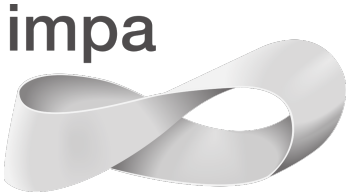
Optimal Investment-Consumption Decision Using Reinforcement Learning

André Lorenzo Bittencourt
Advisor: Rodrigo dos Santos Targino



Instituto de Matemática
Pura e Aplicada

A Thesis submitted for the degree of
Master in Mathematical Methods in Finance

National Institute for Pure and Applied Mathematics
Brazil
August 2023

Abstract

This thesis offers an in-depth analysis of the Optimal Investment-Consumption Decision Problem, often referred to as the Merton's Portfolio Problem, viewed through the lens of Reinforcement Learning (RL), with a specific focus on the Q-Learning algorithm. Within the Q-Learning framework, two distinct implementations are examined: the traditional Tabular approach and the more advanced Deep Q-Learning method, which employs Artificial Neural Networks. Through experimentation and evaluation, this research compare the results obtained from each approach. The findings illuminate the inherent strengths and limitations of both methods, providing insights into their suitability for various scenarios. Moreover, this study aims to serve as a foundational reference for those seeking to implement RL algorithms in the financial sector.

I first dedicate this work to my family and friends who believed in my potential and gave me the confidence to pursue my master's degree. I also wish to thank my professors in the Master in Mathematical Methods in Finance program at IMPA for delivering a world-class curriculum and imparting invaluable knowledge that will guide me throughout my career. Special mention goes to my advisor, Rodrigo Targino, whose advice extended beyond this work. My colleagues in the program deserve recognition for their support and enlightening conversations.

Most importantly, I dedicate this thesis to my beloved Thaline Fransceschi, who endured the sacrifices made during this intense period of pursuing a degree while working. She has always believed in me and my potential, often more than I believed in myself. Without her, this journey would have been significantly more challenging.

Table of Contents

Detailed Contents

Chapter 1

Introduction

1.1 Background

Despite early asset allocation history attracting limited interest and lacking comprehensive studies, it's safe to presume that the question of how to distribute financial wealth among available assets has been an issue since humans began accumulating wealth. An early piece of advice can be traced back to the Judaic book of Talmud, which reads, *"Let every man divide his money into three parts, and invest a third in land, a third in business, and a third let him keep by him in reserve."* This recommendation can be easily followed today using ETFs, REITS, and Treasuries or other money market instruments.

Throughout history, we find instances of this idea of diversification, which is paramount in financial management. A case in point is Shakespeare's *Merchant of Venice*:

> "My ventures are not in one bottom trusted, Nor to one place; nor is my whole estate
> Upon the fortune of this present year; Therefore, my merchandise makes me not sad."

Diversification is crucial not only to avoid the risk of losing all wealth in a single venture but also to reduce risk in general. Humans are risk-averse, a direct consequence of diminishing marginal utility. This concept was first noted by Daniel Bernoulli while exploring the St. Petersburg Paradox in 1738, and later developed by John von Neumann and Oskar Morgenstern [Neumann and Morgenstern, 1953]. Utility theory plays a central role in this study.

In the 19th century, asset allocation evolved into an industry of its own [Turnbull and Farago, 2019], dominated by insurance companies that needed to invest cash to meet their actuarial liabilities. In 1862, one of the first papers on investment allocation was published [Bailey, 1862], advocating risk minimization and, what is known today as, illiquidity premium harvesting. In 1924, the book *Common Stocks As Long Term Investments* was published, shifting the general perspective of stocks from speculative instruments to tools of wealth accumulation.

However, it wasn't until 1952, when Harry Markowitz's groundbreaking work, *Portfolio Selection* [Markowitz, 1952], was published, that the field of asset allocation found its theoretical foundation. Markowitz formalized the concepts of diversification and risk aversion with the mean-variance analysis, a milestone some authors consider as the birth of modern finance theory [Campbell and Viceira, 2001, Rubinstein, 2006b]. This theory triggered numerous studies and publications on the subject, despite many pitfalls of the mean-variance optimization (MVO) approach, which has been the focus of countless articles and methodological developments [Litterman, 2004, Kinlaw et al., 2017, Scherer, 2007].

It has recently been acknowledged [Rubinstein, 2006a] that a decade prior to Markowitz, the Italian statistician Bruno De Finetti had worked on a problem similar to Markowitz's [Finetti, 1939], but his work remained largely unknown at the time.

The MVO and its related variations, referred to as myopic portfolios [Campbell and Viceira, 2001], consider only a single period, be it a day or fifty years. All that matters is the probability distribution of returns at the end of the period. However, investors typically engage in multi-period investments, a series of "single-period" decisions, where the allocation can change and cash can be infused or withdrawn. Under certain conditions [Samuelson, 1963, Samuelson, 1969, Merton, 1969], the solution for a single period is the same as for the multi-period, readjusted for the original allocation, as the weights will diverge due to drift in asset values.

In certain scenarios, however, the MVO allocation might not be ideal. Consider a case where an investor has to invest all their wealth in a single asset. The first asset could triple the investment

or lead to total loss. The second, more conservative, could result in a loss of 10% or a gain of 15%. Depending on the level of risk aversion, an MVO investor might choose the first investment. However, if a series of such investments is considered, the probability of ruin (i.e., losing everything) approaches 1 as the number of periods increases, making it a worse investment, regardless of the level of risk aversion.

The complexity of analyzing many periods led to the proposal of a few solutions, in part due to the mathematical complexity and computational resources only available with modern computers. Two strategies stood out for their simplicity: Thomas Cover's Universal Portfolio [Cover, 1991], with impressive theoretical results but rarely used in practice, and the Growth-Optimal portfolio that maximizes the expected log-return, also known as the *Kelly Criterion* [Kelly, 1956, Thorp, 1975, Thorp, 2011]. Theoretically, the Kelly Criterion yields the highest expected return over a sequence of periods, an attribute that garnered much attention but did not escape criticism [Samuelson, 1971].

However, all models to date have overlooked one key aspect of investing: the *Why*. Few investors invest solely with the aim of leaving a bequest. Most invest to finance future consumption, with the expectation of consuming more than they currently do. Robert C. Merton [Merton, 1969] and Paul Samuelson [Samuelson, 1969] finally addressed the complete, multi-period problem with consumption in what is known today as Merton's Portfolio Problem. The goal is to maximize the expected sum of consumption's utility over many periods. Although this problem is in continuous time (i.e., consumption and investment occur continuously), an idealization that makes the problem more manageable, it remains mathematically challenging with analytical solutions known only for a few cases [Merton, 1969, Merton, 1971, Rao and Jelvis, 2022, Kim and Omberg, 1996, Wachter, 2002], not necessarily in realistic settings. With advances in computing power and numerical methods, more realistic cases were solved using discrete-state approximations [Balduzzi and Lynch, 1999, Barberis, 2000, Brennan et al., 1997].

The aim of this work is to present Merton's Portfolio Problem (MPP) within the Reinforcement Learning Framework and implement a flexible, realistic solution using Artificial Neural Networks. The MPP has been more of a theoretical result or an introductory example of stochastic controls in graduate courses than a practical problem. This is largely due to the complex nature of solving the MPP in realistic scenarios, making it impractical as a "product". However, I believe that advances in computational power and Machine Learning can overcome this issue. Schemes such as the one presented in this work could be used to create investment policies for everyday people, thus democratizing high finance.

1.2 Organization

This work is structured as follows: The chapter 2 introduces the field of Machine Learning, with particular emphasis on Artificial Neural Networks as tools for approximating unknown functions, and the framework of Reinforcement Learning. The chapter 3 details Merton's Portfolio Problem, deriving an analytical solution and presenting the discrete version necessary for our implementation. The chapter 4 focuses on computational implementation and its results. Lastly, conclusions are drawn, and suggestions for further research are proposed.

Chapter 2

Machine Learning

> "I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines."
>
> *Attributed to Claude Shannon*

2.1 Introduction

Machine Learning (ML) is a field that investigates the capability of machines to learn autonomously, without relying on specific algorithms for a given task. While machine learning is often associated with complex algorithms developed to solve intricate tasks, even mechanical structures can demonstrate the ability to "learn." A notable example is MENACE [Michie, 1963], a mechanical device assembled from matchboxes and colored beads, capable of learning how to play tic-tac-toe by engaging in the game against a (presumably) human opponent. Each matchbox represents a possible state of the game, and the beads denote the actions to be taken. MENACE learns by reinforcing actions that lead to a victory (by adding more beads of the same color). In recent decades, the field of machine learning has experienced exponential growth. Presently, ML algorithms are ubiquitous, powering movie recommendations on streaming services, enabling text translation, driving autonomous vehicles, assisting in disease detection, and more. Within the financial sphere, ML is utilized for fraud detection, algorithmic trading, derivatives pricing, and asset management.

But what exactly is "learning"? Probably the most widely accepted definition is proposed by Mitchell [Mitchell, 1997]:

Definition 2.1.1 (Learning)**.** *A computer program A is capable of learning from experience E with respect to a class of tasks T and a performance measure P_T, if it performance improves with E. In other terms: Let $E_1 \subset E_2 \in \mathcal{E}$ be sets of experiences and $A(E)$ a program that received the experience E. It is said capable of learning if:*

$$P_T(A(E_1)) < P_T(A(E_2))$$

For example, the task T could be playing chess, E could be a match played by A with its outcome or a replay of someone else's match with annotated comments about each move, and P_T could be the probability of winning against another benchmark program.

In reality, 2.1.1 should be understood in terms of expectations, as an algorithm might perform worse with the addition of a single experience but should improve with more and more experiences.

There are countless algorithms in Machine Learning, suitable for many different tasks, some more adaptable, some more specific. They are generally classified into three methods or paradigms:

- **Supervised Learning**: In this method, data consisting of a set of features with corresponding labels is used for training. The algorithm seeks to learn how to label a new data point based on its features. An example could be the classification of stocks into economic sectors based on their financial ratios.
- **Unsupervised Learning**: Some tasks may not have associated labels, and the algorithm should find hidden patterns in the data. Sector classifications may not reflect the true nature

of a stock in our current complex landscape; clustering algorithms could group stocks in a more meaningful way [de Prado, 2020].

- **Reinforcement Learning**: This paradigm resembles how humans and animals typically learn by interacting with the environment and receiving positive and negative feedback. The program usually takes actions, receives feedback either immediately or later, and has to evaluate and adjust its behavior based on this feedback. A stock trading algorithm, learning from its Profit and Loss (PnL), would be a typical example. Reinforcement Learning will be the main focus of this work.

Some popular algorithms include:

- Linear and Logistic Regression;
- Decision Trees and Random Forests;
- Support Vector Machines;
- K-Means Clustering;
- Genetic Algorithms;
- Artificial Neural Networks.

While most of these algorithms are specific to their respective paradigms, Artificial Neural Networks (ANNs) have demonstrated versatile applications, ranging from simple tasks to advanced chatbots, owing to their interesting properties which we will explore further in Section 2.3. ANN will also be a central topic in this work.

2.2 Reinforcement Learning

Reinforcement Learning can be seen as a process of "learning by doing." It involves learning how to map a situation or state to an action with the goal of maximizing feedback or reward. The learner is not instructed on which actions are "good" or "bad" but must infer this from the feedback. For example, consider a mouse in a lab cage that receives food if it presses a button after a light turns on. Here, the state is the status of the light (on or off), the action involves pressing the button or not, and the reward is the food. It is expected that the mouse will quickly learn this association. While this is a simple and direct example, more complex applications may not be as straightforward. The reward might not be received immediately but only sometime after the action, the reward signal might be noisy, and the action could impact not only the reward received but also the state of the world itself, affecting subsequent states and rewards.

In this section, we will introduce and define the fundamental concepts of Reinforcement Learning.

2.2.1 Finite Markov Decision Processes

Much of the field of RL is developed around the concept of a Markov Decision Process (MDP). An MDP formalizes sequential decision-making, involving the interaction between an *agent* and an *environment* through *actions* and *rewards*.

Let's define some basic terms:

- **Agent**: The decision-making entity that interacts with and learns from the environment.
- **Environment**: The world or problem that receives the agent's actions and provides feedback in the form of rewards and next states.
- **State**: A representation of the environment, containing current information about the world or problem.
- **Reward**: A numerical value received by the agent as feedback from the environment after taking an action.

Definition 2.2.1 (Markov Decision Process)**.** *A Markov Decision Process (MDP) is defined as the tuple* $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$*, where:*

- $\mathcal{S}$: *The finite state space, representing the set of possible states.*
- $\mathcal{A}$: *The finite action space, representing the set of possible actions.*
- $\mathcal{P}$: *The state transition probability function, denoted as $P_a(s, s')$, which represents the probability of transitioning from state s to state s' when taking action a. $P_a(s, s') = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$.*
- $\mathcal{R}$: *The reward function, denoted as $\mathcal{R}(s, a, s')$, which represents the immediate reward received when transitioning from state s to state s' under action a. The reward could be either deterministic or sthocastic.*

An MDP generates a sequence of interactions. The agent receives a state $S_t \in \mathcal{S}$ and selects an action $A_t \in \mathcal{A}$. The environment processes the action and returns a reward $R_{t+1} \in \mathcal{R}$ and a new state S_{t+1} in the next time step. This interaction continues until a terminal state is reached, either at fixed time steps or a specific state, resulting in a complete episode:
$S_0, A_0, R_1, S_1, A_1, ..., R_T$
The Markovian property is reflected on the fact that the state, action and reward depends only on the last step and not the steps before.

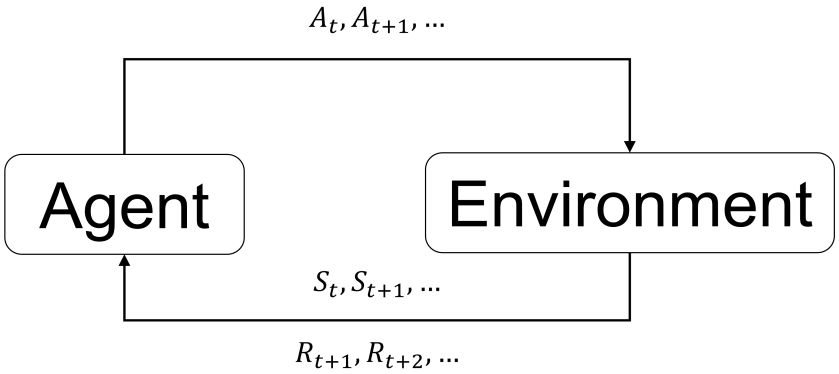


Figure 2.1: The interaction between the agent and the environment in an MDP

To select which action to take, the agent follows a *policy*. Policies can be deterministic or stochastic.

Definition 2.2.2 (Deterministic Policy)**.** *A deterministic policy maps each state to a single action.*

$$\pi : \mathcal{S} \to \mathcal{A} \tag{2.2.1}$$

Definition 2.2.3 (Stochastic Policy)**.** *A stochastic policy selects actions according to a probability distribution conditioned on the state.*

$$\pi : \mathcal{A} \times \mathcal{S} \to [0, 1] \tag{2.2.2}$$

$$\pi(a, s) = \mathbb{P}[A_t = a \mid S_t = s] \tag{2.2.3}$$

The goal of the agent's actions is to maximize the total reward received over an entire episode, not just the immediate reward. If an action leads to a large immediate reward but sacrifices subsequent rewards, it is considered inferior to an action that yields a small immediate reward but leads to greater rewards later on. This reflects the idea of "playing the long game".

Definition 2.2.4 (Return)**.** *The return G_t is the sum of discounted rewards from t to the end of the episode, using the discount factor $\gamma \in [0, 1]$:*

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{(T-1)} R_T \tag{2.2.4}$$

The use of a discount factor γ serves several purposes. It allows for problems that do not have a natural termination or that may continue indefinitely to have infinite returns. Additionally, in many problems, a reward received sooner is more desirable than one received later. In economic-related problems, the discount rate can be interpreted as the opportunity cost or the rate of

intertemporal preference for consumption. Furthermore, the discount factor allows for controlling the preference for shorter-term or longer-term rewards.

Due to the stochastic nature of the environment, the agent's objective is to maximize the expected return.

Definition 2.2.5 (Value Function)**.** *The Value Function $V_\pi^t : \mathcal{S} \to \mathbb{R}$, or V-function, in a finite MDP represents the expected return when starting from a given state s, in time t and following policy π. If the :*

$$V_\pi^t(s) = \mathbb{E}_\pi[G_t|S_t = s, t] \quad \forall s \in \mathcal{S} \tag{2.2.5}$$

Therefore, we seek:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, V_\pi^t(.) \tag{2.2.6}$$

Could also be of interest to know how good a single action is, if we return to follow π, afterwards. For this end, we define the Action-Value Function:

Definition 2.2.6 (Action-Value Function)**.** *The Action-Value Function, Q-function, $Q_\pi^t : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, is the expected return, conditional to the current state s, taking action a now and following the policy π afterwards:*

$$Q_\pi^t(s, a) = \mathbb{E}_\pi[G_t|S_t = s, t, A_t = a] \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A} \tag{2.2.7}$$

For brevity of notation, the time will be included in the state and the superscript will be ommited.

2.2.2 Bellman Equations

The Value function 2.2.5 can be rewritten as a recursive equation:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{(T-1)} R_T|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(s')|S_t = s] \end{aligned} \tag{2.2.8}$$

The same result can be achieved for the Action-Value function:

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(s', a')|S_t = s, A_t = a] \tag{2.2.9}$$

These equations are known as Bellman Equations, named after Richard Bellman [Bellman, 1957].

As mentioned before, we are interested in finding the optimal policy π^*, i.e., the one with the highest $V_{\pi^*}(s)$. We can define the optimal Value and Action-Value functions:

$$V^*(s) = \max_\pi V_\pi(s) \quad \forall s \in \mathcal{S} \tag{2.2.10}$$

$$Q^*(s, a) = \max_\pi Q_\pi(s, a) \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A} \tag{2.2.11}$$

Notice that V^* and Q^* are related by:

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a) \tag{2.2.12}$$

By substituting 2.2.8 and 2.2.9 into 2.2.10 and 2.2.11, respectively, we obtain:

$$V^*(s) = \max_\pi \mathbb{E}_\pi[R_{t+1} + \gamma V^*(s')|S_t = s] \tag{2.2.13}$$

$$\begin{aligned} Q^*(s, a) &= \max_\pi \mathbb{E}_\pi[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')|S_t = s, A_t = a] \\ &= \max_\pi \mathbb{E}_\pi[R_{t+1} + \gamma V^*(s')|S_t = s, A_t = a] \end{aligned} \tag{2.2.14}$$

These equations are consistent with the Bellman's Principle of Optimality:

Definition 2.2.7 (Bellman's Principle of Optimality). *"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." [Bellman, 1957]*

Therefore, these equations are known as Bellman Optimality Equations.

Optimal Policy

It was always assumed that there is an optimal policy, but the reader may question if this is indeed true.

Theorem 2.2.1 (Optimal Policy Existence). *For any Markov Decision Process, $\exists$ a policy π^*, called the optimal policy, such that $V_{\pi^*}(s) \geq V_{\pi'}(s)$, $\forall s \in \mathcal{S}$ and any acceptable policy π'.*

Proof. The detailed proof can be found in [Kumar, 2020, Modirshanechi, 2020]. Here, I will provide a sketch of the idea. We can define an operator T, called the Bellman Optimality Operator, that takes a Value or Action-Value function and returns another Value/Action-Value function:

$$TV(s) = \max_a \mathbb{E}[R_{t+1}(s, a, s') + \gamma V(s')] \tag{2.2.15}$$

Applying the Bellman operator 2.2.15 on 2.2.13:

$$V^* = TV^* \tag{2.2.16}$$

Hence, to prove the existence of a optimal V^* it is enough to prove that T operator has a fixed point. It can be shown that for $\gamma \in [0, 1)$, T is a contraction mapping in sup-norm in a complete metric space. Then, by the Banach Fixed Point Theorem, such V^* exist and hence the optimal policy π^* exists. □

2.2.3 Solution Algorithms

To obtain $V^*(s)$, $Q^*(s, a)$, and $\pi^*(s)$ for a given problem, countless algorithms have been developed, each with its strengths and limitations. In some situations, the dynamics of the MDP (transition probabilities and reward function) are known, such as in games or non-chaotic physical systems. In these cases, we can expand equations 2.2.8, 2.2.9, 2.2.13, and 2.2.14 further:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} \mathbb{P}[s', r|s, a](r + \gamma V_\pi(s')) \tag{2.2.17}$$

$$Q_\pi(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} \mathbb{P}[s', r|s, a](r + \gamma \sum_{a' \in \mathcal{A}} \pi(a|s) Q_\pi(s', a')) \tag{2.2.18}$$

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} \mathbb{P}[s', r|s, a](r + \gamma V^*(s')) \tag{2.2.19}$$

$$Q^*(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} \mathbb{P}[s', r|s, a](r + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')) \tag{2.2.20}$$

The first two equations are linear and can potentially be solved explicitly or numerically if the state and action spaces are not too large. The last two equations are nonlinear and require numerical methods for solving systems of nonlinear equations. The set of methods based on these equations is known as Dynamic Programming.

For more interesting and complex problems where the dynamics are not known, direct computation of the Q and V functions is not feasible. In such cases, two contrasting category of methods were developed: Monte Carlo and Temporal-Difference methods. A comprehensive exposition of these methods can be found at [Sutton and Barto, 2020] and in finance-oriented [Rao and Jelvis, 2022].

Monte Carlo

Monte Carlo methods are based on simulating a complete episode following a given policy, get the realized return and updating the value function, of that initial state, as a weighted average between the current estimation of the value function and the realized return of the episode:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{2.2.21}$$

The parameter α is the learning rate. A larger α leads to faster moving towards the true value, but the estimation may bounce around it or even diverge. A smaller value reduces this risk but may be slower to converge. The solution to this trade-off is discussed in section 2.2.4. A con of this method is that you either only improve the estimation of a single state, the initial, or a vector of Returns, progressively discarding the older reward, has to be carried until the update step.

Temporal-Difference

Temporal-Difference (TD) methods, on the other hand, update the estimation at each step, allowing for online estimation (no need to wait until the end of the episode) and learning in problems without terminal states. This is done using the reward R_{t+1} plus the estimated expected value of G_{t+1} :

$$\hat{G}_t = R_{t+1} + \gamma \mathbb{E}[G_t | S_t = s] \tag{2.2.22}$$

$$\hat{G}_t = R_{t+1} + \gamma V(S_{t+1}) \tag{2.2.23}$$

$$V(S_t) \leftarrow V(S_t) + \alpha[(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.2.24}$$

This method has the advantage of immediately update the estimation, updating for all states in the episode. It also reduce the memory usage and usually leads to simpler codes. A con is the use of the function being estimation to improve itself. A poor initial value, will propagate and my slow down the convergence.

2.2.4 Q-Learning

Among TD methods, Q-Learning is one of the most widely used algorithms, maybe one of the most used in the whole field of reinforcement learning. It was developed by Watkins [Watkins, 1989] in 1989 and remains a simple yet powerful algorithm. Q-Learning is classified as an off-policy algorithm, meaning that during learning, it samples actions not from the optimal policy at that time, but from another policy. The most common approach is the ϵ-greedy policy:

$$a = \begin{cases} \sim \mathcal{U}(\mathcal{A}), & \text{with probability } \epsilon \\ \text{argmax}_a \, Q(s,a), & \text{w.p. } 1-\epsilon \end{cases} \tag{2.2.25}$$

Here, $\sim \mathcal{U}$ denotes uniform random selection. The ϵ-greedy policy allows for exploration of the action space by occasionally selecting non-optimal actions. This is important to avoid getting stuck in suboptimal actions and to encourage learning of potentially better actions. We usually do not follow purely exploratory policies while learning to not waste computational resources improving the estimation of actions we know are bad (I don't have to go 10 times to the same restaurant to be sure how bad it is).

The Q-Learning algorithm updates the estimation of the action-value function using a weighted average between the current estimation and the reward received plus the discounted estimation of the state-value for the next state as target:

$$Q(S,A) \leftarrow (1-\alpha)Q(S,A) + \alpha(R + \gamma V^*(S')) \tag{2.2.26}$$

Here, α is the learning rate, which determines the weight given to the new information compared to the current estimation. The Q-Learning algorithm can be summarized as follows:

Algorithm 1 Q-Learning

```
Initialize Q(S, A);
Initialize α ∈ [0, 1];
Initialize n as total length of time steps;
for i = 1, ..., n do
    Initialize S;
    while S ¬ terminal do
        Sample A ← ϵ-greedy(A);
        Take action A;
        Observe R, S'
        Q(S, A) ← (1 − α)Q(S, A) + α(R + γV(S'))
        S ← S'
```

One advantage of Q-Learning is that it approximates the optimal action-value function regardless of the policy followed, as long as all state-action pairs continue to be updated. This allows for flexibility in exploration and the ability to access the entire action space.

Convergence

Another advantage of this algorithm is that it's guaranteed to converge under the conditions:

$$\sum_t \alpha_t = \infty \quad \sum_t \alpha_t^2 < \infty \tag{2.2.27}$$

Before prove this [Jaa, 1993], let's see some preliminary results:

Theorem 2.2.2 (Q Contraction and Fixed Point)**.** *The operator* $\mathbf{H}$ *is a contraction in the sup-norm and* Q^**, optimal Q-function, is a fixed point in it. The operator* $\mathbf{H}$*, for a function* $Q : \mathcal{S} \times \mathcal{A} \to \mathbf{R}$*, is defined as:*

$$(\mathbf{H}Q)(s,a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} \mathbb{P}[s', r|s, a](r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')) \tag{2.2.28}$$

Proof. Start with:

$$\begin{aligned}
&||\mathbf{H}Q_1 - \mathbf{H}Q_2||_\infty = \\
&= \max_{s,a} \left| \sum_{s' \in \mathcal{S}} \mathbb{P}[s'|s,a][r(s,s',a) + \gamma \max_{a' \in \mathcal{A}} Q_1(s',a') - r(s,s',a) - \gamma \max_{a' \in \mathcal{A}} Q_2(s',a')] \right| \\
&= \gamma \max_{s,a} \left| \sum_{s' \in \mathcal{S}} \mathbb{P}[s'|s,a][\max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a')] \right| \\
&\leq \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} \mathbb{P}[s'|s,a] \left| \max_{a' \in \mathcal{A}} Q_1(s',a') - \max_{a' \in \mathcal{A}} Q_2(s',a') \right| \\
&\leq \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} \mathbb{P}[s'|s,a] \max_{a' \in \mathcal{A}, s^* \in \mathcal{S}} |Q_1(s^*,a') - Q_2(s^*,a')| \\
&= \gamma ||Q_1 - Q2||_\infty
\end{aligned} \tag{2.2.29}$$

$$||\mathbf{H}Q_1 - \mathbf{H}Q_2||_\infty \leq \gamma ||Q_1 - Q_2||_\infty \tag{2.2.30}$$

As the optimal action-value function is:

$$\mathbf{H}Q^* = Q^* \tag{2.2.31}$$

It is a fixed point in $\mathbf{H}$. □

Theorem 2.2.3 (Random Process Convergence)**.** *The random process* $\{\Delta Q_t\}$ *defined as:*

$$\Delta Q_{t+1}(.) = (1 - \alpha_t)\Delta Q_t(.) + \alpha_t F_t(.) \tag{2.2.32}$$

Converges to zero with probability 1, if:

1. $0 \leq \alpha_t \leq 1$, $\sum_t \alpha_t = \infty$ *and* $\sum_t \alpha_t^2 < \infty$ *(Robbins-Monro condition);*

2. $||\mathbb{E}[F_t(.)]|| \leq \gamma ||\Delta Q_t||$, *with* $\gamma < 1$;

3. $Var[F_t] \leq C(1 + ||\Delta Q_t||^2)$, *for* $C > 0$;

Proof. See [Jaa, 1993]. □

Theorem 2.2.4 (Q-Learning Convergence)**.** *Given a finite MDP* $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$*, the Q-learning update rule:*

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t [R_t + \gamma \max_{a^* \in \mathcal{A}} Q_t(s_{t+1}, a^*) - Q_t(s_t, a_t)] \tag{2.2.33}$$

obeying the Robbins-Monro condition, converges to the optimal Q-function with probability 1.

Proof. Let just clarify that the subscript t in Q_t refers to the sequence of updating the Q estimation and not the step in the MDP. To prove the convergence, we should verify the Q-learning algorithm satisfies the condition of theorem 2.2.3. The first condition of theorem 2.2.3 was imposed. To show the second condition, rewrite 2.2.33 as:

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t) Q_t(s_t, a_t) + \alpha_t [r_t + \gamma \max_{a^* \in \mathcal{A}} Q_t(s_{t+1}, a^*)] \tag{2.2.34}$$

Subtract $Q^*(s_t, a_t)$ from both sides and defining:

$$\Delta Q_t(s_t, a_t) = Q_t(s_t, a_t) - Q^*(s_t, a_t) \tag{2.2.35}$$

yields

$$\Delta Q_t(s_t, a_t) = (1 - \alpha_t) \Delta Q_t(s_t, a_t) + \alpha_t [r_t + \gamma \max_{a^* \in \mathcal{A}} Q_t(s_{t+1}, a^*) - Q^*(s_t, a_t)] \tag{2.2.36}$$

Letting

$$F_t(s_t, a_t) = r_t + \gamma \max_{a^* \in \mathcal{A}} Q_t(s_{t+1}, a^*) - Q^*(s_t, a_t) \tag{2.2.37}$$

Then, using the definition 2.2.28:

$$\mathbb{E}[F_t(s_t, a_t)] = (\mathbf{H}Q)(s, a) - Q^*(s_t, a_t) \tag{2.2.38}$$

As Q^* is the fixed point of $\mathbf{H}$:

$$F_t(s_t, a_t) = (\mathbf{H}Q)(s, a) - (\mathbf{H}Q^*)(s_t, a_t) \tag{2.2.39}$$

From 2.2.2 we achieve the second condition:

$$||F_t(s_t, a_t)||_\infty \leq ||\mathbf{H}Q_t - \mathbf{H}Q^*||_\infty \leq \gamma ||Q_t - Q^*||_\infty = \gamma ||\Delta Q_t||_\infty \tag{2.2.40}$$

The third condition is simple to demonstrate:

$$\begin{aligned} Var[F_t(s_t, a_t)] &= \mathbb{E}[(F_t(s_t, a_t) - \mathbb{E}[F_t(s_t, a_t)])^2] \\ &= \mathbb{E}[(r_t + \gamma \max_{a^* \in \mathcal{A}} Q_t(s_{t+1}, a^*) - Q^*(s_t, a_t) - (\mathbf{H}Q)(s, a) + Q^*(s_t, a_t))^2] \\ &= \mathbb{E}[(r_t + \gamma \max_{a^* \in \mathcal{A}} Q_t(s_{t+1}, a^*) - (\mathbf{H}Q)(s, a))^2] \\ &= Var[r_t + \gamma \max_{a^* \in \mathcal{A}} Q_t(s_{t+1}, a^*)] \end{aligned} \tag{2.2.41}$$

As r and Q_t are bounded, $\exists C$ such:

$$Var[F_t(s_t, a_t)] \leq C(1 + ||\Delta Q_t||^2) \tag{2.2.42}$$

□

Tabular Implementation

So far, we have discussed the Q-Learning algorithm without explicitly mentioning the representation or updating process of the Q-function. When both the action and state spaces are discrete, or at least can be discretized, a straightforward approach is to use a tabular representation for the Q-function.

The tabular implementation of Q-Learning involves representing the Q-function as a table, which is often referred to as the Q-table. This table's dimensions correspond to the number of states and actions in the environment, making it more appropriate to call it a Q-tensor if $|\mathcal{A}| + |\mathcal{S}| > 2$. Each entry in the table stands for the Q-value for a specific state-action pair.

Initially, the Q-table is filled with arbitrary values or all zeros. As the agent follows algorithm 1, the Q-table gets updated.

This implementation provides a simple and intuitive representation of the Q-function. However, it is constrained to discrete state and action spaces. With large state and action spaces, the tabular representation becomes unfeasible due to the exponentially growing size of the Q-table, a problem known as the *Curse of dimensionality*. Moreover, the Q-table can only learn the state-action value for a specific pair (s, a) if this pair is encountered many times, which could be problematic if a state or action is rarely visited.

Despite its limitations, the Q-table is a straightforward and powerful tool for small dimension problems.

Deep Q-Learning

The tabular implementation becomes impractical when the number of actions and states is too large, or if the state space is continuous. In such cases, we can rely on function approximation methods to tackle the problem.

Definition 2.2.8 (Function Approximation)**.** *Suppose $f : \mathcal{X} \to \mathcal{Y}$ is a function, and $\Phi = \phi(\cdot, \mathbf{w})$ is a set of functions parameterized by θ. A function $\phi_{\mathbf{w}'} \in \Phi$ is considered an approximation of f over a subset $X \subseteq \mathcal{X}$, provided there exists an error tolerance $\epsilon > 0$ such that:*

$$\sup \|f(X) - \phi_{\mathbf{w}'}(X)\| < \epsilon \tag{2.2.43}$$

Here, $\|\cdot\|$ denotes a suitable norm or distance metric in the space $\mathcal{Y}$.
Function $\phi_{\mathbf{w}_2}$ is deemed to be a better approximation of f than $\phi_{\mathbf{w}_1}$ if:

$$\sup \|f(X) - \phi_{\mathbf{w}_2}(X)\| < \sup \|f(X) - \phi_{\mathbf{w}_1}(X)\| \tag{2.2.44}$$

Essentially, $\phi_{\mathbf{w}_2}$ is a superior approximation if it achieves a smaller error on X compared to $\phi\mathbf{w}_1$.

In the instance of approximating f with a polynomial, w would represent the coefficients of this polynomial, for example.

In Q-Learning, since Q^* cannot be directly observed, we aim to discover a sequence of progressively better approximations of Q^*, derived from rewards.

There are many common choices of functional forms such as Gaussian kernels and polynomials. However, Artificial Neural Networks (ANN) have become particularly popular [Sutton and Barto, 2020] due to their function approximation capabilities, as explored in more detail in Section 2.3.

The application of ANN for Q-Learning was pioneered by DeepMind [Mnih et al., 2013], giving rise to Deep Q-Learning, which has been gaining popularity ever since. Its primary advantage lies in handling large action and state spaces, even continuous state spaces. Furthermore, it can extrapolate Q-Values for previously unvisited state-actions. The overall popularity of ANNs has grown in recent years, thanks to user-friendly programming libraries like PyTorch [Paszke et al., 2019] and TensorFlow [Abadi et al., 2016].

Nonetheless, these advantages come at a significant cost. Theorem 2.2.4 is valid for the *exact* Q-function representation and not approximations, hence convergence is not guaranteed. Worse still, the training could be unstable and carries a risk of divergence. This can occur when the *deadly triad* [Sutton and Barto, 2020] is present:

- Function approximation.
- Bootstrapping (using the estimation itself in the target).
- Off-policy training.

Deep Q-Learning checks all the three boxes. Despite this, its usage is frequently justified by its empirical successes [Fan et al., 2019, Ramaswamy and Hullermeier, 2022].

2.3 Artificial Neural Network

Artificial Neural Networks (ANNs) [Mitchell, 1997, Sutton and Barto, 2020, Ertel, 2017] are mathematical models that take loose inspiration from biological brains. The foundational connection between these concepts was initially made in [McCulloch and Pitts, 1943]. Human brains contain approximately 10^{11} neurons interconnected by over 10^{14} synapses. ANNs are barely capable of simulating a fraction of this size, yet achieve remarkable results. For instance, an ANN with merely 1291 "neurons" was capable of driving a car [Pomerleau, 1989], and a state-of-the-art ANN managed to outperform humans in the game of Go [Silver et al., 2016].

ANNs consist of interconnected units, where information flows from input units to output units, undergoing a series of transformations. Numerous different architectures exist, reflecting how the units are organized [Veen, 2019]. One of the most basic and widely used architectures is the feed-forward neural network.

Definition 2.3.1 (Feed-forward Neural Network)**.** *Let $\mathcal{NN}$ denote the space of feed-forward ANNs. The $NN \in \mathcal{NN}$ is a map $NN : \mathcal{X} \to \mathcal{Y}$, with $\mathcal{X} \subseteq \mathbb{R}^{d_0}$ and $\mathcal{Y} \subseteq \mathbb{R}^{d_L}$. Here, d_0 and d_L represents the dimensionalities of the input and output spaces, respectively.*

The NN consists of an ordered sequence of L mappings, referred as layers. Each layer $l = 1, 2, ...L$ is characterized by a function $f_l : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$, where d_l is the dimensionality of the l^{th} layer's output.

The function f_l is expressed as as the composition of a linear transformation, it a weight matrix $\mathbf{W}_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and a bias vector $\mathbf{b}_l \in \mathbb{R}^{d_l}$, and a nonlinear activation function $\sigma_l : \mathbb{R} \to \mathbb{R}$. That is, $f_l(\mathbf{x}; \theta_l, \sigma_l) = \sigma_l(\mathbf{W}_l\mathbf{x} + \mathbf{b}_l)$, where $\theta_l = (\mathbf{W}_l, \mathbf{b}_l)$.

Therefore, we can define a NN as:

$$NN(\mathbf{X}, \boldsymbol{\Theta}) = f_L \circ f_{L-1} \circ \cdots \circ f_1(\mathbf{X}) \tag{2.3.1}$$

where $\mathbf{X} \in \mathcal{X}$ is the input vector, and $\boldsymbol{\Theta} = \{\theta_1, \theta_2, \ldots \theta_L\}$ is the set of all network's parameters. $\mathbf{W}$ is usually called weights *and $\mathbf{b}$* bias*.*

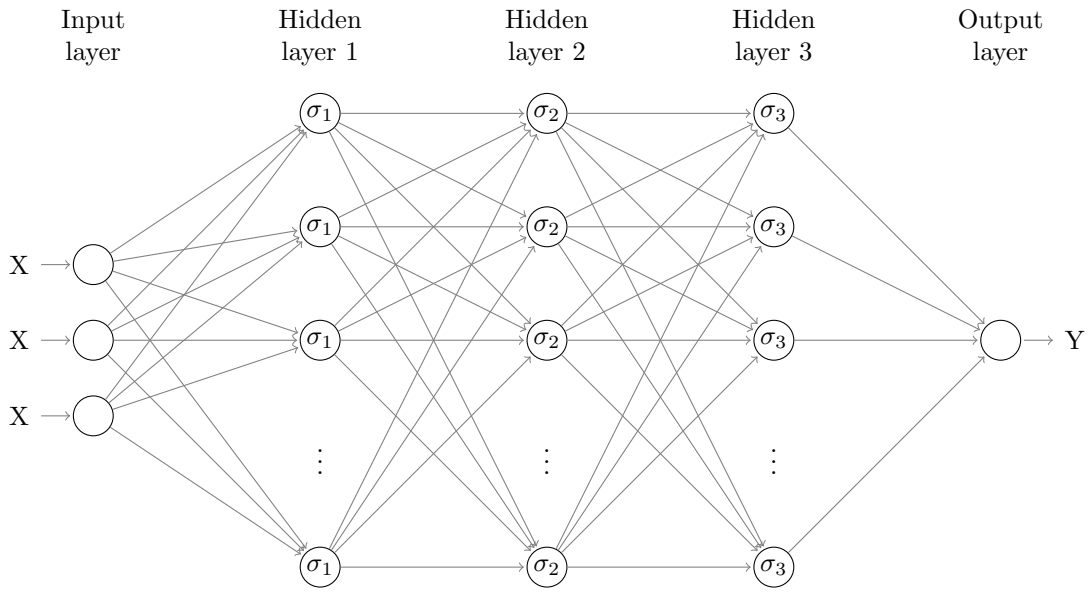


Figure 2.2: Example of a feed-forward neural network

The input layer serves as the starting point for information, and the output layer represents the final result of the ANN. Any layers situated between these two are termed hidden layers.

2.3.1 Activation Function

The selection of activation functions is wide-ranging and continues to grow [Lederer, 2021]. The choice of which one to use often involves as much trial-and-error as theory.

Here are a few of the more commonly used activation functions:

ReLU

The Rectified Linear Unit (ReLU) is frequently used as an activation function due to its simplicity and computational efficiency. It is defined as:

$$\begin{cases} \mathbb{R} \to [0, \infty) \\ \sigma(x) = \max(0, x) \end{cases} \tag{2.3.2}$$

The ReLU activation function allows for faster and more effective training of neural networks due to its property of activating a neuron only when the input is positive. This means it reduces unnecessary computations by not activating all the neurons at the same time.

However, a drawback with ReLU is the so-called dying-ReLU problem. Occasionally, during the training process, the bias becomes so negatively large that the ReLU consistently outputs 0 for most or all possible inputs. As the training process of an ANN involves taking the gradients of each neuron, and this particular neuron has a gradient of zero, it ceases to improve, and effectively "dies".

ELU

To deal with the dying-ReLU problem, several variations were introduced, one of which is the Exponential Linear Unit (ELU):

$$\begin{cases} \mathbb{R} \to (-a, \infty) \\ \sigma(x) = \begin{cases} x, & \text{if } x \geq 0 \\ a(e^x - 1), & \text{if } x < 0 \end{cases} \end{cases} \tag{2.3.3}$$

Here, $a \in [0, \infty)$ is a parameter.

The ELU function introduces a small negative slope for negative input values, ensuring that ELU neurons never completely die out. This leads to more robust and accurate training. However, the ELU function is computationally more expensive than the ReLU, which can be a limitation in scenarios where computational resources or speed are a concern.

Binary

The Binary activation function is specifically designed for binary classification problems, where the output should be either 0 or 1. It is defined as:

$$\begin{cases} \mathbb{R} \to \{0, 1\} \\ \sigma(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0 \end{cases} \end{cases} \tag{2.3.4}$$

The Binary activation function can be an effective choice when a clear binary decision is required from the model. However, because it is not differentiable, it can't be used with standard backpropagation training methods. This can limit its applicability in practical scenarios.

Logistic

The Logistic (or sigmoid) activation function is a differentiable and therefore trainable alternative to the Binary function. It is defined as:

$$\begin{cases} \mathbb{R} \to (0, 1) \\ \sigma(x) = \frac{1}{1+e^{-x}} \end{cases} \tag{2.3.5}$$

The Logistic function introduces a smooth, differentiable transition between the 0 and 1 outputs, making it suitable for training via gradient descent. Additionally, its outputs can be interpreted as probabilities, making it a popular choice for output layers in binary classification problems.

However, it can suffer from the "vanishing gradient" problem, where the gradients become very small if the input is too positive or too negative, slowing down the learning process.

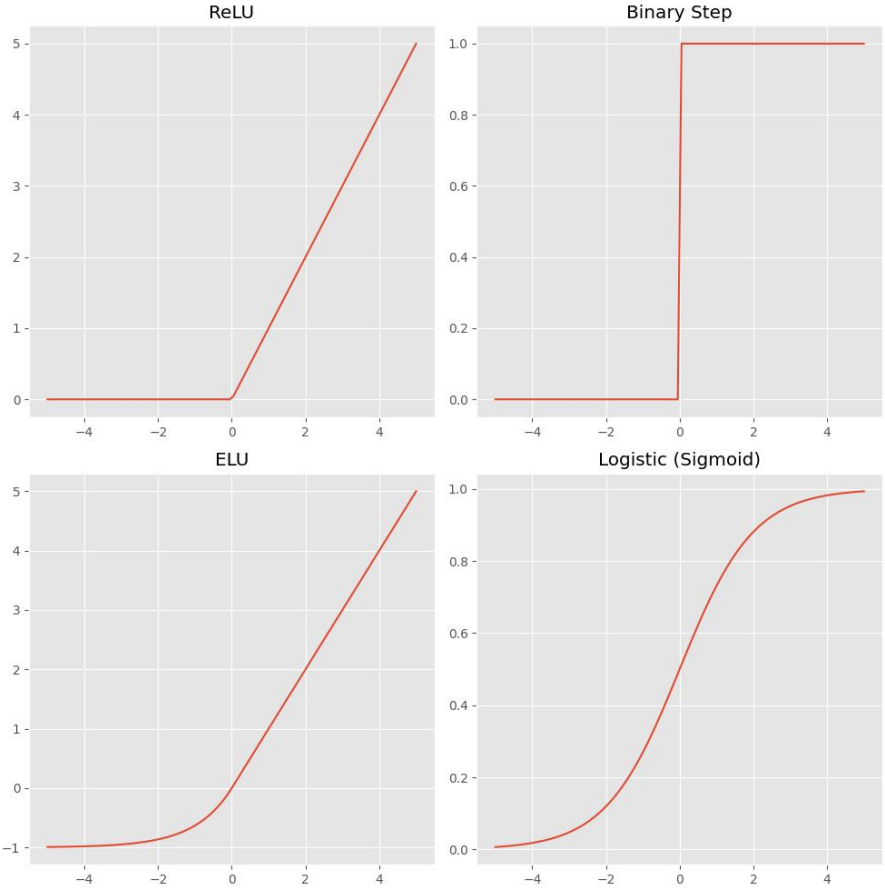


Figure 2.3: Different types of Activation Functions.

2.3.2 Universal Approximation Theorem

The flexibility of Artificial Neural Networks (ANNs) largely explain their popularity, as they can approximate a vast variety of functions. Specifically, a feed-forward ANN, equipped with a single hidden layer (alongside a linear combination in the output layer), is capable of approximating *any* function with a *any* degree of accuracy, given sufficient width. This is known as the *Universal Approximation Theorem* (UAT). In fact, there exist various versions of UATs, for different architectures. Cybenko's UAT [Cybenko, 1989] is arguably the most recognized:

Theorem 2.3.1 (Universal Approximation Theorem)**.** *Let $\mathcal{C}([0,1]^n)$ represent the set of all continuous functions $[0,1]^n \to \mathbb{R}$, and let σ denote any sigmoidal function. Define:*

$$g(X) = \sum_{j=1}^{N} a_j \sigma(\mathbf{w}_j X + b_j) \tag{2.3.6}$$

where $\mathbf{w}_j$ is a matrix in $\mathbb{R}^{n\times n}$ and b_j is a vector in $\mathbb{R}^{1\times n}$. Then, $g(X)$ is dense in $\mathcal{C}([0,1]^n)$. That is, $g(.)$ can approximate any function in $\mathcal{C}([0,1]^n)$.

It's important to clarify that the term *sigmoidal function* here doesn't refer to the *sigmoid function*, but rather any function where:

$$\sigma(t) \to \begin{cases} 1, & \text{as } t \to +\infty \\ 0, & \text{as } t \to -\infty \end{cases} \tag{2.3.7}$$

Although the ReLU function is not a sigmoidal function, as it grows towards infinity, a linear combination of ReLUs (one increasing, the other decreasing, thereby offsetting the infinite growth) can function as such. Consequently, the theorem remains valid if we double the number of neurons at most.

Proof. The detailed proof can be found in [Cybenko, 1989]. It involves functional analysis, which falls outside the scope of this work. However, the concept can be grasped intuitively. Suppose we aim to approximate a function f within the domain $[0, 1]$. We then consider an ANN with a single neuron in the hidden layer that outputs 1 if the input exceeds 0.5, or 0 otherwise. With appropriate selection of weights and bias, the ANN can yield the average of the function f over $[0, 0.5]$ if the input is < 0.5 and deliver the average of f over $(0.5, 1]$ if it is greater. Employing two neurons allows the domain to be divided into three parts, which approximates the average of each part in the image, resulting in improved approximation. This process can be indefinitely repeated, enhancing the approximation with each additional neuron in the hidden layer. □

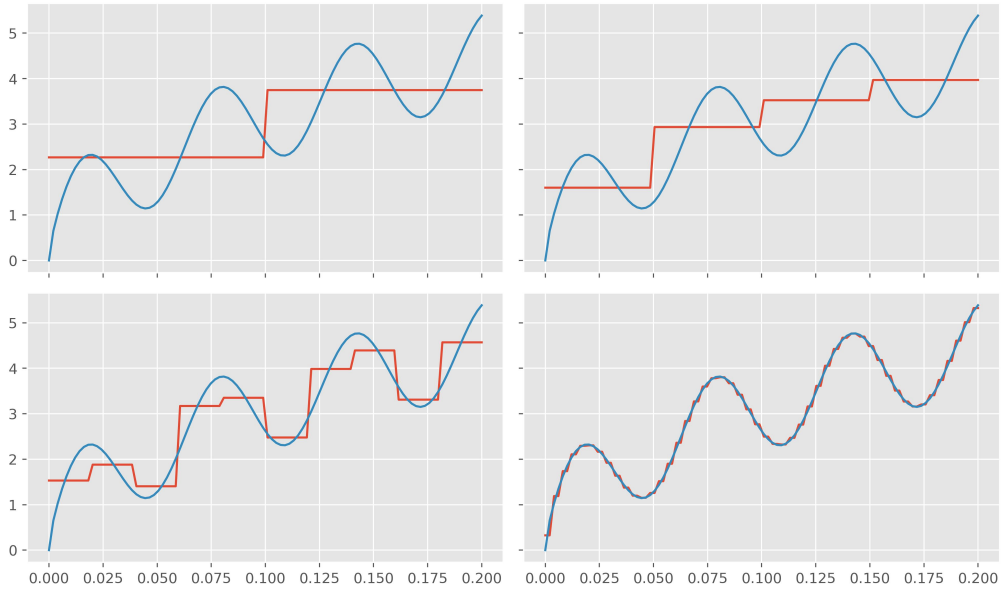


Figure 2.4: Universal Approximation Theorem. Approximations for N = 1, 3, 9 and 50 hidden neurons. Function $f(x) = \sqrt{x} + sin(x)$, $x \in [0, 0.2]$.

The theorem is immediately expanded to functions with image $\mathbb{R}^m$, by staking m neural networks. Other findings expand on this theorem to accommodate bounded width and arbitrary depth under varying conditions [Hornik et al., 1989, Hornik, 1993]. The debate whether use deep (numerous hidden layers) or shallow structures is addressed in [Bianchini and Scarselli, 2014]. For an overview of significant approximation results in Artificial Neural Networks, readers are directed to [Petersen, 2022].

Intriguingly, a parallel result in finance, albeit unrelated to the main subject, stipulates that any European contingent claim can be replicated with a static portfolio of vanilla calls and puts, with strikes forming a continuum [Carr and Madan, 2002, Bossu et al., 1998]. A portfolio of call options is analogous to a single hidden layer ANN with ReLU activation.

2.3.3 Training Process

The ability of an ANN to approximate any function to an arbitrarily close degree, provided enough breadth or depth, has been established. However, the challenge lies in finding the optimal set $\boldsymbol{\Theta}$ within the weight space for an ANN of a fixed size. As the size of $\boldsymbol{\Theta}$ can quickly escalate – for example, a small network with layers dimensions $[2, 4, 4, 2]$ would encompass 44 parameters (the total sum of the connections and neurons). Trying different values mindlessly is unfeasible.

The method to find the optimal $\boldsymbol{\Theta}^*$ is referred to as training. Before diving into the training of a complex network, let's examine the training process for a single-layer ANN.

Delta Rule

Consider an $NN \in \mathcal{NN}$ with J inputs, I outputs and a single layer with I neurons:

$$y_i = \sigma_i(h_i) \tag{2.3.8}$$

$$h_i = \sum_{i=1}^{I} w_{ij} x_j \tag{2.3.9}$$

And the define the Error Function:

$$E = \frac{1}{2} \sum_{i=1}^{I} (t_i - y_i)^2 \tag{2.3.10}$$

This function measures the discrepancy between our target output t_i and our actual output y_i. Our goal is to identify either the global or at least a local minimum of E. Gradient Descent is a well-known and efficient algorithm to locate a local minimum. For a function F, the argument $\mathbf{a}$ that minimizes the function (locally) is obtained from an arbitrary starting point $\mathbf{a}_0$ by:

$$\mathbf{a} \leftarrow \mathbf{a} - \alpha \nabla F(\mathbf{a}) \tag{2.3.11}$$

Here, ∇ denotes the gradient and α represents the learning rate. This can be seen as "taking a step of size α in the direction of steepest descent".

In terms of the function E, the derivative with respect to a single weight can be computed using the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial \theta_{ij}} \tag{2.3.12}$$

Applying the chain rule again to the second term:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial h_i} \frac{\partial h_i}{\partial w_{ij}} \tag{2.3.13}$$

This gives us:

$$\frac{\partial E}{\partial y_i} = (t_i - y_i) \qquad \frac{\partial y_i}{\partial h_i} = \sigma'(h_i) \qquad \frac{\partial h_i}{\partial w_{ij}} = x_j \tag{2.3.14}$$

which then implies:

$$\frac{\partial E}{\partial w_{ij}} = (t_i - y_i)\sigma'(h_i)x_j \tag{2.3.15}$$

The gradient descent for each weight then becomes:

$$w_{ij} \leftarrow w_{ij} - \Delta w_{ij} \tag{2.3.16}$$

$$\Delta w_{ij} = \alpha(t_i - y_i)\sigma'(h_i)x_j \tag{2.3.17}$$

Commonly, it is denoted as:

$$\Delta w_{ij} = \alpha \delta_i x_j \tag{2.3.18}$$

$$\delta_i = (t_i - y_i)\sigma'(h_i) = \frac{\partial E}{\partial h_i} \tag{2.3.19}$$

Backpropagation

When dealing with multilayer networks, the delta rule cannot be directly applied as there is no specific target for each layer, except for the output layer. However, as ANNs are essentially compositions of functions, we can still leverage the chain rule.

For the output layer, the delta rule remains applicable, but with the previous layer's output instead of the network input. For the hidden layers, each neuron contributes to the error of several neurons in the subsequent layer. Therefore, we need to sum these contributions to determine the error term:

$$\frac{\partial E}{\partial h_j^l} = \delta_j^l = \sum_i \left(\frac{\partial E}{\partial h_i^{l+1}} \frac{\partial h_i^{l+1}}{\partial o_j^l} \right) \frac{o_j^l}{\partial h_j^l} \tag{2.3.20}$$

In this equation, the superscript l represents the layer, and o_j^l symbolizes the output of the jth neuron of the lth layer, as a substitute for y_j, the network output. Here, $\frac{\partial E}{\partial h_i^{l+1}}$ is the error term for neuron i in layer $(l+1)$, $\frac{\partial h_i^{l+1}}{\partial y_j^l}$ is the weight w_{ji}^{l+1}, and $\frac{\partial y_j^l}{\partial h_j^l}$ is the derivative of the activation function $\sigma_j'(h_j^l)$. Hence, the error term for the jth neuron in the lth layer, denoted as δ_j^l, can be rewritten as:

$$\delta_j^l = \left(\sum_i \delta_i^{l+1} w_{ji}^{l+1} \right) \cdot \sigma_j'(h_j^l) \tag{2.3.21}$$

Observe that we initiate calculations from the output layer and proceed backward to the input, hence the nomenclature of the technique. Backpropagation can be viewed as a particular instance of a programming abstraction termed Automatic Differentiation (AD) [Baydin et al., 2018], which encompasses techniques to efficiently compute derivatives in computer programs.

The weight update process remains consistent with the earlier process:

$$w_{jk}^l \leftarrow w_{jk}^l - \alpha \delta_j^l x_k \tag{2.3.22}$$

Chapter 3

Merton's Portfolio Problem

3.1 Problem Definition

Consider a scenario where an investor who retires or inherits a substantial sum of money at time $t = 0$. The investor plans to live for $T \in \mathbb{R}$ more years and doesn't foresee any additional income aside from the portfolio. All consumption is discretionary, meaning that there are no fixed costs or minimum levels of consumption. Regarding the market, we assume that the investor can trade any fractional amount of wealth without incurring transaction costs. We are operating in continuous time. Some of these assumptions are more realistic than others, but they are necessary to make the problem manageable.

Let's denote W_t as the wealth at any given time t. The investor will withdraw a certain amount, c_t, from W_t for consumption, providing a utility $u(c_t)$. Considering that current consumption is more desirable than future consumption, future utilities are subject to discounting at a rate ρ. The investor might wish to leave a bequest for their family or a charity, which is evaluated by a "bequest function". This function is typically assumed to be ϵ^γ for simplicity, where $0 < \epsilon << 1$ results in no bequest (instead of 0, for technical reasons), and γ is a utility-related parameter.

The objective is to maximize the expected time-aggregated utility of consumption by controlling the asset allocation and the consumption, $a_t = [\pi_t, c_t]$:

$$\max_{\pi_t, c_t} \mathbb{E}\left[\int_t^T e^{-\rho(s-t)} u(c_s) ds + \epsilon^\gamma e^{-\rho(T-t)} u(W_T)\right] \tag{3.1.1}$$

In Stochastic Control language, we have a Markovian Decision Process with:

- *State*: (W_t, t)
- *Action*: (π_t, c_t)
- *Reward* at $t < T$: $u(c_t)$
- *Reward* at T: $\epsilon^\gamma u(W_T)$

So far, we have made very few assumptions, which could be adjusted to accommodate more realistic scenarios. For instance, a minimum level of consumption could be imposed in the utility function, and a fixed transaction cost could be added as an additional term in the equation. This flexibility makes Merton's Portfolio Problem (MPP) a robust and realistic framework for such work.

3.2 Analytical Solution

As previously mentioned, 3.1.1 has analytical solution under specific conditions. In this section, we will derive one of these solutions, following [Rao and Jelvis, 2022]. The aim is to work with a stochastic control example to gain a deeper understanding and to have a benchmark for comparison with the Tabular and ANN implementations.

We will further consider:

- A riskless asset, R_t, accruing a known constant rate of return:

$$dR_t = rR_t dt \tag{3.2.1}$$

- A risk asset, S_t following a Geometric Brownian Motion with known μ and σ:

$$dS_t = \mu S_t dt + \sigma S_t dZ_t \tag{3.2.2}$$

- $\mu > r > 0$ and $\sigma > 0$
- The fraction of wealth allocated in the risk asset is a function of the state only, $\pi_t = \pi(W_t, t)$.
- The fraction allocated in the risk-free asset is the complement, i.e. $1 - \pi_t$.
- Constant Relative Risk Aversion (CRRA) Utility function, with risk-aversion parameter γ:

$$u(x) = \begin{cases} \frac{x^{1-\gamma}}{1-\gamma}, & \text{for } 0 < \gamma \neq 1 \\ \log(x), & \text{for } \gamma = 1 \end{cases} \tag{3.2.3}$$

Under these conditions, the Wealth process W_t is defined as:

$$dW_t = ((\pi_t(\mu - r) + r)W_t - c_t)dt + \pi_t \sigma W_t dZ_t \tag{3.2.4}$$

The equation 3.1.1 can be considered an Optimal Value Function for a given state, given the utility function 3.2.3:

$$V^*(W_t, t) = \max_{\pi_t, c_t} \mathbb{E}\left[\int_t^T \frac{e^{-\rho(s-t)} c_s^{1-\gamma}}{1-\gamma} ds + \frac{\epsilon^\gamma e^{-\rho(T-t)} W_T^{1-\gamma}}{1-\gamma} \middle| \mathcal{F}_t\right] \tag{3.2.5}$$

Under careful inspection, it's evident that 3.2.5 is a recursive equation, and the solution would require solving a Hamilton-Jacobi-Bellman (HJB) equation, a common approach in stochastic control theory:

$$V^*(W_t, t) = \max_{\pi_t, c_t} \mathbb{E}\left[\int_t^{t'} \frac{e^{-\rho(s-t)} c_s^{1-\gamma}}{1-\gamma} ds + e^{-\rho(t'-t)} V^*(W_{t'}, t')\right] \tag{3.2.6}$$

The first term corresponds to the reward accrued within the interval $[t, t_1]$ while the second term is the expected rewards, discounted over time. Reformulating this as a Stochastic Differential Equation and reordering the terms, we have:

$$\rho V^*(W_t, t)dt = \max_{\pi_t, c_t} \mathbb{E}\left[dV^*(W_t, t) + \frac{c_t^{1-\gamma}}{1-\gamma} dt\right] \tag{3.2.7}$$

The term $dV^*(W_t, t)$ can be expanded using Itô's lemma:

$$\rho V^*(W_t, t)dt = \max_{\pi_t, c_t} \mathbb{E}\Bigg[\left(\frac{\partial V^*}{\partial t} + \frac{\partial V^*}{\partial W}((\pi_t(\mu - r) + r)W_t - c_t) + \frac{\partial^2 V^*}{\partial W^2}\frac{\pi_t^2 \sigma^2 W_t^2}{2}\right) dt + \pi_t \sigma W_t dZ_t + \frac{c_t^{1-\gamma}}{1-\gamma} dt\Bigg] \tag{3.2.8}$$

The term dZ_t is a martingale and thus $\mathbb{E}[\pi_t \sigma W_t dZ_t] = 0$. By dividing both sides by dt, we obtain the Partial Differential Equation (PDE) for the Hamilton-Jacobi-Bellman (HJB) equation:

$$\rho V^*(W_t, t) = \max_{\pi_t, c_t}\left[\frac{\partial V^*}{\partial t} + \frac{\partial V^*}{\partial W}((\pi_t(\mu - r) + r)W_t - c_t) + \frac{\partial^2 V^*}{\partial W^2}\frac{\pi_t^2 \sigma^2 W_t^2}{2} + \frac{c_t^{1-\gamma}}{1-\gamma}\right] \tag{3.2.9}$$

The right-hand side of the equation can be noted with $\Phi(t, W_t; \pi_t, c_t)$. To obtain the optimal values π_t^* and c_t^*, the partial derivatives with respect to these variables are equated to zero:

$$\frac{\partial \Phi}{\partial \pi_t} = (\mu - r)\frac{\partial V^*}{\partial W_t} + \frac{\partial^2 V^*}{\partial W_t^2}\pi_t \sigma^2 W_t = 0 \tag{3.2.10}$$

$$\pi_t^* = \frac{-(\mu - r)\frac{\partial V^*}{\partial W_t}}{\sigma^2 W_t \frac{\partial^2 V^*}{\partial W_t^*}} \tag{3.2.11}$$

$$\frac{\partial \Phi}{\partial c_t} = -\frac{\partial V^*}{\partial W_t} + (c_t^*)^{-\gamma} = 0 \tag{3.2.12}$$

$$c_t^* = \left(\frac{\partial V^*}{\partial W_t}\right)^{-\frac{1}{\gamma}} \tag{3.2.13}$$

By substituting equations 3.2.11 and 3.2.13 into equation 3.2.9, the following result emerges:

$$\rho V^*(W_t, t) = \frac{\partial V^*}{\partial t} - \frac{(\mu - r)^2}{2\sigma^2} \frac{\left(\frac{\partial V^*}{\partial W_t}\right)^2}{\frac{\partial^2 V^*}{\partial W_t^2}} + \frac{\partial V^*}{\partial W_t} r W_t + \frac{\gamma}{1-\gamma} \left(\frac{\partial V^*}{\partial W_t}\right)^{\frac{\gamma-1}{\gamma}} \tag{3.2.14}$$

With the terminal condition:

$$V^*(W_T, T) = \epsilon^\gamma \frac{(W_T)^{1-\gamma}}{1-\gamma} \tag{3.2.15}$$

It can be confirmed that the second-order conditions are satisfied for $c_t > 0$, $W_t > 0$, $\frac{\partial^2 V^*}{\partial W} < 0$, and $\gamma > 0$. Economically, these conditions can be understood as positive consumption and wealth, diminishing marginal utility, and risk-aversion, respectively.

Now, assume the ansatz:

$$V^*(W_t, t) = f(t)^\gamma \frac{(W_t)^{1-\gamma}}{1-\gamma} \tag{3.2.16}$$

We have the following partial derivatives:

$$\begin{aligned} \frac{\partial V^*}{\partial t} &= \gamma f(t)^{\gamma-1} f'(t) \frac{W_t^{1-\gamma}}{1-\gamma} \\ \frac{\partial V^*}{\partial W_t} &= f(t)^\gamma W_t^{-\gamma} \\ \frac{\partial^2 V^*}{\partial W_t^2} &= -f(t)^\gamma \gamma W_t^{-\gamma-1} \end{aligned}$$

Putting all together, we have the ODE:

$$f'(t) = \nu f(t) - 1 \tag{3.2.17}$$

with:

$$\nu = \frac{\rho - (1-\gamma)(\frac{(\mu-r)^2}{2\sigma^2\gamma} + r)}{\gamma} \tag{3.2.18}$$

The ODE solution is:

$$f(t) = \begin{cases} \frac{1+(\nu\epsilon-1)e^{-\nu(T-t)}}{\nu}, & \text{for } \nu \neq 0 \\ T - t + \epsilon, & \text{for } \nu = 0 \end{cases} \tag{3.2.19}$$

Making the proper substitutions, we finally get:

$$\pi^*(W_t, t) = \frac{\mu - r}{\sigma^2 \gamma} \tag{3.2.20}$$

$$c_t^* = \frac{W_t}{f(t)} = \begin{cases} \frac{\nu W_t}{1+(\nu\epsilon-1)e^{-\nu(T-t)}}, & \text{for } \nu \neq 0 \\ \frac{W_t}{T-t+\epsilon}, & \text{for } \nu = 0 \end{cases} \tag{3.2.21}$$

$$V_t^*(W_t, t) = f(t)^\gamma \frac{W_t^{1-\gamma}}{1-\gamma} = \begin{cases} \frac{(1+(\nu\epsilon-1)e^{-\nu(T-t)})^\gamma}{\nu^\gamma} \frac{W_t^{1-\gamma}}{1-\gamma}, & \text{for } \nu \neq 0 \\ (T-t+\epsilon)^\gamma \frac{W_t^{1-\gamma}}{1-\gamma}, & \text{for } \nu = 0 \end{cases} \tag{3.2.22}$$

The wealth process evolve according:

$$\frac{dW_t}{W_t} = \left(r + \frac{(\mu - r)^2}{\sigma^2 \gamma} - \frac{1}{f(t)}\right) dt + \frac{\mu - r}{\sigma\gamma} dZ_t \tag{3.2.23}$$

The consumption c_t^* is annualization of the instantaneous consumption, i.e. the consumption at given point in time is $c_t^* dt$. For $0 << t < T$ and/or small ϵ the consumption could be larger than 1, which doesn't mean one should borrow to consume. A quantity of interest could be the expected consumption over a year. The direct calculation of $\int_t^T c_\tau^* d\tau$ probably does not yield an analytical solution and numerical procedure is necessary. But we may approximate the percentage consumption over two periods in time as:

$$\%C_{t,t'}^* \approx \int_t^{t'} \frac{1}{f(\tau)} d\tau \tag{3.2.24}$$

Resulting in:

$$\%C_{t,t'}^* \approx \begin{cases} (t'-t)\nu + \log\left(\frac{(\epsilon\nu-1)e^{-\nu(T-t)}+1}{(\epsilon\nu-1)e^{-\nu(T-t')}+1}\right), & \text{for } \nu \neq 0 \\ \log\left(\frac{T-t+\epsilon}{T-t'+\epsilon}\right), & \text{for } \nu = 0 \end{cases} \tag{3.2.25}$$

Earlier in the process, for $\nu \neq 0$, ν dominates the consumption, later the second part of the equation dominates.

The reader may wonder how the action-value function in continuous-time is, but it does not exist [Kim et al., 2021]. The action-value function calculate the expected return, deviating the policy momentarily, and return to the prescribed policy afterwards. The shorter the time Δt the policy is deviated, the smaller the impact, i.e. smaller the difference between the Q-value for a given action and the V-value, until:

$$\lim_{\Delta t \to 0} |Q(s,a) - V(s)| = 0 \quad \forall a \in \mathcal{A} \tag{3.2.26}$$

Therefore, the action-value function in continuous time is not defined.

3.2.1 Remarks

The solution derived herein have interesting implications, some of which are consistent with common investment wisdom, while others conflict with it.

The portion of wealth allocated to the risky asset is time-homogeneous, challenging the age-old belief that younger individuals should invest more in risky assets than their older counterparts. The conditions under which this is true have been previously explored by Samuelson [Samuelson, 1963]. While this holds under the assumptions made in this work, it does not always apply [Kritzman and Rich, 1998, Ross, 1999, Bianchi et al., 2016]. Another fascinating revelation is that the share of wealth invested in the risky asset is reduced to:

$$\pi_t = \frac{\mu - r}{\sigma^2} \tag{3.2.27}$$

For $\gamma = 1$. This equation is also known as Kelly's Criterion and represents the optimal solution for a log-utility investor. This formula can be derived independently by maximizing the logarithm of a portfolio invested in a risky and a risk-free asset, with respect to the fraction invested in the risky asset.

3.2.2 Results

In this section, we illustrate two instances utilizing the analytical solution.

Case 1

The first case have the following set of parameters:

Parameter	W_0	T	μ	σ	r	γ	ρ	ϵ
Value	100	40	8%	35%	5%	0.3	0.06	0.6

The prescribed percentages allocated to the risky asset and consumption throughout the years are depicted as follows:

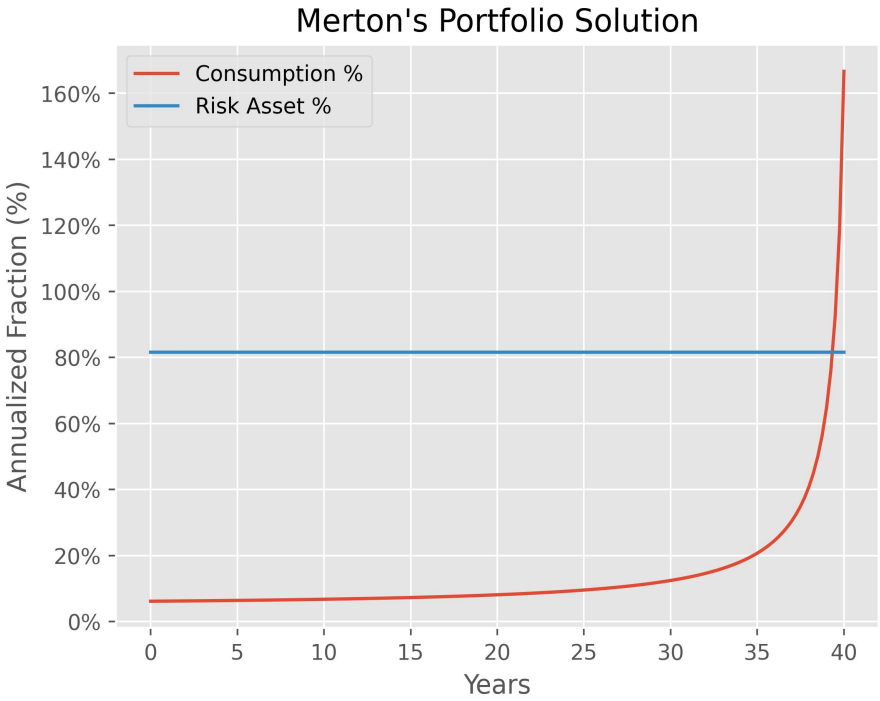


Figure 3.1: Optimal consumption and Investment policy

Notice, the consumption escalates sharply at the end of the period, attributable to the term $\frac{1}{e^{-\nu(T-t)}}$ in equation 3.2.21. At the period's end, the consumption surpasses 100%, but it's important to remind that this represents the consumption rate over dt.

The resulting State-Value function (V-Value) can be visualized in the heatmap below. As it would be expected, higher values of wealth provide higher expected discounted returns. Starting with $W_t = 100$ and $t = 0$, the V-Value is $V(100, 0) = 82.878$.

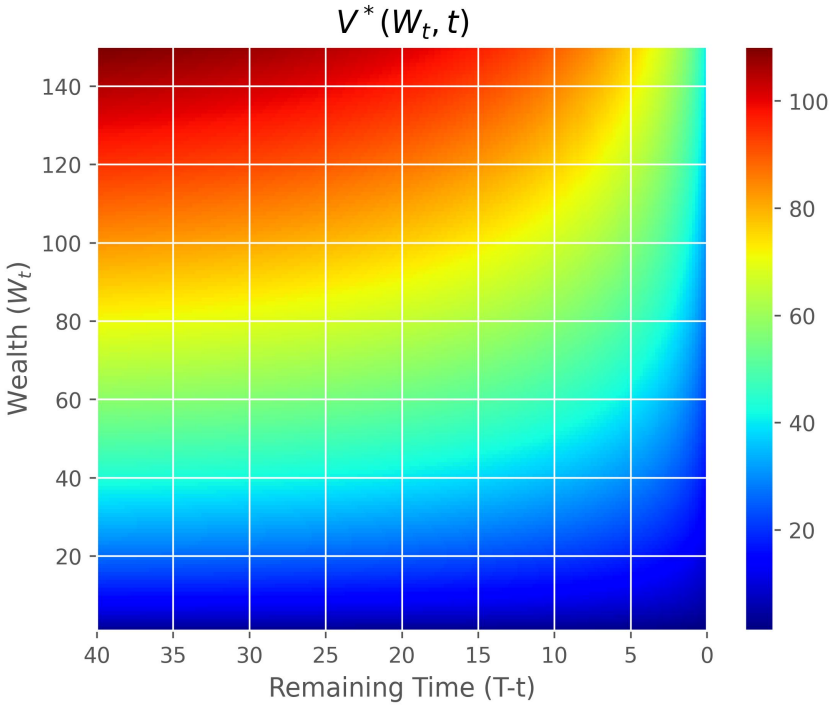


Figure 3.2: State Value (V-Value)

Lastly, a sample of possible wealth processes paths, with two highlighted paths.

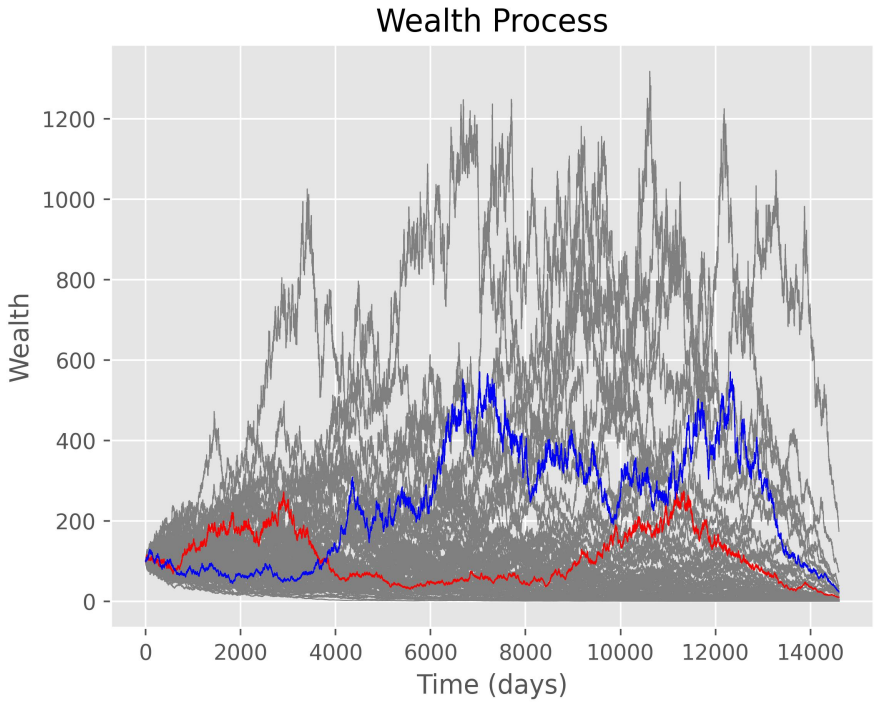


Figure 3.3: Sample of wealth process. Two arbitrary samples highlighted.

Case 2

Due to computational limitations, the discrete numerical implementations will have a shorter time length. In this example, we maintain all parameters consistent with Case 1, except for the time T, which is reduced to $T = 5$, to serve as a benchmark against the solutions in the forthcoming section.

Parameter	W_0	T	μ	σ	r	γ	ρ	ϵ
Value	100	5	8%	35%	5%	0.3	0.06	0.6

Figure 3.4: Optimal consumption and Investment policy

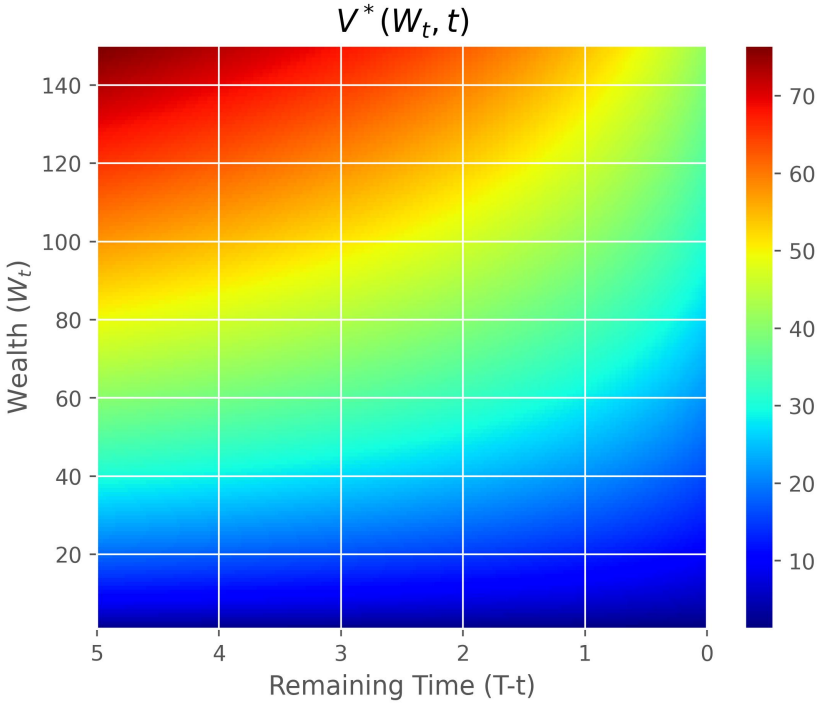


Figure 3.5: State Value (V-Value)

With $V(100, 0) = 57.556$.

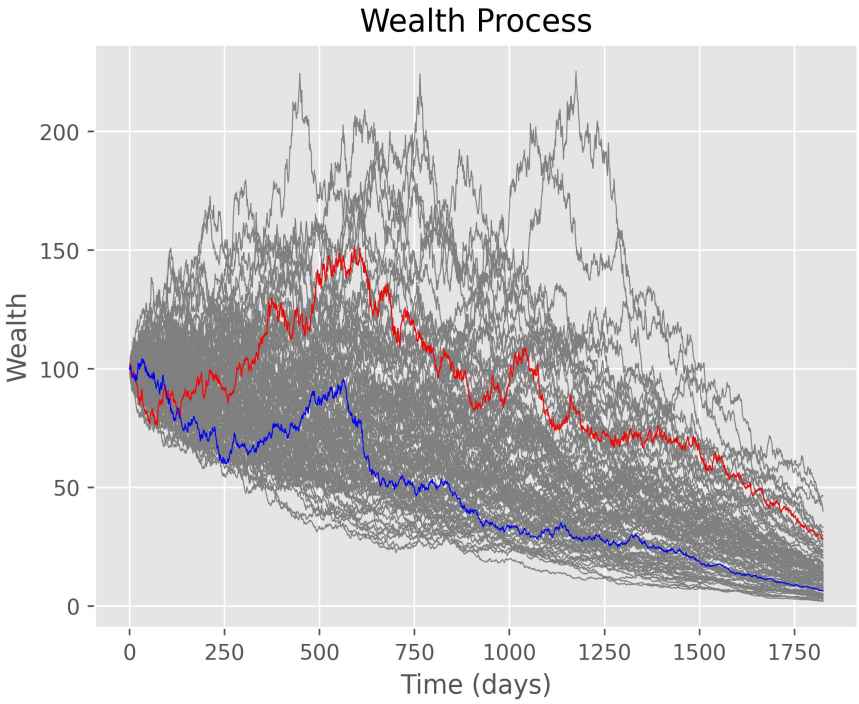


Figure 3.6: Sample of wealth process. Two arbitrary samples highlighted.

3.3 Discrete Merton's Portfolio

In practice, neither consumption nor investment can occur in continuous time. Instead, these actions are performed at discrete time intervals, demanding certain modifications to the model.

Let's consider a partition τ_n on the time interval $[t, T]$, $\tau_n : t = t_0 < t_1 < \cdots < t_n = T$ and $t_{i+1} - t_i = \Delta t$, $\forall t_i \in \tau_n$. The objective now becomes:

$$V_t^*(W_t, t) = \max_{(\pi_t, c_t) \in \mathcal{A}} \mathbb{E}\left[\sum_{\tau=t}^{T} e^{-\rho(T-\tau)} u(c_\tau)\Delta t + \epsilon^\gamma e^{-\rho(T-\tau)} u(W_T) \middle| W_t = w\right] \tag{3.3.1}$$

The utility function remains the same, therefore, we have:

$$V_t^*(W_t, t) = \max_{(\pi_t, c_t) \in \mathcal{A}} \mathbb{E}\left[\sum_{\tau=t}^{T} e^{-\rho(T-\tau)} \frac{c_\tau^{1-\gamma}}{1-\gamma}\Delta t + \epsilon^\gamma e^{-\rho(T-\tau)} \frac{W_T^{1-\gamma}}{1-\gamma} f \middle| W_t = w\right] \tag{3.3.2}$$

The risk-free and risky assets now follow the discrete versions of Equations 3.2.1 and 3.2.2:

$$R_{t+1} = R_t \cdot \Delta R_t \tag{3.3.3}$$

$$\Delta R_t = e^{r\Delta t} \tag{3.3.4}$$

$$S_{t+1} = S_t \cdot \Delta S_t \tag{3.3.5}$$

$$\Delta S_t = e^{(\mu - \frac{\sigma^2}{2}) + \sigma\sqrt{\Delta t}\xi} \tag{3.3.6}$$

In the continuous case, investment and consumption are assumed to occur "simultaneously". However, in the discrete case, we first consume and then invest the remaining amount until the next period. The wealth, therefore, evolves as follows:

$$W_{t+1} = (W_t - c_t)(\pi_t(\Delta S_t - \Delta R_t) + \Delta R_t) \tag{3.3.7}$$

From equation 3.3.2 is direct to see it is a **Bellman optimality equation**:

$$V_t^*(W_t,t) = \max_{(\pi_t,c_t)\in\mathcal{A}} \mathbb{E}\left[\frac{c_t^{1-\gamma}}{1-\gamma} + \sum_{\tau=t}^{T} e^{-\rho(T-\tau)}\frac{c_\tau^{1-\gamma}}{1-\gamma}\Delta t + \epsilon^\gamma e^{-\rho(T-\tau)}\frac{W_T^{1-\gamma}}{1-\gamma}\middle| W_t, t\right] \tag{3.3.8}$$

$$V_t^*(W_t,t) = \max_{(\pi_t,c_t)\in\mathcal{A}} \mathbb{E}\left[\frac{c_t^{1-\gamma}}{1-\gamma} + e^{-\rho\Delta t}\left(\sum_{\tau=t}^{T} e^{-\rho(T-\tau)}\frac{c_\tau^{1-\gamma}}{1-\gamma}\Delta t + \epsilon^\gamma e^{-\rho(T-\tau)}\frac{W_T^{1-\gamma}}{1-\gamma}\right)\right] \tag{3.3.9}$$

$$V_t^*(W_t,t) = \max_{(\pi_t,c_t)\in\mathcal{A}} \mathbb{E}\left[\frac{c_t^{1-\gamma}}{1-\gamma} + e^{-\rho\Delta t}V_{t+1}^*\right] \tag{3.3.10}$$

The optimal action-value function is:

$$Q_t^*(W_t,t;c_t,\pi_t) = \max_{(\pi_{t+1},c_{t+1})\in\mathcal{A}} \mathbb{E}\left[\frac{c_t^{1-\gamma}}{1-\gamma} + e^{-\rho\Delta t}V_{t+1}^*(W_{t+1},t+1)\middle| c_t,\pi_t\right] \tag{3.3.11}$$

Equations 3.3.10 and 3.3.11 are nonlinear equations that should be solved numerically. Moreover, they rely on the calculation of expectations. Even though the dynamics are given in this scenario, the calculation of the expectation can be complex. In other cases, the dynamics might not be known at all.

In the next section, the Discrete Merton's Portfolio problem will be solved using Q-Learning with Tabular and Deep Q-Learning Implementations.

Chapter 4

Numerical Implementation

In this chapter, two implementations of the Q-learning algorithm will be presented. Both implementations, as well the analytical solution can be found at my GitHub repository andrequant/Q-Merton.

First, the Tabular implementation, where the estimated Q-Values are stored in a 4-dimensional tensor, one dimension for each state variable and one for each action variable. Second, the Deep Q-Learning implementation, using a Neural Network to approximate the functional form of the Action-State function.

4.1 Tabular

The main advantage of the Tabular Q-Learning is the guarantee of convergence 2.2.4. With sufficient training, we are assured the true state-action values and hence, the optimal policy are achieved. The main drawback is the agent's size in memory of order $\mathcal{O}(|\mathcal{S}| \times |\mathcal{A}|)$, with $|.|$ being size of each set. In a 100-points discretization in each dimension, the tensor has 10^8 entries, and each entry has to be visited many times during the training process to achieve satisfactory convergence. If we want to generalize the model, say, by adding a second risk asset, the tensor will have 10^{10} entries, and training that starts to become unfeasible. A second issue is that each entry learns individually. A given $Q(s, a)$ does not gain any information from its surrounding Q-values, even if we may expect them to be correlated with each other. Interpolations may help [Szepesvari, 2001].

As the reader remembers from equation 2.2.26, the update of the Q-value depends on the estimated Q-value for the next state (bootstrapping). Therefore, bad estimations on the Q-value for later states impact the estimation for earlier states. A poor estimation also affects the actions taken during the learning process when the ϵ-greedy policy selects to follow the optimal step.

But there is an exception to the bootstrapping: the last step. As our problem has a fixed terminal condition (when remaining times reach zero), the update at $t = T$ is:

$$Q'(s_{T-1}, a_{T-1}) \leftarrow (1 - \alpha)Q(s_{T-1}, a_{T-1}) + \alpha \cdot (r_T + \gamma B(W_T)) \tag{4.1.1}$$

Where $B(.)$ is our terminal condition, the bequest, which is known at this point in the training. In the last step, the bootstrapping problem disappears.

Due to the Markovian nature of the process and fixed episode length, instead of training the whole episode, we can start by training only at the last step. With this, Q_{T-1} will converge faster. Next, with a good estimation of Q_{T-1} on hand, restart the training with two steps; the propagation of the error of Q_{T-1} to Q_{T-2} will be small, resulting in faster convergence of Q_{T-2} (note Q_{T-1} will keep improving as well). Continue this process until training for the desired number of time steps.

The algorithm is below:

Algorithm 2 Backward Training

```
Initialize Q-table;
Initialize n as total length of time steps;
for i = 1, ..., n do
    while not stopping training criteria do
        Initialize Environment with steps = i
        for step = 1, ..., i do
            Take action ε-greedy
            Get reward and next state
            Update Q
```

This process is repeated a certain number of times. Each time the backward training restart, the learning rate α and the ϵ of the ϵ-greedy policy decay:

$$\alpha_{i+1} = \lambda_\alpha \alpha_i \qquad \epsilon_{i+1} = \lambda_\epsilon \epsilon_i \tag{4.1.2}$$

with $\lambda_\alpha, \lambda_\epsilon < 1$. At this moment, the agent's performance is evaluated, measuring the return of following the optimal policy many times. The evaluation is not performed at each Q-Table update due to the computational time required, which would slow down the training process significantly.

4.1.1 Results

Due to computational restrictions, the problem was limited to only five time steps. The complete discretization is:

	Min Value	**Max Value**	**Nº of steps**
Wealth	0	250	126
Time	1	5	5
Consumption %	0%	90%	19
Risk Investment %	0%	150%	16

In the event that wealth exceeds the maximum value, the Reward is assigned to the maximum wealth in the table. The following were the environment parameters used:

Parameter	W_0	T	μ	σ	r	γ	ρ	ϵ_{beq}
Value	100	5	8%	35%	5%	0.3	6%	0.6

and training parameters:

Parameter	**Initial α**	**Final α**	**Initial ϵ**	**Final ϵ**	**Nº Episodes**
Value	1	1e-4	2	0.5	6e4

Every 2,000 episodes of training were followed by a estimation of the true return by simulating additional 2,000 episodes following the (currently) optimal policy. Below are is the evolution of the Return.

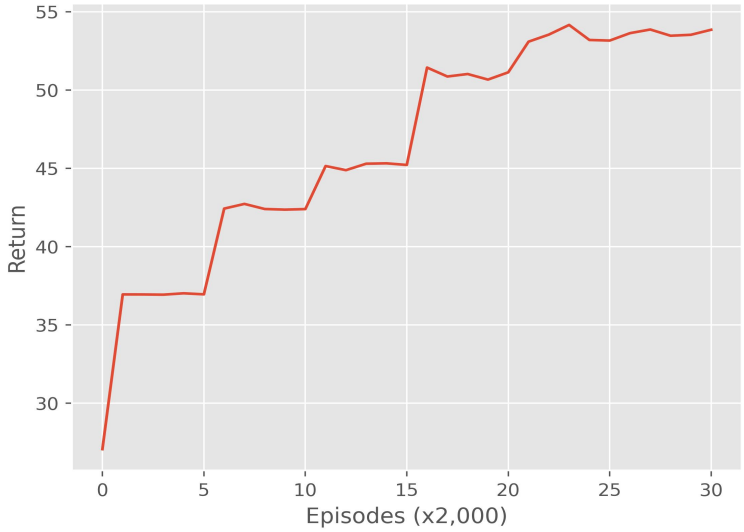


Figure 4.1: Return vs Training

The initial point in the graph represents an agent following a uniform random policy. Backward training was then implemented, with each leap in return corresponding to the training moving on to the next timestep. From the 20 to 30 (x2,000) marks, the training was performed on the entire time length. For each episode, the initial wealth was set according to a uniform distribution $\mathcal{U}[0, 150]$, except when the complete timeframe was trained (i.e., remaining time of five years), in which case the initial wealth was set at 100. The simulations always used the complete timeframe and set the initial wealth as 100.

The average simulated actual return was 53.851, with a standard error of 0.226, but the expected Return, or V-value, was estimated at 82.407, representing a significant overestimation.

The lack of improvement towards the end of the training process may suggest the agent had converged to the optimal policy, (even with the overestimation of individual Q-values, the policy can be optimal if the overestimation is homogeneous across states and actions). While we cannot definitively say if the agent had converged to the optimal solution and true state and action-state values, the action-state values may provide some insights.

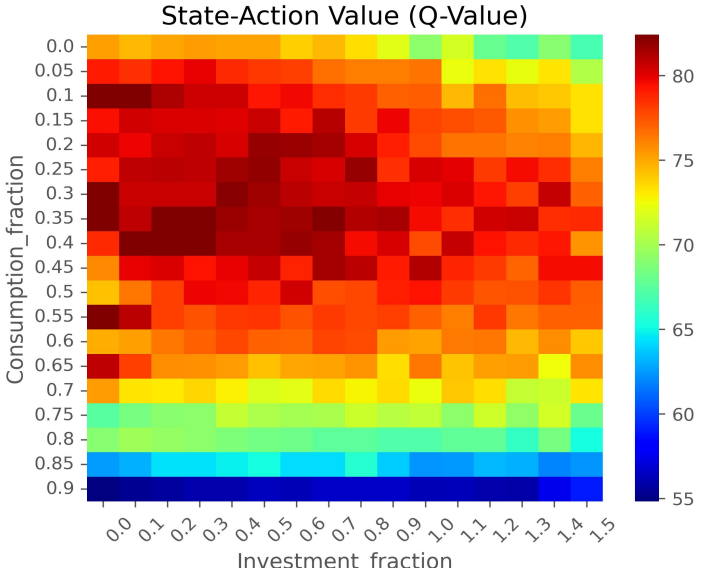


Figure 4.2: Q-Values for Wealth = 100 and Remaining Time = 5.

The Q-Values at this point do not depict a smooth graph, and a comparison with the continuous case might suggest that the agent hasn't converged yet. However, the graph does give a sense of

what the true solution might look like. Note that with the full timeframe, the training always starts with wealth set at 100, thus, this state is visited many times. Other states are visited either through uniform initialization or as the subsequent state of a previous state, hence, they are visited much less frequently.

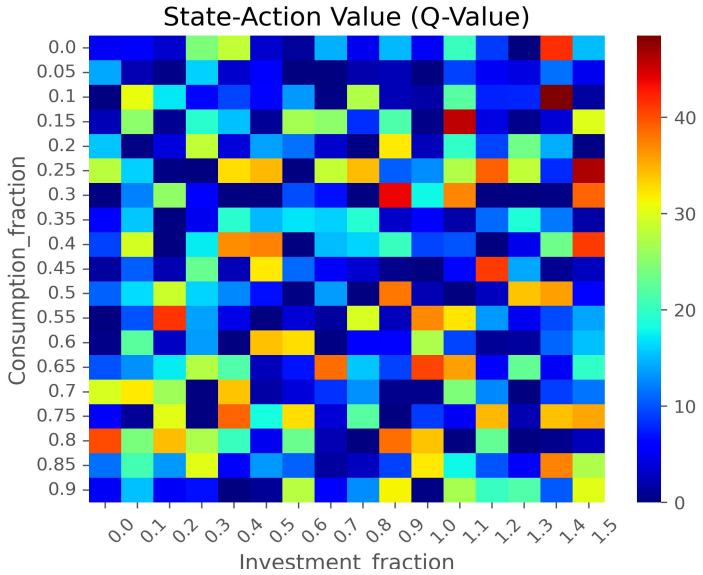


Figure 4.3: Q-Values for Wealth = 100 and Remaining Time = 1.

The state at figure 4.3 clearly haven't achieved a good estimation of the Q-Values at this point in training.

A characteristic of the ϵ-greedy policy is its tendency to better estimate the Q-Value of the optimal action than for other actions, as it transitions from an exploratory to an exploitative behavior. Hence, it is possible to have a poor estimation of the Q-Values in general, but a good estimation of the V-Values.

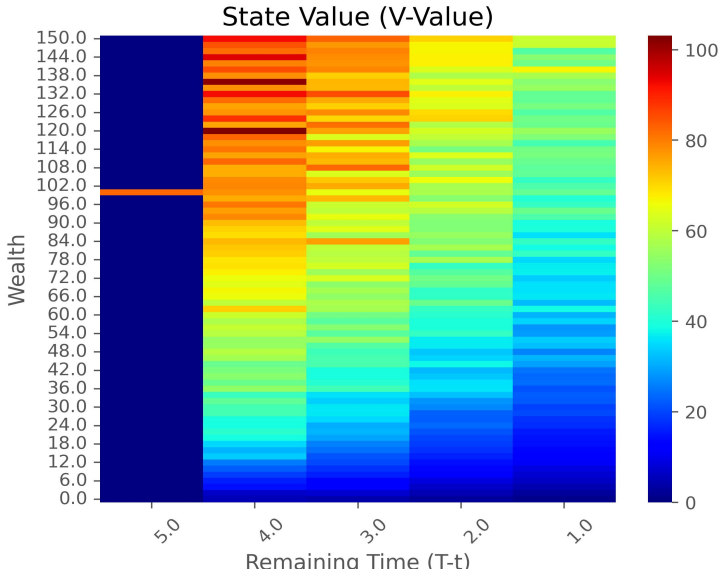


Figure 4.4: V-Value for each wealth and remaining time. For remaining time = 5, other levels of wealth different of 100 are not trained.

While the V-value graph visually resembles the continuous case, it fails to be monotonic on each axis and also reaches higher values, indicating an overestimation.

Training for an additional 150,000 episodes resulted in an average return of 56.215, with a standard error of 0.302, which is very close to the continuous solution in case 3.2.2. The estimated

V-value was 70.935, still overestimating.

The resulting graphs were:

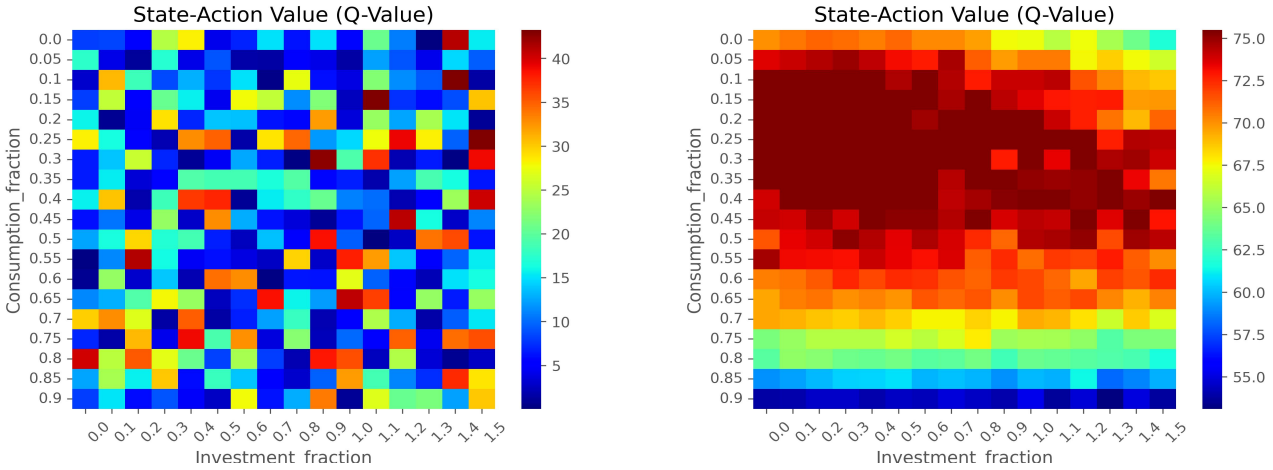


Figure 4.5: Q-Values for Wealth = 100 and Remaining Times = 1 and 5 respectively.

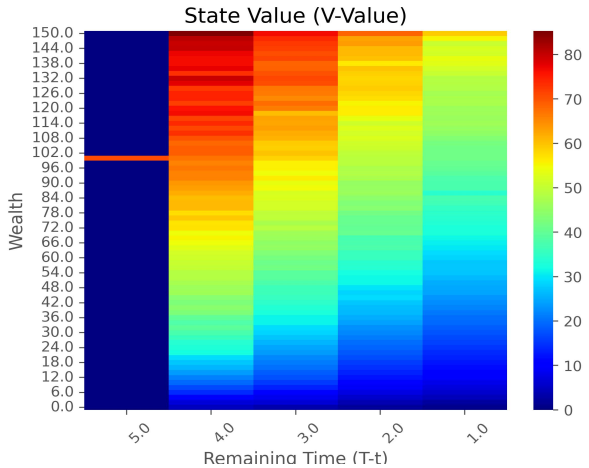


Figure 4.6: V-Value for each wealth and remaining time.

Although the Q-Value for values other than $W_t = 100$ and $T - t = 5$ are poorly estimated, the V-value seems much closer than the continuous case. Here are some V-values for comparison:

$t \setminus W_t$	**50**	**100**	**150**
0	35.43	57.55	76.44
2	31.53	51.23	68.04
4	25.15	48.85	54.26

Table 4.1: Selected V-Values for the Analytical Solution

$t \setminus W_t$	**50**	**100**	**150**
0	NA	70.93	NA
2	37.80	58.63	76.32
4	25.19	41.36	59.25

Table 4.2: Selected V-Values for Tabular Q-Learning, trained with 210,000 episodes.

To provide a sense of the time scale involved in training, the process of training 150,000 episodes took approximately 17 minutes. The code was implemented in Python, utilizing the Pandas package

for tensor manipulation. Faster execution could have been achieved with the use of Numpy or TensorFlow, but such tools might have complicated the implementation. The code was executed in Google Colab, a cloud-based Python notebook with specifications equivalent to an Intel Xeon @2.20 GHz processor and 13GB of RAM.

4.2 Deep Q-Learning

Deep Q-Learning (DQL) offers several advantages over the Tabular approach. First, only the action space is discretized, the state space can be continuous. This allows DQL to handle the curse of dimensionality more effectively. Additionally, by approximating a smooth function with respect to state variables, states that haven't been visited can still benefit from the learning of other states. However, these benefits come at the expense of a lack of guaranteed convergence.

Leveraging Artificial Neural Networks (ANNs) to approximate the Q-Function brings forth several benefits:

- **Function Approximation:** ANNs excel in handling continuous states and vast state-action spaces.
- **Generalization:** They can estimate the Q-value for state-action pairs that have never been encountered.
- **Efficient Implementation:** Contemporary ANN programming tools employ cutting-edge techniques, enhancing both implementation and training efficiency.
- **Parallelization:** Training ANNs can be parallelized, allowing for independent training across multiple processing units. Modern graphics cards, with over 3000 processing units, can expedite the training process significantly.

However, using ANNs also introduces challenges. The main issue is the absence of a convergence guarantee. Without this, it's uncertain if a chosen NN architecture and training procedure will converge. Even if convergence is achieved, the selection of architecture, activation function, hyperparameters (like learning rate and ϵ), and optimizer [1] can greatly influence the rate of convergence. Another challenge is the steep learning curve associated with using these tools. In this study, the TensorFlow v2 package with the Keras API was employed. While Keras offers a user-friendly syntax for basic applications, its complexity can increase with more intricate problems, necessitating a deep understanding of the documentation. Lastly, efficient utilization of the package can be challenging. TensorFlow employs its unique datatype, the *Tensor*, which is efficient within its ecosystem but can have compatibility issues with other Python packages. Converting Tensors to Numpy arrays, for example, can be a slow process and may become a program's bottleneck. Thus, it's crucial to design the code to minimize such conversions.

4.2.1 Implementation

Architecture

The Artificial Neural Network architecture follows:

- **Layer 1**: 2 input neurons.
- **Layers 2**, **3** and **4**: 512 neurons with ELU activation.
- **Layer 5**: 256 neurons with Linear activation.
- **Layer 6**: Reshaping of Layer 5 into a 16x16 grid.

Each output in Layer 6 represents the Q-value for each consumption-investment fraction pair. While the ANN allows for continuous input, the output remains discretized. The consumption fraction, ranging from [0,1], is divided into 16 levels, and the investment, ranging from [0,1.5], is also divided into 16 levels.

The NN is fully connected, meaning every neuron in a layer connects to all neurons in the subsequent layer. The network has 723,968 trainable weights.

[1] ANN packages offer more efficient training techniques than the simple Gradient Descent presented in 2.3.3 such as Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (ADAM) [Kingma and Ba, 2014] and Nesterov-accelerated Adaptive Moment Estimation (NADAM) [Dozat, 2016].

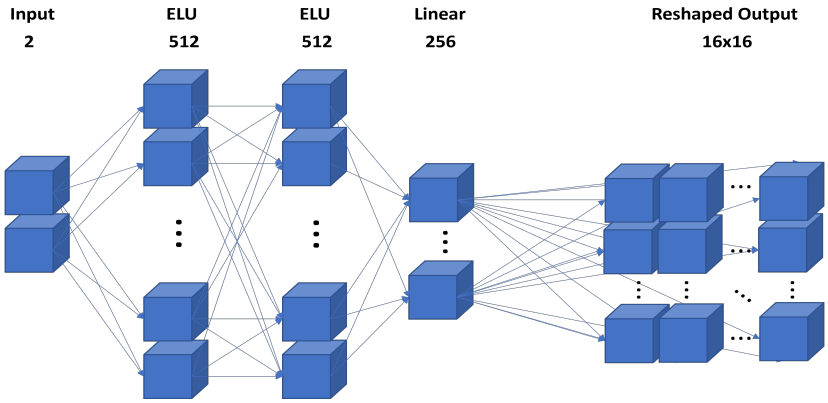


Figure 4.7: ANN Architecture

The activation functions ReLU, ELU and Logistic were tested. The ReLU often encountered the dying ReLU problem. The Logistic, constrained to $(0, 1)$, struggled to learn higher rewards even with the linear layer. ELU yielded satisfactory outcomes.

Training Procedure

Training this model required adjustments to the training procedure used in the tabular implementation. ANNs typically perform better with inputs rescaled to values closer to one. With wealth values around 100, the network can easily diverge. Given our utility function choice, both the V and Q functions can be scaled using the following identities:

$$f(W_t, t) = g(a) f\left(\frac{W_t}{a}, t\right) \tag{4.2.1}$$

$$g(a) = a^{1-\gamma} \tag{4.2.2}$$

Before inputting into the network, the wealth will be scaled by a factor of 100. The results will then be rescaled to ensure comparability across the ANN, Tabular, and Analytical solutions.

The invariance could be further explored as the Q-values, properly scaled, should have the same values. This could be explored by modifying the Error Function to take this fact into account or by averaging the values, across different W_t, in the output. In this work, neither of those options were applied.

The training procedure doesn't directly update the estimated Q-value for a specific state and action. Instead, it modifies the network's weights as described in section 2.3.3. The Stochastic Gradient Descent algorithm was employed, where the Error Function is derived from batches of samples. While the network outputs a 16x16 grid of estimated Q-values, a sample of realized return provides only a single reward. To address this, the Error Function was adapted to:

$$E = \frac{1}{2} \sum_{i=1}^{I} \sum_{\substack{s,a \\ \in \mathcal{S},\mathcal{A}}} \delta^{s,a} (t_i - y_i^{s,a})^2 \tag{4.2.3}$$

Here, $\delta^{s,a} = 1$ for the network output that matches the sample's state-action and 0 otherwise.

The backward training yielded poor results. Once you trained for $T - t = 1$ and goes to train for $T - t = 2$, this new input will have the estimation improved, regardless if it deteriorate what was already trained. The best result was achieved by training all T in parallel.

As mentioned earlier, converting back and forth Tensors is computationally intensive. The environment was built within the Numpy framework, not TensorFlow. Thus, each interaction between the agent and the environment requires conversion, slowing down the training process. However, this issue can be overcome by leveraging the problem's Markovian property. The Q-value for $W_t = 100$ and $t = 3$ is the same, irrespective of whether the environment was $W_t = 120$ or $W_t = 90$ in the previous step. Hence, simulating an entire episode isn't necessary. Instead, multiple environments can be initialized with different W_t and t values, an random or greedy action can be chosen, its respective reward obtained, and the network updated accordingly. This approach avoids the time-consuming *Agent* $\rightarrow$ *Environment* interaction.

For each set of realizations, the ANN is trained with a decreasing learning rate (Robbins-Monro condition). The algorithm follows a purely random policy, as opposed to an ϵ-greedy one. This

choice might slow down the V-value estimation since the optimal action is visited less frequently. However, it provides a more accurate Q-value estimation for other actions, and avoid the Agent-Environment interaction.

The complete training algorithm is:

Algorithm 3 ANN Training

Initialize ANN;
Set n (Number of batches);
Set α_0 (Initial Learning Rate)
Set α_T (Final Learning Rate)
Set α-decay = $(\alpha_T/\alpha_0)^{1/n}$
Set Size (number of different initial states)
Set Epochs

for $i = 1, \ldots, n$ **do**
    Initialize *Size* states with uniformly random W_t and t
    Select *Size* random actions
    Receive the Return R_{t+1} of each state-action
    Define $Q(W_t, t; a) = R_{t+1} + \gamma \max_{a'} Q(W'_t, t')$ for each state-action
    $\alpha = \alpha_0$

    for $i = 1, \ldots, Epochs$ **do**
        Sample (Size/Epochs) samples
        Update the NN using the pairs $\langle (W_t^i, t^i, a^i), Q^i \rangle$
        $\alpha \leftarrow \alpha * \alpha$-decay

The *Epochs* loop refines the network based on the given sample $\langle (W_t^i, t^i, a^i), Q^i \rangle$, but it doesn't necessarily converge to the true Q-value. This convergence happens across the n loop. The pair $\langle (W_t^i, t^i, a^i), Q^i \rangle$ represents a sample input-output relationship, making the update essentially a supervised learning process.

4.2.2 Results

The environment parameters remained the same as the previous settings. The training parameters are outlined below:

Parameter	**Initial α**	**Final α**	**n**	**Size**	**Epochs**
Value	5e-1	1e-3	10	105e3	3

The evolution of return is presented below:

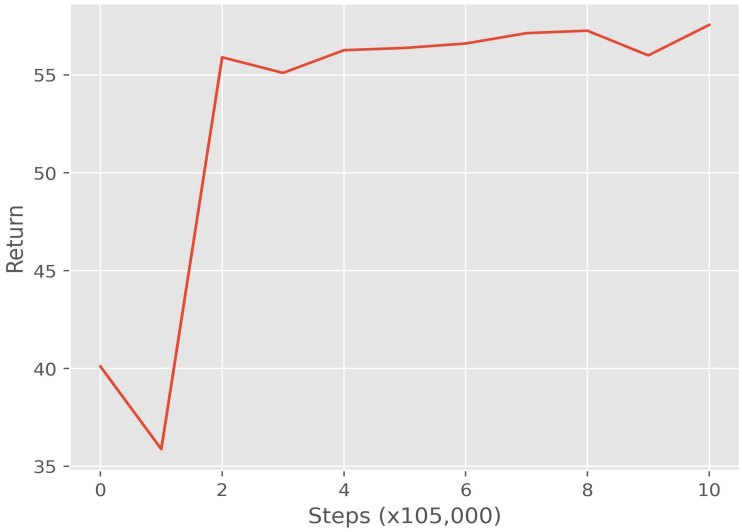


Figure 4.8: Return vs Training

The starting point represents the average return of the untrained model, which is influenced by the network's initial random state. The first training batch is heavily influenced by the random initialization. Notably, the second training batch shows a significant improvement. By the final batch, the average return for $W_t = 100$ and $t = 0$ is 57.558 with a standard error of 0.066, remarkably close to the analytical result of 57.556. Simulating 10,000 episodes for increased precision yielded an average return of 57.348 with a standard error of 0.030.

This striking result is consistent across various states. For example, for $W_t = 50$ and $t = 2$, the result was 31.432 (SE: 0.152) compared to the analytical value of 31.533. For $W_t = 150$ and $t = 3$, the result was 62.386 (SE: 0.027) against 62.224. The model even demonstrated (limited) extrapolation capabilities for untrained states. For instance, for $W_t = 100$, $t = 7$ and $t = 10$, the results were 61.401 (SE: 0.050; Analytical: 62.103) and 65.270 (SE: 0.068; Analytical: 67.080), respectively. These outcomes suggest that the derived policy is very close to the optimal one.

The Q-values, as shown in figure 4.9, may not appear smooth, but they now offer consistent estimations across different times. It's intriguing to observe how the higher Q-values shift from low consumption levels at the episode's beginning to higher consumption levels towards its end. The heatmap also indicates that higher expected returns are more concentrated on its right side. This experimental result suggests that the region surrounding the optimal policy for a given state is relatively flat, making pinpointing the optimal point challenging.

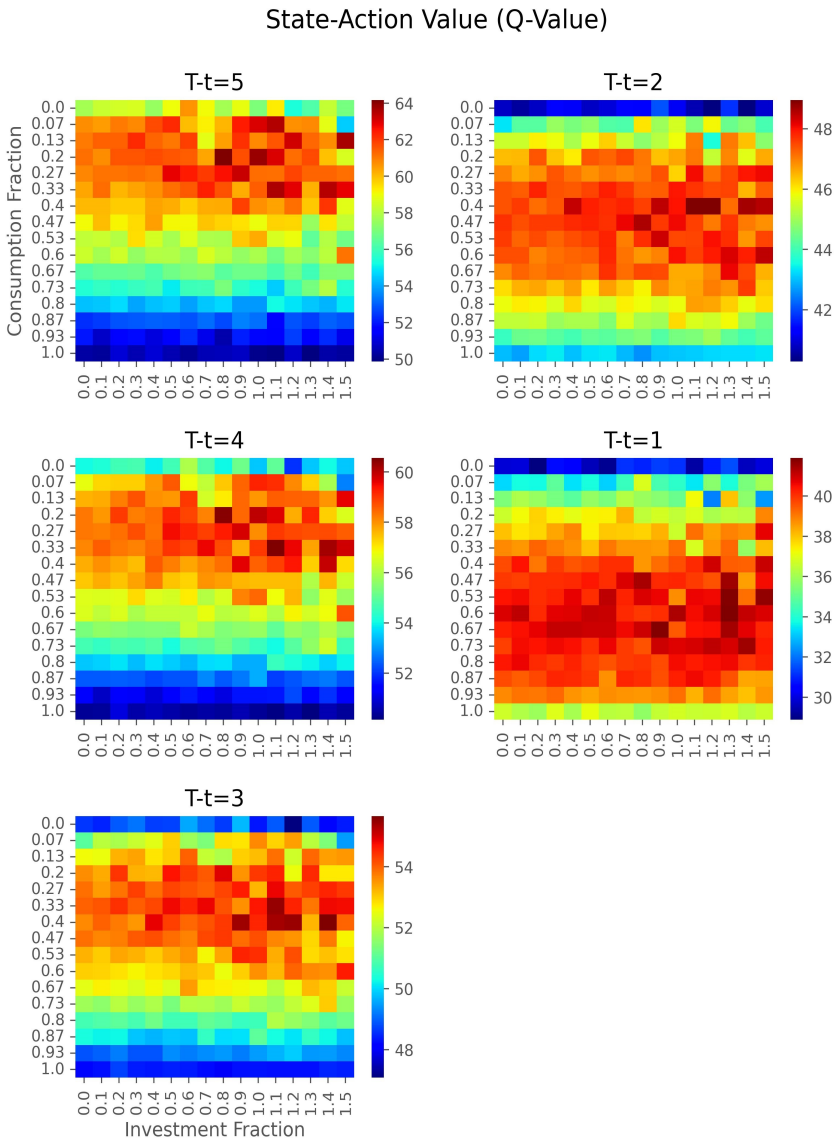


Figure 4.9: Q-Values for Wealth = 100

Given the ANN's ability to accept continuous input, we can plot a continuous heatmap of the V-Value as shown in figure 4.10. The DQL implementation effectively captures the functional form of the State Value. However, the scale suggests a tendency to overestimate. This observation can be further validated by comparing tables 4.1 and 4.3.

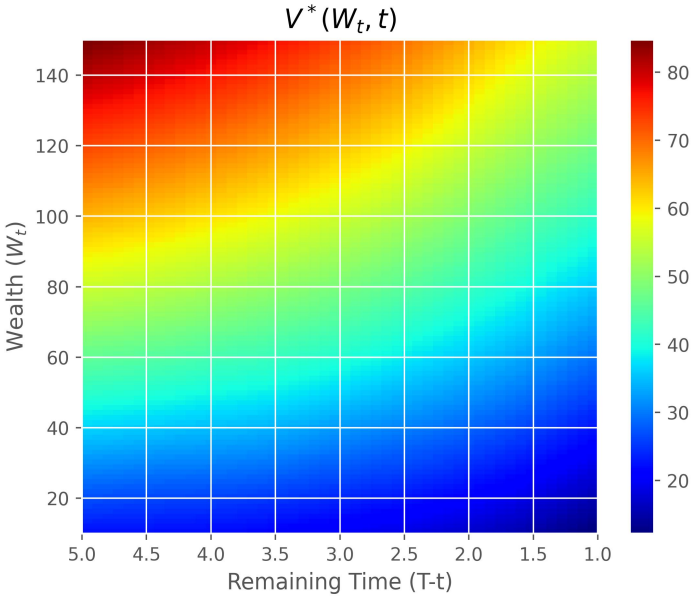


Figure 4.10: State Value (V-Value)

$t \setminus W_t$	**50**	**100**	**150**
0	41.49	64.17	84.75
2	36.65	55.66	72.92
4	26.42	41.89	55.76

Table 4.3: Selected V-Values for DQL

The three models can also be compared by they resulting wealth process for the same underlying risk asset process. In the image 4.11, we have four samples. For the three models they tend to have similar processes, but due to the still erratic behaviour of the action, we find relevant deviations.

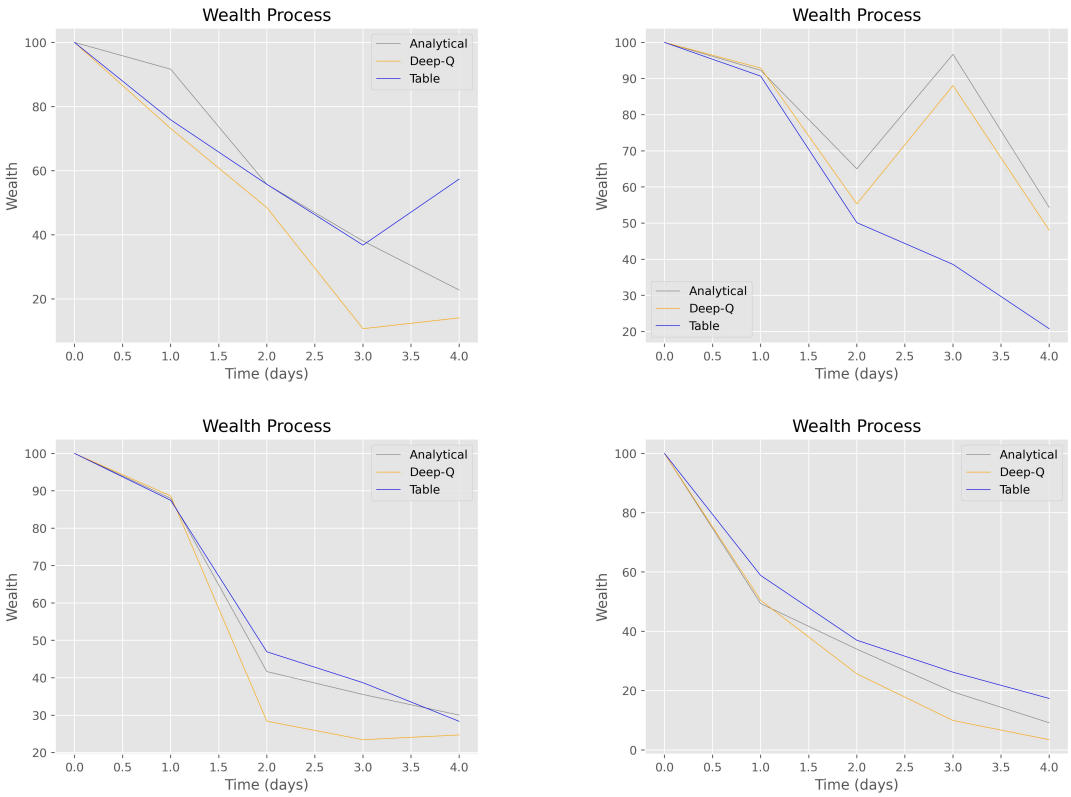


Figure 4.11: Four samples of the wealth process for each model.

Chapter 5

Conclusion

The Merton's Portfolio Problem (MPP) is a fundamental problem in financial mathematics. It encapsulates the challenges and decisions faced by investors, providing a mathematical representation of real-world dilemmas. Academically, the problem encompass different areas of mathematical modelling such as stochastic calculus, stochastic control, dynamic programming and even utility theory and econometrics in the estimation of parameters, offering many possibilities of exploration. But its practical implications extend far beyond the academic interest. It is a problem with immediately implications for pension funds, insurance companies and even retail investors. But in my opinion, the problem is less explored than it could be and even less used as a financial product. I believe it can be attributed to the advanced level of mathematics needed to work with the Merton's Portfolio Problem and the lack of incentives to use this type of resources in retail investors' problems.

The MPP can also be notable challenge. With the set up used in this work, the region surrounding the optimal solution is notoriously flat, it means that find the optimal solution is specially challenge. On the other hand it makes that less-than-optimal solutions performs almost as good as the optimal one. The numerical results presented here also highlights an paradox, investors devote much more time trying to find the optimal investment policy, while the consumption policy is the one with the higher impact in the expected return.

In our exploration of Reinforcement Learning as a tool to solve the MPP, two algorithms were explored: Tabular Q-Learning and Deep Q-Learning. The former with its simple implementation and guaranteed convergence is a powerful tool for scenarios characterized by smaller state and action spaces. However, it might not be well-suited for problems with many decisions variables or large state-action spaces.

Deep Q-Learning, on the other hand, is a more sophisticated object, better equipped to grapple with larger state-action spaces, bringing a robustness that Tabular Q-Learning might lack. However, this robustness comes at a cost. Deep Q-Learning is computationally intensive, requiring a careful implementation or the training time becomes impracticable. Additionally, the complexity of tools used in its development have a steeper learning curve for the one who is implementing.

The use of Machine Learning into the financial sector has been a topic of considerable debate and exploration. Much of the focus has been on harnessing ML's predictive capabilities, especially in forecasting returns, probably due to ML's predictive capabilities in other areas. While this approach has its merits, due to the level of market efficiency faced today, it will problem redeem unfruitful results and overfitting. This focus also overlooks the broader potential of ML in finance. Instead of a narrow focus on prediction, there is a vast landscape of intricate challenges within finance that ML can address, with more realistic and rewarding results. The Merton's Portfolio Problem is a prime example of such problem. It is essential for the financial sector to recognize and embrace this broader perspective, ensuring that ML's full potential is realized.

The exploration of the MPP has raised ideas for potential future research. On the problem itself, variations of the problem such as different utility functions or more nuanced asset behavior models add layers of complexity as well bring the problem closer to real-world scenarios, enhancing its practical relevance. A particularly intriguing propositions is the integration of stochastic time, instead of having a fixed known time length, modeled after actuarial tables. This approach would introduce an additional layer of uncertainty, mirroring the uncertainties of real-life investment scenarios.

On the algorithmic side, there is also room for refinement and exploration. A notable concern is the Q-Learning tendency to overestimate expected returns. This overestimation, result of the maximization term in the Bellman equation, can skew results. The introduction of the Double

Q-Learning algorithm [Hasselt, 2010] offers a potential solution to this challenge. By reducing the overestimation bias, Double Q-Learning promises more accurate and reliable outcomes, enhancing the algorithm's efficacy.

In summary, it is crucial to reflect on the broader implications and potential impact on the machine learning, whether RL or not, in finance. The methodologies and insights gathered on this thesis are not confined to the Merton's Portfolio Problem. Their adaptability means they can be applied to a myriad of similar problems within finance. As the world of finance continues to evolve, and the Brazilian market start to embrace the use of advance mathematical methods, I hope this work serves as a starting point for those interesting in working with Reinforcement Learning within financial problems, whether academically or in the industry.

Bibliography

[Mel,] Convergence of Q-learning: a simple proof.

[Jaa, 1993] (1993). On the Convergence of Stochastic Iterative Dynamic Programming Algorithms.

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

[Anderson and Rosenfeld, 1988] Anderson, J. A. and Rosenfeld, E., editors (1988). *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, MA, USA.

[Bailey, 1862] Bailey, A. H. (1862). On the principles on which the funds of life assurance societies should be invested. *Journal of the Institute of Actuaries*, 10:142–147.

[Balduzzi and Lynch, 1999] Balduzzi, P. and Lynch, A. W. (1999). Transaction costs and predictability: some utility cost calculations. *Journal of Financial Economics*, 52:47–78.

[Barberis, 2000] Barberis, N. (2000). Investing for the long run when returns are predictable. *The Journal of Finance*, 55:225–264.

[Baydin et al., 2018] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: a Survey. *Journal of Machine Learning Research*, 18(153):1–43.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*.

[Bernoulli, 1954] Bernoulli, D. (1954). Exposition of a new theory on the measurement of risk. 22:23–36.

[Bianchi et al., 2016] Bianchi, R. J., Drew, M. E., and Walk, A. N. (2016). The time diversification puzzle: A survey. *Financial Planning Research Journal*, 1.

[Bianchini and Scarselli, 2014] Bianchini, M. and Scarselli, F. (2014). On the complexity of shallow and deep neural network classifiers. *European Symposioum on Artificial Neural Networks. Computational Intelligence and Machine Learning*.

[Bossu et al., 1998] Bossu, S., Carr, P., and Papanicolaou, A. (1998). A functional analysis approach to the static replication of european options. 29:417–427.

[Bozinovski, 2020] Bozinovski, S. (2020). Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica*, 44.

[Brennan et al., 1997] Brennan, M. J., Schwartz, E. S., and Lagnado, R. (1997). Strategic asset allocation. *Journal of Economic Dynamics and Control*, 21:1377–1403.

[Campbell and Viceira, 2001] Campbell, J. Y. and Viceira, L. M. (2001). *Strategic Asset Allocation: Portfolio Choice for Long-Term Investors*.

[Carr and Madan, 2002] Carr, P. and Madan, D. (2002). Towards a theory of volatility trading.

[Cover, 1991] Cover, T. M. (1991). Universal portfolios. *Mathematical Finance*, 1:1–29.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function*. *Math. Control Signals Systems*, 2:303–314.

[de Prado, 2020] de Prado, M. M. L. (2020). Machine learning for asset managers. *Elements in Quantitative Finance*.

[Dozat, 2016] Dozat, T. (2016). Incorporating nesterov momentum into adam. 2016.

[Ertel, 2017] Ertel, W. (2017). Introduction to Artificial Intelligence.

[Fan et al., 2019] Fan, J., Wang, Z., Xie, Y., and Yang, Z. (2019). A Theoretical Analysis of Deep Q-Learning. *Proceedings of Machine Learning Research*, 120:486–489.

[Finetti, 1939] Finetti, B. D. (1939). La teoria del rischio e il problema della rovina dei giocatori. *Istituto italiano degli attuariali*.

[Hasselt, 2010] Hasselt, H. V. (2010). Double q-learning.

[Hornik, 1993] Hornik, K. (1993). Some new results on neural network approximation. *Neural Networks*, 6:1069–1072.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

[Kelly, 1956] Kelly, J. L. (1956). A new interpretation of information rate.

[Kim et al., 2021] Kim, J., Shin, J., and Yang, I. (2021). Hamilton-jacobi deep q-learning for deterministic continuous-time systems with lipschitz continuous controls. *Journal of Machine Learning Research*, 22:1–34.

[Kim and Omberg, 1996] Kim, T. S. and Omberg, E. (1996). Dynamic nonmyopic portfolio behavior. *Review of Financial Studies*, 9:141–161.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. L. (2014). Adam: A method for stochastic optimization.

[Kinlaw et al., 2017] Kinlaw, W., Kritzman, M. P., and Turkington, D. (2017). *A Practioner's Guide to Asset Allocation*. John Wiley & Sons.

[Kritzman and Rich, 1998] Kritzman, M. and Rich, D. (1998). Beware of dogma: The truth about time diversification. *Journal of Portfolio Management*, 24.

[Kumar, 2020] Kumar, V. (2020). Mathematical analysis of reinforcement learning — bellman optimality equation. https://web.archive.org/web/20230712184328/https://towardsdatascience.com/mathematical-analysis-of-reinforcement-learning-bellman-equation-ac9f0954e19f?gi=f8c3c1665c34. Accessed: 2023-07-12.

[Lederer, 2021] Lederer, J. (2021). Activation Functions in Artificial Neural Networks: A Systematic Overview.

[Litterman, 2004] Litterman, B. (2004). *Modern investment management: an equilibrium approach*. John Wiley & Sons.

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

[Markowitz, 2019] Markowitz, H. M. (2019). The early history of portfolio theory: 1600–1960. *https://doi.org/10.2469/faj.v55.n4.2281*, 55:5–16.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

[Merton, 1969] Merton, R. C. (1969). Lifetime portfolio selection under uncertainty: The continuous-time case. 51:247–257.

[Merton, 1971] Merton, R. C. (1971). Optimum consumption and portfolio rules in a continuous-time model. *Journal of Economic Theory*, 3:373–413.

[Michie, 1963] Michie, D. (1963). Experiments on the mechanization of game-learning part i. characterization of the model and its parameters. *The Computer Journal*, 6:232–236.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning.

[Modirshanechi, 2020] Modirshanechi, A. (2020). Why does the optimal policy exist? https://web.archive.org/web/20230719074454/https://towardsdatascience.com/why-does-the-optimal-policy-exist-29f30fd51f8c?gi=2ccea0a71bc0. Accessed: 2023-07-12.

[Neumann and Morgenstern, 1953] Neumann, J. V. and Morgenstern, O. (1953). *Theory of Games and Economic Behavior (Commemorative Edition)*. Princeton University Press.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Petersen, 2022] Petersen, P. C. (2022). Neural network theory.

[Pomerleau, 1989] Pomerleau, D. (1989). Alvinn: An autonomous land vehicle in a neural network. In Touretzky, D., editor, *Proceedings of (NeurIPS) Neural Information Processing Systems*, pages 305 – 313. Morgan Kaufmann.

[Ramaswamy and Hullermeier, 2022] Ramaswamy, A. and Hullermeier, E. (2022). Deep Q-Learning: Theoretical Insights from an Asymptotic Analysis. *IEEE Transactions on Artificial Intelligence*, 3(2):139–151.

[Rao and Jelvis, 2022] Rao, A. and Jelvis, T. (2022). *Foundations of Reinforcement Learning with Applications in Finance*. Chapman and Hall/CRC.

[Ross, 1999] Ross, S. A. (1999). Adding risks: Samuelson's fallacy of large numbers revisited. *The Journal of Financial and Quantitative Analysis*, 34:323.

[Rubinstein, 2006a] Rubinstein, M. (2006a). Bruno de finetti and mean-variance portfolio selection.

[Rubinstein, 2006b] Rubinstein, M. (2006b). *A History of the Theory of Investments*. John Wiley & Sons, Inc.

[Samuelson, 1963] Samuelson, P. A. (1963). Risk and uncertainty: A fallacy of large numbers. *Scientia*, 57(98):108.

[Samuelson, 1969] Samuelson, P. A. (1969). Lifetime portfolio selection by dynamic stochastic programming. *The Review of Economics and Statistics*, 51:239–246.

[Samuelson, 1971] Samuelson, P. A. (1971). The "fallacy" of maximizing the geometric mean in long sequences of investing or gambling. *Proceedings of the National Academy of Sciences*, 68:2493–2496.

[Scherer, 2007] Scherer, B. (2007). *Portfolio Construction and Risk Budgeting*. Risk Books.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Smith, 1924] Smith, E. L. (1924). *Common Stocks As Long Term Investments*.

[Sutton and Barto, 2020] Sutton, R. S. and Barto, A. G. (2020). *Reinforcement Learning: An Introduction*. The MIT Press, 2 edition.

[Szepesvari, 2001] Szepesvari, C. (2001). Convergent Reinforcement Learning with Value Function Interpolation. Technical report.

[Thorp, 1975] Thorp, E. O. (1975). Portfolio choice and the kelly criterion. *Stochastic Optimization Models in Finance*, pages 599–619.

[Thorp, 2011] Thorp, E. O. (2011). Understanding the kelly criterion. *The Kelly Capital Growth Investment Criterion: Theory And Practice*, pages 511–525.

[Time, 1961] Time (1961). Science: The goof button - time.

[Turnbull and Farago, 2019] Turnbull, C. and Farago, R. (2019). 200 years of asset allocation.

[Veen, 2019] Veen, F. V. (2019). The neural network zoo. https://web.archive.org/web/20230616211637/https://www.asimovinstitute.org/neural-network-zoo/. Accessed 2023-06-16.

[Wachter, 2002] Wachter, J. A. (2002). Portfolio and consumption decisions under mean-reverting returns: An exact solution for complete markets. *The Journal of Financial and Quantitative Analysis*, 37:63.

[Watkins, 1989] Watkins, C. J. (1989). Learning from delayed rewards.

[Yang et al., 2020] Yang, Z., Xie, Y., and Wang, Z. (2020). A theoretical analysis of deep q-learning.