visit unlikely states that would be otherwise unvisited under the Q-Learning method, and approximate optimal actions in such situations (Sutton and Barto, 2018). Prémont applied a PG approach by initializing the weights of the model to pre-specified values. Despite the model's success in approximating the optimal policy in an oversimplified investment environment, it ultimately failed in a more realistic environment (Prémont, 2021).

While GBWM is intuitive and represents a viable investment approach for many, the use of a multi-goal approach where goals are competing amongst each other is closer to the reality experienced by various investors (Das et al., 2022). Multi goals wealth management (MGWM) can be an investment philosophy of interest for investors who foresee having to meet financial obligations on a periodic basis (e.g.: monthly mortgage payments) or for those who consider making several major purchases during a given time horizon (e.g.: buying a house and a boat in the next 10 years). Das et al. (2022) introduce a framework to optimize the sequence of investment decisions in a multi-goal investment environment with the objective of maximizing the investor's overall utility throughout the investment period. The dynamic optimization process to solve a MGWM problem proposed by Das et al. (2022) is utterly inspired by the original optimization framework developed for a GBWM problem (containing a single investment goal). More precisely, just like

the GBWM problem, the optimization process consists of backwardly propagating utility expectations in order to derive the optimal sequence of actions all the way to the beginning of the investment period. The goals' success probabilities derived from the optimization process take into account actions that reflect the investor's preferences. Such an approach has proven to outperform static portfolio strategies that are commonly used across the wealth management industry (Das et al., 2022).

3 A Goal-Based Wealth Management Problem with a Single Goal

Dynamic Programming was first introduced by Richard Bellman in 1954 in his publication entitled *The Theory of Dynamic Programming* (1954). Bellman developed the DP framework to solve mathematical problems composed of multi-stage decision processes (Bellman, 1954). While the development of the DP framework was an important step toward optimizing sequential decisions, when first developed, its level of practicality was scant given the lack of computational power to perform large amounts of repetitive calculations often arising when employing such frameworks. Since 1954, breakthroughs in computational capabilities have led DP methods to undergo numerous improvements that increased their rigor. The focus of this chapter is on elucidating the application of the DP framework to dynamically solve a multi-period GBWM problem with a single investment goal. A replication of results included in Das et. al (2020) is also detailed to have a clearer picture of the approach which will serve as a benchmark in later sections as additional methods will be subject to testing. In the description of concepts, a specific notation practice is maintained all throughout: vectors are labelled in bold characters and elements of finite sets are labelled using an upper index between parentheses. The number of elements in these finite sets are labelled by using the letter n, followed by a corresponding subscripted letter (e.g.: n_k).

3.1 The Multi-Period Optimization Problem

The GBWM optimization problem is subject to a finite time horizon composed of discrete time increments where the objective is to optimize $\mathbf{x}_t$, the fund's asset allocation, at different periods from a finite set of n_x different allocations. The allocations available to a given investor are a choice from n_a different assets that each have expected levels of returns $m^{(i)}$ and volatility $v^{(i)}$ where $i = 1$ to n_a. Given that the fund's allocation is reevaluated at the beginning of every period, the optimization problem is considered to be dynamic as the investor seeks to optimize the distribution of funds among the n_a available assets. The basic case of the optimization problem

focuses on the objective of attaining a wealth level G by the end of the investment term T by solely investing the fund's initial balance, therefore ignoring the option to consider cash inflows and outflows during the investment period. While the set of asset allocations (or actions) available to the investor are subject to stochastic periodic returns, each action is characterized by a constant expected return level $\mu^{(i)}$ and constant volatility $\sigma^{(i)}$ where $i : 0, ..., n_x$. At a given period t, an action is deemed to be optimal if it maximizes the investor's probability of attaining its goal G by time T, which can be expressed using the following expression:

$$\max_{\{\mathbf{x}_t\}_{t=0}^{T-h}} Pr(W_T > G), \tag{1}$$

where time increments of h years are denoted as $0, h, \ldots, n_t h$. The set of available asset allocations and the fund's terminal wealth are expressed as $\mathbf{x}_t$ and W_T, respectively. By repeating this condition all the way to the initial period, the sequence of optimal actions derived from backpropagating is expected to maximize the investor's success probability with respect to the investment objective. Additionally, the set of actions represented by unique asset allocations can be expressed as:

$$\mathbf{x}_t \in \{\mathbf{x}_t^{(1)}, \mathbf{x}_t^{(1)}, ..., \mathbf{x}_t^{(n_x)}\}. \tag{2}$$

Each allocation is defined by a set of expected mean and volatility measures $\{\mu^{(i)}, \sigma^{(i)}\}$ that lies on the mean-variance efficient frontier. Furthermore, the wealth level is assumed to follow a geometric Brownian motion defined as:

$$W_{t+h} = W_t e^{(\mu_t - \frac{1}{2}\sigma_t^2)h + \sigma_t \sqrt{h} Z_t}, \tag{3}$$

where μ_t and σ_t take the values of the expected mean and volatility of returns under the selected allocation $\mathbf{x}_t$. Return dynamics, μ_t and σ_t, are explicit functions of $\mathbf{x}_t$ and are under the control of the investor. The stochastic component in (3) is defined as a standard normal random variable Z_t.

3.2 The Benchmark Dynamic Programming Approach

The fully discrete muti-period DP algorithm requires the integration of a wealth grid in the policy optimization process (Das et al., 2020). The grid contains n_ω wealth levels, where the set of elements can be expressed as $\{W^{(1)}, W^{(2)}, ..., W^{(n_\omega)}\}$. Neighboring elements in the grid are equally spaced in their log-transformed form (Denault and Simonato, 2022). An important characteristic of the wealth grid is its time-homogenous nature throughout the investment time horizon.

When expressing transition probabilities associated with moving from one wealth point to another, $w^{(i)}$ represents the log-transformed wealth value of the agent at the time of the decision (time t), and $w^{(j)}$ represents the log-transformed wealth level to be potentially attained in the following period (time $t+h$).

Given the discrete and finite nature of the wealth grid, transition probabilities are calculated using the following expression (Das et al., 2018):

$$p^{(i,j,k)} = \frac{\phi(w^{(j)}|w^{(i)}, \mathbf{x}^{(k)})}{\sum_{l=1}^{n_\omega} \phi(w^{(l)}|w^{(i)}, \mathbf{x}^{(k)})}, \tag{4}$$

with the normal conditional density function $\phi(\cdot)$ being expressed as:

$$\phi(w^{(j)}|w^{(i)}, \mathbf{x}^{(k)}) = \frac{1}{\sqrt{2\pi}\sigma^{(k)}} exp\left(-\frac{\frac{1}{2}\left(w^{(j)} - w^{(i)} - \mu^{(k)}h + \frac{h}{2}(\sigma^{(k)})^2\right)^2}{(\sigma^{(k)})^2 h}\right), \tag{5}$$

In GBWM, the investor's primary focus is to attain its investment objective G. At the final period T, if the investment goal is attained, a return of one is earned by the agent, otherwise the return is zero. Regardless of whether the portfolio's balance meets or exceeds the investment objective at the end of the investment period, the return attributed will remain identical. Since it is not possible to know whether a portfolio has succeeded or fallen short of achieving a minimum value of G until the end of the time horizon, all returns are equal to zero for periods $t < T$. The value function at time T can therefore be expressed as:

$$V^{(T,i)} = \begin{cases} 0, & \text{if } W^{(i)} < G \\ 1, & \text{if } W^{(i)} \geq G. \end{cases} \tag{6}$$

To derive a sequence of optimal actions that maximize expected returns in subsequent periods, the Bellman equation can be utilized to determine the optimal action $\mathbf{x}^{(k)}$ to be taken in the current period (Denault and Simonato, 2022). The Bellman equation can be expressed as:

$$V^{(T,i)} = \max_{\mathbf{x}^{(k)}} \left[\sum_{j=1}^{n_w} V^{(t+1,j)} \times p^{(i,j,k)} \right], \tag{7}$$

As a result, for each state in the investment environment, an optimal action is derived which ultimately forms the investor's optimal policy denoted as $I^{(t,i)}$ for a portfolio worth $W^{(i)}$ at time t. By distributing expected returns in accordance with the Bellman equation and deriving the investment's optimal policy all the way to the initial period (time 0), the state value at time 0 will be representative of the probability of attaining the investment goal when following the optimal policy.

3.3 Case Study: A Single Goal Problem

This section takes a look at an application of the DP framework in a GBWM environment that is detailed in Das et al. (2020). In this numerical example, the state-space is subject to $n_\omega = 200$ attainable values bounded by $W_{\min}$ and $W_{\max}$. Their values are a function of the lowest expected return $\mu_{\min}$, highest expected return $\mu_{\max}$ and highest expected return volatility $\sigma_{\max}$ in the following expressions:

$$W_{\min} = W_0 e^{(\mu_{\min} - \frac{\sigma_{\max}^2}{2})T - 3\sigma_{\max}\sqrt{T}}, \tag{8}$$

$$W_{\max} = W_0 e^{(\mu_{\max} - \frac{\sigma_{\max}^2}{2})T + 3\sigma_{\max}\sqrt{T}}. \tag{9}$$

In Das et al. (2020), an analytical approach was used to determine the grid size which is not the case in this section's replication of results. The action-space vector $\mathbf{x}^{(k)}$ is composed of $n_x = 15$ actions, each representing a unique asset mix available to the investor, that lie on the efficient frontier. Allocations are subject to three distinct asset classes ($n_a = 3$): U.S. bonds, international stocks, and U.S. stocks.

Table 1: Asset characteristics table.

Asset class	**Mean return**	**Covariance of returns**		
U.S. bonds	0.0493	0.0017	-0.0017	-0.0021
International stocks	0.0770	-0.0017	0.0396	0.0309
U.S. stocks	0.0886	-0.0021	0.0309	0.0392

Table 2, below, displays the different allocations between the three asset classes detailed in Table 1 along with expected returns and volatility levels:

Table 2: Portfolio weights and expectations

	Portfolio Weights				
Portfolio number (k)	U.S. bonds	International stocks	U.S. stocks	**Mean return** ($\boldsymbol{\mu^{(k)}}$)	**Standard deviation** ($\boldsymbol{\sigma^{(k)}}$)
1	0.9098	0.0225	0.0677	0.0526	0.0370
2	0.8500	0.0033	0.1467	0.0552	0.0396
3	0.7903	-0.0160	0.2257	0.0577	0.0463
4	0.7305	-0.0352	0.3047	0.0604	0.0557
5	0.6707	-0.0545	0.3837	0.0629	0.0664
6	0.6110	-0.0737	0.4628	0.0655	0.0781
7	0.5512	-0.0930	0.5418	0.0680	0.0904
8	0.4915	-0.1122	0.6208	0.0706	0.1031
9	0.4317	-0.1315	0.6998	0.0732	0.1159
10	0.3719	-0.1507	0.7788	0.0757	0.1290
11	0.3122	-0.1700	0.8578	0.0783	0.1421
12	0.2524	-0.1892	0.9368	0.0809	0.1554
13	0.1927	-0.2085	1.0158	0.0834	0.1687
14	0.1329	-0.2277	1.0948	0.0860	0.1821
15	0.0731	-0.2470	1.1738	0.0886	0.1955

With the objective being to optimize the agent's sequence of decisions and invest in optimal asset mixes, all 15 actions are return-maximizing allocations for the amount of risk borne by the investor as they lie on the efficient frontier of Markowitz. In Figure 1, the efficient frontier is represented by the blue curve, and the points represent each of the 15 asset mixes (or actions) available to a given investor.

Figure 1: Efficient frontier illustration

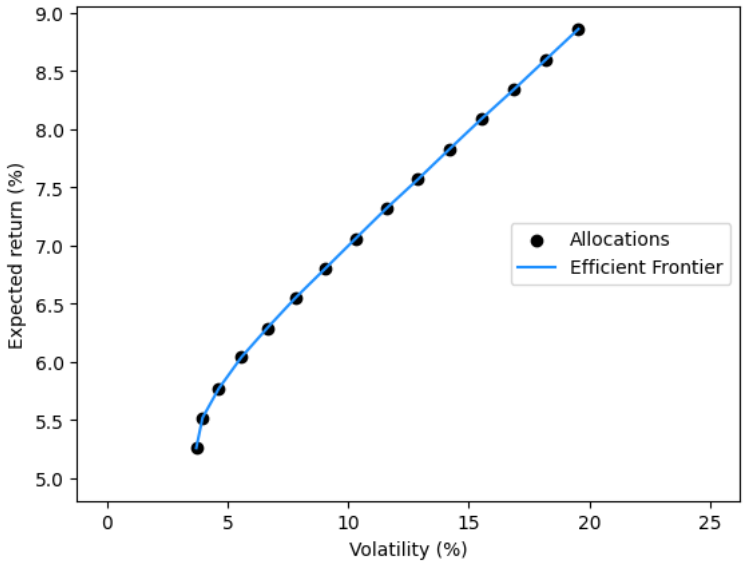


Figure 2: Heatmap of the optimal policy for a 10-year investment period

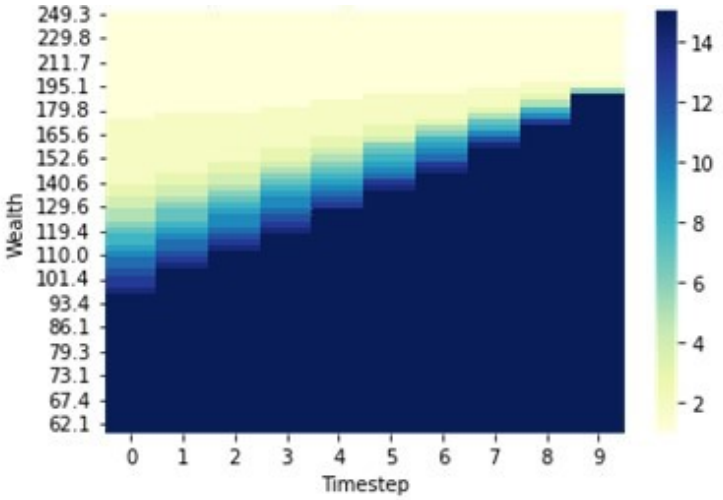
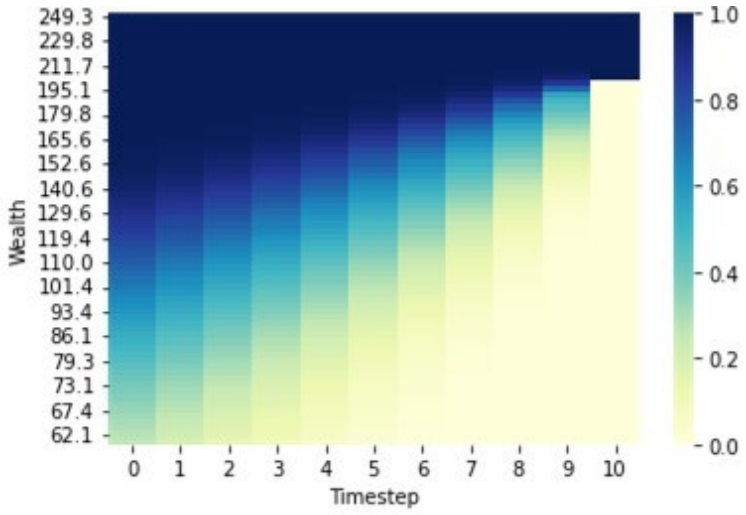


Figure 2 illustrates the optimal policy's evolution using a 10-year investment horizon where the investor's initial wealth W_0 and goal G are 100 and 200, respectively. The set of actions are displayed on the right-hand side of the figure. Dark colors represent riskier actions by the agent, while lighter colors represent conservative investment allocations. The action numbers are representative of the portfolio numbers in Table 2. As the agent approaches the end of the investment period, riskier actions are taken in order to maximize the fund's chances of attaining the goal if it is below the desired threshold. On the other hand, if the goal is attained prior to time 10, the agent's preferred allocations involve reduced risk levels in order

Figure 3: Heatmap of state values for a 10-year investment period



to minimize the likelihood of experiencing undesired decreases in wealth. Both of the above heatmaps show that for lower state values, the agent favors allocations that are increasingly riskier as it approaches the end of the investment period.

Obtaining the investment strategy's success rate is done using out-of-sample (OOS) simulations. In the scope of a GBWM problem, performing OOS simulations consists of simulating paths where returns follow a normal random variable defined by the optimal policy's choice of action (in the form of a probability distribution function) derived prior. The success rate among all paths generated in the OOS simulations is then used to evaluate the current policy's performance. The table below indicates the success rate for a given period-and-goal combination where a total of 100,000 OOS paths were simulated.

Table 3: Dynamic programming solutions

T	G	Success rate
10	200	0.6654

Since the agent's initial fund value does not appear in this example's wealth grid, neighbouring values below and above W_0 were used in the OOS simulations making it possible to use an interpolation function to obtain a solution for the desired starting point.

4 Policy Approximation in GBWM

This section focuses on replicating the results of Das et al. (2020) by employing a policy approximation approach to determine the optimal policy in goal-based investing. To achieve this, the REINFORCE algorithm is used to train the policy using a neural network replicating the investment environment detailed in Das et al. (2020). It is worth noting that this approach also involves revisiting some of the results and conclusions from *Reinforcement Learning Algorithms for a Dynamic Goal-Based Wealth Management Problem* by Maxence Prémont (2021). In Prémont's work, the PG REINFORCE algorithm was utilized to estimate a polynomial representation of the optimal policy which is replaced by a neural network in this thesis. By developing a well-implemented and robust PG algorithm, the environment's optimal policy can be derived efficiently while benefiting from the additional flexibility that neural networks provide. The potential upsides that can arise from the trials in this section make it a worthwhile pursuit in the domain of goal-based investing.

Policy approximation methods offer greater flexibility in stochastic forecasting, as they allow attainable states to span a continuous range, hence effectively eliminating the limitations associated with discrete and finite state spaces that are required in DP and Q-learning (Prémont, 2021). Their ability to generalize optimal actions to states not visited throughout the training process makes them applicable for real-world situations as financial markets are often not constrained by finite state spaces where unique market outcomes can always occur. Policy approximation methods can be trained using neural networks, which have been at the forefront of recent technological advances given their learning efficiency. They have the ability to include numerous customizations in their training process making them a useful tool that can be adapted to any learning environment. In addition, building and implementing neural networks can be done through the use of well-known Python libraries like *PyTorch* and *Tensorflow*, which provide generic training environments that can be adapted to various problems. As a result, such implementations can be efficiently developed while potentially excelling at estimating the optimal policy for

a wide range of problems.

4.1 Policy Gradient Approach

The objective of policy gradient algorithms is to adjust the policy's parameterization with respect to action selection in a way that maximizes the agent's expected returns (Sutton and Barto, 2018). To maximize performance, the model's parameters are updated by approximating the gradient ascent in the performance measure $J(\pi_\theta)$, whose value is essentially the state-value at the starting point (t=0) of an episode in the context of a GBWM problem (Sutton and Barto, 2018). The gradient form of the performance measure serves as the objective function (also known as the loss function) that guides the optimization process with respect to the model's set of parameters $\boldsymbol{\theta}$ (Sutton and Barto, 2018). More specifically, the gradient form of the performance measure $J(\pi_\theta)$ for a model using a PG approach is used to derive the optimized expected finite-horizon discounted returns of the policy:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\tau} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right], \tag{10}$$

where τ represents the number of trajectories (or iterations) performed by the agent in a given episode and $\pi_\theta(a_t|s_t)$ denotes the probability of selecting action a in state s at time t. The advantage function $A^{\pi_\theta}(s_t, a_t)$ is used to express the sum of discounted future returns from the agent's point of view at time t and can be reformulated as:

$$A^{\pi_\theta}(s_t, a_t) = \sum_{k=t+1}^{T} r_k \gamma^k, \tag{11}$$

with r_k being the return earned by the agent at period k, and γ^k being the corresponding discount factor.

Calculating the gradient of the performance measure $J(\pi_\theta)$ is a mathematical task that is automatically calculated by *PyTorch* or *Tensorflow* (depending on which of the two libraries is used to build a neural network). The model's parameters are updated according to the expression below once the stochastic gradient ascent in

the performance measure expressed in (10) is calculated:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_k). \tag{12}$$

The policy's parameters $\boldsymbol{\theta}$ are updated using a learning rate expressed as α. For a given optimization problem, a learning rate α can be kept constant or adjusted during the training process. While there are different approaches to adjust a learning rate, the objective always remains to enhance the model's learning ability throughout the training phase (Sutton and Barto, 2018).

4.2 REINFORCE Algorithm

The REINFORCE algorithm, also known as the vanilla PG method, is one of the most common algorithms used in policy approximation. The policy, characterized by a set of parameters $\boldsymbol{\theta}$, can be initialized with predetermined values or by random assignment which is a common practice in neural networks (de Sousa, 2016). The REINFORCE method enables the training of the policy's parametric function by generating Monte Carlo simulations in which the agent selects actions with respect to the probability distribution function outputted by the model. In each epoch, the performance measure's gradient is calculated and used to update the policy (Sutton and Barto, 2018). The policy's set of parameters is iteratively updated through this approach with the objective being to progressively improve performance and ultimately reach the optimal policy that maximizes the investor's likeliness of attaining the predefined investment goal.

A key component of the vanilla PG algorithm is its use of the cross-entropy method in the calculation of the log function, which consists of taking the logarithm of the action selection probability multiplied by the action's corresponding reward (Sutton and Barto, 2018). Using the expressions detailed in section 4.1, the REINFORCE algorithm goes as follows:

Algorithm 1 REINFORCE pseudo-code

i) Initialization of the policy ($\boldsymbol{\theta}$).

ii) Loop N times (determined prior to launching the training process):

1. Generate a Monte Carlo simulation of states and actions according to the current optimal policy $\pi(\cdot|\cdot, \boldsymbol{\theta}) : S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$.
2. Calculate the value of discounted returns for each time step.
3. Calculate the update component and apply it to the policy's parameters $\boldsymbol{\theta}$ using equation (12).

4.3 Artificial Neural Network Implementation

Neural networks are responsible for the engineering behind advanced artificial intelligence applications across a wide scope of industries and topics. The ease of access and efficiency of machine learning frameworks like *PyTorch* and *Tensorflow* (*PyTorch* was used for the production of this thesis) make them ideal tools to train a policy approximation model in the context of a GBWM problem. More specifically, these tools free users from performing complex tasks such as initializing the model's set of parameters $\boldsymbol{\theta}$ and calculating the performance measure's gradient, therefore offering a solution to some of the difficulties encountered by Prémont (2021) in his attempt to implement a PG approach to solve a GBWM problem. The use of feedforward artificial neural networks (ANN) allows the inclusion of several specifications, also known as hyperparameters, in the training process that can support the agent in its attempt to identify an environment's optimal policy. In other words, hyperparameters define the training environment in which the learning process takes place. Examples of hyperparameters are the number of layers and weights that form the neural network, learning rate, activation functions, batch size and exploration rate.

For tests performed in this thesis, a series of initial learning rates were tested to determine which one benefited the most the model's learning ability (results are available in Appendix - A). More specifically, the learning rate α is initialized prior to the launch of the training process and is optimized using the ADAM approach, which is an adaptive technique provided by the *PyTorch* library, whose objective is to increase the policy's speed of convergence towards the optimal policy during

training.

4.4 Improving Robustness in the Training Process

In the context of RL, robustness refers to the model's ability to perform its intended task, which is to approximate the optimal policy of a given problem. Robustness is an essential feature that is worth measuring and enhancing as it increases stability, effectiveness, and reliability across a wide range of applications. Within the context of this section which consists of solving a GBWM problem, the theme of robustness is narrowed to the model's ability to learn the optimal policy and the speed at which it does so.

4.4.1 Scaling of the Artificial Neural Network's Input Variables

In a GBWM investment environment, the agent's action selection heavily depends on the agent's temporal positioning and the fund's balance. Herein, time and the fund's balance serve as essential input variables for the model, which then results in a set of probabilities forming the probability distribution function used as reference by the agent to select an asset allocation (or action) at the beginning of the subsequent period. Neural networks typically require input variables to be scaled - a practice commonly referred to as *feature scaling* (Subasi, 2020). Feature scaling helps prevent model parameters from remaining in local extrema. By facilitating the agent's interpretation of its environment, feature scaling provides additional support to enhance the agent's ability to navigate within the environment (Subasi, 2020).

The most common scaling methods in neural networks are *normalization* and *standardization*. Normalization scales input variable by subtracting each raw value by the variable's minimum value and dividing the resulting by the difference between the variable's maximum and minimum values. When normalized, input variables take values between zero and one. A given input variable y is normalized using the following approach:

$$y^{\text{normalized}} = \frac{y - y_{\min}}{y_{\max} - y_{\min}}. \tag{13}$$

The implementation of a policy approximation approach to estimate the optimal policy in the scope of a GBWM problem gives the agent the flexibility to navigate through a continuous space of fund values across time increments. As a result, there are not any known upper and lower bounds that can be inputted in expression (13) to normalize a given fund value. To address this limitation, a large number of simulations using random action sequences can be generated to establish the theoretical values of $W_{\min}$ and $W_{\max}$ (to be inserted in (13)), enabling the normalization of the portfolio's wealth level which serves as an input variable to the model.

Standardization scales input variables by subtracting them by their mean value and dividing the resulting by their respective standard deviation. Once scaled, the set of values for each input variable has a mean of zero and standard deviation of one. A given input variable y is standardized using the following approach:

$$y^{\text{standardized}} = \frac{y - \bar{y}}{\sigma_y}, \tag{14}$$

with $\bar{y}$ representing the variable's mean and σ_y being its standard deviation.

Since the REINFORCE algorithm in an ANN environment trains the model without a wealth grid, finding the mean and standard deviation of the fund's wealth would be a complex task given the infinite amount of potential states. The complexity associated with identifying the mean and standard deviation in an effort to standardize the model's input variables makes standardization an unworthy scaling method for a GBWM problem. On the other hand, finding minimum and maximum attainable values, which are essential for normalization, can be obtained through a series of simulations. Variations in their values have little to no impact on the model's ability to learn the optimal policy.

While normalization may at first glance appear to be the superior approach to scale input variables in a goal-based investing context, a centering approach tailored for a GBWM problem is explored. Traditional centering consists of subtracting a variable's value by its mean. Once centered, variables have a mean of zero. Using a basic centering approach, a given input variable y is centered using the following

expression:

$$y^{\text{centered}} = y - \bar{y}. \tag{15}$$

Given the continuous nature of the environment's state space in PG algorithms, an adjusted centering method where variables are centered around a target value can be employed with the objective of avoiding the difficulties linked to estimating the fund's mean balance. In theory, to reach its investment objective by time T, the fund should follow a certain progression towards the goal throughout the investment period. By setting a specific target for a given period and centering the fund's value around it, the agent will have an indication of the fund's positioning as a function of time and the goal. The fund's balance and investment goal are initially adjusted by dividing both by the fund's initial value. This step is done in an attempt to better support the model in its learning of the optimal policy as such scaling approaches can help PG models avoid remaining in suboptimal local extrema (the chance that this occurs increases when large values are used to train a model). The scaled investment goal g_t and centered fund value W_t^{centered} are obtained using the following expressions:

$$G^* = \frac{G}{W_0}, \tag{16}$$

$$W_t^* = \frac{W_t}{W_0}, \tag{17}$$

$$g_t = G^{*\frac{t}{T}}, \tag{18}$$

$$W_t^{\text{centered}} = W_t^* - g_t. \tag{19}$$

The input variable relating to time, however, is centered using a traditional centering approach. Since it is known that decisions are made at the beginning of each period, the time variable is subtracted by the mean value of decision periods.

The times at which the agent is making a decision are stored in a vector $\mathrm{v}_{\text{decision}}$:

$$\mathrm{v}_{\text{decision}} = [0, 1, ..., T-1], \tag{20}$$

$$t_i^{\text{centered}} = t_i - \bar{\mathrm{v}}_{\text{decision}}. \tag{21}$$

Case Study: A Comparison of Scaling Methods

The method used to scale the model's input variables can have an impact on the training process' ability to converge toward the optimal policy. In the result comparisons conducted in this section, the policy's learning rate α^{policy} is initialized to 0.001. The learning rate's initialized value is selected according to a series of trials that were performed using different learning rates. The objective of the trials was to determine which learning rate enhanced the model's convergence rate (refer to Appendix-A).

To compare the performance of both scaling approaches, the model was trained five separate times under both approaches. To ensure that both methods could be compared on the same basis, the sequences of random numbers generated throughout the training processes were maintained by setting seeds in the scripts.

In order to normalize input variables on an identical basis across training processes, the fund's minimum and maximum attainable values, $W_{\min}$ and $W_{\max}$, are obtained using a fixated seed to regenerate, once again, an identical sequence of random actions and stochastic shocks to the wealth levels across all 100,000 simulated paths.

The policy's performance is evaluated using 10,000 OOS paths at a sequential frequency throughout the training phase. The figures below display the OOS success rate paths in cases where the model's input variables were scaled using normalization (Figure 4) and the adjusted centering approach (Figure 5).

Figure 4: Individual paths: success rate evolution using normalization to scale input variables

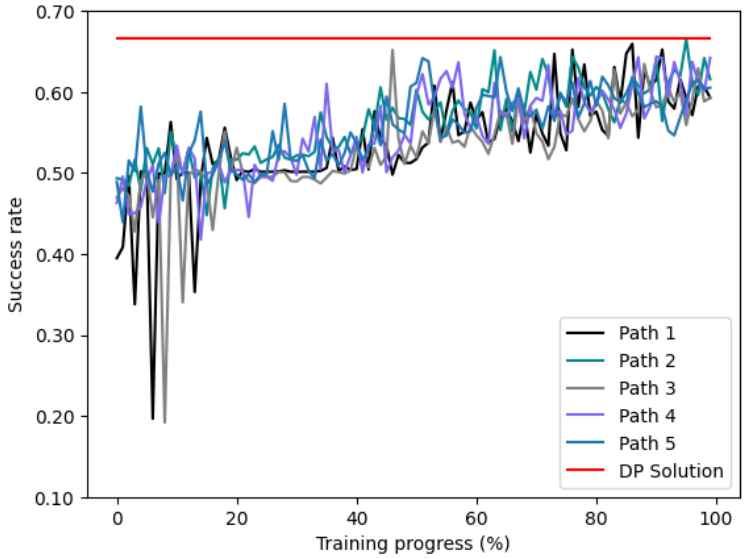
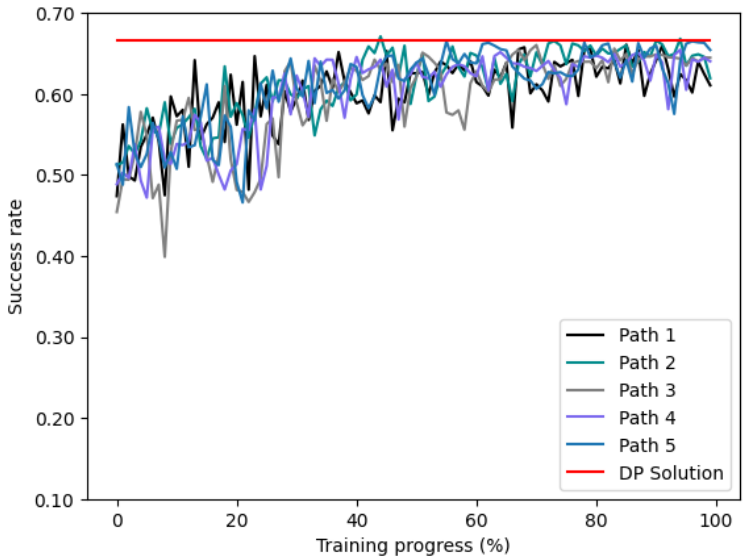


Figure 5: Individual paths: success rate evolution using an adjusted centering approach to scale input variables

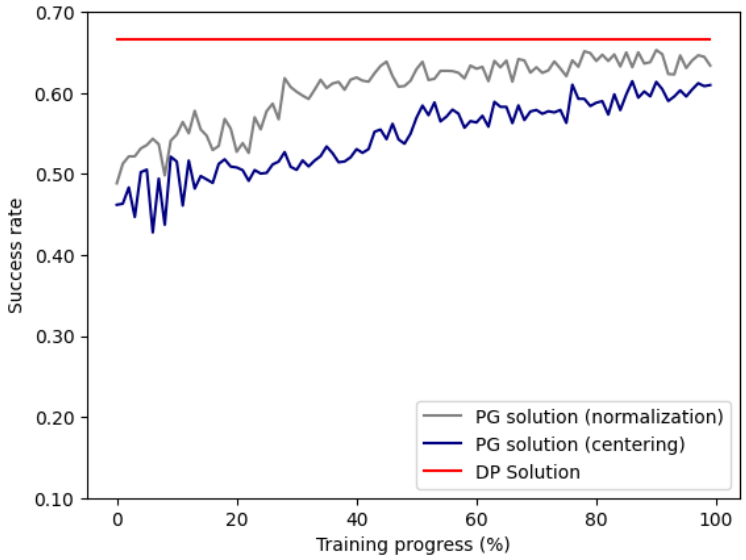


Figures 4 and 5 display different trends as a result of different scaling methods in the training process. In Figure 4, it is apparent that the various paths attain success rates that are close to the DP solution, which is considered the optimal point. In the final phase of the training process, the success rate paths remain volatile as they tend to fluctuate, meaning that the policy struggles to converge to an optimal level. In Figure 5, however, different observations can be made. Individual paths

appear to attain success rates similar to the DP solution sooner than in Figure 4. As the training process progresses, success rates also appear to be more stable, hence indicating a superior ability to converge towards the optimal policy. The average standard deviation in OOS success rate paths is 5.37% when using normalization and 4.83% when using the adjusted centering approach.

A smoother progression of simulated success rates can be achieved by averaging the success rates at each stage of the training processes displayed in Figures 4 and 5. Figure 6 displays the average success rates under both normalization and adjusted centering scaling methods as a result of their respective training process.

Figure 6: Success rate evolution



In Figure 6, it is clear that using an adjusted centering scaling approach results in a higher speed of convergence towards the optimal policy. In the final 20 percent of the training process, the average success rate under the adjusted centering method shows a somewhat constant progression as it has narrowed its gap with the DP solution, suggesting that the policy has converged. On the other hand, when using a normalization approach, it can be determined that the policy has not converged to the optimal policy after 50,000 training iterations given that the average of all five paths (represented by the blue curve in Figure 6) is still progressing towards the optimal policy established by the DP method.

Given the analysis detailed in this section, it appears that an adjusted centered

scaling approach results in a more efficient learning process in the context of a GBWM problem as it accelerates the model's learning of the optimal policy while maintaining stability.

4.4.2 Batch Training for a Goal-Based Wealth Management Problem

In addition to scaling input variables using a centering approach, batch training can be considered a potential source of enhancement that improves the model's speed of convergence towards the optimal policy (Hambly et al., 2023). Training in batches consists of generating multiple episodes (for each epoch) and then averaging the resulting loss functions in the calculation of the model's gradient. Batch training has been commonly used in training NN models given its supposed ability to better approximate the gradient in the steps leading to the weights' ($\boldsymbol{\theta}$) update (Hambly et al., 2023).

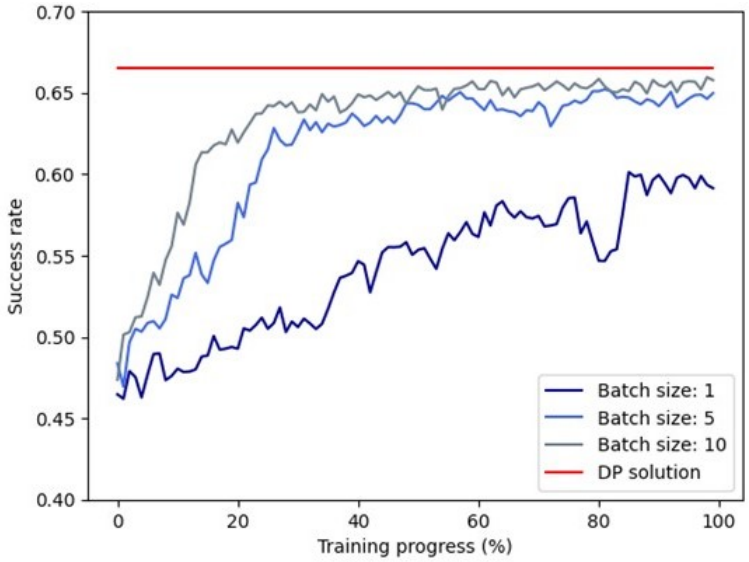
While batch training can improve stability and efficiency in the training process, there is a delicate balance to be considered with respect to the number of episodes generated for each batch and the total number of epochs used to train the model as longer than desired running times can arise when increasing the value of one or both hyperparameters.

Case Study: Comparing Results Under Different Batch Sizes

The effect of using batches in the context of a GBWM problem can be further analyzed by comparing results following 50,000 training epochs where batches of size 1, 5 and 10 are used. The resulting success rate evolution paths for different batch sizes can be observed in Figure 7.

Figure 7 illustrates the effect of larger batch sizes on the model's ability to convergence towards the optimal policy. An obvious observation is the little difference in the rate of convergence between training processes using batches of size 5 and 10. Given that batches of size 5 and 10 lead the model to learn the optimal policy at similar rates, a determining factor in establishing the superior batch size is their running times. While 50,000 epochs requires 21 minutes of running time when using batches of size 5, the running time increases to 37 minutes when raising the size of

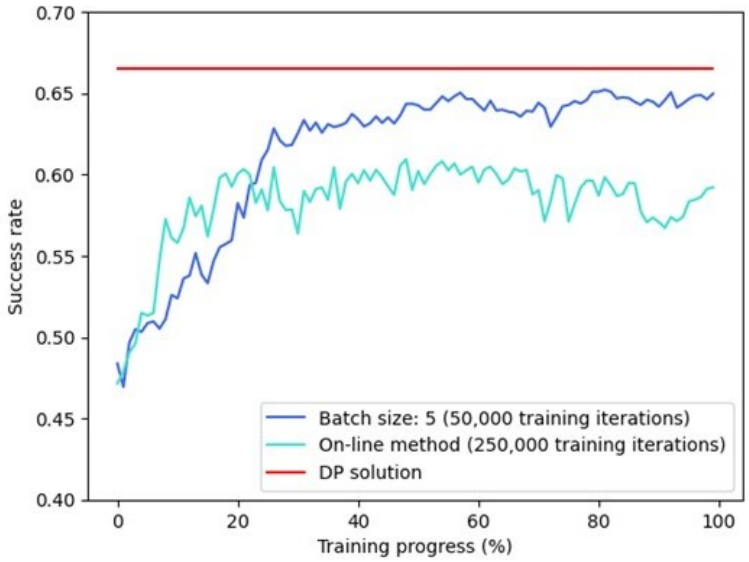
Figure 7: Success rate evolution for different batch sizes in training



batches to 10. The runtime burden that comes with using batches of size 10 proves to be inefficient considering the marginal improvement in the model's convergence towards the optimal policy.

While the policy's success rate evolution path under an on-line approach (batches of size 1) consistently lies below the trajectory of OOS success rates using batches of size 5 after 50,000 training epochs, it remains unclear which of the two approaches yields superior performance. To improve the basis of comparison between training using an on-line approach and batches of size 5, the on-line method is used with 250,000 training epochs in order to have both approaches generate the same number of episodes (250,000 each). Figure 8 illustrates the policy's evolving dynamics under both the on-line approach (with 250,000 training epochs) and training using batches of size 5 (with 50,000 training epochs):

Figure 8: Success rate evolution for training using batches of size 5 and the on-line method



In the above figure, it can be observed that batches of size 5 improve the model's ability to converge toward the optimal policy. Despite using a larger amount of training iterations, an on-line approach displays a lower ability to converge toward optimality when compared to batch training, where the model is able to approach the benchmark solution within 60% of the training progress.

4.4.3 Training with a Baseline

In reinforcement learning (RL), the learning process can be accelerated and stabilized (subject to less variability) through the inclusion of a baseline in the model's training process. As detailed in section 4.1, the policy's update is performed by calculating the gradient of the performance measure:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\tau} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right]. \tag{22}$$

The use of a baseline, however, alters the calculation of the advantage function, consequently impacting the policy's update. For a given state s_t, the advantage function is defined as the sum of discounted future returns (total rewards earned by

the agent), minus the baseline's state value $b(s_t)$:

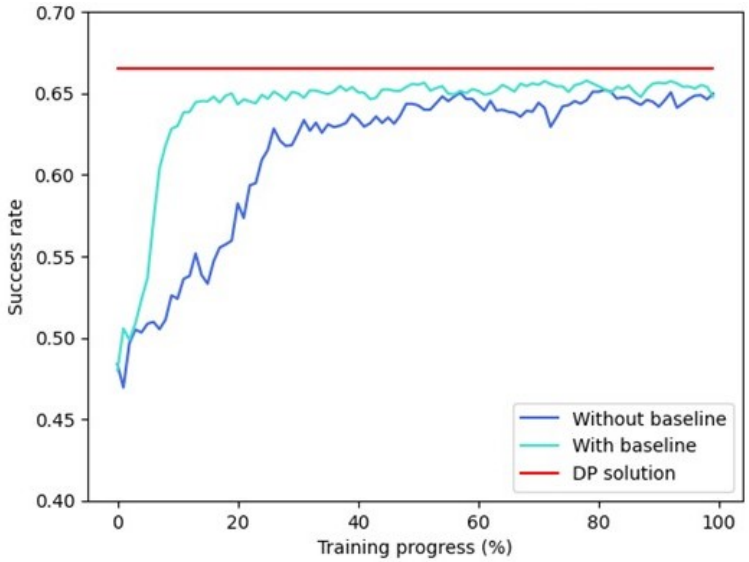
$$A^{\pi_\theta}(s_t, a_t) = \sum_{k=t+1}^{T} r_k \gamma^k - b(s_t). \tag{23}$$

Baselines are often used as an estimate of the policy model's state value, and therefore act as a benchmark solution to the model. The use of baselines in policy approximation training processes has been found to often reduce variance in policy evolution trajectories. By subtracting the estimated state value from the rewards obtained in a training epoch, the model's weights are adjusted in a way that accelerates the model's path towards the optimal policy. Additionally, the baseline is trained using a distinct update function. It calculates the gradient of the mean-loss errors (MSEs) obtained from simulated rewards (observed value) and the baseline's predicted values (forecasted value). Just like the model's policy, all steps leading to the baseline policy's update have the benefit of being performed using the tools available in *Pytorch*.

Case Study: Assessing The Impact Of A Baseline on Model Convergence

In an effort to build on the series of tests previously conducted in this section, the effect of the baseline is tested using batches of size 5 and 50,000 training epochs. Figure 9 illustrates the average success rate evolution of five training processes under both with and without a baseline in the training process. The learning rates of the policy's model and baseline are initialized to 0.001 and 0.002, respectively. Both learning rates are optimized throughout the training phase using the ADAM optimization tool in *PyTorch*. Seeds were set in the scripts to ensure an equal comparison basis of results:

Figure 9: Success rate evolution: baseline comparison



The baseline shows obvious signs of improvement to the model given an accelerated ability to converge towards the optimal policy. The line representing the model that includes a baseline (turquoise line) approaches the benchmark solution established by the DP method within 20% of the training process, which is an outright improvement compared the blue line representing the model trained without a baseline. Besides resulting in a superior speed of convergence relative to the model trained without a baseline, the use of a baseline consistently generates solutions that are closer to the benchmark solution, displaying an ability to generate stable solutions as the training phase progresses. Additionally, incorporating a baseline in the training process results in slightly higher runtimes than training processes that exclude the use of a baseline. A training process using a baseline and performing 50,000 epochs results in a runtime of 27.5 minutes, up from 21 minutes when excluding a baseline. Considering the increased speed of convergence and stability in results, the use of a baseline proves to enhance performance across all key measures.

4.4.4 An Attempt to Improve Convergence Through Exploration

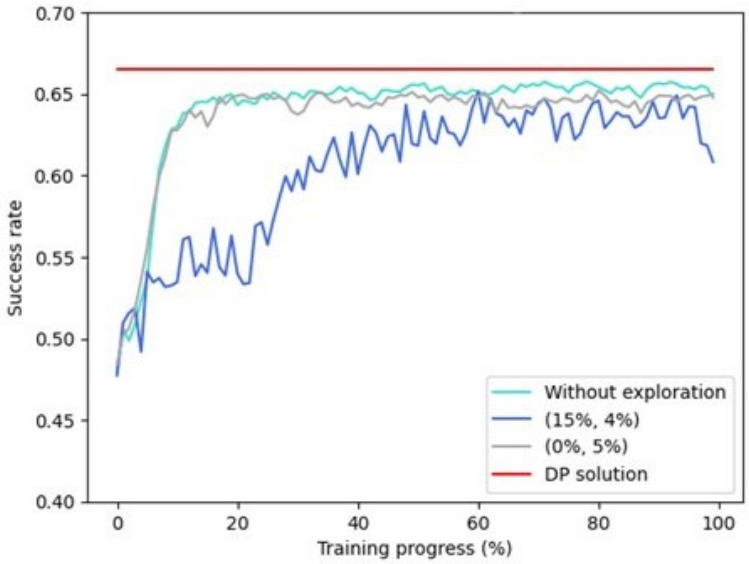
In RL, determining the adequate exploration rate for training the model can be a complex task. While the agent learns as training epochs are performed and the model's policy improves, exploration remains an integral source of consideration in

order to support the agent in its search for potential reward maximizing actions. It can occur that the agent finds itself trapped in a local optima that is suboptimal in the overall scheme. Exploration can therefore help the agent identify an improved set of parameters by allowing it to explore actions that would not be selected if the greedy action was always the one of choice (Sutton and Barto, 2018). Exploration can also help establish a generalized optimal policy where the agent is able to select reward maximizing actions when encountering a state with no prior visit (Sutton and Barto, 2018).

Case Study: Adding Exploration in the Training Process

To support the model's comprehension of the environment in which it navigates, the early stages of the training process can be solely focused on exploring the environment. Once the exploration phase is completed, an ϵ-greedy policy can be used to continue exploring the environment on an occasional basis. In other words, under an ϵ-greedy policy, the model selects its preferred action with a probability of (1-ϵ)%. The base case consists of having the first 15% of the model's training process be in full exploration mode, and apply a 4% ϵ-greedy policy afterwards. The model is trained using 50,000 training epochs where each epoch generates batches of five episodes. The model was trained using the same model specifications and procedures as in the case study of section 4.4.3 (including a baseline) with the addition of exploration. Figure 10, below, compares success rate paths for methods with and without exploration in the model's training process. (X%, Y%) combinations are to be interpreted as such: the inital X% of the model's training process is subject to a constant exploration of the environment (referred to as the exploration phase), while Y% defines the ϵ-greedy policy (Y% indicates the percentage of actions that do not take the greedy action once the initial exploration phase is completed).

Figure 10: Success rate evolution: baseline comparison



It can be observed in Figure 10 that the model using a pure exploration approach in the initial 15% of the training process (blue line) significantly underperforms the model without any exploration (turquoise line) and the one using a 5%-greedy policy (grey line). As for the model without exploration and the one using a 5%-greedy policy, the speed of convergence is highly similar for both models, making it difficult to identify either one as superior to solve a GBWM problem.

5 A Goal-Based Wealth Management Problem with Multiple Goals

The focus of this section is on applying the REINFORCE algorithm to solve a multi goals wealth management (MGWM) problem. In comparison to GBWM where there is a single investment objective that is used as a reference point in the training of the policy, MGWM brings a new complexity to the optimization process as the agent is faced with the option to purchase or forego a goal-oriented item with the focus being on maximizing the investor's total utility. In the context of MGWM, goal-oriented items encompass a variety of consumable products or services.

The policy is derived with respect to maximizing the investor's total expected utility throughout the investment period. More precisely, an investor will seek to maximize its probability of attaining a certain wealth level at specific time periods in order to purchase goal-oriented items that will result in utility gains if purchased (Das et al., 2022). This differs from the approach used for a GBWM problem, where the policy is optimized with respect to maximizing the probability of attaining a single predetermined investment goal G. While the REINFORCE implementation was successful in estimating the optimal policy in a GBWM environment (as shown in section 4), the additional complexities to the training process in MGWM led to obtaining suboptimal solutions when reusing a similar training environment to the one in section 4. While the resulting solutions make general sense as they are somewhat aligned with benchmark solutions, the optimization process appears to be lacking rigor as utility levels are trailing their benchmark. Given the time limitations of this thesis, the research process to develop a framework leading to an optimized policy in the context of a MGWM problem remains incomplete.

5.1 The Multi-period Optimization Problem for Multi Goals Wealth Management

The investment environment studied by the agent is composed of a finite set of n_b investment objectives that represent a set of purchasing costs for corresponding goal-

oriented items. Individual item costs are expressed as $C^{(i)}$ and utility levels earned by the investor at the moment of purchase are expressed as $U^{(i)}$, where i= 1 to n_b. As it is the case for a GBWM problem with a single investment goal, the policy is optimized using a finite set of available asset allocations $\mathbf{x}_t$ (homogenous across periods) that are defined by a constant expected return level $\mu^{(i)}$ and volatility $\sigma^{(i)}$ with $i = 1$ to n_x. Given the agent's option to purchase goal-oriented items, the number of actions expands to $2n_x$. More specifically, the first n_x actions forego any purchase, and the following n_x actions are used to complete purchases. Asset allocations, again, are subject to n_a different assets where each asset's expected return and volatility dynamics are expressed as $m^{(i)}$ and $v^{(i)}$, respectively, where i = 1 to n_a.

As detailed in section 3, return dynamics are defined by sets of expected returns and volatility pairs $\{\mu^{(i)}, \sigma^{(i)}\}$ that lie on the efficient frontier of Markowitz. Wealth levels continue to follow a geometric Brownian motion in the simulated trajectories. If the agent selects an action resulting in a purchase (assuming that the fund is appropriately funded), the fund's wealth trajectory is subject to the following expression:

$$W_{t+h} = (W_t - C_t)e^{(\mu_t - \frac{1}{2}\sigma_t^2)h + \sigma_t\sqrt{h}Z_t}, \tag{24}$$

where μ_t and σ_t are the expected return and volatility associated with the action selected by the agent. C_t takes the value of the purchase cost at time t. If the agent forgoes the opportunity to purchase any goal-oriented item at time t, C_t takes a value of zero.

5.2 The Benchmark Dynamic Programming Approach for a MGWM Problem

The approach to establish the wealth grid's maximum and minimum values, $W_{\max}$ and $W_{\min}$, slightly differs from the approach used for a GBWM problem, as a MGWM problem requires taking purchasing costs into account. In a similar sense, transition probabilities must take into account the option of purchasing goal-oriented

items all throughout the investment period. In the realm of a MGWM problem, normalized transition probabilities are therefore expressed as (Das et al., 2022):

$$p^{(i,j,k)} = \frac{\phi(w^{(j)}|w^{(i)} - c^{(k)}, \mathbf{x}^{(k)})}{\sum_{l=1}^{n_\omega} \phi(w^{(l)}|w^{(i)} - c^{(k)}, \mathbf{x}^{(k)})}, \tag{25}$$

with $c^{(k)}$ being the log-transformed purchasing cost in the current period for action k, $w^{(i)}$ being the log-transformed wealth at the time of decision (time t), and $w^{(j)}$ being the log-transformed wealth at the subsequent period $(t + h)$. The vector of allocations k selected by the agent at time t is expressed as $\mathbf{x}^{(k)}$. Given that the log-transformed value of zero is undefined, $c^{(k)}$ is automatically set to zero if no cost is borne by the investor at the time of decision. As a result of including purchasing costs in the computation of transition probabilities, the normal density function is defined as:

$$\phi(w^{(j)}|w^{(i)}, \mathbf{x}^{(k)}) = \frac{1}{\sqrt{2\pi}\sigma^{(k)}} exp\left(-\frac{\frac{1}{2}\left(w^{(j)} - (w^{(i)} - c^{(k)}) - \mu^{(k)}h + \frac{h}{2}(\sigma^{(k)})^2\right)^2}{(\sigma^{(k)})^2 h} \right). \tag{26}$$

With different investment goals allowing the purchase of various items and services, the distribution of returns in the training process differs from the dynamic optimization process detailed in section 3.2 for goal-based investing with a single goal. The agent earns a return when a purchase is completed, with the return being equal to the level of utility $u(t)$ earned as a result of a purchase at time t. An outstanding difference between goal-based investing with single and multiple goals is that a MGWM environment allows the agent to earn returns in intermediate periods whenever a purchase is completed.

5.2.1 Case Study: A Benchmark Solution for a Multi Goals Wealth Management Problem

This section has the objective of deriving a series of optimal policies for different sets of utilities whose solutions will serve as benchmarks in numerical case studies involving the REINFORCE algorithm for a MGWM problem.

The investment environment used in this numerical experiment is subject to a

wealth grid with $n_\omega = 200$ values. The action-space vector $\mathbf{x}^{(k)}$ includes the same 15 allocations from section 3.3 with and without the option to purchase (bringing the total number of actions to 30). Additionally, two investment goals are included in the investment environment. The below table details the different times and costs associated to the different goal-oriented items available for purchase to the investor.

Table 4: Costs

$\boldsymbol{t}$	$\boldsymbol{C(t)}$
5	100
10	150

While the timing and costs associated with the purchase of items remains static in the scope of this case study, utility levels $u(t)$ are subject to various values to help analyze different investment behaviours by the agent. The below table details the investor's resulting expectations with respect to utility levels and success probability dynamics following 100,000 OOS paths under each scenario:

Table 5: Dynamic programming solutions

Scenario	$u(5)$	$u(10)$	E*[u]	P($W(5) \geqslant 100$)	P({purchase at t=5})	P($W(10) \geqslant 150$)
1	1000	1000	1164.6	0.8680	0.8680	0.2966
2	1000	2000	1895.9	0.9241	0.1260	0.8849
3	1000	3000	2817.6	0.9721	0.0167	0.9336
4	2000	1000	2095.5	0.9630	0.9630	0.1696
5	2000	2000	2334.3	0.8699	0.8699	0.2972
6	2000	3000	2938.4	0.8466	0.2630	0.8041
7	3000	1000	3072.7	0.9839	0.9839	0.1209
8	3000	2000	3242.8	0.9402	0.9402	0.2111
9	3000	3000	3499.2	0.8708	0.8708	0.2957

The second and third columns in Table 5 are utility levels at times 5 and 10, respectively, inputted prior to the optimization process, and contain nine different scenarios to better understand the optimal policy under varying settings. Solutions obtained following the OOS simulations as a result of the optimized policy and are displayed in columns 4 and on. The above table clearly indicates the impact of utility

levels on the probabilities of purchasing a given item. When desired items in further periods result in higher utility attainment by the agent, these goals are prioritized, hence leading earlier goals to be ignored if this jeopardizes the investor's chances of attaining the preferred objective. On the other hand, when desired items in earlier periods provide the investor with additional utility, these goal-oriented items are prioritized (hence purchased) by the investor, and the later items are purchased if the investment performance allows for the purchase of the desired items. Figures 11 and 12, below, present a narrowed look of scenarios 2 and 4 (from Table 5) to better comprehend the investor's course of actions:

Figure 11: Heatmap of the optimal policy: scenario 2

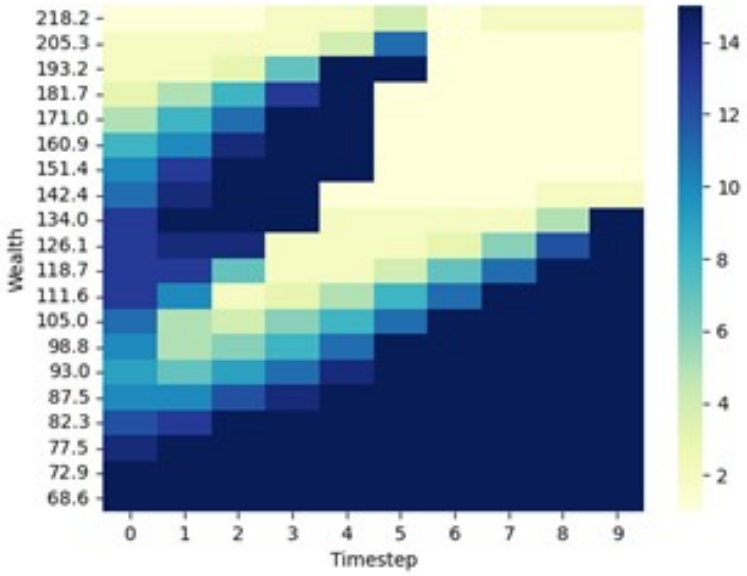
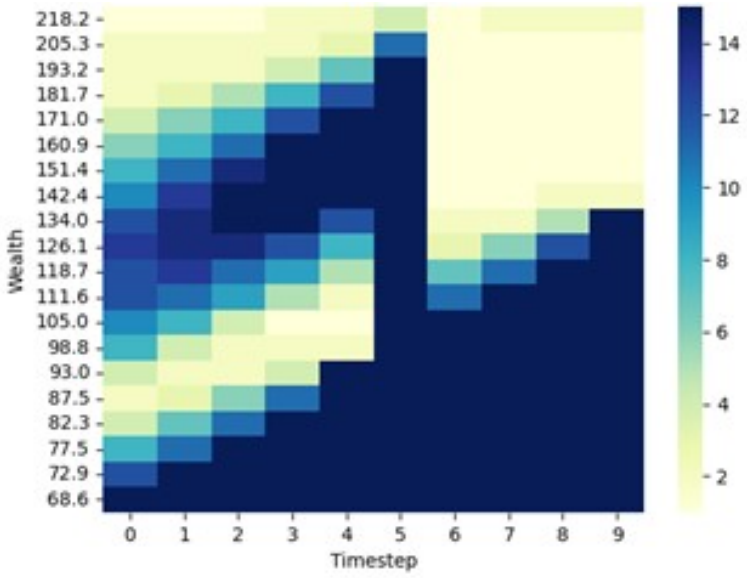


Figure 12: Heatmap of the optimal policy: scenario 4



In comparing Figures 11 and 12, an outstanding difference is the investor's utility maximizing actions at the end of the fourth period. Given that scenario 4 offers greater utility for the item at time 5, the investor would prioritize attaining the investment objective associated with the item to move along with the purchase, which explains the investor's willingness to take on additional risk leading up to the fifth period. As the agent gets closer to the second (and final) investment objective, optimal actions are rather similar under scenarios 2 and 4; aggressive actions are taken if the portfolio's balance is below the minimum threshold while more conservative actions are taken if the portfolio is well-funded.

5.3 A Policy Gradient Approach with REINFORCE to Solve a Multi Goals Wealth Management Problem

While the benchmark solution was established in section 5.2, the ultimate objective from a testing point of view is to solve a MGWM optimization problem using a PG approach in conjunction with the REINFORCE algorithm. While the PG approach was able to efficiently approximate the optimal policy when the investment environment was subject to a single investment objective, the model's ability to approximate the optimal policy for a MGWM problem remains in question given the

added complexity coming from the agent's option to purchase multiple goal-oriented items.

5.3.1 Scaling of the Artificial Neural Network's Input Variables

Section 4.4 put forth an adjusted centering scaling approach tailored for GBWM, ultimately increasing the model policy's convergence towards the optimal policy. The centering approach was viable given that a single investment objective formed the training environment and acted as one of only two guiding factors for the model throughout the training process. In cases where multiple goals are included in the investment environment, using a centering method that scales input variables in consideration of the various wealth objectives can be a non-trivial task. As a result of the difficulties that surface from attempting to use a centering scaling method in the context of a MGWM problem, employing a normalization approach to scale input variables emerges as the most suitable technique.

5.3.2 Case Study: Attempt to Solve a Multi Goals Wealth Management Problem Using the REINFORCE Algorithm

Applying the REINFORCE algorithm to solve a MGWM problem and comparing the resulting solutions to the benchmark solutions established using the DP approach will provide information on the implementation's ability to estimate the optimal policy.

The boundaries used as scaling parameters to normalize inputted portfolio balances, $W_{\min}$ and $W_{\max}$, were obtained by generating 100,000 paths under a random action selection process. Given the agent's option to purchase, $W_{\min}$ was subject to negative values, which would be replaced by a wealth level of zero as the portfolio is not subject to negative wealth levels. Since purchases shift the portfolio's value downward, the value of $W_{\max}$ was obtained by rejecting the option to purchase any of the targeted items in order to obtain the highest attainable portfolio value. Also, as done for a GBWM problem, a seed was used to ensure that $W_{\min}$ and $W_{\max}$ maintain their values in each training process in order to assure that results are compared to each other on the same basis.

A single neural network is employed to train the optimal policy, thus the model is responsible for optimizing decisions related to asset allocation and the choice of purchasing or foregoing a given goal-oriented item. If the portfolio is adequately funded to complete a purchase, the model will consider each of n_x possible asset allocation, both with and without purchase (therefore all $2n_x$ actions). On the other hand, if the portfolio is not able to afford a targeted item, only actions foregoing the option to purchase are considered (only n_x of the $2n_x$ actions). As it was the case in the tests using an ANN in section 4, the policy network uses a learning rate of 0.001 while the baseline uses a learning rate of 0.002. Given the added complexity that comes with having multiple goals throughout the investment period, the model's policy is training using 100,000 training epochs (rather than 50,000 as done for section 4) with each epoch generating batches of five simulated paths. Additionally, the training process is subject to an 5%-greedy policy (5% of actions taken by the agent being random).

In order to test the model's policy progression towards the optimal policy, 10,000 OOS simulations are generated to determine the model policy's performance at various points of the training phase. The set of weights that generated the highest level of expected utility across OOS simulations was saved and considered to be optimal for each of the nine scenarios of interest:

Table 6: Policy Gradient with REINFORCE solutions

Scenario	$u(5)$	$u(10)$	E*$[u]$	P$(W(5) \geqslant 100)$	P($\{$purchase at t=5$\}$)	P$(W(10) \geqslant 150)$
1	1000	1000	1082.1	0.8189	0.8189	0.2632
2	1000	2000	1792.8	0.9961	0.0136	0.8896
3	1000	3000	2639.4	0.9906	0.0000	0.8798
4	2000	1000	1998.7	0.9992	0.9992	0.0003
5	2000	2000	2163.8	0.8360	0.8360	0.2459
6	2000	3000	2790.7	0.8231	0.3143	0.7207
7	3000	1000	2999.9	0.9983	0.9983	0.0050
8	3000	2000	3050.4	0.9734	0.9734	0.0651
9	3000	3000	3265.2	0.8586	0.8586	0.2298

Table 6 (continued)

Scenario	P({purchase at t=10})
1	0.2632
2	0.8896
3	0.8798
4	0.0030
5	0.2459
6	0.7207
7	0.0050
8	0.0651
9	0.2298

The fourth column in Table 6 displays the agent's highest expected utility level, denoted as $E^*[u]$, from the series of OOS simulations conducted throughout the training process of all nine tested scenarios. Its results indicate that the training process, as it is defined in the context of a MGWM problem, is unable to lead the model's policy to optimality, which is responsible for the lower levels of expected utility earned by the agent than the ones presented in Table 5.

Figure 13: Expected utility progressions against benchmark solutions: scenarios 1, 2 & 3

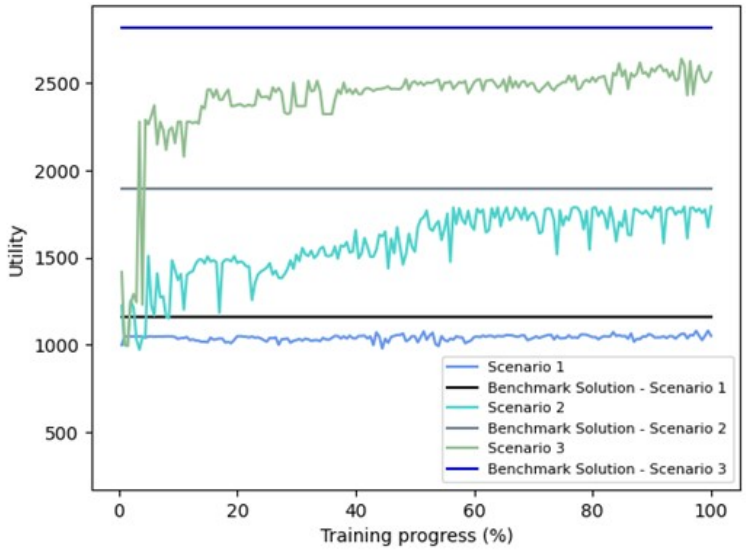
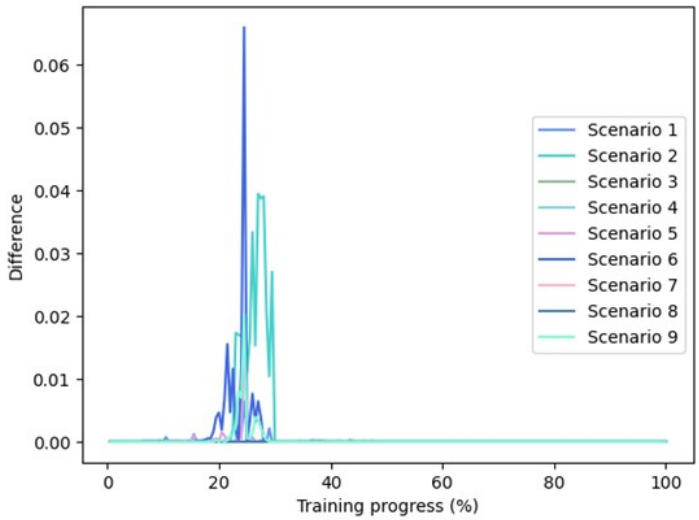


Figure 13 illustrates expected utility progression paths for scenarios 1, 2 and 3 against their respective benchmarks (which are DP solutions). For all three demonstrated scenarios, utilities derived from the model's optimal policy lay below the benchmark solutions established by the DP framework. While the model's policy appears to improve in the initial half of the training process for scenarios 2 and 3, the trajectories appear to be relatively stagnant for the remainder of the training process. The training process appears to help the model improve its decision-making within the investment environment it navigates, but struggles to converge to expected utility levels when following the optimal policy obtained through the DP method (serving as the benchmark). A source of the model policy's underperformance relative to the benchmark solution can be attributed to the model's inability to successfully derive optimal actions with respect to purchasing the first of the two goal-oriented items. The training process, however, is able to recognize the importance of purchasing the final item if the portfolio is adequately funded. This demonstrates that the model is able to learn the optimal action when the decision in question is for the most part obvious (since purchasing the final goal-oriented item is the obvious optimal action as it guarantees to increase utility). On the other hand, when there is more ambiguity around the optimal action, as it is the case for intermediate time periods, the model policy's action selection remains suboptimal.

Figure 14: Difference between attaining and purchasing probabilities at time 10



The path evolution of the difference between the probabilities of attaining and purchasing the final item at time 10 is illustrated in Figure 14 for all scenarios. Values different from zero are an indication that the model struggles to identify the optimal action to take at the final period. The two last columns of Table 6 provide valuable insight as far as coming to the observation that the model favors actions that purchase the targeted item at the final period if the fund can bare the cost. The fact that differences between attaining and purchasing probabilities at time 10 are zero in the entire second half of the training process for all studied scenarios leads to the determination that the model is able to learn the optimal action sequence at the final period.

5.4 An Attempt to Optimize the Model's Policy Through Additional Exploration

The results in section 5.3 show that under the current implementation of a policy approximation approach to solve a MGWM problem, the model's ability to converge toward the optimal policy is limited. While the model's policy still benefits from the training process as it is shown to approach the optimal policy to a certain extent, its inability to optimize its decisions relating to purchases in intermediate periods can be an important factor for its shortfall relative to benchmark solutions. Given the added complexity to the investment environment that comes with the inclusion of goal-oriented items throughout the investment period, it is a possibility that the model's parameters remain trapped in a local optima and struggle to identify an optimal parametrization within the global domain. Given the low amount of exploration permitted by the training process in section 5.3 (5% greedy policy), the lack of exploration could explain the model's inability to escape suboptimal positioning when calculating the performance measure's gradient.

5.4.1 Case Study: Adding Exploration to the Training Process

To address the questions around exploration and the overall objective of supporting the model in its training process to converge towards optimal levels, increasing the

amount of exploration in the training process can be a viable option in supporting the model expand its knowledge of the environment. Essentially, the objective is to improve the model's decision-making within the investment environment that is defined. To increase exploration, the initial exploration phase will result in the agent selecting random actions in the initial 15% of the training process, and then a 5% ϵ-greedy policy will come into effect for the remaining training epochs.

Figure 15: Expected utility progressions against benchmark solutions: scenarios 1, 2 & 3

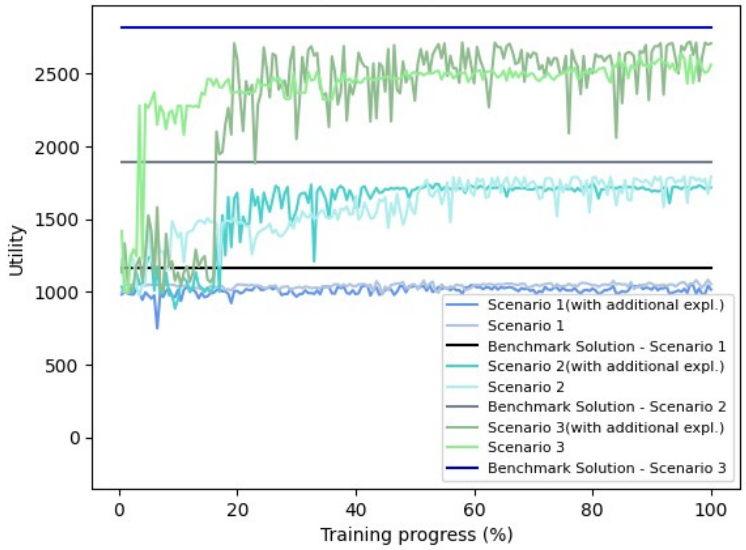


Figure 15 illustrates the evolution of expected utility levels for scenarios 1, 2 and 3 all throughout the training process of the MGWM problem under the different exploration approaches used to train the model against their respective benchmark solution. The curves representing OOS success rates for models trained using a pure exploration approach in the initial 15% of the training phase do not display improved convergence compared to models trained with a standard 5%-greedy policy. It is not possible to draw any conclusion on the stability of results as neither of the two approaches generate more stable solutions across all observed scenarios. As shown in Figure 15, the additional exploration for scenario 3 results in high variations of the policy, while the opposite is true for scenarios 1 and 2 as training progresses.

6 Conclusion

The work presented in this thesis has the objective of developing a robust policy approximation framework to solve a GBWM problem. More specifically, while reinforcement learning algorithms have been used to solve GBWM problems in previous research projects, the environment in which the policy was trained was subject to a number of limitations, opening the door to additional considerations with respect to the use of a policy approximation method to solve the main drawbacks of previous applications.

First, the implementation of a Policy Gradient approach using the REINFORCE algorithm within an artificial neural network proved to be a capable training method that led the model to learn the optimal policy and ultimately solve a GBWM problem. Various concepts were used and combined to optimize the training process' efficiency, essentially supporting the model's path towards the optimal policy. This included adapting the scaling of parameters accordingly, using a baseline, and adding exploration to the training process. In fact, two scaling approaches were developed to better support the model's training within an artificial neural network environment. The first approach focused on generating theoretical minimum and maximum portfolio values to allow the normalization of the fund's value as an input variable. The second scaling approach, which proved to significantly increase the speed at which the policy converges towards optimality, is a centering approach tailored to solve a GBWM problem. In addition to enhancing the policy's convergence rate towards the optimal policy, the adjusted centering approach proved to increase stability in the process. The additional efficiency from utilizing such an approach increases its usability for investors and decision-makers in the financial sector.

Second, with the application of a Policy Gradient approach in the scope of a GBWM problem being successful, the focus shifted to extending such an application onto a MGWM problem, where multiple investment goals can be included in the environment. While the agent is aware of the goals that are at its disposal, the selection of actions is further complexified as the objective is to maximize overall utility throughout the investment period. A single neural network was utilized to

estimate the optimal policy with respect to selecting the appropriate asset mix for the investor and the option to purchase goal-oriented items. While the training process showed an ability to improve the policy, it was unable to bring the trained policy to converge to the benchmark solution. Additional efforts were made to support the model learn the optimal policy through the addition of an exploratory phase in the training process. While this approach did not improve the policy's performance against the benchmark solution, the objective of the additional freedom provided (through additional exploration) was to help the agent develop a more profound understanding of its environment.

The findings in this thesis open the door to additional work. In the context of a MGWM problem, different adjustments should be made to the neural network's environment to provide the model with additional support in an attempt to improve the policy and narrow the gap with the benchmark solution. A potential solution is to employ two distinct neural networks within the policy model: one dedicated to estimating the asset mix policy and another one focused on decisions related to purchasing or foregoing targeted goals. Additionally, solving GBWM and MGWM problems requires initializing the values of numerous hyperparameters. Selecting the values of hyperparameters, in many policy approximation implementations, is a sensitive task that often has an impact on a given model's performance. For this reason, hyperparameter optimization tools can be considered to improve the model's ability to learn the optimal policy.

References

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515.

Das, S., Markowitz, H., Scheid, J., and Statman, M. (2011). Portfolios for investors who want to reach their goals while staying on the mean–variance efficient frontier. *The Journal of Wealth Management*, 14(2):25–31.

Das, S. R., Ostrov, D., Radhakrishnan, A., and Srivastav, D. (2020). Dynamic portfolio allocation in goals-based wealth management. *Computational Management Science*, 17:613–640.

Das, S. R., Ostrov, D., Radhakrishnan, A., and Srivastav, D. (2022). Dynamic optimization for multi-goals wealth management. *Journal of Banking & Finance*, 140:106192.

Das, S. R., Ostrov, D. N., Radhakrishnan, A., and Srivastav, D. (2018). A new approach to goals-based wealth management. *Available at SSRN 3117765*.

Das, S. R. and Varma, S. (2020). Dynamic goals-based wealth management using reinforcement learning. *Journal Of Investment Management*, 18(2):1–20.

de Sousa, C. A. (2016). An overview on weight initialization methods for feedforward neural networks. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 52–59. IEEE.

Denault, M. and Simonato, J.-G. (2022). A note on a dynamic goal-based wealth management problem. *Finance Research Letters*, 46:102404.

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

Markowitz, H. (1952). Portfolio selection. *Journal of finance*, 6(1): 77-91:77–91.

Parker, F. J. (2022). *Goals-Based Portfolio Theory*. John Wiley & Sons.

Prémont, M. (2021). *Reinforcement Learning Algorithms for a Dynamic Goal-based Wealth Management Problem*. PhD thesis, HEC Montréal.

Subasi, A. (2020). *Practical machine learning for data analysis using python*. Academic Press.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*.

MIT press.

Appendix - A

Table A: Highest success probability for a GBWM problem: training using normalization to scale input variables

		Training epochs				
$\alpha_{\text{init}}^{\text{policy}}$	Weights per layer	50,000	100,000	150,000	200,000	250,000
0.001	10	0.5048	0.5402	0.5402	0.5485	0.5532
	20	0.6278	0.6278	0.6278	0.6278	0.6278
	40	0.6133	0.6133	0.6133	0.6133	0.6133
0.002	10	0.5370	0.5756	0.5756	0.5756	0.5756
	20	0.5903	0.5903	0.5903	0.5903	0.5903
	40	0.5406	0.5712	0.5712	0.5712	0.5712
0.005	10	0.5835	0.5835	0.5835	0.5835	0.5835
	20	0.6093	0.6093	0.6093	0.6093	0.6093
	40	0.6078	0.6078	0.6078	0.6078	0.6078
0.01	10	0.5502	0.6097	0.6097	0.6097	0.6097
	20	0.6110	0.6110	0.6110	0.6110	0.6478
	40	0.5150	0.5960	0.5960	0.5960	0.5960

[1] Training performed under an on-line approach. Seeds were fixed to assure an appropriate comparison basis between results.

[2] The model's policy is trained using a 5%-greedy policy (e.g.: the probability that model's preferred action is selected is 95%).

Table B: Highest success probability for a GBWM problem: training using an adjusted centering approach to scale input variables

$\alpha_{\text{init}}^{\text{policy}}$	Weights per layer	**Training epochs**				
		50,000	100,000	150,000	200,000	250,000
0.001	10	0.6451	0.663	0.663	0.663	0.6664
	20	0.6642	0.6642	0.6642	0.6642	0.6642
	40	0.6482	0.6532	0.6532	0.6619	0.6619
0.002	10	0.6487	0.6492	0.6492	0.6589	0.6589
	20	0.6614	0.6614	0.6614	0.6614	0.6614
	40	0.6308	0.6308	0.6427	0.6629	0.6629
0.005	10	0.6480	0.6591	0.6591	0.6591	0.6591
	20	0.6264	0.6475	0.6475	0.6475	0.6475
	40	0.6288	0.6288	0.6297	0.6407	0.6407
0.01	10	0.6330	0.6330	0.6490	0.6490	0.6490
	20	0.6045	0.6060	0.6060	0.6152	0.6152
	40	0.5329	0.5329	0.5609	0.5609	0.5612

[1] Training performed under an on-line approach. Seeds were fixed to assure an appropriate comparison basis between results.

[2] The model's policy is trained using a 5%-greedy policy (e.g.: the probability that model's preferred action is selected is 95%).

The results in Table C utilize an initial learning rate of 0.001 for the model's policy ($\alpha_{\text{init}}^{\text{policy}}$) and an adjusted centering approach to scale input variables as their combination has resulted in the fastest convergence speed towards the benchmark amongst all solutions in Tables A and B.

Table C: Highest success probability for a GBWM problem: training using an adjusted centering approach and baseline

$\boldsymbol{\alpha_{\text{init}}^{\text{policy}} = 0.001}$		**Training epochs**				
$\alpha_{\text{init}}^{\text{baseline}}$	Weights per layer	50,000	100,000	150,000	200,000	250,000
0.001	10	0.6611	0.6611	0.6625	0.6625	0.6625
	20	0.6602	0.6661	0.6678	0.6678	0.6678
	40	0.6474	0.6474	0.6474	0.6484	0.653
0.002	10	0.6594	0.6598	0.6598	0.6598	0.6598
	20	0.6602	0.6624	0.664	0.664	0.6702
	40	0.6406	0.6406	0.6406	0.6406	0.6524
0.005	10	0.6565	0.6627	0.6627	0.6627	0.6627
	20	0.6557	0.6558	0.6586	0.6626	0.6631
	40	0.6537	0.6537	0.6537	0.6537	0.6537
0.01	10	0.6663	0.6663	0.6663	0.6674	0.6689
	20	0.6209	0.6399	0.6399	0.6399	0.6403
	40	0.6603	0.6603	0.6603	0.6661	0.6661

[1] Training performed under an on-line approach. Seeds were fixed to assure an appropriate comparison basis between results.

[2] The model's policy is trained using a 5%-greedy policy (e.g.: the probability that model's preferred action is selected is 95%).

Table D: Highest expected utility for a MGWM problem: training using normalization to scale input variables

Scenario 1: $u(5) = 1000$, $u(10) = 1000$

		Training epochs				
$\alpha_{\text{init}}^{\text{policy}}$	Weights per layer	50,000	100,000	150,000	200,000	250,000
0.001	10	1049.8	1049.8	1049.8	1049.8	1049.8
	20	1041.0	1043.9	1043.9	1045.5	1049.8
	40	1051.8	1051.8	1051.8	1051.8	1051.8
0.002	10	1045.2	1045.2	1047.6	1047.6	1047.6
	20	1077.0	1077.0	1077.0	1077.0	1077.0
	40	1044.8	1052.1	1055.2	1055.2	1055.2
0.005	10	1054.7	1054.7	1054.7	1054.7	1057.2
	20	1043.7	1043.7	1049.0	1053.3	1053.3
	40	1054.7	1054.7	1054.7	1054.7	1054.7
0.01	10	1054.7	1054.7	1054.8	1054.8	1054.8
	20	1043.3	1043.3	1056.1	1056.1	1056.1
	40	1053.9	1053.9	1054.7	1056.3	1056.3

Scenario 2: $u(5) = 1000$, $u(10) = 2000$

		Training epochs				
$\alpha_{\text{init}}^{\text{policy}}$	Weights per layer	50,000	100,000	150,000	200,000	250,000
0.001	10	1438.4	1543.8	1682.4	1682.4	1682.4
	20	1563.7	1720.2	1720.2	1720.2	1720.2
	40	1530.3	1619.5	1705.8	1778.4	1778.4
0.002	10	1435.6	1707.4	1707.4	1707.4	1711.4
	20	1634.6	1731.4	1731.4	1731.4	1731.4
	40	1477.2	1477.2	1503.7	1645.2	1688.0
0.005	10	1587.2	1644.4	1730.4	1730.4	1730.6
	20	1494.9	1730.4	1730.4	1730.4	1730.4
	40	1631.2	1631.2	1631.2	1722.8	1730.4
0.01	10	1730.8	1730.8	1730.8	1730.8	1730.8
	20	1730.6	1730.6	1730.6	1730.6	1730.6
	40	1604.2	1713.2	1713.2	1713.2	1713.2

Table D (continued)

Scenario 3: $u(5) = 1000$, $u(10) = 3000$

$\alpha_{\text{init}}^{\text{policy}}$	Weights per layer	**Training epochs**				
		50,000	100,000	150,000	200,000	250,000
0.001	10	2296.8	2561.1	2561.1	2561.1	2561.1
	20	2282.4	2561.1	2561.1	2561.1	2561.1
	40	2461.8	2547.6	2547.6	2547.6	2564.7
0.002	10	2497.5	2542.5	2542.5	2564.7	2564.7
	20	2561.1	2587.8	2587.8	2587.8	2587.8
	40	2478.0	2551.2	2551.2	2595.9	2595.9
0.005	10	2550.0	2550.0	2586.6	2597.1	2597.1
	20	2481.6	2481.6	2507.4	2558.4	2558.4
	40	2413.6	2507.7	2507.7	2507.7	2535.3
0.01	10	2542.5	2595.9	2595.9	2595.9	2595.9
	20	2457.0	2584.2	2584.2	2584.2	2598.6
	40	2548.8	2548.8	2548.8	2548.8	2595.9

[1] Training performed under an on-line approach. Seeds were fixed to assure an appropriate comparison basis between results.

[2] The model's policy is trained using a 5%-greedy policy (e.g.: the probability that model's preferred action is selected is 95%).

Table E: Highest expected utility for a MGWM problem: training using normalization and baseline

Scenario 1: $u(5) = 1000$, $u(10) = 1000$

$\boldsymbol{\alpha}_{\mathbf{init}}^{\mathbf{policy}} = \mathbf{0.001}$		**Training epochs**				
$\alpha_{\text{init}}^{\text{baseline}}$	Weights per layer	50,000	100,000	150,000	200,000	250,000
0.001	10	1050.8	1050.8	1056.8	1056.8	1083.4
	20	1033.3	1058.8	1058.8	1058.8	1058.8
	40	1052.6	1055.4	1058.9	1091.6	1091.6
0.002	10	1050.3	1050.3	1051.4	1052.0	1053.9
	20	1032.4	1044.2	1070.4	1070.4	1070.4
	40	1021.3	1042.1	1052.3	1052.3	1069.5
0.005	10	1051.7	1051.7	1041.7	1055.2	1055.2
	20	1046.5	1053.1	1056.1	1088.8	1088.8
	40	1030.6	1036.2	1041.9	1051.8	1070.7
0.01	10	1026.8	1026.8	1026.8	1026.8	1036.9
	20	1002.3	1048.2	1056.7	1056.7	1062.1
	40	1030.9	1046.4	1046.4	1059.7	1059.7

Scenario 2: $u(5) = 1000$, $u(10) = 2000$

$\boldsymbol{\alpha}_{\mathbf{init}}^{\mathbf{policy}} = \mathbf{0.001}$		**Training epochs**				
$\alpha_{\text{baseline}}^{\text{policy}}$	Weights per layer	50,000	100,000	150,000	200,000	250,000
0.001	10	1451.8	1463.8	1661.4	1787.6	1793.6
	20	1620.5	1738.6	1761.1	1766.7	1794.7
	40	1640.3	1785.0	1791.0	1791.0	1791.0
0.002	10	1453.2	1497.8	1751.5	1781.2	1786.2
	20	1651.8	1704.6	1726.2	1726.2	1760.4
	40	1635.8	1701.6	1730.3	1758.2	1771.8
0.005	10	1424.4	1529.8	1603.0	1759.1	1780.5
	20	1552.0	1801.2	1801.2	1801.2	1801.2
	40	1679.9	1779.6	1779.6	1791.8	1791.8
0.01	10	1476.0	1538.2	1759.8	1768.4	1781.0
	20	1574.6	1708.9	1763.8	1804.0	1804.0
	40	1652.8	1786.9	1793.2	1821.6	1821.6

Table E (continued)

Scenario 3: $u(5) = 1000$, $u(10) = 3000$

$\boldsymbol{\alpha_{\text{init}}^{\text{policy}} = 0.001}$		**Training epochs**				
$\alpha_{\text{init}}^{\text{baseline}}$	Weights per layer	50,000	100,000	150,000	200,000	250,000
0.001	10	2343.9	2497.5	2539.8	2539.8	2539.8
	20	2455.2	2610.0	2619.0	2619.0	2638.8
	40	2507.7	2698.2	2717.1	2717.1	2720.1
0.002	10	2303.4	2595.6	2595.6	2595.6	2595.6
	20	2417.4	2601.6	2601.6	2628.9	2628.9
	40	2590.5	2590.5	2590.5	2595.9	2595.9
0.005	10	2535.3	2585.7	2585.7	2590.8	2590.8
	20	2456.7	2564.4	2596.5	2596.5	2650.2
	40	2438.7	2637.0	2637.0	2646.3	2646.3
0.01	10	2423.4	2545.2	2553.0	2607.0	2649.9
	20	2283.0	2628.9	2666.7	2715.6	2726.4
	40	2598.6	2598.6	2629.5	2629.5	2661.9

[1] Training performed under an on-line approach. Seeds were fixed to assure an appropriate comparison basis between results.

[2] The model's policy is trained using a 5%-greedy policy (e.g.: the probability that model's preferred action is selected is 95%).