POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Exploiting Continuous Action Space in Fx Trading with Reinforcement Learning

Tesi di Laurea Magistrale in
Computer Science and Engineering -
Ingegneria Informatica

Author: **Vito Alessandro Monaco**

Student ID: 970876
Advisor: Prof. Marcello Restelli
Co-advisors: Lorenzo Bisi, Luca Sabbioni
Academic Year: 2022-23

Abstract

In recent years, the popularity of artificial intelligence (AI) has surged due to its widespread application in various fields. The financial sector has harnessed its advantages for multiple purposes, including the development of automated trading systems. These systems are designed to interact autonomously with the markets and have demonstrated superior performance compared to human agents. In any case, due to the great volatility and unpredictability of the markets, the need for systems capable of recognizing and managing market trends while limiting potential losses has become increasingly crucial. In this work, we focus on the Foreign Exchange market, known for its extensive liquidity and flexibility. Our approach involves the implementation of a Reinforcement Learning algorithm called Fitted Natural Actor-Critic. This algorithm enables the training of an agent that can effectively interact with the market by utilizing continuous actions, thanks to which it is possible to define both the type and the size of the order to be executed. Furthermore, the adoption of a continuous action space allows for more realistic modelling of transaction costs, as they are dependent on the size of the executed order. In addition, it also facilitates the integration of risk-averse approaches, able to induce more conservative behaviour in the agent.

Keywords: FX-Trading, Actor-Critic, Reinforcement Learning, Risk-aversion

Abstract in lingua italiana

Negli ultimi anni l'intelligenza artificiale è divenuta sempre più popolare grazie al suo crescente utilizzo in ambiti di ogni tipo. Il settore finanziario ha saputo sfruttarne le sue caratteristiche per diversi impieghi, fra cui lo sviluppo di sistemi di trading automatico. Questi sistemi sono in grado di interagire in maniera autonoma con i mercati, dimostrando anche performance superiori rispetto a quello ottenute da trader umani. In ogni modo a causa della grande volatilità e imprevedibilità dei mercati, è diventata sempre più cruciale la necessità di sviluppare sistemi in grado di poter riconoscere e gestire i vari trend di mercato limitando le potenziali perdite. In questo lavoro ci siamo soffermati sul Foreign Exchange market, conosciuto per la sua enorme liquidità e flessibilità. Il nostro approccio ha previsto l'implementazione di un algoritmo di Reinforcement Learning chiamato Fitted Natural Actor-Critic. Questo algoritmo ci ha permesso di addestrare un agente in grado di interagire con il mercato utilizzando azioni continue, con il quale è possibile definire sia la tipologia che l'importo dell'ordine da eseguire. Attraverso l'utilizzo di azioni in un spazio continuo siamo anche riusciti a modellare in maniera più realistica i costi di transazione, rendendoli dipendenti dall'importo dell'ordine eseguito. Infine, ha facilitato l'integrazione di approcci di avversione al rischio, in grado di imprimere un comportamento più conservativo all'agente.

Parole chiave: FX-Trading, Actor-Critic, Reinforcement Learning, Avversione al rischio

Contents

1 | Introduction

The increasing interest in applying machine learning techniques in the financial field has been driven by the widespread adoption of artificial intelligence (AI) and advancements in computational capabilities. Due to the decision-making nature of traders, trading markets have particularly benefited from these techniques, providing a fertile ground for the application of reinforcement learning techniques.

In our work, we focus on the Foreign Exchange market, also called Forex Market, which is a global decentralized or over-the-counter (OTC) market where it is possible to trade currencies.

Due to its decentralized nature, low transaction costs, and continuous trading throughout the week, it is considered the largest and most liquid market in the world. According to the *Bank for International Settlements* (BIS), it had recorded an average daily trading volume of $7.5 trillion in April 2022 [1], a volume that has been growing steadily over the years which reflects the growing interest in Forex currencies trading.

Furthermore, thanks to the technological progress of recent years, it has been possible to interact with the market always with higher frequencies. Combining this factor with the market's high liquidity, which allows for quick buying or selling of assets, has intensified the interest in using techniques that can automatically execute transactions.

An effective approach for automatic trading involves modelling the agent's interaction with the market as a sequential decision-making process, where at each step the agent observes the state of the market and then decides which actions to perform in order to maximize their returns. This continuous interaction between the market and the agent can be modelled as a Markov Decision Process (MDP). Due to the non-stationarity and unknown dynamics of the markets, solving this MDP requires the application of Reinforcement Learning (RL) techniques.

Using RL techniques, we want to create agents able to interact with the market in a realistic and profitable way, taking into account its non-stationarity to reduce any possible risk.

1.1. Motivation and Goals

This work is motivated by the necessity to discover, explore, and expand the existing RL techniques in order to uncover profitable strategies in the Forex trading environment. Achieving this objective requires enhancing the realism of the interaction between the agent and the market, incorporating behaviours to mitigate and limit potential losses resulting from the market's high level of stochasticity.
In fact, many of the approaches currently used in this research field model the market through limiting assumptions, such as the adoption of fixed allocation amounts and a simplified representation of transaction costs. This causes these models to have limited application in real markets. Moreover, the high volatility of the markets underscores the growing need for risk-averse models capable of identifying any uncertainties in the market and enabling agents to interact in a more cautious manner.
Through the use of an Actor-Critic architecture, precisely using the *Fitted Natural Actor Critic* (FNAC, [2]) algorithm, we wanted to combine the benefits of value-based and policy-based approaches. These benefits include the ability to employ any regressor to approximate the value function and the adoption of a continuous action space for the policy. At the same time, thanks to the policy-based nature of the actor component, we investigate the possibility of integrating risk measures and complex transaction costs into the objective function.

1.2. Contributions

This thesis builds upon the research presented in [3], and expands on their principles through the use of an actor-critic architecture that allows us to work in continuous action space.

The main contributions developed in this work can be summarized as follows:

- The implementation from scratch of the FNAC algorithm which, to the best of our knowledge, has been applied for the first time in a real trading environment. This implementation allowed us to thoroughly explore the use of the natural gradient for training the policy-based component.
- The use of a continuous action space to model the interaction with the agent in the market. Thanks to this, we reduce the *model risk* [4] improving the stability of the learning process. Furthermore, this continuous space permits modelling the reward by using a more realistic transaction cost, able to reflect the true costs charged by brokers for large orders.

- The integration of persistence and risk-averse measures into policy-based approaches in Forex trading environments, resulting in more cautious and profitable policies.

1.3. Structure of the Thesis

This thesis, divided into six other chapters excluding this introduction, is organized as follows:

- Chapter 2: In this chapter, we introduce the main theoretical basis of Reinforcement Learning useful for understanding our work. We first explain the Markov Decision Process mathematical framework, then we illustrate the Dynamic Programming approach for solving basic MDPs problems. To extend the DP approaches to be able to work in real problems we present the Monte-Carlo and Time-Difference techniques, introducing then the related approximate solution methods useful for enhancing the generalization capabilities of RL methods. Finally, we introduce the risk-averse approaches for controlling the risk of agent interactions and the concept of action persistence.
- Chapter 3: By initially introducing the Actor-Critic architecture, we explain the Fitted Natural Actor-Critic algorithm which we implemented in this thesis. Then we illustrate the main popular state-of-art Actor-Critic methodologies.
- Chapter 4: We illustrate a brief description of the Reinforcement Learning approaches applied in the financial trading market environment, focusing later on the policy gradient approaches. In addition, we will describe some works that use the value-based FQI [5, 6] algorithm, in which some aspects have been reused in this thesis.
- Chapter 5: here we present the problem formulation of the trading Forex environment used in our work, by also providing additional details about our implementation of the FNAC algorithm.
- Chapter 6: In the first section we describe the model selection details which we used to validate our models. In subsequent sections, we describe the experimental results obtained in the discrete and continuous action space setting, including the integration of size-dependent fees and risk-averse approaches.
- Chapter 7: In this final chapter we will describe the conclusions and obtained goals of our work, providing further insights for future approaches that can be explored to improve our results.

2 | Reinforcement Learning

Reinforcement Learning (RL) is a branch of Machine Learning (ML) and can be defined as a computational trial and error approach for training agents to learn an optimal behaviour to achieve a specific goal. The learning process takes place through interactions performed by the agent towards an external environment, these interactions generate rewards that measure the effectiveness of the chosen actions in pursuing his predetermined goal.

The main peculiarity of RL consists in the fact that actions do not only affect the immediate reward, but also the dynamics of the environment and, consequently, the subsequent rewards.

The trial and error approach and the delayed reward distinguish the RL subfield from supervised learning, for which each training sample is provided with the true target given by an informed external supervisor, and from unsupervised learning, which exploits patterns without any ground truth value to build a representation to be used for reasoning or prediction.

RL has a rich and extensive history that dates back to the 1950s when it was initially studied in the field of artificial intelligence [7, 8], with the goal of creating computational agents capable of demonstrating rational behaviour. Over the years, RL has also experienced significant advancements from its integration with diverse disciplines, including statistics, which played a crucial role to understand and model the stochastic nature of the environment. Additionally, mathematical optimization techniques, such as Dynamic Programming, have contributed to further enrich the capabilities of RL algorithms, enabling them to achieve notable successes in real-world and complex tasks.

In this chapter, we summarize the basic theoretical foundations of RL, paramount for understanding the techniques employed in this work. We start from the *Markov Decision Process* formulation that permits us to model the agent-environment interaction, after that we introduce the Dynamic Programming principles with their related approximate solution methods, which are necessary to be able to work in real-world environments. Considering the need to control the risk of the choices made by agents to minimize any negative consequences, we present the risk-aversion approaches in the RL context. Fi-

nally, in order to use a control frequency that provides an effective signal for the learning process we introduce the concept of action persistence.

2.1. Markov Decision Process

Markov Decision Process (MDP, [9]) is the classic mathematical framework for modelling sequential decision-making problems, where the actions influence not only the immediate reward but also the future ones through the subsequent states. The decision maker, usually denoted as *agent*, interacts with an external *environment*, by selecting an action and receiving an observation. As a direct consequence of the action, the environment provides the agent with a reward that reflects the goodness of the action taken; this reward allows the agent to learn and understand how to achieve its prefixed goal.
In a more specific way, the interaction between the agent and the environment happens through a sequence of discrete timesteps $t = 0, 1, 2, 3...$, where at each time step the agent receives an observation that represents the environment's state $S_t \in \mathcal{S}$. Based on such observation, the agent selects an action $A_t \in \mathcal{A}$, receiving a consequent reward $R_t \in \mathcal{R}$. In the meanwhile, the environment evolves to a new state S_{t+1} based on the previous state and the performed action. This iterative procedure generates a *trajectory* τ that starts and proceeds in this way:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_3, R_3, \dots$$

The formal definition of an MDP is provided below [9]:

Definition 2.1.1. *(Markov Decision Process)*

A Markov Decision Process is a stochastic dynamical process defined by the tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle$:

- $\mathcal{S}$ is a continuous or finite set of states;
- $\mathcal{A}$ is a continuous or finite set of actions;
- P is a state-transition probability function, i.e. $P(s'|s, a)$ represents the probability of a transition to a state $s' \in \mathcal{S}$ starting from the pair $(s, a) \in \mathcal{S} \times \mathcal{A}$;
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function that, for each state-action pair, returns a scalar feedback;
- γ is a discount factor: $\gamma \in [0, 1]$;
- μ is the distribution of the initial state.

In an MDP, the state-transition probability completely characterizes the environment's dynamics, in fact, the probability of each random value S_t and R_t is dependent only on the previous state and action. In this way, the state must include all the information from the history, and when it happens the state satisfies the *Markov property*.

Definition 2.1.2. *Markovian Process*

A stochastic process is said to be Markovian if and only if:

$$P(S_t|A_{t-1}, S_{t-1}, A_{t-2}, S_{t-2}, ...) = P(S_t|A_{t-1}, S_{t-1}).$$

Thanks to this property, once the state is known, the history of the past states may be neglected.

2.1.1. Rewards and Returns

As previously mentioned, the agent's objective is to maximize the cumulative reward over a sequence of timesteps. This concept can be formally expressed through the reward hypothesis [8]:

> All of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

In general, we want to maximize the *expected return*, where the return, denoted by G_t is the sum of the reward between the timesteps:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T, \tag{2.1}$$

where T is the final time step. When the interaction between the agent and the environment can be divided into subsequences with a finite *horizon* T, the MDP task is called an *episodic task*. Instead, when it is not possible to define a final state, the MDP task is denoted as a continuous task and in this case the final time step is equal to $T = \infty$.

Using the previously introduced discount factor γ we define the discounted return as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

The discount factor γ has a special meaning: in fact, it represents how much we would weigh the immediate reward compared to future rewards. If γ is close to 0 we will have

a myopic evaluation, otherwise, if γ is close to 1 we will have a far-sighted evaluation.

2.1.2. Policy and Value Function

A *policy* decides for each given state which action the agent must select, therefore defining the agent's behaviour according to the observation received from the environment. More formally, a *stochastic* policy π assigns a probability distribution over the action space for each specific state s, i.e. $\pi(a|s)$ is the probability that the action a is selected in the specific state s. Due to the dependence of the policy upon the only current state (neglecting the past trajectory), this type of policy is denoted as *Markovian*.
Furthermore, a policy is said to be *stationary* if it is independent of the time step t, otherwise, the policy is defined as *non-stationary*. However, it is important to underline that any non-stationary policy can be made theoretically stationary by including the time step as a feature in each state.

Given a policy π, it is possible to define the utility of the agent to be in a given state, this utility is defined as the expected return starting from state s and following the policy π.

Definition 2.1.3. *State-Value Function*

The state–value function $V^\pi(s)$ of a MDP is the expected return starting from state s, and then following the policy π:

$$V^\pi(s) = E_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right]. \tag{2.3}$$

Similarly, we define the utility of an agent given a state s, to select an action a and then follow policy π.

Definition 2.1.4. *Action-Value Function*

The action-value function $Q^\pi(s, a)$ of a MDP is the expected return starting from state s, taking action a and then following the policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]. \tag{2.4}$$

It is possible to exploit the recursive properties of the state-value function to decompose its formulation into two terms, one representing the immediate reward and a second

representing the state-value function of the next state:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma V^{\pi}(s_{t+1})|s_t = s] \\ &= \sum_{a \in A} \pi(a,s) \Bigg(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s') \Bigg). \end{aligned} \tag{2.5}$$

Similarly, the action-value function can be decomposed in the same way:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1})|s_t = s, a_t = a] \\ &= R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s'). \end{aligned} \tag{2.6}$$

These recursive relationships are known as *Bellman equations* and play a crucial role in the resolution method of Reinforcement Learning.

The Bellman equations can also be represented in matrix form, through which it is possible to notice its solution in closed form:

$$V^{\pi} = R^{\pi} + \gamma P^{\pi} V^{\pi} = (I - \gamma P^{\pi})^{-1} R^{\pi} \tag{2.7}$$

2.1.3. Optimal Policies and Optimal Value Functions

Searching the solution of an RL task means finding the *optimal policy* that guarantees the maximum possible expected return. For a finite MDP, we can define an optimal policy exploiting the partial ordering over the policies defined by the Value Function:

$$\pi \geqslant \pi' \text{ if } V^{\pi}(s) \geqslant V^{\pi'}(s), \forall s \in \mathcal{S}.$$

For any MDP exists an optimal policy $\pi^{\star}$ that is better than or equal to all other policies π. It is possible that there exist multiple optimal policies, anyway, all these policies achieve the same optimal value function $V^{\pi^{\star}}(s) = V^{\star}(s)$.

Definition 2.1.5. *Optimal Value Function*

$$V^{\star}(s) = \max_{\pi} V_{\pi}(s) \quad \forall s \in \mathcal{S}. \tag{2.8}$$

Definition 2.1.6. *Optimal Action Value Function*

$$Q^{\star}(s,a) = \max_{\pi} Q_{\pi}(s,a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{2.9}$$

It can be proven that there always exists a deterministic optimal policy and this optimal policy can be found maximizing over $Q^\star(s,a)$ w.r.t. the action space.

Since $V^\star(s)$ is the value function for a policy, it satisfies the condition given by the Bellman equation (2.5). However, this consistency condition can be written in a special form without reference to any specific policy using the *Bellman optimality equation.*

Definition 2.1.7. *Bellman Optimality equation*

$$V^\star(s) = \max_{a\in\mathcal{A}} Q_{\pi^\star}(s,a) = \max_{a\in\mathcal{A}} \left[R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) V^\star(s') \right]. \tag{2.10}$$

$$Q^\star(s,a) = R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \left[\max_{a'} Q^\star(s',a') \right]. \tag{2.11}$$

For finite MPDs, the Bellman optimality equation for $V^\star(s)$ has a unique solution, therefore solving it allows us to obtain the optimal policy. However, this equation is actually a non-linear system of equations and its resolution is usually not feasible.
The reasons behind this difficulty mainly lie in three assumptions that are not usually encountered in real problems: the dynamics of the environment is accurately known, the computational resources are sufficient for complete the resolution of the system and the states satisfy the Markovian property.

As we shall see in the next sections, the approaches usually used to overcome these difficulties are those relating to approximate methods.

2.2. Dynamic Programming

Dynamic Programming (DP) is a general solution approach for solving many classes of problems. Its approach consists of decomposing the problem into subproblems and combining their subsolution. DP can be used when the problem satisfies the principle of optimality substructure (when the optimal solution can be retrieved by the optimal solutions of its subproblems) and the overlapping subproblems (to reuse and cache of the optimal subsolutions).

MDP problems satisfy both the previous properties, but unfortunately, the classic DP approaches are of limited use in RL. This limitation is due to the fact that in real problems the assumption of perfect knowledge of the dynamics required for the application of DP techniques is not satisfied. Furthermore, the application of these techniques usually requires too much computational effort, hardly acceptable in real contexts.

Despite having these restrictions, the DP approaches are fundamental for RL because they lay the foundations on which the approximate methods are based, which usually require an imperfect knowledge of the dynamics of the problems and a minor computational effort.
In any case, DP can be used in RL for planning purposes, in fact, we can use this technique for prediction tasks or for control tasks. In the prediction task we want to predict the value function of a specific state, while in the control task we want to find the optimal policy to find the optimal solutions.

2.2.1. Policy Evaluation

Policy evaluation is used for prediction tasks, it essentially computes the value function of a state s given a policy π. If the environment's dynamics is completely known, we can recall the Bellman equation in (2.7) and compute the value function of a policy π by solving the linear system of the $|\mathcal{S}|$ equations. However, for previously mentioned reasons, iterative methods are usually adopted.
Precisely, considering a sequence of approximate value functions $V_0, V_1, V_2, ...$ where V_0 is arbitrarily chosen, the subsequent value functions can be approximated by applying the Bellman equation as an update rule:

$$\begin{aligned} V_{k+1} &= E_\pi[r_{t+1} + \gamma V_k(s_{t+1})|s_t = s] \\ &= \sum_{a \in A} \pi(a, s) \Bigg(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V_k(s') \Bigg). \end{aligned} \tag{2.12}$$

This update rule can be employed to update the old value function V_k with the new value function V_{k+1}. By recalling the Bellman Equation that ensures the equality of each optimal value function, we can guarantee that $V_k = V_\pi$ is a fixed point for the previous equation. Furthermore, it can can be proven that V_k converges to V_π with $k \to \infty$. This procedure is called *iterative policy evaluation.*

2.2.2. Policy Improvement

Considering a deterministic policy π, we are interested to know if for a given state s we should change the policy to choose a better action $a \neq \pi(s)$.
We know how good it is the policy π through V_π, and one way to improve this measure

is to act greedily by choosing an action that maximizes our action value function:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^{\pi}(s, a). \tag{2.13}$$

Theorem 2.1. *Policy Improvement.*
Let π and π' be any pair of deterministic policies such that:

$$Q_{\pi}(s, \pi'(s)) \geq V_{\pi}(s),\ \forall s \in \mathcal{S},$$

then the policy π' must be good as, or better than π:

$$V_{\pi'}(s) \geq V_{\pi}(s),\ \forall s \in \mathcal{S}.$$

Now suppose that the new greedy policy π' is as good but not better than the old policy π, then we have that $V_{\pi'} = V_{\pi}$ and from (2.11) we know that for all $s \in \mathcal{S}$:

$$\begin{aligned} V_{\pi}(s) &= \max_{a \in \mathcal{A}} \mathbb{E}\Big[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a\Big] \\ &= \max_{a \in \mathcal{A}} \Big[R(s, a) + \gamma \sum_{s'} P(s'|s, a) V_{\pi}(s')\Big]. \end{aligned} \tag{2.14}$$

But this equation is the same as the Bellman Optimality equation and so $V_{\pi'} = V^{\star}$, making both π and π' optimal policies. Therefore, this procedure allows us to retrieve a better policy except when the initial policy is already optimal.

2.2.3. Policy Iteration and Value Iteration

Once we have obtained a new best policy π' and the corresponding value function $V_{\pi'}$, it is possible to apply again these steps to obtain a new best policy π''. Consequently, we can repeatedly alternate between policy evaluations and policy improvements, and obtain a sequence of policies and value functions that consistently improve over time:

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} V^{\star},$$

where $\xrightarrow{E}$ denotes a policy evaluation step and $\xrightarrow{I}$ denotes a policy improvement.
This procedure is called *Policy Iteration* and each step ensures an improvement until convergence to the optimal policy $\pi^{\star}$. One drawback of this approach is the computational effort required, this is mainly due to the iterative policy evaluation step which can be expensive for multiple passes throughout the state set.

In any case, the policy evaluation step of policy iteration can be truncated without losing

the convergence guarantee. One of the most important cases is when the policy evaluation is stopped after just one sweep, this algorithm is called *Value Iteration* and is the iterative application of Bellman Optimality as update Rule:

$$\begin{aligned} V_{k+1}(s) &= \max_{a \in \mathcal{A}} \mathbb{E}\Big[r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a\Big] \\ &= \max_{a \in \mathcal{A}} \Big[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s')\Big], \quad \forall s \in \mathcal{S}. \end{aligned} \tag{2.15}$$

Like policy evaluation, value iteration can require an infinite number of iterations to converge exactly to $V^\star$, so usually the procedure is stopped when the value function changes by only a small amount in a single sweep.

2.2.4. Generalized Policy Iteration

The term *Generalized Policy Iteration* refers to the general idea of letting the steps of policy evaluation and policy improvement interact, regardless of the specific level of detail of both processes.
If both the evaluation and improvements process stabilizes, it means that the value function and the policy produced must be both optimal. In another way, we can see that both processes compete and cooperate to find a single joint optimal solution. In fact, the improvement of policy generates greedy policies with respect to the value function, making the old value function incorrect with respect to the new one. In the same way, the subsequent policy evaluation makes the value function consistent with the policy, with the consequence of making the policy not aligned with the new value function.

2.3. Optimal Solution with Reinforcement Learning

In real-world problems, the full knowledge of the MDP is usually not available. In addition, the state spaces and the action spaces are usually very large or not discrete, making the application of the previous exact methods limited.

All modern RL techniques deal with the problems of learning how to predict the value functions, and how to control the MDPs in the best way as possible.
We can resume the main RL techniques through the following classification:

- *Model-Free* and *Model-Based*. Model-Based approaches aim to estimate the model or the reward in order to solve the MDP respectively, by approximation or by exact solutions. Model-free techniques work without the necessity of estimating the exact

model with the aim of learning the value function of the policy.

- *On-policy* and *Off-policy*. In the Off-policy case, the learning process uses a behavioural policy to explore and collect additional information for learning a target policy, while in the On-policy case, the target and behavioural policies coincide.
- *Online* and *Offline*. Online approaches consider a scenario where the agent can learn through direct real-time interaction with the environment. In offline approaches, all the samples are available from the beginning of the learning process. It is possible to have an intermediate approach, called *semi-batch* that permits to have limited interaction with the environment to collect a new batch of samples.
- *Tabular* and *Function Approximation*. When the MDP is finite the setting is called tabular because we can imagine value functions and policies as a table. Otherwise, when the state-action space is not finite or too large, we have to rely on function approximation approaches to estimate the value function or the policy using a generalization of the state-action samples encountered.
- *Value-Based* and *Policy-Based* and *Actor-Critic*. Value-Based methods try to learn the optimal value function as done in value iteration. Policy-Based provide a direct parametrization of the policy, and try to learn the optimal policy following a gradient in the parameter space. The combination of the two previous approaches is called Actor-Critic: this approach tries to combine the best characteristic of the two methods.

2.3.1. Monte Carlo Learning

Monte Carlo (MC) is a model-free method. In fact, it does not require the full knowledge of the environment but directly learns from episodes of experience, with the only caveat that the episodes must be complete and finite.
MC adopts the simple idea to approximate the value function from the average return computed by the past experience sample. This average is obtained incrementally and is updated through the new experience samples collected.

As usual in the prediction task, the goal is to estimate $V_\pi(s)$. MC executes this task through the collection of trajectories collected by the policy π. For instance, given N trajectories with length T collected by the policy π, the empirical mean used for estimating

the value function is calculated by:

$$\widehat{V}_\pi = \sum_{i=0}^{N-1} G_i = \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \gamma^t R_{i,t+1}.$$

Thanks to the *law of large numbers* and since this estimate is the average of i.i.d samples, the estimator has mean $\mathbb{E}[\widehat{V}_\pi] = V_\pi$ and variance $\mathbb{V}ar[\widehat{V}_\pi] = \frac{\mathbb{V}ar[V_\pi]}{N}$.
Furthermore, if our estimator makes the average of returns only in the first time where each state s is visited, we have an *unbiased* and *consistent* estimation of the value function called *first-visit* estimator. Instead, if our estimator makes the average of returns every time where s is visited, we have an *biased* estimator but still *consistent*, called *every-visit* estimator.

2.3.2. Time Difference Learning

Temporal Difference learning is a combination of Monte Carlo Methods and dynamic programming ideas. TD methods are still model-free and can learn directly from episodes of experience, but like dynamic programming methods, they update the new estimates using the previously computed one. This approach is also known as *bootstrapping* and allows to use incomplete episodes without waiting for their end.

The prediction task for a policy π of TD methods follows this general rule:

$$V_{k+1}(s_t) = V_k(s_t) + \alpha[G_t - V_k(s_t)], \tag{2.16}$$

where α is the *learning rate*, G_t is some estimation of the return starting from state s, t is the current timestep and k is the current iteration of the learning process.
If we set G_t equal to the discounted return, we reduce the rule to a Monte Carlo every-visit update. Instead, using $G_t = R_{t+1} + \gamma V(s_{t+1})$, we obtain this transformed update rule:

$$V_{k+1}(s_t) = V_k(s_t) + \alpha\underbrace{[\underbrace{R_{t+1} + \gamma V_k(s_{t+1})}_{\text{TD target}} - V_k(s_t)]}_{\text{TD error}}, \tag{2.17}$$

where the quantity *TD error* measures the difference between the estimate of the current state s and the new estimate of the subsequent state. The previous update rule is called *TD(0)*, or *one step TD* because it considers only the one-time step for updating the current value function estimate.

TD methods compared to MC methods exhibits a lower variance and a higher bias. To

manage the trade-off between these two quantities it is possible to change the TD target to look a higher amount (say n) of future steps, thus obtaining the TD(n) target:

$$G_t^n = \sum_{i=0}^{n-1} \gamma^i R_{t+i+1} + \gamma^n V_k(s_{t+n}).$$

In another approach is possible to exponentially average the n-step TD to calculate the λ-*return* and obtain the TD(λ) target:

$$G_t^\lambda = (1-\lambda) \sum_{n=0}^{\infty} \lambda^{n-1} G_t^n,$$

using λ, or n in the simply case of TD(n), for regulate the trade-off between the n-step updates. When λ=0 we recover the TD(0) case, while with λ=1 we recover the MC case. TD(λ) is an approach for moving between the TD and MC cases, providing a *Forward–view* for computing the n-step updates and a *Backward–view* through the *Eligibility traces* concept, used for creating a temporal credit assignment on the past events.

2.3.3. Control Approaches

Control approaches have the aim of finding the optimal policy or the optimal value function. The RL methods used for this purpose face the challenge of balancing the need to try new actions, potentially worse than the ones suggested by the policy, to gather more knowledge about the environment and select more promising actions that yield the highest final return. This challenge is known as *Exploration-Exploitation dilemma* and arises from the trade-off between exploring new actions and exploiting the knowledge gained from past experience. In order to balance these two different needs the previously mentioned on-policy and off-policy approaches are used:

- on-policy approach forces the policy used by the agent to be exploratory making it select sub-optimal actions in some cases.
- off-policy approach uses two policies: the agent use samples gathered from an explorative behavioural policy to learn a potentially optimal target policy.

In this subsection, we will focus only on two TD control methods: *SARSA* and *Q-Learning*.

SARSA

SARSA is the on-policy TD algorithm proposed by Rummery and Niranjan [10]. This method follows the pattern of GPI where the TD prediction is used for the policy evalu-

ation part.
This algorithm, instead of using the same approach as the TD update rule (Equation 2.17), learns the action-value function $Q(s, a)$ for the current behavioural policy π in this way:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]. \tag{2.18}$$

This update is performed taking into account the transitions from one state-action pair to another state-action pair. As can be seen, the equation use the tuple $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ from which the name of the method derives.
The convergence of Sarsa depends on the exploration of the behavioural policy. Common alternatives for not having a purely deterministic policy are the $\epsilon - greedy$ policy, which randomly selects an action with a probability ϵ and chooses the greedy one with a probability of $1 - \epsilon$, or the softmax policy where the probability of choosing an action is determined by a normalized exponential function.

Theorem 2.2. *Sarsa Convergence*
Sarsa converges to the optimal action-value function $Q^\star(s, a)$ under the following conditions:

- *All state-action pairs are explored infinitely many times and the policy converges to the limit to the greedy policy (Greedy in the Limit of Infinite Exploration condition).*
- *Robbins–Monro condition of the learning rate ($\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$).*

Q-Learning

Q-learning [11] is an off-policy TD algorithm where the learned action-value function $Q(s, a)$, obtained by the interaction with a behavioural policy π_b, directly approximates the optimal action-value function $Q^\star$. This process creates the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]. \tag{2.19}$$

Q-learning convergence is ensured by the same conditions as Sarsa.

2.4. Approximate Solution Methods

The previously presented RL methods, usually called tabular methods, work well only in the framework of finite MDPs framework, where the state and action spaces have a small and finite number of elements.

This assumption is frequently limiting for the majority of real-world RL tasks, where is common to interact with continuous or huge discrete state-action spaces. This large amount of elements means that most of the states encountered have probably never been encountered before.
Because of this, it is necessary to generalize the experience gathered from samples encountered in the past and use it on new states that are potentially similar to past ones.
This type of generalization is commonly referred to as *function approximation* and has already been fully examined in the literature [8]. These methods are defined in this way because they take samples from a desired function (for instance the previously explained state-value function and action-value function) and attempt to construct an approximation of the complete function.
Function approximations are an instance of *supervised learning* Machine Learning methods, with additional issues such as no-stationarity, bootstrapping and delayed targets.

2.4.1. Value-Function approximation

The Value Function approximation is usually represented by a parameterized function with weight vector $w \in \mathbb{R}^d$. For instance, given a policy π, we will write $\widehat{v}(s, w) \approx v_\pi(s)$ to represent the approximate value function of state s with weight vector w. Usually, the number of weights used to approximate the value function is much lower than the number of states ($d \ll |\mathcal{S}|$), with the consequence that the modification of a single weight can potentially affect the prediction of the value function of each state.
In principle, the approximation can be made by any parametric supervised learning function usually considered in ML. For example, we can represent the approximation by a linear regressor with weights w, with a multi-layer artificial neural network with the connection weights in the layers described by the weights w or by a decision tree where w represents the parameters for defining the split points and leaf values.
As in the classic supervised learning method, we need to specify a scalar objective for our optimization process and since we have a number of states greater than the number of weights, we need a state distribution ($\mu(s) \geq 0$ with $\sum_{s \in \mathcal{S}} \mu(s) = 1$) for denoting how much we care about the approximation error in each state s. Usually, this represents the time spent in a specific state s and under the on-policy training is called *on-policy distribution.*

A typical objective function is the *mean squared value error*, obtained by the mean squared error between the approximate value $\widehat{v}(s, w) \approx v_\pi(s)$ and the true value $v_\pi(s)$ weighted

by the state distribution $\mu(s)$:

$$\overline{\text{VE}}(w) = \sum_{s \in \mathcal{S}} \mu(s)[v_\pi(s) - \hat{v}(s, w)]^2. \tag{2.20}$$

An ideal goal is the minimization of $\overline{\text{VE}}$ to find a global optimum for which $\overline{\text{VE}}(w^*) \leq \overline{\text{VE}}(w)$ for all possible w. Nevertheless, this goal is achievable only when the objective is convex, as in the case of linear approximators; in other cases the best possible result is to find a local optimum.

2.4.2. Policy Approximation

Now we consider Policy Approximation approaches. These methods use a *parameterized policy* that could select an action from a given state without the use of any type of value function. Even if the value function is not used for the selection process, it can be used to train the model in order to have a better understanding of the effectiveness of each performed action.
The policy parameter vector can be defined by the notation $\theta \in \mathbb{R}^{d'}$, i.e., the probability of taking an action a in a state s is denoted as $\pi(a|s, \theta)$.

Now we consider a class of methods called *Policy Gradient methods*. These methods update the policy parameters using the gradient of some scalar and differentiable performance measure $J(\theta)$, with respect to the policy parameter θ. Usually, our goal is the maximization of the performance measure $J(\theta)$:

$$\max_{\theta \in \Theta} J(\theta).$$

A policy π can be parametrized in any way by making $\pi(a|s, \theta)$ differentiable with respect to its parameters θ. In practice, we only need that the policy must be not deterministic to ensure enough exploration in the learning process. The *Policy Gradient theorem* [12] provides an analytic expression for calculating the gradient of the performance measure with respect to the policy parameter:

Theorem 2.3. *Policy Gradient Theorem.*
Let π_θ be a stochastic policy differentiable in θ, the policy gradient can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\substack{s \sim d_{\pi_\theta} \\ a \sim \pi_\theta(\cdot, s)}} [\nabla_\theta \log \pi_\theta(a, s) Q_\pi(s, a)], \tag{2.21}$$

where $\nabla_\theta \log \pi_\theta(a, s)$ is commonly referred to as a *score-function* and the discounted occupancy, denoted by d_{π_θ}, defines the stationary distribution over the state-action pair induced by the policy π [12].

2.4.3. REINFORCE

REINFORCE algorithm [13] is a variation of the classic policy gradient method that replaces the expectation under the state distribution with the sample mean. Considering a set of N trajectories of length T sampled from the policy π_θ, and using the equation (2.21) we can derive the REINFORCE estimator:

$$\widehat{\nabla}_\theta^{RF} J(\theta) = \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) G_t, \tag{2.22}$$

where G_t is the usual return. Using this estimator we can use stochastic gradient ascent to update our policy parameters:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla}_\theta^{RF} J(\theta). \tag{2.23}$$

This update modifies the policy parameters in the direction that increase the probability of repeating actions that favour the highest return.

As we can see, REINFORCE uses a single Monte Carlo estimate G_t to calculate the derivative of the performance measure J_θ. This return estimation can result in high variance producing slow learning.
One approach for reducing the variance is the introduction of a baseline function $b : \mathcal{S} \to \mathbb{R}$ used to include an action value comparison:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\substack{s \sim d_{\pi_\theta} \\ a \sim \pi_\theta(\cdot, S)}} [\nabla_\theta \log \pi(a, s)(Q_\pi(s, a) - b(s))]. \tag{2.24}$$

The baseline can be any function as it remains constant with respect to the action a. Indeed, considering an episodic case we have:

$$\sum_a b(s) \nabla \pi_\theta(a, s) = b(s) \nabla \sum_a \pi_\theta(a, s) = b(s) \nabla 1 = 0.$$

A common choice for the baseline is the use of a value function approximation $\widehat{v}(s_t, w)$.

2.4.4. Policy Parametrization for Continuous Actions

Policy-based methods provide effective approaches for managing large action spaces, including continuous spaces with an unlimited number of actions. Rather than calculating learned probabilities for each action, these approaches focus on learning statistical measures regarding the probability distribution.
One possible way to express continuous action in the real domain is to use a normal distribution with the following probability density function:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{2.25}$$

where μ and σ are the mean and standard deviation of the normal distribution.
In order to produce a policy parameterization, the policy can be formulated using this normal probability density function, where μ and σ are determined by this policy parametric function:

$$\pi_\theta(a|s) = \frac{1}{\sigma_\theta(s)\sqrt{2\pi}} \exp\left(-\frac{(a-\mu_\theta(s))^2}{2\sigma_\theta(s)^2}\right). \tag{2.26}$$

Using a practical example, we can divide the policy's parameter vector into two distinct parts $\theta = [\theta_\mu, \theta_\sigma]^T$, each used to approximate the mean or the standard deviation. More specifically, the mean can be approximated by a linear function and the standard deviation by an exponential of a linear function:

$$\mu_\theta(s) = \theta_\mu^T x_\mu(s); \quad \sigma_\theta(s) = \exp(\theta_\sigma^T x_\sigma(s));$$

where $x_\mu(s)$ and $x_\sigma(s)$ are state feature vector of the state s.

2.5. Risk-Averse Approaches

Real-world environments present challenges regarding the types of strategies performed by the agents. In RL, the usual goal is the maximization of the expected return; however, in some contexts like finance, robotics and health care, it is also important to consider the risk of the chosen strategy to minimize any negative consequences.
Precisely, risk is defined as the *dispersion measures* of the expected return, or in other words, the variance of the return w.r.t. its expected value. The types of risk in reinforcement learning can be classified into three main categories [4, 14]:

- *Inherent risk* when is due to the inherent uncertainty of the environment.
- *Model risk* relates to a limited understanding of the environment, making it hard

to predict the outcomes of certain actions.

- *Action risk* which is related to the random choice of the action by the agent, typically related to exploration purposes.

Various approaches can be used to face the risk in RL: the *inherent risk* is typically handled by transforming the objective function, taking into account the probability of visiting states with negative consequences; the model risk instead can be reduced using *safe policy updates* in the learning process.

2.5.1. Risk-Averse Measures

Risk-averse objectives are a subject of investigation in various fields dealing with risky situations. In the realm of finance, these objectives are commonly referred to as *risk-measures*.

Definition 2.5.1. *Risk-Measure [15, 16]*

A general *Risk-measure* can be defined by the mapping:

$$\eta : \mathcal{G} \to \mathbb{R}, \tag{2.27}$$

where $\mathcal{G}$ is the set of all the possible return functions and can be intended as a cumulated cost or cumulated reward.

A common measure of risk is the variance of returns [17, 18]:

$$\sigma_\pi^2 = \mathbb{E}_{\substack{s_0 \sim \mu \\ a \sim \pi_\theta(\cdot, s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}} \left[\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) - J_\pi \right)^2 \right], \tag{2.28}$$

where J_π is the usually expected return given a policy π.

Another technique for addressing risk is to consider a trade-off between the expected return J_π and its variance σ_π. This approach is generally called *Mean-Variance* [18] and is commonly used in finance due to its simplicity [19]. More precisely, this approach includes a set of different risk-averse objectives:

- *Multi-objective optimization* between the maximization of the expected value objective and the minimization of the variance objective.
- *Constrained optimization problem* between the maximization/minimization of a single objective that represents the expected value or the variance subject to a precise

constraint.

- A *penalized Mean-Variance* objective:

$$\eta_{MV}^{\pi,\lambda} = J_\pi - \lambda\sigma_\pi^2, \tag{2.29}$$

where λ is a risk-aversion coefficient used to regulate the trade-off penalization.

- *Sharpe Ratio* maximization:

$$\eta_S^\pi = \frac{J_\pi}{\sigma_\pi}. \tag{2.30}$$

Risk measures can be used to control the risk of the agents in order to minimize the probability of undesirable situations. In order to do that, is important to ensure the rationality of the risk measure used. For this reason, we can check if a risk measure η is a Coherent Risk-Measure (CRM).

Definition 2.5.2. *Coherent Risk-Measure [16]*

Given G as the return random variable, a Risk-Measure η is said to be a *Coherent Risk-Measure* if it satisfies the following properties:

1. *Translation Invariance* or *Convexity*: $\forall G \in \mathcal{G}, \forall \alpha \in \mathbb{R} : \eta(G + \alpha) = \eta(G) + \alpha$.
2. *Subadditivity*: $\forall G_1 \in \mathcal{G}, \forall G_2 \in \mathcal{G} : \eta(G_1 + G_2) \leq \eta(G_1) + \eta(G_2)$.
3. *Positive homogeneity*: $\forall \lambda \geq 0, \forall G \in \mathcal{G} : \eta(\lambda G) = \lambda\eta(G)$.
4. *Monotonicity*: $\forall G_1 \in \mathcal{G}, \forall G_2 \in \mathcal{G} : G_1 \leq G_2 \implies \eta(G_2) \leq \eta(G_1)$.

Unfortunately, not all the risk-measures satisfy these properties. For instance, the penalized Mean-Variance does not satisfy the monotonicity property.

One of the most important risk-measures that satisfies all the previous properties is the *Conditional Value-at-Risk* (CVaR, [20]):

$$\rho_{CVaR}^\pi(G, \alpha) = \max_\rho \left\{ \rho - \frac{1}{\alpha}\mathbb{E}_\pi \left[G - \rho\right]^- \right\}, \tag{2.31}$$

where $\alpha \in (0, 1)$ defines the level of the risk-aversion, while the value of $\rho^\star$ that maximizes the previous equation is defined as Value-at-Risk (VaR) [1]. Rigorously the CVaR risk measure, which is also known as expected shortfall, extends the popular VaR considering not only the probability of extreme events identified by the $\alpha - quantile$ but also the

[1]The Value-at-Risk value represents the quantile of the distribution of returns.

severity of those events. In fact, the CVaR risk measure considers the expected values of the worst outcomes encountered in the tail of the distribution identified by the previous $\alpha - quantile$.

Previously, we have described only risk-measures based on the variance of the return. However, it is also possible to define risk measures based on reward: these types of risk measures are able to estimate the short-term risk in a better way by considering the variation of the one-step reward of the visited states [21].
Bisi et al. [21] introduced a new risk measure based on the variance of the reward w.r.t. the state occupancy distribution named *reward volatility*:

$$\begin{aligned} \nu_\pi^2 &= \mathbb{E}_{\substack{s\sim d_{\mu,\pi} \\ a\sim\pi_\theta(\cdot,s)}} \left[\left(R(s,a) - \overline{J}_\pi \right)^2 \right] \\ &= (1-\lambda)\, \mathbb{E}_{\substack{s_0\sim\mu \\ a\sim\pi_\theta(\cdot,s_t) \\ s_{t+1}\sim P(\cdot|s_t,a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t \left(R(s,a) - \overline{J}_\pi \right)^2 \right], \end{aligned} \tag{2.32}$$

where $\overline{J}_\pi$ is the normalized expected return defined as $\overline{J}_\pi = (1-\gamma)J_\pi$.

Furthermore, it is possible to derive a relationship between the reward volatility and the return variance:

$$\sigma_\pi^2 \leq \frac{\nu_\pi^2}{(1-\gamma)^2}. \tag{2.33}$$

This upper bound is important because it demonstrates that by minimizing the reward volatility, it is possible to reduce the return variance.

Following the mean-variance approach in (2.29), we can define a new risk-measure objective called *Mean-Volatility*:

$$\eta_\pi = \overline{J}_\pi - \lambda \nu_\pi^2. \tag{2.34}$$

This new risk measure allows to establish a trade-off between the expected return and the risk minimization.

2.5.2. Risk averse Optimization

Given a risk-averse objective, we need to find a way to optimize our model to learn a policy that includes risk-aware behaviour. For this purpose, we recall the policy gradient theorem (2.21) using, when possible, an adapted version of it for each risk measure.

Regarding the penalized Mean-Variance, it is possible to derive a policy gradient formulation by taking the gradient of the variance with respect to the expectation over the

trajectories [22]:

$$\nabla_\theta \mathbb{V}ar_\theta = \mathbb{E}_{\tau \sim p_\theta(\cdot)}[\nabla_\theta \log p_\theta(\tau)(G - J_\theta)^2]. \tag{2.35}$$

where τ is a trajectory that defines the action-state interaction, and $p_\theta(\tau)$ represents its overall probability. Regarding the CVaR risk measure, Tamar et al [23] derive a formulation using the policy gradient formula:

$$\nabla_\theta \eta_{\alpha,\theta}^{CVaR} = \mathbb{E}_{\tau \sim p_\theta(\cdot)}[\nabla_\theta \log p_\theta(\tau)(G - \rho)^-]. \tag{2.36}$$

An alternative approach to optimize the CVaR measure is to adopt a block-coordinate approach [24] using two nested loops. The external loop optimizes the classical CVaR formulation to find the optimal $\rho^\star$, while the internal loop uses a return transformation to optimize the policy with the classic policy gradient approach.

2.6. Action Persistence

Continuous time control problems are usually transformed into discrete time control problems. This discretization is performed by sampling (in the offline case) or by interacting with the environment (in the online case) with a certain control frequency. Intuitively, we could think that by increasing the frequency we will only have benefits: unfortunately, this is not true for most cases. In most real-world problems faced by the RL framework, the dynamic of the environment is not exactly known; for this reason excessively high frequency can produce negative effects making the problem difficult to solve. In fact, by increasing the control frequency, the advantage of choosing one specific action over another [25] could be reduced, making the signal returned by the action useless due to the noise in the dynamics of the environment. Instead, the use of low control frequencies allows better signal-to-noise feedback and minor sample complexity, useful for reducing the required computational effort. Moreover, exploration is also affected by the control frequency, as described in [26].

For all of the above reasons, it is crucial to find a trade-off between high and low in control frequencies. Metelli et al. [27] introduced the concept of *action persistence* in order to modify the control frequency by using the repetition of a chosen action for a fixed number of steps.

By exploiting the concept of action-persistence, the agent-environment interaction is modified in the following way: given a persistence parameter k, at the time-step $t = 0$ the agent selects an action according to its policy $a_0 \sim \pi(\cdot, s_0)$. Afterwards, this action is

persisted, or more simply repeated, for the subsequent $k-1$ steps; at the time-step $t = k$, the agent retrieves the new action by using the policy $a_k \sim \pi(\cdot, s_k)$, repeating this process until the end of the episode.
Therefore, finding the proper value of the persistence index k allows us to take advantage of a stronger and more valid signal, potentially allowing a better learning process. Clearly, we can resort to the classic agent-environment interaction by choosing $k = 1$.

3 | Actor-Critic Methods

Actor-Critic (AC) methods try to combine the best characteristics of policy-based and value-based methods using two components called *Actor* and *Critic*. The actor represents the component that represents the policy, i.e. it is involved in the selection of an action for each specific state. Instead, the critic represents the value approximator and it is denoted in this manner because he has the role of evaluating the choices made by the actor.
AC methods are mainly based on TD techniques; in fact, after the selection of the action by the actor component, the critic evaluates the new state to determine if the outcome is better or worse than expected by usually relying on the TD error.
This architecture has several notable advantages: firstly, it demands minimal computational effort for action selection since the actor is an approximated policy, which further allows the usage of continuous actions. Additionally, thanks to the decomposed architecture, it is possible to parametrize the two components in a different way.
In this chapter, we will review the key features of Actor-Critic methods and their state-of-the-art implementations. Afterwards, we will introduce the Fitted Natural Actor-Critic algorithm, an extended version of the Natural Actor-Critic architecture that enables general function approximation and data reuse within the Actor-Critic framework.

3.1. Actor-Critic Architecture

The Actor-Critic architecture allows the integration of value-approximation and policy-approximation methods in a unique algorithm. For this reason, the algorithms maintain the value function parameters w for the estimator $\hat{v}(s, w)$ and the policy parameters θ for the policy estimator $\hat{\pi}(a|s, \theta)$.
As seen in Section 2.4.3, it is possible to use a baseline $b(s)$ to reduce variance in the learning process; a common approach consists in the adoption of the critic's value function as the baseline, obtaining the following quantity:

$$\delta_t = R_{t+1} + \gamma \hat{v}(s_{t+1}, w) - \hat{v}(s_t, w). \tag{3.1}$$

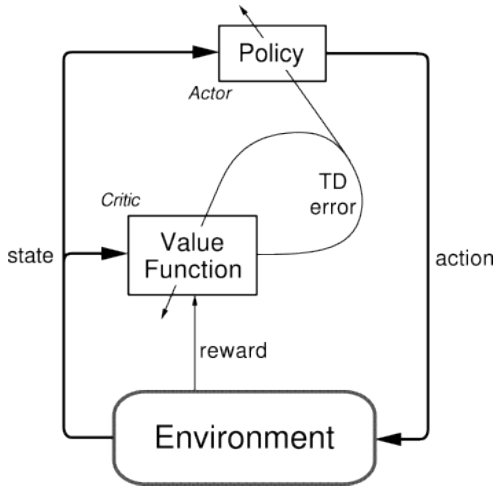


Figure 3.1: Actor-Critic Architecture [28]

As we can notice δ_t is the TD error of the estimated value function. Thanks to this measure we can update our critic and approximate the policy gradient using the policy gradient theorem.

Algorithm 3.1 Episodic One-Step Actor Critic

Input: a differentiable policy parametrization $\pi(a|s,\theta)$
Input: a differentiable value-function parametrization $\hat{v}(s,w)$
Parameter: learning rates $\alpha_w>0$ and $\alpha_\theta>0$
Initialize w and θ (for example to 0)
while Until stopping condition is reached **do**
    Initialize S as the first state of the episode
    $I = 0$
    while Until S is not terminal **do**
        a $\sim \pi(\cdot|S,\theta)$
        Take action a, observe S and R
        $\delta = R + \gamma\hat{v}(S',w) - \hat{v}(S,w)$ (if S' is terminal, then $\hat{v}(S,w) = 0$)
        $w = w + \alpha_w\delta\nabla\hat{v}(S,w)$
        $\theta = \theta + \alpha_\theta\delta\nabla \ln\pi(a|S,\theta)$
        $I = \gamma I$
        $S' = S$
    end while
end while

Using a bootstrap approach, the critic estimator $\hat{v}(s_t,w)$ is a biased estimate of $v_{\pi_\theta}(s_t)$. For this reason, the update of θ may not follow the gradient of the performance measure

$\nabla_\theta J(\theta)$ thus leading to an incorrect solution; therefore, it is very important to choose a correct estimation of the value function to avoid any bias in the gradient. In order to fix this issue, we can rely on the *Compatible Function Approximation* theorem:

Theorem 3.1. *Compatible Function Approximation [12]*
An action-value function $q_w(s,a)$ is compatible with the policy space π_θ if:

- *The following identity between gradients holds:*

$$\nabla_w q_w = \nabla_\theta \log \pi_\theta(a,s) \ \forall (s,a) \in \mathcal{S} \times \mathcal{A}$$

- *Value function parameters w minimize the mean square value error under the γ-discounted occupancy:*

$$w \in \arg\min_{w \in \mathbb{R}^d} \overline{VE}(w) \coloneqq \frac{1}{2}\mathbb{E}_{\substack{S\sim d_{\pi_\theta} \\ A\sim\pi_\theta(\cdot|S)}} [(q_{\pi_\theta}(S,A) - q_w(S,A))^2]$$

Then, the policy gradient computed replacing $q_{\pi_\theta}(S,A)$ with $q_w(S,A)$ is exact

Thanks to this theorem, we can use fulfil the assumptions of the policy gradient theorem and overcome the bias issue using the same critic estimator $\hat{v}(s_t, w)$.

3.2. Natural Actor Critic

The *Natural Actor Critic* (NAC, [29]) is a method in which the update of the actor policy is performed using a natural gradient approach [30], while the critic obtains both the natural policy gradient and additional parameters of a value function through linear regression.
It has been shown that policy gradients have strong convergence properties even when used in conjunction with value function approximation [12, 31], however, when applied to simple samples with few states, these methods are often inefficient due to the presence of large plateaus in the expected return calculation. These plateaus result in small gradients that do not lead to the optimal solution in an efficient and usable way.
The natural gradient effectively mitigates the impact of the stochastic policy through an empirical mean; therefore, it needs fewer data points to obtain a reliable gradient estimate compared to vanilla gradient approaches.

3.2.1. Natural Policy Gradient

The *Natural gradient*, unlike vanilla gradients that follow the steepest direction in parameter space, follow the steepest ascent direction in *Riemannian space* [32]:

$$\widetilde{\nabla}_\theta f(\theta) = G(\theta)^{-1}\nabla_\theta f(\theta), \tag{3.2}$$

precisely, $G(\theta)$ is a positive definite matrix named *metric tensor*. A common choice in machine learning and specifically in the natural gradient is the *Fisher Information Matrix* $F(\theta)$ as metric tensor, given by:

$$F(\theta) = \mathbb{E}_{\tau\sim p_\theta}[\nabla_\theta \log p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)^T], \tag{3.3}$$

where p_θ is a probability distribution conditioned by the value θ.

Using the policy gradient definition in (2.21) with the addition of a baseline $b_\pi(s)$, we can replace the term $q_\pi(s,a) - b_\pi(s)$ with a compatible function approximation:

$$q_w(s,a) = w^T(\nabla_\theta \log \pi_\theta(s,a)). \tag{3.4}$$

Thanks to this replacement and using (3.2) and (3.3), the natural policy gradient simplifies to:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\substack{s\sim d_{\pi_\theta}\\ a\sim\pi_\theta(\cdot,S)}}[\nabla_\theta \log \pi(a,s) q_w(s,a)] \\
&= \mathbb{E}_{\substack{s\sim d_{\pi_\theta}\\ a\sim\pi_\theta(\cdot,S)}}[\nabla_\theta \log \pi(a,s)\nabla_\theta \log \pi(a,s)^T w] \\
&= F(\theta)w \\
\widetilde{\nabla}_\theta J(\theta) &= F(\theta)^{-1}\nabla_\theta J(\theta) = w.
\end{aligned} \tag{3.5}$$

Consequently, we only need to estimate w instead of $G(\theta)$, thus leading to a policy improvement step of:

$$\theta_{t+1} = \theta_t + \alpha w. \tag{3.6}$$

One important property of the natural gradient is its invariance to the parametrization p_θ, in addition, it has been observed that opting for a more direct path to the optimal solution allows faster convergence, preventing premature convergence compared to vanilla gradients.

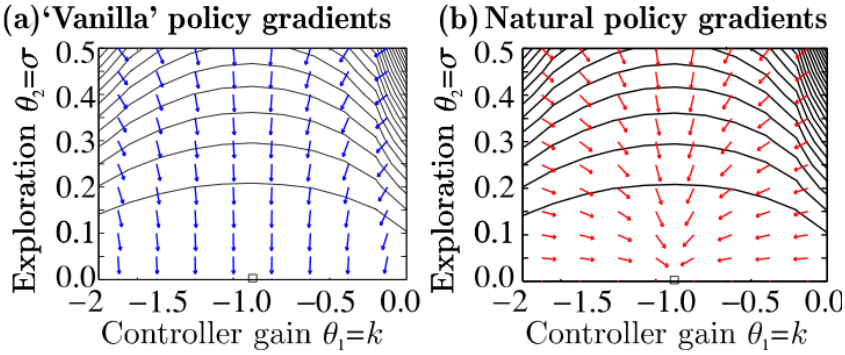


Figure 3.2: Comparision of expected return landscape for simple problem as 1d linear-quadratic regulation between vanilla and natural policy gradients [29].

3.2.2. Critic Estimation and Evaluation

The critic must evaluate the current policy π in order to provide the information for the actor improvement with respect to the change $\nabla\theta = \alpha w$ of policy parameters.
Considering the compatible function approximation $q_w(s, a)$ defined in (3.4), we can notice that it has zero mean w.r.t. the action distribution. Therefore, it represents an *advantage function* $A_w(s, a)$ and it quantifies the relative benefit of selecting one action over others in a particular state.
Anyway, we cannot learn this advantage function with a classic TD-like approach since we do not know the value function $v_w(s)$. One way to resolve this issue is to approximate the advantage $A_w(s, a)$ by exploiting the Bellman Equation in (2.6):

$$Q_\pi(s, a) = A_w(s, a) + V_\pi(s) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V_\pi(s'). \tag{3.7}$$

using this equation and parametrizing $V_\pi(s)$ as a linear regressor, we can use an LSTD(λ) [33] approach for determining w and consequently use it for updating the actor component.

3.3. Fitted Natural Actor-Critic

Fitted Natural Actor-Critic (FNAC, [2]) is an extension of the Natural actor-critic architecture, and it modifies the TD-based critic to allow the use of general value function approximations implementing a variant of Fitted Value-Iteration by relying on importance sampling.
In order to obtain an efficient estimation of the value function used to compute the policy gradient, FNAC combines the advantages of the usage of natural policy gradients with

the adoption of value function approximation. This approach results in a policy that can potentially handle continuous action spaces, making it possible to handle a broader class of real-world problems.
The algorithm makes use of a set $\mathcal{D}$ of samples obtained from the environment either offline or online way: each sample is composed of a tuple (s_t, a_t, r_t, y_t) where y_t is the next state according to the state-transition probability.

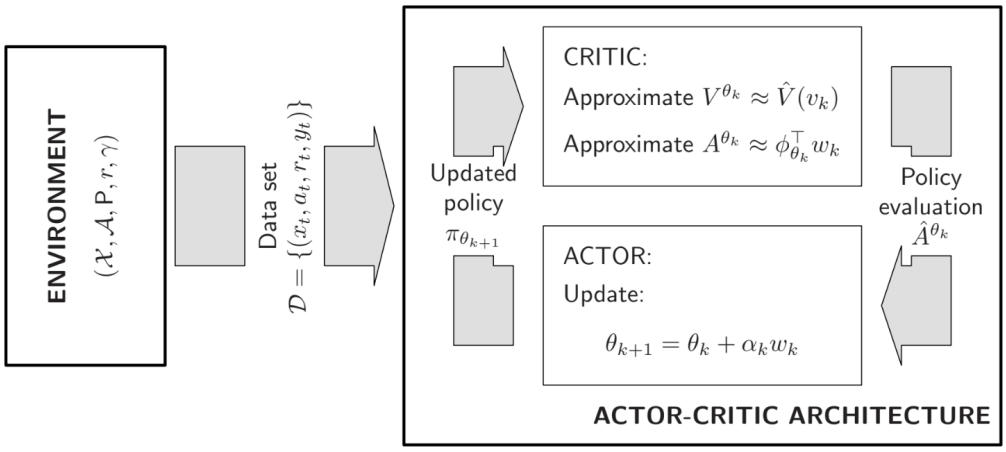


Figure 3.3: Scheme of FNAC architecture [2].

During each iteration of the algorithm, the samples are processed by the critic component using a generic value function approximator to compute an approximation $\hat{V}_\theta$ related to the policy π_θ. This approximation is then used to estimate an approximate form of the advantage function $\hat{A}_\theta$ by employing a linear function approximation with compatible basis functions. In the end, the actor employs this last estimation to update the current policy π_θ using a standard policy natural gradient update.
More specifically, we can provide an estimate $\widehat{V}_\theta$ of the value function by fitting any type of regressor model and by minimizing the Bellman Error [1] using the samples in $\mathcal{D}$. Furthermore, the corresponding advantage function can be approximated by tackling the following regression problem:

$$w^\star = \arg\min_w \sum_t \frac{1}{\widehat{\mu}(s_t)} (r_t + \gamma \widehat{V}_\theta(y_t) - \widehat{V}_\theta(s_t) - \phi^T(s_t, a_t) w)^2, \tag{3.8}$$

where parameter w denotes a linear coefficient vector. This vector corresponds to the orthogonal projection of the advantage function $\widehat{A}_\theta$ within the linear space defined by the

[1] The Bellman Error represents the difference between the estimation of the value function and the target value function in the Bellman Equation.

compatible basis functions $\phi(s_t, a_t)$:

$$\phi(s_t, a_t) = \frac{\partial \log(\pi_\theta)}{\partial \theta}(s, a). \tag{3.9}$$

This regression problem can be solved using an ordinary least square approach:

$$M = \sum_t \frac{1}{\widehat{\mu}(s_t)} \phi(s_t, a_t)\phi^T(s_t, a_t),$$

$$b = \sum_t \frac{\phi(s_t, a_t)}{\widehat{\mu}} (r_t + \gamma \widehat{V}_\theta(y_t) - \widehat{V}_\theta(s_t)).$$

To finally obtain $w^\star = M^{-1}b$, which is used to update the policy using the natural gradient formula in (3.6).

In addition, thanks to the use of *importance sampling*, it is possible to approximate the value function related to the policy π_θ also with samples in $\mathcal{D}$ collected with other policy $\pi_0 \neq \pi_\theta$ using a likelihood ratio $\frac{\pi_\theta(s,a)}{\pi_0(s,a)}$. This allows to reuse all data in every iteration of the algorithm, providing a crucial benefit in scenarios where data collection is expensive or time-consuming.

Algorithm 3.2 Episodic Fitted Natural Actor Critic

1: Input: a differentiable policy parametrization $\pi(a|s, \theta)$
2: Input: a value-function parametrization $\widehat{v}(s, \eta)$
3: Parameter: learning rates $\alpha_\theta > 0$
4: Initialize θ and η (for example to 0)
5: **while** Until stopping condition is reached **do**
6: Collect a batch of trajectories in $\mathcal{D}$ with the policy π_θ
7: Fit $\widehat{v}(s, \eta)$ with samples in $\mathcal{D}$ a generic function approximator
8: Estimate the advantage function through the OLS approach and retrieve w_i
9: $\theta_i = \theta_i + \alpha_\theta w_i$
10: **end while**

3.4. State of the Art Architectures

In this section, we briefly describe two state-of-the-art actor-critic approaches in reinforcement learning: Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO).
Both methods are based on the natural gradient approach and have made significant contributions to the field, offering improved stability and sample efficiency.

3.4.1. TRPO

The classic policy gradient approaches aim to find the optimal policy by optimizing the current parameters in a step-by-step way, using the learning rate to regulate the update. This process is usually problematic when the update step is not done in a flat region because it can bring the new solution too far from the steepest point, with the risk of a *catastrophic learning*.
Trust Region Policy Optimization (TRPO, [34]) tries to face this problem by considering the type of the solution space and using a *trust region* in which the update step can be performed in a safer way.
Specifically, TRPO tries to find an optimal solution defining a trust region with the following constrained optimization problem:

$$\begin{aligned} \max_{\theta} \quad & J_{\theta_{old}}(\theta) \\ \text{s.t.} \quad & D_{KL}(\theta_{old}, \theta) \leq \delta, \end{aligned} \tag{3.10}$$

where $D_{KL}(\theta_{old}, \theta)$ is the *Kullback–Leibler divergence* between the old policy parameters θ_{old} and the current policy parameters θ, while $\delta > 0$ is a threshold used to limit the distance between the old and the new policy.
We can rewrite the previously constrained objective by considering the performance improvement of the new policy π_θ over the old policy $\pi_{\theta_{old}}$:

$$J(\theta) - J(\theta_{old}) = \underbrace{\mathbb{E}_{\substack{s\sim d_{\pi_{\theta_{old}}} \\ a\sim\pi_\theta(\cdot|s)}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\theta_{old}}(s,a) \right]}_{\mathcal{L}_{\pi_{\theta_{old}}}(\pi_\theta)}, \tag{3.11}$$

in which we use *importance sampling* to use the advantage function calculated by using the old policy. This approach increases the sample efficiency but can bring a high variance in the learning process: for this reason, the approximation of the improvement is expressed by setting a lower bound with respect to the previous term $\mathcal{L}$ and the KL divergence between the two policies:

$$J(\theta) - J(\theta_{old}) \leq \mathcal{L}_{\pi_{\theta_{old}}}(\pi_\theta) + C D_{KL}^{max}(\pi_{old}, \pi), \tag{3.12}$$

where $C = \frac{2\epsilon\gamma}{(1-\gamma)^2}$ determines the size of the trust region and influences the trade-off between exploration and exploitation in the policy optimization, while the inner ϵ is a term that represents the maximum possible advantage between the two policies.
Finally, by employing the *Minorization-Maximization algorithm*, we enhance the earlier

lower bound and thereby we maximize our performance improvement.

In the practical implementation, the estimation of KL divergence involves an unreasonably high amount of computational effort and, for this reason, is typically approximated by selecting its mean. In addition, to transform the previous lower bound into a differentiable objective function that can be optimized through a policy gradient method, we use the gradient $g = \nabla_\theta \mathcal{L}(\theta)|_{\theta=\theta_{\text{old}}}$. Furthermore, $\mathcal{L}$ is approximated as the first-order term of its Taylor series expansion. Likewise, the mean of the KL-Divergence is approximated by the second-order term of the Taylor series expansion of the KL-divergence at θ_{old}:

$$\hat{D}_{KL}(\theta_{old}, \theta) = \frac{1}{2}(\theta - \theta_{old})F(\theta_{old})(\theta - \theta_{old})^T, \tag{3.13}$$

where $F(\theta_{old})$ is the *Fisher Information Matrix* that represent the Hessian of the KL-divergence.

Finally, by employing a constrained optimization version of this formulation and using the *Lagrangian duality* we obtain the policy gradient updates equation:

$$\theta = \theta_{old} + \sqrt{\frac{2\delta}{g^T F(\theta_{old}) g}} \underbrace{F(\theta_{old})^{-1} g}_{\text{Natural Gradient}} . \tag{3.14}$$

However, even the inverse of the *Fisher Information Matrix* is complicated to calculate in terms of computational resources: therefore, a common solution consists in resorting to an approximation using the *conjugate gradient* to resolve the linear problem $F(\theta_{old})x = g$.

3.4.2. PPO

Proximal Policy Optimization (PPO, [35]) extends the trust-region approach of TRPO simplifying it by using only a first-order optimization with a clipped surrogate objective for regulating the safeness of the updating step.
By defining the probability ratio $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$, PPO modifies the unconstrained $\mathcal{L}(\theta)$ objective of TRPO in this way:

$$\mathcal{L}^{CLIP}(\theta) = \mathbb{E}_{\substack{s\sim d_{\pi_{\theta_{old}}} \\ a\sim\pi_\theta(\cdot,s)}} \left[\min\Big(r(\theta) A_{\theta_{old}}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon) A_{\theta_{old}}(s,a) \Big) \right], \tag{3.15}$$

where ϵ is a hyperparameter for defining an interval $[1-\epsilon, 1+\epsilon]$ that is used for clipping the probability ratio. This new objective takes the minimum of the clipped and unclipped objective $\mathcal{L}(\theta)$ making the final objective $\mathcal{L}^{CLIP}(\theta)$ a lower bound of the original unclipped

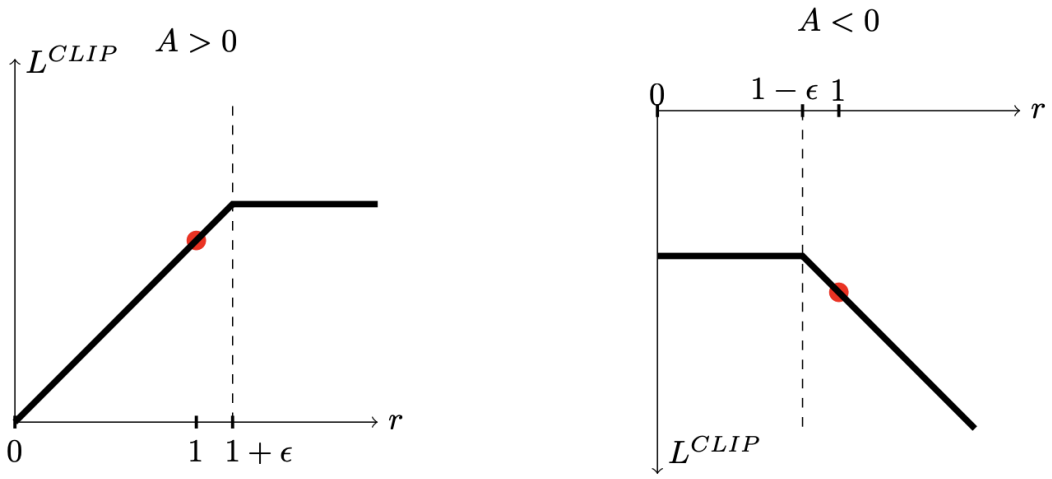


Figure 3.4: Plot that shows the clipping of $\mathcal{L}^{CLIP}$ as a function of the probability ratio r, respect to a negative and positive advantage function [35].

objective.

PPO is easier to implement compared to TRPO and exhibits improved sample efficiency by making more efficient use of collected experience. Furthermore, PPO is computationally less demanding as it does not necessitate any approximation of the natural gradient. For the previous reasons, PPO is currently one of the most employed algorithms in RL.

4 | Related Works

The use of Artificial Intelligence has undergone a massive diffusion thanks to its effectiveness in different types of fields. The increase in available computational power and the growing availability of data have been the determining factors for its development. From the beginning, the financial sector has shown a deep interest in using these techniques to create models that can help to maximize profits.

Initially, Machine Learning techniques in the financial sector were exploited to create models capable of making forecasts regarding the market trends. In recent decades, thanks also to the possibility of interacting with the market with ever higher frequency, there has been a growing interest in creating trading systems in order to create models capable of interacting autonomously with the markets. In this context, Reinforcement Learning has been able to put itself in the foreground, thanks to its ability to interact autonomously with the financial markets and to be able to make decisions dependent on the market situations learned through the use of historical data, thanks to which it is possible to create profitable strategies that can outperform even human traders.

At the same time, RL techniques have developed to be able to include approaches capable of limiting the risk of losses due to uncertain market situations.

In this chapter, we initially introduce a brief description of the Reinforcement Learning technique applied to trading markets, then we describe some techniques in which some aspects have been reused in this work. Finally, we will present some of the related works of our greatest interest that use policy-based techniques.

4.1. Reinforcement Learning for Trading

One of the initially significant successes in applying reinforcement learning to trading was the application of *Recurrent Reinforcement Learning* (RRL). Moody et al. [36] used an RRL model applied to the USD/GBP foreign exchange trained by maximizing a variant of the Sharpe Ratio, known as the *differential Sharpe ratio*, incorporating exponential moving average and standard deviation of returns as risk-adjusted measures. In addition,

the RRL agent was trained using a fixed-size amount, and it was limited to taking long, neutral, or short positions.
Gold [37], based on Moody's work, uses a recurrent neural network trained across different currency markets, including recent price history and the previous market position as features. Meanwhile, Dempster [38] employed a Q-Learning approach trained with a frequency of 15 minutes, incorporating only the buy-sell actions. Notably, the study revealed that this approach outperformed other techniques, such as *Genetic Algorithms* and *Markov Chains* in the same setting.

With the rise in popularity of *Deep Learning* and the advent of value-based approaches, such as Deep Q-Network [39], Carapuço, João et al. [40] employ it to FX trading using a three-layer Neural Network with *Rectified Linear Unit* activations function, testing it in a period of 8 years between 2010 to 2017 obtaining an average profit of 15%.
Huang [41] uses a DQN model trained with a feature set based on open, high, low, close prices and tick volume values (*OHLCV* data) sampled at 15-minute intervals of 12 different currency pairs. The performances were calculated by considering the effect of the spread, obtaining the best average annualized return of 26.3% with the value of 0.1% base point of spread.
Similarly, Sornayura [42] employs a DQN model in the EUR/USD and USD/JPY currencies pair using 15 years of OHLC data, obtaining a value of 43.88% of annualized return for the EUR/USD pair.

Finally, Briola [43] uses Proximal Policy Optimization (PPO) algorithm in the context of *Limit Order Book* (LOB). The PPO agent was implemented by Multilayer Perceptron trained considering three different scenarios and using the spread as transaction costs to make the experiment more realistic.

4.2. Forex Trading with Fitted Q-Iteration

Due to the high non-stationarity of the markets data, it is necessary to integrate risk-averse approaches that allow to limit the risks of the actions taken by the agents, moreover, the noise of the same data make difficult to obtain signals able to learn the agents in an efficient way.

Bisi et al. [44] employed a Fitted Q-Iteration in the forex market using a Multi-Objective formulation for keeping the risk of noisy profits under control and increment the risk-aversion behaviour of the algorithm. Lately, Riva et al.[3] extend the use of FQI integrating the concept of *persistence* for tuning the control frequency and handling in a better way the signal-to-noise ratio.

In the following section, we initially present the Fitted Q-Iteration algorithm, describing its use in the previously mentioned works.

4.2.1. Fitted Q-Iteration

Fitted Q-Iteration (FQI, [5, 6]) is a batch-mode reinforcement learning algorithm, it applies the idea of fitted value iteration reformulating the Q-function determination problem as a sequence of regression problems, making it possible to take full advantage of the function approximation of any regression algorithm.

In particular, FQI acquires an estimate of the Q-function by repeatedly expanding the optimization perspective in an infinite horizon optimal control problem with discounted rewards. The learning process is based on the four-value tuple $(s_t, a_t, s_{t+1}, r_{t+1})$ that represents the experience gathered by one agent through the interaction with the environment in a specific time t. Being a batch algorithm, FQI uses a pre-collected dataset $\mathcal{D}$ containing all the available four-value tuples.
At each iteration, the FQI algorithm uses the training set derived from the dataset $\mathcal{D}$ and by using the Q-function of the previous iteration, to create a new training set used to build a new approximation of the Q-function.

More precisely, in the first iteration, an approximation of the Q_1-function is produced by using a training set $TS = \{(i^k, o^k)|k = 1, 2, ..., |\mathcal{D}|\}$ with inputs i equal to the state-actions pairs (s_t, a_t) and outputs o equal to the related immediate rewards r_t of $\mathcal{D}$. In this training set the output o^k represents the immediate reward obtained by performing the action a^k in the state s^k.
In the subsequent iteration N, a new training set TS is obtained by updating the output values of the previous training set. This update is performed using the *value-iteration* update rule and the Q_{N-1} of the previous step:

$$o^k = r^k_{t+1} + \gamma \max_a Q_{N-1}(s^k_{t+1}, a), \tag{4.1}$$

instead, the approximation of the Q_N-function corresponds to an N-step optimization horizon, which is derived using the regression function approximator and the updated training set TS.

It is also necessary to define a stopping condition for the training iterations: usually, it is sufficient to define a priori the total number of iterations N, at which the algorithm must be stopped. Otherwise, it is possible to stop the algorithm when the difference between two successive Q-functions is lower than a certain predefined threshold.

Defining a stop condition, such as a predefined number of iterations (N), is important because in each iteration of FQI, the horizon being considered is extended by one step. As a result, any errors arising from the Q-function approximation are progressively propagated across the iterations, limiting the convergence of the Q-function.
This creates the need to balance two factors, first, there is a desire to increase the number of iterations in order to expand the horizon under consideration, on the other hand, it is crucial to limit the impact of approximation errors deriving from the regressor algorithm. Therefore, a trade-off must be carefully managed between the number of algorithm iterations, which expands the horizon, and the mitigation of approximation errors caused by the regressor algorithm. This need is usually handled by tuning the number of iterations N through model-selection techniques.

Algorithm 4.1 Fitted Q-iteration

1: Input: Batch of transitions $\mathcal{D} = \{(s_t^k, a_t^k, s_{t+1}^k, r_{t+1}^k) | k = 1, 2, 3..., |\mathcal{D}|\}$
2: Initialize $Q_N(s, a) = 0 \ \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
3: Initialize N = 0
4: Initialize $TS = \emptyset$
5: **while** Until stopping condition is reached **do**
6: N = N + 1
7: Build the training set $TS = \{(i^k, o^k) | k = 1, 2, ..., |\mathcal{D}|\}$ with Q_{N-1}:

$$i^k = (s_k, a_k), \qquad o^k = r_{t+1}^k + \gamma \max_a Q_{N-1}(s_{t+1}^k, a).$$

8: Use the regression algorithm to fit the new $Q_N(s, a)$
9: **end while**

4.2.2. Risk-Aversion formulation approach

Bisi et al. [44] generalize the FQI algorithm by incorporating a multi-objective optimization problem that simultaneously incorporates both profit maximization and risk minimization objectives, naming this generalization as Multi-Objective Fitted Q-Iteration (MOFQI).
In the Multi-Objective MDP (MOMDP,[45]) formulation, the quantities related to the problem, such as the return or the value functions, are represented in the form of vectors with dimension d equal to the number of objectives, with the consequence that it is not possible to define a direct maximization over a scalar Q-function like FQI. A common approach for resolving MOMDP is to reduce the problem to a single objective, using a

weight λ chosen in the $d-1$ dimensional simplex Λ. Although, finding a value of λ that ensures a proper trade-off between the objectives is challenging.
In contrast to the aforementioned approach, MOFQI tries to learn the Pareto frontier in order to find a Pareto-optimal policy [1], by incorporating the weight vector λ in the state and learning through the regressor model a Q-function $Q^\star(s, \lambda, a)$.
Regarding risk minimization, Bisi et al. considers the mean-volatility objective defined in (2.34), maximizing it by means of a transformation of the reward (as in [21]) :

$$R_\pi^\lambda(s,a) = R(s,a) - \lambda(R(s,a) - J_\pi)^2. \tag{4.2}$$

This transformation can be applied only on on-policy based methods due to the dependence of the new reward on the policy in the formula: hence, in order to work with off-policy methods like FQI, MOFQI uses an exponential reward transformation which represents a first-order approximation of $R_\pi^\lambda(s,a)$:

$$\widetilde{R}_\pi^\lambda(s,a) = \frac{1 - e^{-\lambda R(s,a)}}{\lambda}. \tag{4.3}$$

In order to train the model, the authors considered the pairs 2015-2016, 2016-2017, 2017-2018 using the years before and after the training set respectively as validation set and test set. In addition, Extra Trees were used for the regression algorithm, tuning the model as suggested in [46] by regulating only the min-split parameter.
To represent the state of the MDP problem, they considered data from the EUR/USD exchange rate by considering the last 60 open prices , the current time (denoted as the ratio of time left until the close of the trading day), and the portfolio allocation of the previous step $x_t \in \{-1, 0, 1\}$ expressed by the *Short*, *Stay* and *Long* position. The action consists in the position that the agent wants to keep in the next minute, so is defined by the same $a_t \in \{-1, 0, 1\}$ of the state portfolio allocation. Finally, the reward was modelled by the following formula:

$$r_t = a_t(p_{t+1} - p_t) - f|a_t - x_t|, \tag{4.4}$$

where the first term describes the profit obtained by the action performed and the changed price, the last term consists of a fee f that the agent must pay when he modifies her position, precisely they set as a fee equal to 2\$ on a total traded amount equal to 10^5\$. They achieved the best result on the model trained in the two years 2015-2016, obtaining in the test year 2017, through multiple runs, an average return of 11218 with a standard

[1] A policy is Pareto-optimal if it cannot improve on one objective without worsening another.

deviation of 1785, outperforming various baselines including the *Buy&Hold* and *Sell&Hold* strategies.

4.2.3. FQI with Persisted Actions

Riva et al. [3] exploit the use of the FQI using Extra Trees as the regression algorithm to learn the optimal Q-function. In addition, they incorporate the concept of persistence to examine the influence of control frequency in a realistic multi-currency trading scenario. To model the multi-currency trading scenario as MDP they used the last 60 exchange rate variations of the base currency, the corresponding time of the day and the current portfolio allocation as features, expressing the allocations with respect to the foreign currency. Finally. they used the same reward formulation in 4.4.
For the experimental phases, they tested the FQI model using market Forex data collected between the years 2017 to 2020, using $1e5$\$ as a fixed allocation amount and by employing an action persistence value equal to 1,5 and 10 minutes. Under this formulation, it is observed that models trained with a higher value of persistence outperform the models with a value of persistence equal to 1: this is attributed to the lower signal-to-noise ratio that arises from employing a control frequency that is too high, causing the actions performed by the agents to not receive a significant and useful reward due to the high stochasticity and non-stationarity of the trading environment considered.
With higher persistence values, in fact, as well as providing better feedback to the agent and having less computational effort due to the reduced horizon, the agent represented by the model has shown the ability to learn and exploit temporal patterns by expressing specific behaviour in some particular windows time in the market.

4.3. Trading with Policy-Based methods

In the financial trading context, Policy-based and the related Actor-Critic methods offer several advantages over value-based methods: for example, they can naturally handle continuous action spaces to determine the precise allocation for a specific trade; furthermore they can explicitly manage the uncertainty and the stochasticity of the markets by directly optimizing the policy, making the choice of the agent more robust respect to the changing market conditions.
In addition, the fact that the policy representation is directly learned from the action-state mapping makes policy-based methods more interpretable, enabling us to understand the characteristics of each individual policy. These reasons have led to the development of many policy-based and actor-critic algorithms in the finance field.

Bisi et al. [21] incorporated the mean-volatility objective (Equation 2.34) into the TRPO algorithm (described in Section 3.4.1) to guarantee risk-averse behaviour and achieve risk-averse updates of the policy parameters. This new algorithm, named Trust Region Volatility Optimization (TRVO), extends the usual advantage function to incorporate the mean volatility update ensuring the same theoretical results of TRPO:

$$\begin{aligned} A_\pi^\lambda(s,a) &= Q_\pi^\lambda(s,a) - V_\pi^\lambda(s,a) \\ &= \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[R(s,a) - \lambda(R(s,a)-J_\pi)^2 + \gamma V_\pi^\lambda(s') - V_\pi^\lambda(s)\right], \end{aligned} \tag{4.5}$$

where λ is used as a risk-aversion parameter for the mean-volatility objective.
TRVO was tested in a trading environment on the S&P 500 index, considering buy, sell and flat as possible actions and using the daily price from the 1980s until 2019, achieving performance domination with regard to the reward-volatility and expected return frontier compared to the classical TRPO.

Liu et al. [47] explore the use of the Deep Deterministic Policy Gradient DPPG (algorithm) to find the best strategy in a stock trading environment: DPPG is an actor-critic algorithm proposed in [48] that combines the better features of Deterministic Policy Gradient (DPG, [49]), which is an algorithm that learns deterministic policy by making the expected gradient of the action-value function, and of Deep Q-Networks [39] using neural networks as function approximator with the use of replay buffer for storing and re-using the experience collected by the agent through the interaction with the environment.
Liu et al. modelled the stock trading problem using the prices of the stocks, the number of holdings, and the remaining balance as the state. Furthermore, the available actions for each stock include selling, buying, and holding, which correspond to reducing, increasing, and maintaining the holdings, respectively. Finally, the reward was determined by considering the change in the portfolio value after each action was performed.
The dataset used was obtained from the Dow Jones index using the years between 2009 to the end of September 2018. The training data encompassed the years from 2009 to the end of 2014, the validation data covered the two years between 2015 and 2016, while the remaining available data was utilized to test the performance of the model. Employing the model in the test year, they obtained an annualized return of 22.24%, outperforming Dow Jones Industrial Average and min-variance portfolio allocation baselines.

Yang et al. [50] extended the previous work by employing an ensemble method that combined PPO, A2C (Advantage Actor-Critic), and DDPG algorithms. The authors employed an enriched state space to describe multiple stocks trading using the available balance, the close price, the shares owned by each stock and other special indexes used

for comparing the current and old stock prices.
The action space was represented in an interval $[-k, k]$, where k is the number of shares that the agent can buy or sell for each stock, then normalized to $[-1, 1]$ as the A2C and PPO work with a Gaussian distribution policy that requires normalization and symmetry. Finally they incorporated a transaction cost of 0.1% of the trade value and utilized the *financial turbulence index* [51] as a risk-aversion measure for capturing extreme asset price movements. In fact in cases where this index exceeded a predefined threshold, indicating severe market conditions, they halt the buy action by making the trading agent sell all shares.
Overall, the ensemble strategy involved training and selecting agents iteratively, validating their performance, and using the best-performing agent's predictions for subsequent trading periods.
By applying the extended dataset setting from the previous work and utilizing the ensemble process, they achieved a Sharpe ratio of 1.30%. These results surpassed the performance of each individual algorithm, demonstrating the effectiveness of the ensemble applied strategy.

5 | Problem Formulation

The first prerequisite for working with a problem in Reinforcement Learning, if it is possible, is its modelling in the MDP framework. In our work, we modelled the Forex exchange working environment to try to succeed in finding a profitable strategy that can model a real trading environment as efficiently as possible: for this reason, we extend the proposed model in [3] to support continuous actions that permit to allocate only a portion of the maximum available amount.

In this chapter, we explain the Forex Exchange Trading peculiarity and its transposition to the MDP problem.
Moreover, we explain in more detail the implementation of FNAC with the function approximation algorithms considered, including the integrations for modelling risk measures and the concept of persistence to reinforce the signal received by the decision agent.

5.1. Forex Trading Market

The *Foreign Exchange Market* is the biggest and most liquid financial market in the world; it is the place where currencies are exchanged between traders in *Over-the-Counter* (OTC) markets, so through a distributed computer network which allows the market to be open 24 hours a day on 5 days a week.
Currencies are traded in pairs: in our case, we will mainly consider the EUR/USD pair, where the first abbreviation represents the euro currency €, while the second represents the dollar $. For each pair we have a related price called *exchange rate*, for instance, the price 1.21723 is the amount of the *quote currency* in dollars $ that we can buy with one unit of *the base currency* of €.
Traders exploit fluctuations in the latter price to make profits by positioning in a specific way with respect to one of the two currencies involved. Precisely, traders can position themselves in three ways: if they believe that the price of the base currency will rise, they can assume a *Long* position by selling the quote currency. Conversely, if they sell the base currency thinking that the price will decrease, they are defined as in a *short* position. Finally, if they believe that making any changes in their position would not be profitable

due to the inherent exchange fees, they can simply assume a neutral position, denoted as *flat*.

In order to facilitate transactions between different traders while maintaining the decentralized nature of the market and ensuring liquidity, two distinct prices known as the *bid* and *ask* prices are employed. These prices serve as the basis for transactions and reflect the ongoing exchange of assets between market participants. The *bid* price denotes the price that a buyer is disposed to offer to purchase a currency, while the *ask* price is the price at which sellers are available to sell. From these two prices it is possible to define the *mid* price, which is the average of the two.
The difference between the bid and ask prices is called *spread* and can be viewed as a measure of the buying and selling interest for a pair of currencies. This quantity also represents one of the costs which a trader can encounter by making a transaction in the forex market: the reason for this relationship between the cost and the spread is due to brokers and market makers, who use the spread as a fee to supplant internal costs and risks for the services offered to the market.
The spread can also be understood as an indicator of the liquidity of the pair currency in the market: liquidity, in the financial context, is a measure that indicates how quickly an asset can be bought and sold in the market, ensuring that during periods when the market has a high liquidity presence, there is greater confidence that the exchange price reflects the value of the traded asset. As a result, a smaller spread indicates a higher level of liquidity in the market for the respective currency.

5.2. Forex Trading Model

5.2.1. Forex Trading Data

The original dataset contains observations collected from the forex market at a per-minute frequency: these observations represent the available data during market opening hours, which are from Monday to Friday, 24 hours per day, permitting to capture the key characteristics of the market.
Each sample of the dataset contains the mid-price, the spread, the date and the time in which the observation was collected in Central European Time (CET) hour.

In Table 5.1 there are some examples of the data recovered in the first days of the year 2020.

Date	Time	Mid	Spread
02-01-2020	00:00	1.1215	0.00007
02-01-2020	00:01	1.12171	0.00001
02-01-2020	00:02	1.12187	0.000035
02-01-2020	00:03	1.12197	0.000045

Table 5.1: Sample on 2021 market observation in the EUR/USD currencies pair.

Year	Avg Mid	Std Mid	Avg Spread	Std Spread
2018	1.181242	0.036870	0.000026	0.000031
2019	1.119554	0.013503	0.000018	0.000017
2020	1.141464	0.044322	0.000020	0.000023
2021	1.182661	0.028206	0.000019	0.000022
2022	1.057071	0.052377	0.000025	0.000027

Table 5.2: Summary statistics of the years between 2018 and 2022.

5.2.2. Forex Trading Exploration Analysis

In this section, we present a brief exploratory analysis of the data used in our work; precisely we focus our attention on the EUR/USD currency pairs in the years between 2018 and 2022. The data used were retrieved through the website HistData.com [52], with a multiplicative factor on the spreads to retrieve more realistic costs.

In Table 5.2 we provide the key statistics for each year regarding the mid-price and spread.

Furthermore, we show in Figure 5.1 the trend of the mid-price and spread throughout the years: we can observe, particularly in the spread figure, the presence of macroeconomic events that significantly impact the market. For example, on March 2020, the COVID-19 pandemic began to spread around the world causing macroeconomic disruption, resulting in a marked rise in the spread trend. Additionally, the frequent spikes visible in the plots of the spread are due to the weekend closure, which creates less liquidity in the market.

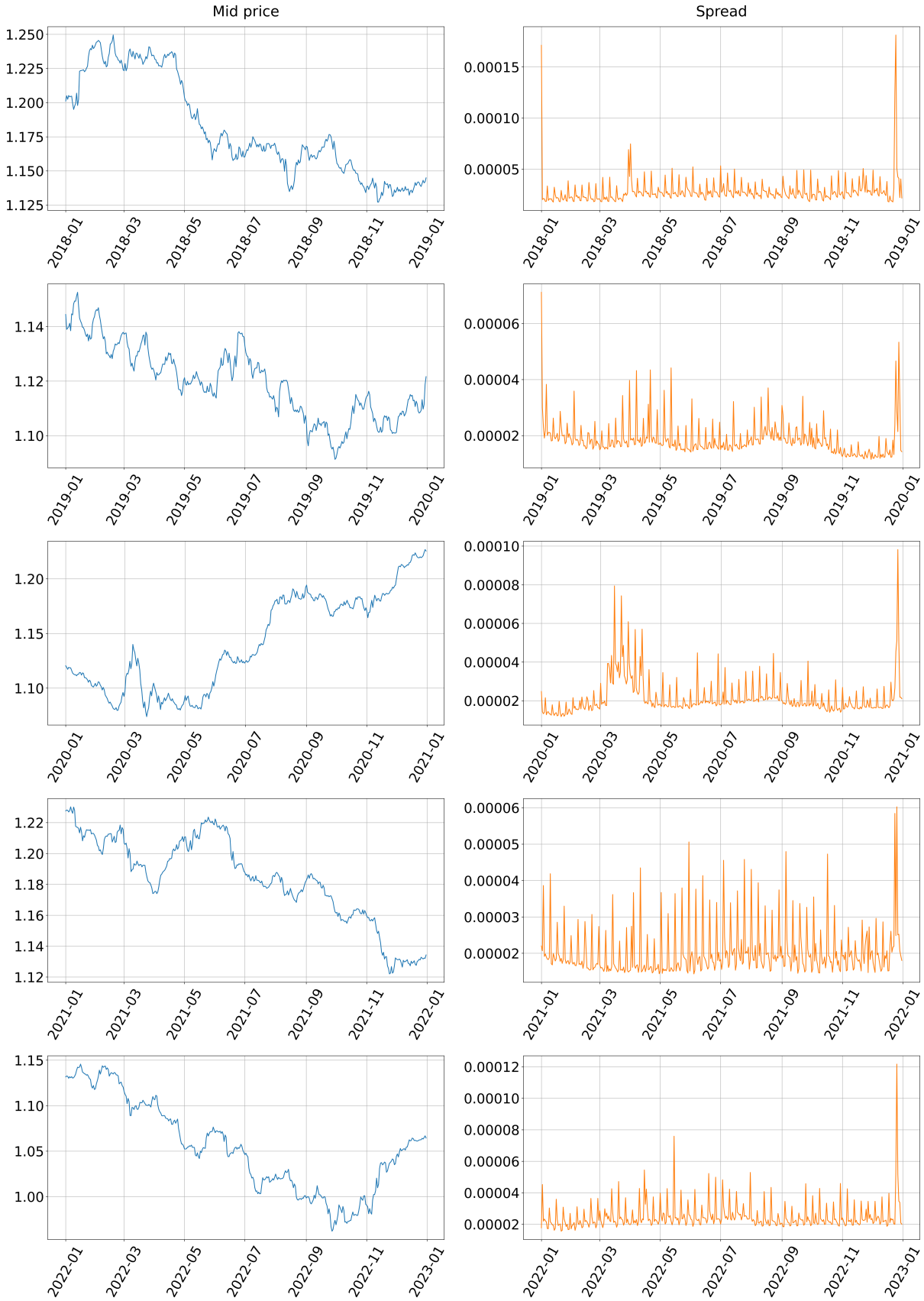


Figure 5.1: Mid and spread trends for each year between 2018 and 2022.

Year	ADF Statistic	P-Value
2018	-1.04	0.74
2019	-2.51	0.11
2020	-0.24	0.93
2021	-1.09	0.72
2022	-1.59	0.48

Table 5.3: ADF Test applied to the original mid-price between the years 2018-2022.

Another meaningful analysis consists in checking the stationarity of the mid prices. The term *Stationary* refers to a time series that does not have any form of seasonality, cyclic or trend patterns, or abrupt changes, maintaining a constant mean, variance, and autocorrelation over time.
Stationarity is an essential property as it ensures that statistical measures of a series remain constant over time. This characteristic facilitates the construction of models for the series, making the process easier and more effective.

First of all, we can easily see by a visual inspection and by Table 5.2 that our data does not have a constant mean and a constant variance over time. Furthermore, to check the stationarity of our data in a more technical way we use the *Augmented Dickey-Fuller Test* (ADF) statistical test, which tests if a unit root [1] is present in the tested time series. More specifically, the null hypothesis of the ADF consists of the presence of the unit root in the time series, suggesting its non-stationarity.

To interpret the result we use a *p-value* with a threshold of 0.05, meaning that if the p-value obtained from the test is less than or equal to this threshold, we can reject the null hypothesis and consider the time series to be stationary, otherwise, we can accept the null hypothesis by considering that the series contains a unit root and is therefore non-stationary.

From Table 5.3, we can observe that all the mid-prices of all years have a p-value significantly greater than 0.05, implying that we cannot reject the null hypothesis and indicating that all the years exhibit a non-stationary behaviour.

Fortunately, we can apply various transformations to our data to try to retrieve stationarity: in our case, as we can see in the next section, we calculate the difference between two consecutive mid-prices. After this transformation, we perform once more the ADF

[1] A unit root is a stochastic trend that can't be predicted, hence making the time-series non-stationary

Year	ADF Statistic	P-Value
2018	-110.89	0.00
2019	-72.71	0.00
2020	-65.23	0.00
2021	-63.54	0.00
2022	-63.90	0.00

Table 5.4: ADF Test applied the differences between two consecutive mid-price between the years 2018-2022.

test to check the stationarity of the obtained data. By observing the results shown in Table 5.4, we can reject the null hypothesis, certifying in this way the stationarity of our transformed data.

5.2.3. Forex Trading MDP formalization

In this section, we outline the formalization of the Forex Trading problem within the framework of MDPs, highlighting the unique aspects introduced in our work, such as continuous actions and variable fees associated with the chosen actions. The following formulation is based on the work of Riva [53]. First, we decided to formalize the Forex Trading problem as an episodic task: each episode corresponds to a trading market day, consisting of a total of 600 minutes between 8:00 a.m. and 6:00 p.m. (CET). The reason for choosing this time slot is based on the fact that it aligns with the trading hours of the New York Stock Exchange, which is known for its high trading volumes. This allows us to more easily identify potential profit strategies due to the presence of higher liquidity in the markets.
The motivations behind the choice of having an episodic task are both financial and practical. From a financial perspective, having a terminal state (the final minute of the episode) allows us to be able to close all positions. This avoids any increase in transaction costs associated with overnight charges imposed by some FX brokers. From a purely operational point of view, having an episode task allows us to have a limited horizon, making it also possible to work in an undiscounted setting.

State Formulation

In order to build a state formulation $\mathcal{S}$ that respects the Markovian property, each state must include enough information to predict the future state without recurring any state

history. For this reason, a usual choice is to introduce in the state formalization data related to past observations. In our formulation, we define the features included in each state as follows:

- the day of the week, expressed as an integer number between 1 (Monday) and 5 (Friday);
- the time of the current day, expressed as an integer (e.g. 8:00 a.m. time corresponds with the number 480);
- the current portfolio position derived from the choice of the previous action expressed by $x \in \{-1, 0, -1\}$ (*Short*, *Flat*, *Long*);
- the value of the *spread* in the specific time sample;
- the last normalized 45 *mid* price variations between consecutive minutes:

$$d_i^k = \frac{p_{t-k+1} - p_{t-k}}{p_{t-k}} \quad \forall k \in \{0, 1, 2, ..., 45\}. \tag{5.1}$$

We want to emphasize that the feature representing current portfolio allocation has two meanings. The first is related to its sign, which indicates the type of current allocation; the second, related to the absolute value of the allocation can be seen as a coefficient to be used to retrieve the amount invested: for example, in the case of a discrete portfolio position, this amount can only be 0 or the maximum possible traded value. Instead, in the case of a continuous portfolio position, this feature is a real number $x \in \mathbb{R}$ such that $-1 \leq x \leq 1$, allowing an amount in the portfolio in the range between 0 and the maximum possible amount.

The maximum amount of the current portfolio allocation can be treated as a parameter for the problem: in our case, it was set to $1e5\$$.

Action Formulation

The action defines the portfolio allocation that the agent is willing to invest. In the discrete setting, the set of possible actions can be defined as follows:

$$\mathcal{A}(s) = \begin{cases} \{0\} & \text{if } s \text{ is a terminal state} \\ \{-1, 0, 1\} & \text{otherwise} \end{cases} \tag{5.2}$$

Similar to the discrete setting, in the case of continuous action we have this definition:

$$\mathcal{A}(s) = \begin{cases} \{0\} & \text{if } s \text{ is a terminal state} \\ [-1, 1] & \text{otherwise} \end{cases} \tag{5.3}$$

at each time-step t, denoting with a_t the selected action, and with x_t the previous portfolio allocation, the absolute difference $|a_t - x_t|$ represents the traded amount performed in each step.

Reward Formulation

The reward received by the agent after performing the action a_t can be computed according to the following Equation:

$$r_t = a_t(p_{t+1} - p_t) - c_t|a_t - x_t|, \tag{5.4}$$

where

$$c_t = f + \frac{1}{2}\sigma_t. \tag{5.5}$$

In those equations, p_t is the current exchange rate at time t, p_{t+1} is the next exchange rate at time $t + 1$ potentially different from the last due to market fluctuation, x_t is the current portfolio allocation, f is a fixed transaction cost and finally σ_t is the current value of the spread at time t.

The reward formula in Equation 5.4 can be divided into two parts: the former represents the gain (or loss) due to the exchange rate variation, and the latter is the total cost charged by the FX brokers due to the allocation changes performed by the agent.
The motivation behind the presence of $\frac{1}{2}\sigma_t$ within transaction costs is due to the presence of bid and ask costs within the FX market: as previously mentioned, the bid price is the price with which a trader can sell his owned currency and the ask price is the price he has to pay when he has to buy other currency, the average value is the mid price used for modelling the price exchange rate, causing $\frac{1}{2}\sigma_t$ to represent the difference between the price transaction amount (respect to the bid/ask operation) and the exchange price considered. FX brokers commonly utilize this measure as a variable cost in relation to the overall transaction cost of trades performed by individual traders.

An important topic which is seldom considered in RL application in finance is the following: in real markets, the transaction costs charged by FX brokers regarding the spread value can depend on the size of the exchanged transaction amount.

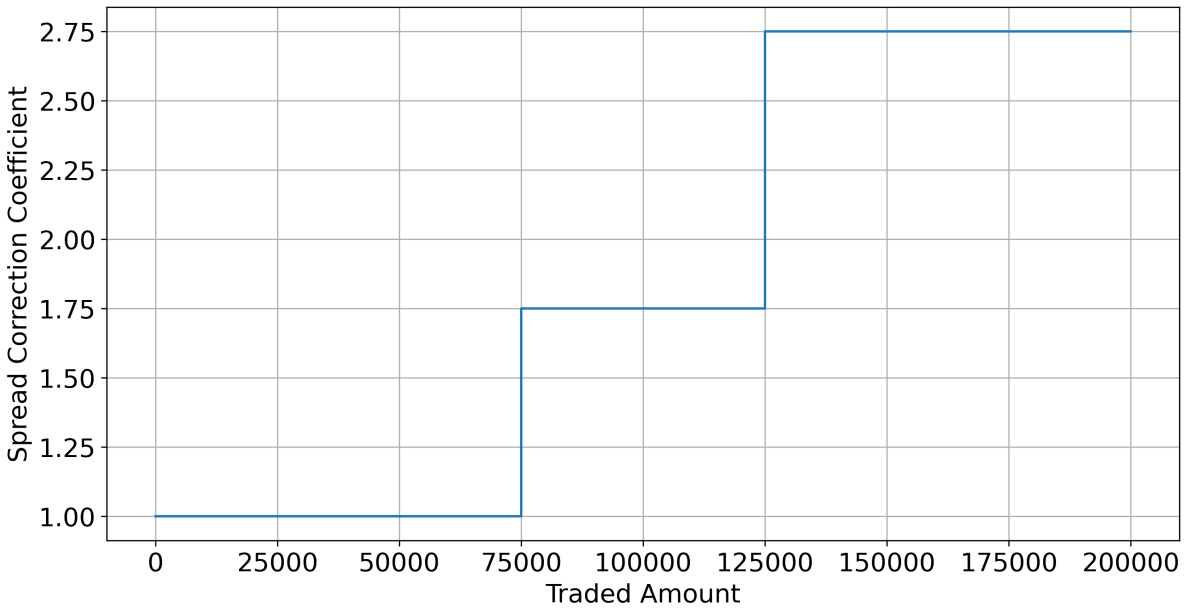


Figure 5.2: Step Function for the spread correction coefficient, where the x-axis indicates the maximum trade in a single transaction order.

More specifically, in the case of continuous actions, the agents can perform actions that bring the new portfolio to have only a portion of the maximum portfolio amount. Thanks to this fact, it is possible to evaluate the amount to set for each order, enabling us to model with more reality the transaction costs resulting from the size exchanged in each specific order.

Using a function that considers the volume of the trading amount, we can obtain a spread correction coefficient that we can use to transform the previous reward formula (5.4) in this way:

$$r_t = a_t(p_{t+1} - p_t) - \left(f - g(|a_t - x_t|)\frac{1}{2}\sigma_t\right)|a_t - x_t|, \tag{5.6}$$

where $g(x)$ can be any positive function $g : \mathcal{X} \to \mathcal{Y}, \mathcal{X} \in \mathbb{R}, \mathcal{Y} \in \mathbb{R}^+$ that models the FX broker variable transaction cost. In some of our settings, by taking $z = 1e5\$$ as the possible maximum portfolio allocation amount and the fixed transaction cost f equal to 0, to reflect the empirical costs observed in the FX markets we use a step function

constructed in this way:

$$g(x) = \begin{cases} 1 & 0 \leq x < \frac{3}{4}z \\ \frac{11}{4} & x > \frac{5}{4}z \\ \frac{7}{4} & \text{otherwise} \end{cases} \tag{5.7}$$

Clearly, we can resort to the base case taking as a function $g(x) = 1 \quad \forall x$.

Transition Probability

Due to the massive liquidity of the FX market, we assume for our model that any actions performed by the agent cannot impact the state dynamics of the environment. This assumption makes the state transition independent of the selected action, with the only exception of the allocation feature.

Discount Factor

Having modelled the problem as an episodic task, for the risk-neutral models we use an undiscounted learning rate equal to $\gamma = 1$. Instead, in the setting in which we use the risk-averse approaches, we used a learning rate equal to $\gamma = 0.9999$: the motivation behind this is related to the fact that the theoretical basis of these approaches mainly follows continuous tasks, with a not defined horizon. However, since the time horizon of our episodes is limited, especially when persistence is used, this does not impact the learning process.

5.2.4. Action Persistence

In the financial context, the frequency of interaction with the market is a fundamental aspect. In fact, a higher frequency interaction allows better control opportunities leading to a greater number of orders to be placed and thus potentially higher profits.

In order to retrieve a better signal-to-noise ratio, we modify the control frequency using the concept of *action persistence* (Section 2.6), where we keep the same allocation for a fixed amount of unit time-steps. Since we employ a batch-mode setting, for each trading day we then produce k new episodes, where k is the action persistence index used, by shifting the starting time of the trading day. Using persistence indexes greater than 1, many states useful for increasing the generalization capabilities of the model would be lost; this is due to the fact that persistence reduces the number of action-state interactions. In policy-based methods, in order to avoid limited generalization capacity that increase the

overfit on the only available data, this availability of additional states is a fundamental aspect.

Finally, to be compliant with the action-persistence definition, we have modified the general reward formula in Equation 5.4 to account for the next persisted exchange price value as follows:

$$r_t = a_t(p_{t+k} - p_t) - c_t|a_t - x_t|. \tag{5.8}$$

5.3. Algorithm Implementation

In order to be able to create an agent capable of adopting a profitable policy in a real FX trading market, we have implemented the Fitted Natural Actor-Critic algorithm (described in Section 3.3) from scratch, training the model using the Forex MDP formulation depicted in the previous section.
More specifically, we have implemented the FNAC architecture with the following three main components:

- for the main critic component used for computing the value function we initially explored the use of a *Feed-Forward Neural Network* (FFNN), focusing later on the use of the *XGBoost* algorithm [54].
- for the critic component, used for estimating the advantage function and for computing the w parameters for the natural gradient update, we use a Ridge regressor with an iterative update method.
- for the actor component we use a *Feed-Forward Neural Network*.

In the following, we detail our FNAC implementation in order to work with the Forex setting with discrete and continuous actions.

5.3.1. Critic components

As previously mentioned, in the FNAC algorithm we can find two different critic components: the first is used to compute the value function; the second is adopted for estimating the w parameters in order to update the actor component with a natural gradient approach.

Regarding the value function approximator, we initially use two different implementations: The first approach was based on a *Feed-Forward Neural Network* architecture. It had an input layer with a number of neurons equal to the dimensionality of the feature space, two hidden layers with 64 neurons each and an output layer with a single neuron representing

the estimation of the value function, where each layer uses a leaky ReLU activation function; the second approach, where we focused our attention, is based on the XGBoost (Extreme Gradient Boosting, [54]) model, which is a decision-tree ensemble algorithm that utilizes the gradient boosting framework.

Both implementations were trained using the trajectories sampled by the policy of the actor component. In addition, we used a validation procedure to control the overfitting on the samples used for the trajectory creation.

For the second critic component, used for computing the estimation of the advantage function, we used a Ridge Linear regressor method using an LSQR solver based on the Scikit-Learn API [55]: more specifically, the least squares (LSQR) algorithm is an adaptation of the conjugate gradients used for resolving and speed up the prediction in rectangular linear systems.

To reduce the computational effort for the advantage estimation, at each iteration of the algorithm we sample a fixed number of days (with the corresponding $N \cdot k$ trajectories due to the use of action persistence): for each sample, we retrieve the partial derivative of the actor prediction and we calculate the compatible basis function (as in Equation 3.9). At this point, we obtain a rectangular matrix $(N \cdot k) \times |\theta|$ that is used as input for the advantage estimator critic to obtain the parameters w, useful for updating the actor policy parameters.

5.3.2. Actor Component

The actor component is in charge of the policy estimation: since we needed a differentiable approximator, we decided to use a Feed-Forward Neural Network using a different architecture depending on the type of action space we intended to use.

Discrete Actions

To support actions within a discrete action space (described in Equation 5.3), we used an input layer consisting of a number of neurons equal to the dimensionality of the feature space, one hidden layer with 32 neurons, both with leaky ReLU activation function and an output layer with three neurons, one each for each action, with a softmax function to normalize the output neurons to a probability distribution over predicted output actions. Finally, we extracted the chosen neuron that represents the action related to the policy $\pi(a|s)$ using a multinomial probability distribution derived from the prior probabilities.

Continuous Actions

When a continuous action space is taken into account, (e.g. Equation 5.3) we used the same input layer as in the previous case, two hidden layers with 32 and 16 neurons using leaky ReLU activation functions, and finally, an output layer with only two neurons defining the mean μ and the standard deviation σ of a normal distribution, used for representing a policy parametrization through the approach described in Section 2.4.4. In order to retrieve a value in the range of feasible actions allowed from the action space $\mu \in [-1, 1]$, we transform the output mean by resorting to the hyperbolic tangent function. Similarly, we use a Softplus activation function to ensure a valid standard deviation that consists of only positive values.

Such solution, implemented to allow continuous actions, gives rise to two new peculiar issues: the first stems from the fact that the normal distribution defined by the network outputs has support in the entire real domain $x' \sim \mathcal{N}(\mu, \sigma^2)$, $x' \in \mathbb{R}$ when in our case we need only the restricted support $x \sim \mathcal{N}(\mu, \sigma^2)$, $x \in [-1, 1]$. To fulfil this requirement and efficiently represent the distribution of the policy used, ensuring a proper signal for the backpropagation phase, we replaced the normal distribution with the *Truncated Normal distribution* [56] that bound the random sample x in the previously restricted range.

The second issue derives from the computation of the natural gradient, where we need the partial derivative of the compatible basis function (Equation 3.9) with respect to the policy parameters. Since we sample the action from a probability distribution, we obtain a stochastic sample that is not directly differentiable, posing a problem to use this type of policy parametrization.
To solve this problem we used the *Reparameterization Trick* [57], which allows us to compute a differentiable sample x obtained with the previous normal distribution $\mathcal{N}(\mu_\theta, \sigma_\theta)$, by recasting it respect to an external random variable ϵ sampled from another normal distribution $\mathcal{N}(0, 1)$:

$$x = \mu_\theta + \epsilon\sigma_\theta, \tag{5.9}$$

through this formula, the stochasticity of the sample x is expressed by the random variable ϵ permitting to calculate the differentiation with respect to the policy parameters θ.

5.4. Risk-Aversion approaches

In order to keep the inherent risk under control, we extend the previous risk-neutral model by modifying the objective function in order two optimize two different risk measures:

- The Reward Conditional Value-at-Risk (RCVaR, [58]), a reward-based version of the classic CVaR:

$$\rho^{\pi}_{RCVaR}(R,\alpha) = \max_{\rho} \left\{ \rho - \frac{1}{\alpha} \mathbb{E}_{\substack{s\sim d_{\pi_\theta} \\ a\sim\pi_\theta(\cdot|s)}} \left[R(s,a) - \rho \right]^{-} \right\}, \tag{5.10}$$

- The Mean-Volatility risk measure (Equation 2.34)

5.4.1. RCVaR optimization

As shown in [58], is possible to take advantage of the RCVaR structure by reversing the order of the maximization and transforming the inner problem in an MDP:

$$\begin{aligned} \max_{\pi} \left\{ \eta^{CVaR}_{\alpha,\pi} \right\} &= \max_{\pi} \left\{ \max_{\rho} \left\{ \rho - \frac{1}{\alpha} \mathbb{E}_{\substack{s\sim d_{\pi_\theta} \\ a\sim\pi_\theta(\cdot|s)}} \left[R(s,a) - \rho \right]^{-} \right\} \right\} \\ &= \max_{\rho} \left\{ \max_{\pi} \rho - \left\{ \frac{1}{\alpha} \mathbb{E}_{\substack{s\sim d_{\pi_\theta} \\ a\sim\pi_\theta(\cdot|s)}} \left[R(s,a) - \rho \right]^{-} \right\} \right\}. \end{aligned} \tag{5.11}$$

For a given value of ρ, the inner problem is equivalent to an MDP with a transforming reward:

$$\widetilde{R}(s,a) = \rho - \frac{1}{\alpha} \left(R(s,a) - \rho \right)^{-}, \tag{5.12}$$

where the shaping additive factor ρ helps to enhance the stability of the learning process [58].

To decrease the non-stationarity of the reward and to reduce the computational effort caused by the external maximization, we used the approach proposed in [59] by using ρ as fixed risk-aversion hyperparameter and treating α only as a scaling factor, allowing us to bypass the outer maximization step. Moreover, with the fixed ρ, we employed the reward transformation described in Equation 5.12 to model our MDP.

5.4.2. Mean-Volatility optimization

For the mean-volatility risk measure objective we follow the approach called *direct-method* proposed in [15] using the reward transformation presented in Equation 4.2.
More precisely, to compute the previous reward transformation we require an estimation of the normalized expected return J_π; for this reason, we use a Monte-Carlo procedure

that employs the current policy π:

$$\widehat{J}_{\pi} = \frac{1-\gamma}{1-\gamma^T} \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t), \tag{5.13}$$

where N is the number of trajectories used for the estimation, T is the horizon of the trajectories and $1-\gamma^T$ is a correction factor to take into account the finite horizon of the geometric series due to the finite length of the trajectories.

6 | Experimental Results

In this chapter, we illustrate the results obtained by the FNAC algorithm using the formulation defined in the previous chapter, describing in a more detailed way the settings and the peculiarity of each implemented model.

We will begin by describing the model selection procedure, specifying the *training set*, *validation set* and *test set* used to train and evaluate the learned agent.
Then, we present the results of each approach, starting from the discrete actions setting in Section 6.2, and moving to the case of continuous actions in Section 6.3. In the latter, thanks to the continuous formulation, we present the result of the extended models using the transaction fees function (Equation 5.6) and the integration of the risk measures explained in Section 5.4.

Each result shown will be derived from the training of the model with different independent runs to understand the robustness of our algorithm with respect to different initializations and learning instances. Moreover, where possible due to our computational availability, we will show the same results with different persistence parameters.

6.1. Model Selection

The actor-critic architecture of the FNAC algorithm necessitates distinct components for value function and policy approximation.[1] This separation leads to increased complexity, both in terms of the model training process due to a larger number of hyperparameters to be selected and the computational effort required to train the two separate components. Additionally, the FNAC implementation introduces a third component for approximating the advantage function, further augmenting the overall complexity of the algorithm.

In any case, finding the right requirements for each component, such as the best hyperparameters, is a key aspect for finding a model that can maximize its performance.

As said in Section 5.3, we have initially explored the use of a Feed-Forward Neural Network

[1] in some cases, as some implementations of PPO, the two components may share some layers of the neural network

for the critic, related to the value function approximation, and for the actor component. However, a number of reasons (the model explainability, the sample complexity and the high number of hyperparameters to be tuned) led us to adopt a FFNN only for the actor component, while the critic is performed by means of an XGBoost model: the choice of XGBoost is motivated by its excellent capability to handle diverse and large datasets, along with its high prediction performances and efficient computational processing. Moreover, XGBoost can provide greater explainability capabilities, providing information about the more influential features on the final prediction, with the possibility of understanding the structure of the final model thanks to its tree-based nature. Finally, being an optimized ensemble implementation that combines gradient boosting and bagging approaches, its hyperparameters are mainly related to these latest techniques.

The validation process is performed through a selection procedures among a set of available hyperparameters. This set includes the learning rate, the number of *boosting rounds* related to the gradient boosting approach and the *min-child weight* that indicates the minimum sum of the weights of the leaf observations to perform the node splitting, allowing us to control the overfitting on the training data.
An additional procedure we performed to limit overfitting is the use of *early stopping* procedure on the boosting rounds by using a selected validation data set.

Taking into account the critic for the advantage approximation with a ridge regressor, the main hyperparameter to be tuned is the α coefficient for regulating the L2 term (Ridge coefficient). Additionally, since we employed an iterative method, it was necessary to determine the hyperparameter ϵ to establish the minimum precision required for the final prediction. This hyperparameter is hence crucial to find an optimal trade-off between the desired precision and the computational effort involved, especially regarding the high dimensionality of the linear system to be solved due to a large number of actor parameters and the samples for the advantage estimation.

Finally, about the FFNN used for the actor component, we mainly tuned the learning rate and the number of parameters of the network, by defining the number of layers and neurons needed considering the computational effort resulting from the computation of the Jacobian.

In order to retrieve an accurate measure of the performance of the trained model through a validation scenario, we decided to split the available data in the following manner:

- Training Set: we used the years 2018 and 2019, obtaining (after a preprocessing procedure) a dataset composed of 508 days.

- Validation Set: we used two different types of sets to validate our model.
  - Validation for the critic component: we used the year 2020, with 236 trading days (after a preprocessing) to perform an early stopping procedure on the XGBoost component.
  - Validation for the actor component: we used the year 2021 with 246 preprocessed working days to perform the model selection of the actor component. These data are not used in any way for the model training but only for selecting the best model with respect to the chosen measure.
- Test Set: we used the 2022 year with 252 preprocessed working days to assess the performance of the selected policy

The decision to use two different validation sets is based on several factors: firstly, the data in the year 2020 exhibited high variance and unusual patterns, which could potentially impact the performance of the model. To mitigate this issue, we excluded the month of March (Covid-19 spread) to ensure more stable data for validation. In a second instance, the use of an additional validation set for the model-selection regarding the agent policy, allows us to make a more reliable assessment of the performances obtained.

6.2. Discrete Setting

In this section, we show the results obtained by our model using the discrete setting formulation (Equation 5.2). We compare the performance obtained with the baselines *Buy&Hold* and *Sell&Hold*. Furthermore, in order to have a more robust comparison with another algorithm, we will compare the performance obtained in the test year 2022 with the performance of the FQI algorithm (used in [3, 44]), using the same discrete formulation.

As a first result, we present a comparative table (Table 6.1) of the performance through the years of the learning process, employing the best models which maximize the cumulative reward in the 2021 validation year using the model selection described in the previous section. In this setting we consider only the spread as transaction costs, hence we set $f = 0$ for the reward presented in Equation 5.8.

From Table 6.1, we can notice that the persistence value has a great and useful impact on the cumulative return yielded at the end of the year: in fact, models that use a higher value of the persistence obtain a higher Sharpe Ratio and a higher P&L in the validation year 2021.

Persistence	% P&L (Mean ± Std)			Sharpe Ratio
	2018-2019	2020	2021	2021
1	36.33 ± 5.31	2.71±3.66	11.98 ± 1.58	2.50
5	36.14 ± 1.17	-1.55 ± 1.24	15.27 ± 1.20	3.20
10	44.65 ± 2.87	4.47 ± 2.62	14.81 ± 2.33	3.18

Table 6.1: Performance obtained with 3 different seeds using the best model obtained in the validation years 2021, measured by the cumulative percentage P&L (mean ± std) and the Sharpe Ratio.

In addition, we present in Figure 6.1 the learning curve of the model with persistence 10. The plot shows the cumulative return obtained over the iterations during the learning process on the training and validation set: as we can see, even if the learning curve is not completely stable, the model improves performance throughout learning, gradually overfitting the training data as the iterations progress.

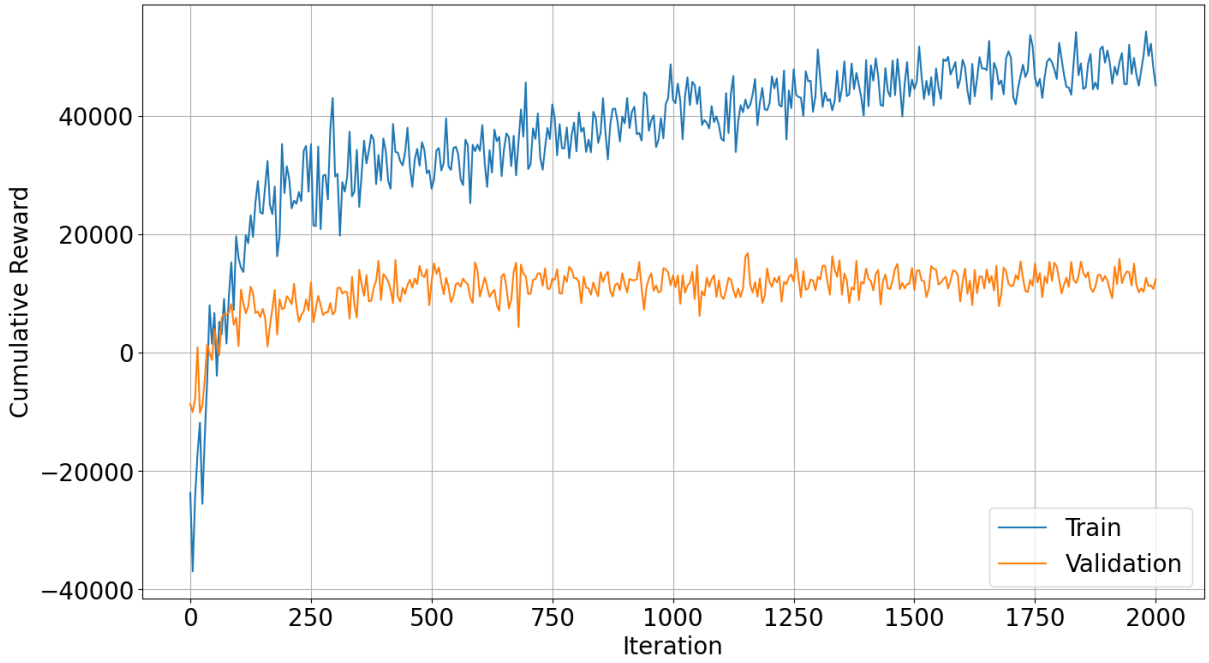


Figure 6.1: Cumulative return over the training process in 2018-2019 years (training set) and in the year 2022 (validation set) using a model with persistence 10 and discrete actions.

From Table 6.2, we can observe that the best model in the test year 2022 is the one obtained with persistence 10. This could be because of the fact that high values of persistence are able to return a signal that can build a model with a higher degree of generalization.

Persistence	% P&L (Mean ± Std)	Sharpe Ratio
1	-11.20 ± 6.67	-1.40
5	0.49 ± 2.37	0.05
10	10.93 ± 2.41	1.25

Table 6.2: Performance obtained in the test year 2022 using the models obtained in the validation year 2021, measured by the cumulative percentage P&L (mean ± std) and the Sharpe Ratio.

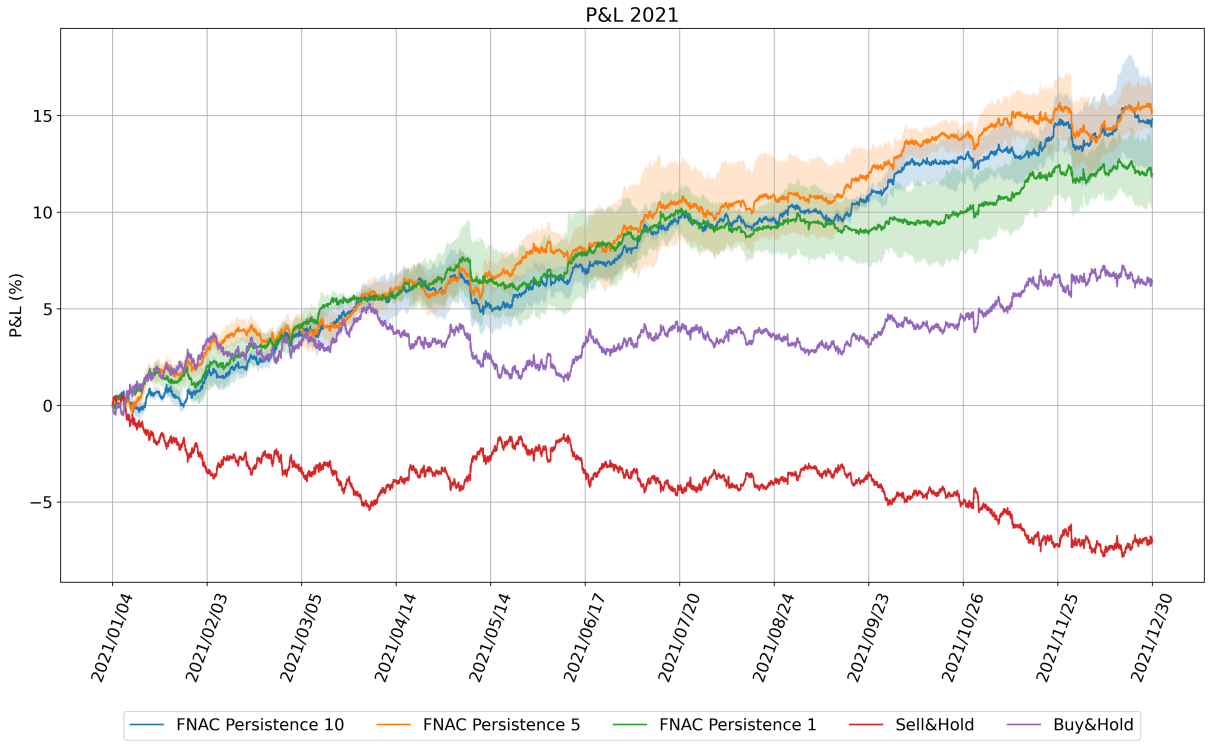


Figure 6.2: Plot of the percentage P&L obtained using the models in table 6.1 in the validation year 2021.

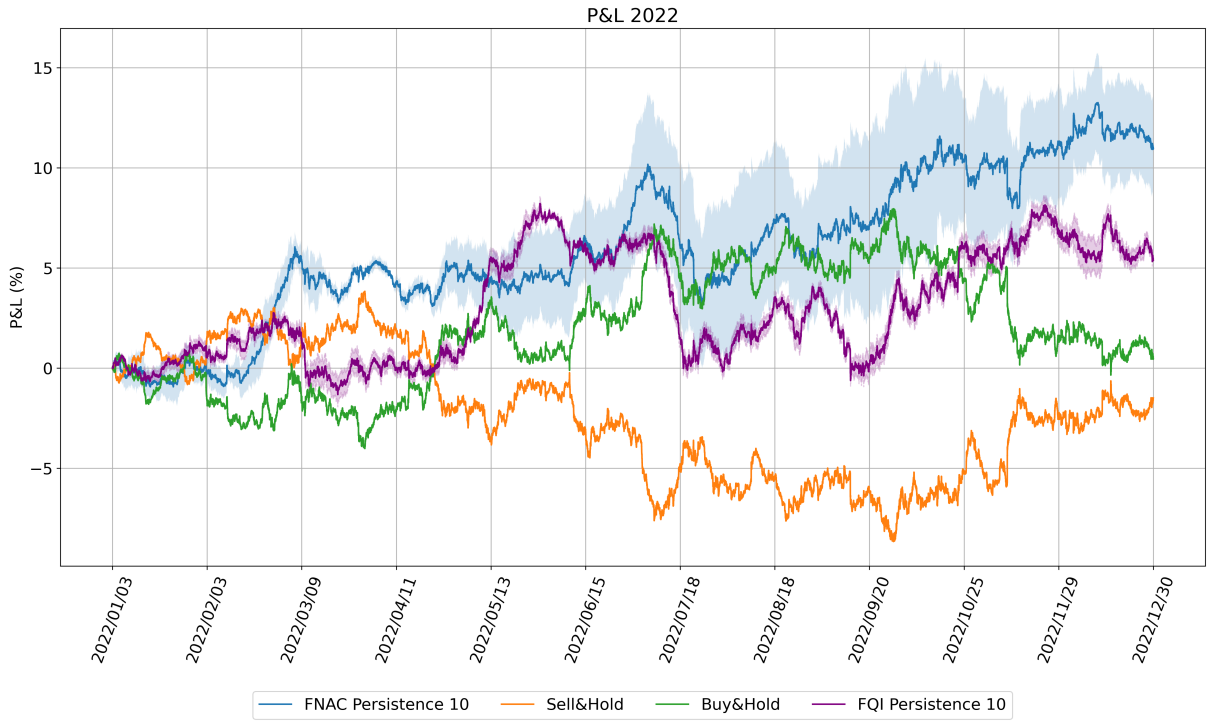
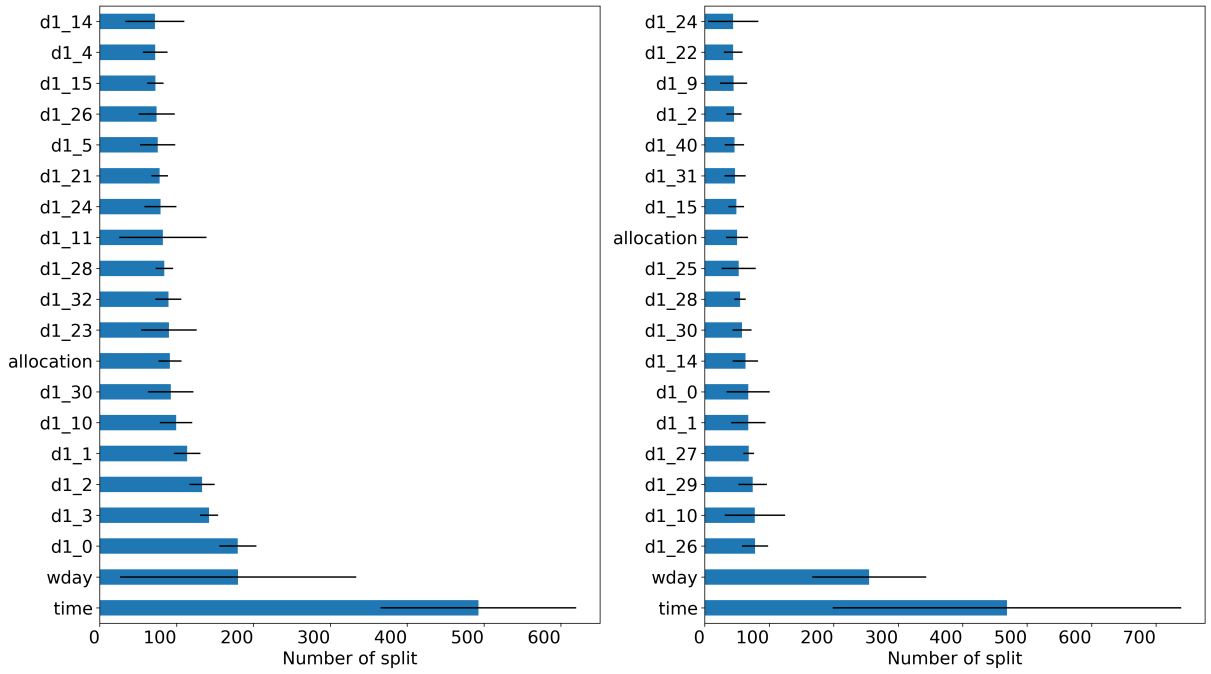


Figure 6.3: Plot of the percentage P&L in the test year 2022, comparing the best model in 6.2 with FQI with persistence 10.

From Figure 6.3, we can notice that the FNAC algorithm with persistence of 10 outperforms FQI, as well as the Buy&Hold and Sell&Hold baseline. However, it exhibits a significant standard variation during the year, not observed in the validation year in Figure 6.2. This high variability can mainly be attributed to the initialization of the neural network weights used to represent the agent's policy in the models trained using different seeds.

To enhance the interpretability of these results, focusing mainly on patterns with higher persistence, we start to analyze the critic component exploring the features that have the most significant influence on the approximation of the value function: this can be accomplished by utilizing the feature importance derived from XGBoost, using as a measure the number of times that each feature is taken into account to split across the trees. The feature importance is represented in Figure 6.4.

We can see that the most characterizing features for prediction are mainly the minute of the day (the *time* feature in the bar plot) and the day of the week (the *wday* feature in the bar plot). Instead, perhaps surprisingly, the importance of the differences regarding the mid prices do not follow a specific chronological order, as well as the previous allocation is not shown in a distinguished position.

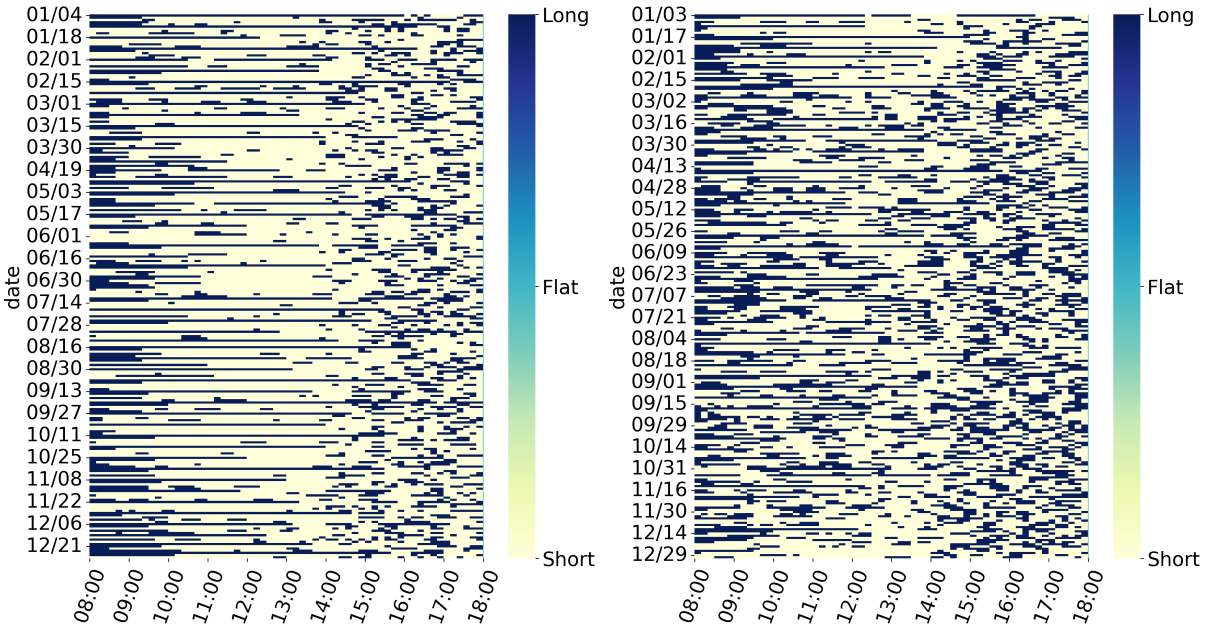


(a) Persistence 5 (b) Persistence 10

Figure 6.4: Bar Plot representing the mean and the standard deviation of feature importance of the first 20 features across different seeds

Being the actor model a neural network, its explainability of the model is relatively limited due to the large number of weights and the utilization of non-linear layers. While we have made some efforts to interpret the final weights, our primary focus will be on representing the final predictions of its represented policy.

In Figure 6.5 are shown the allocation heatmaps of the validation year 2021 and test year 2022. In this heatmaps each row represents a trading day, each column represents a precise timestep (using persistence 10 each column represents a group of 10 minutes) and the colour denotes the allocation made at the specified timestamp. In any case, it is possible to observe the presence of certain temporal patterns, especially during mid-day through the Short action, and at the beginning and end of the day through the Long action.
Similarly, it can be observed that the hold action is practically never executed and that the chosen actions are held for several minutes (in addition to the implicit minutes due to persistence). This conservative behaviour can be induced by the price change predictions generated by the policy and the fact that the analyzed spread (Figure 6.4) does not contribute crucially to the position change.



(a) Allocations of the validation year 2021

(b) Allocations of the test year 2022

Figure 6.5: Heatmaps of the allocations chosen by the best model with persistence 10 identified in the validation year 2021

6.3. Continuous Setting

In this section, we will describe the results obtained from the setting of our greatest interest, namely the one resulting from the use of continuous actions. We will start with the basic setting by simply extending the discrete case. Afterwards, we describe the results obtained through the use of the step function in Equation 5.6. Finally, we use the previous risk measures to obtain a risk-averse behaviour agent.

In addition, we will specifically examine the results obtained only from models with persistence equal to 10 (in which each allocation is maintained for 10 minutes). This is mainly because the results obtained with smaller values of persistence showed the generalization problems present in the discrete case.

6.3.1. Base Setting

Following the formulation described in Section 6.1, we added an additional layer to the neural network with respect to the discrete setting, in order to add non-linearity in the prediction capabilities in a continuous range. We now describe the main results obtained by our model starting from the performance through the years used in our formulation,

Years	% P&L (Mean ± Std)	Sharpe Ratio
2018-2019	36.76 ± 6.90	5.92
2020	8.31 ± 0.90	1.63
2021	12.73 ± 0.63	3.03
2022	8.91 ± 4.32	1.29

Table 6.3: Performance obtained with 3 different seeds using the best model with persistence 10 obtained in the validation years 2021, measured by the cumulative percentage P&L (mean ± std) and the Sharpe Ratio.

depicted in Table 6.3 and Figure 6.6.

As we can see in the plots presented in 6.6, the performance in the test year 2022 is slightly worse than that of the discrete setting: this can be attributed to the greater difficulty in finding the most effective strategies due to the continuous action space. However, similarly to the discrete part, we can see an increase in performance variability between the different initializations concentrated at the end of the year, with a common significant performance drop at the beginning of July where the models seem unable to control and to limit losses effectively.

In order to have a better understanding of the differences between this setting and the previous one, the cumulative return during the learning process compared to the iterations in the training years 2018-2019 and validation year 2021 is presented in Figure 6.7: In this case, the training seems much more stable than that of the discrete setting, We attribute the improved stability of the policy across different iterations to the use of the truncated normal distribution. This provides a more consistent natural gradient update to the network parameters and contributes to a more uniform policy update.

To analyze the allocations chosen by the model, we can consider the heatmap allocation. Due to the use of a continuous action space, its interpretability is more limited with respect to the discrete counterpart. For this reason, in addition to the heatmap (presented in Figure 6.8), we show in Figure 6.9 a histogram representing the distribution of the allocations chosen by the model in the test year 2022.

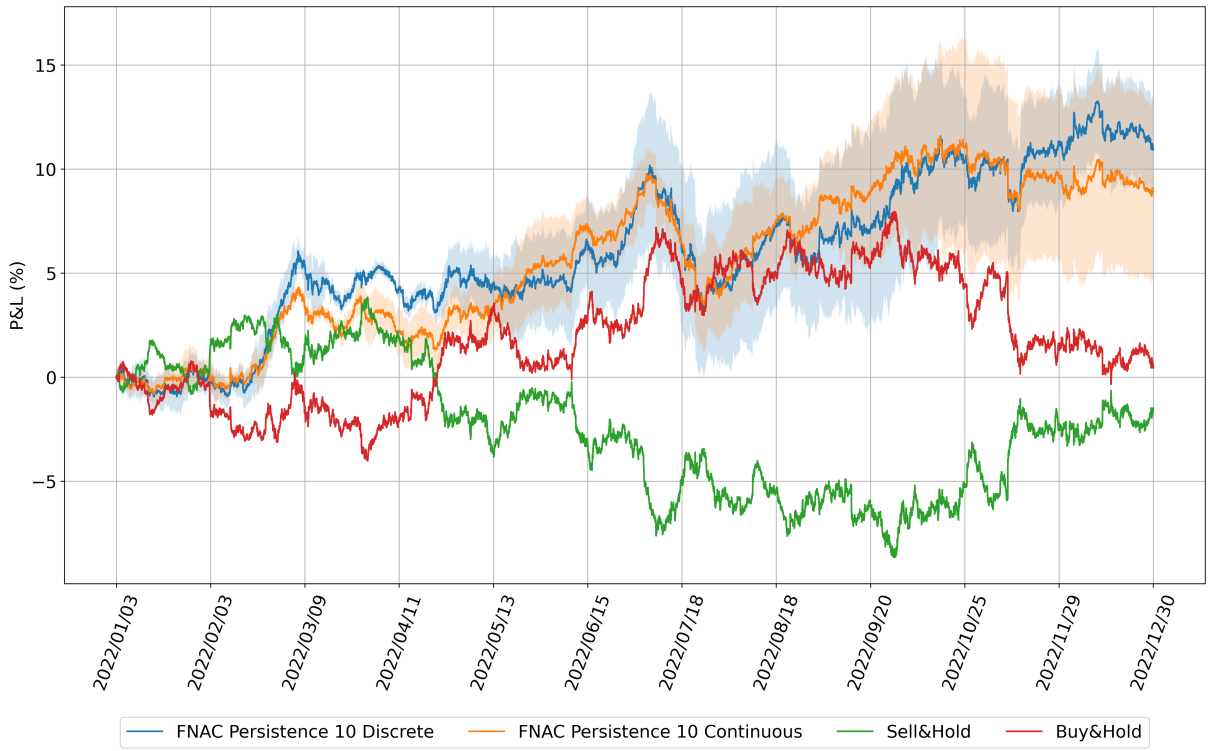


Figure 6.6: Plot of the percentage P&L in the test year 2022, comparing the best models in 6.3 with the best discrete models FNAC with persistence 10

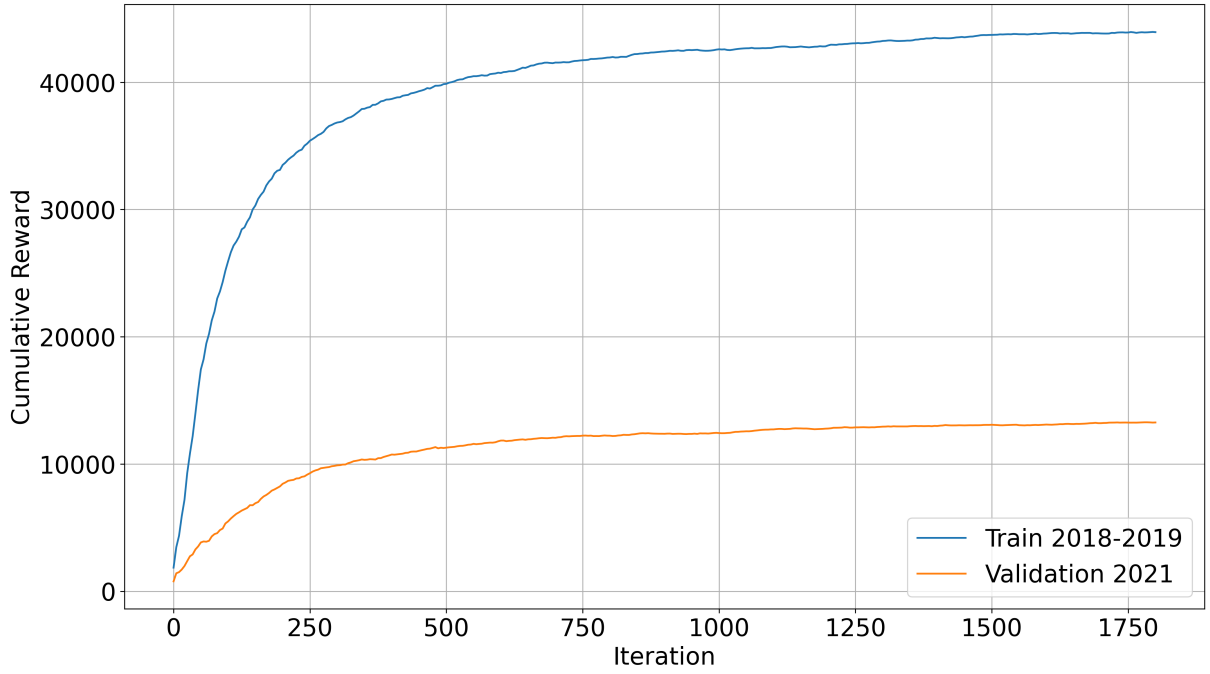
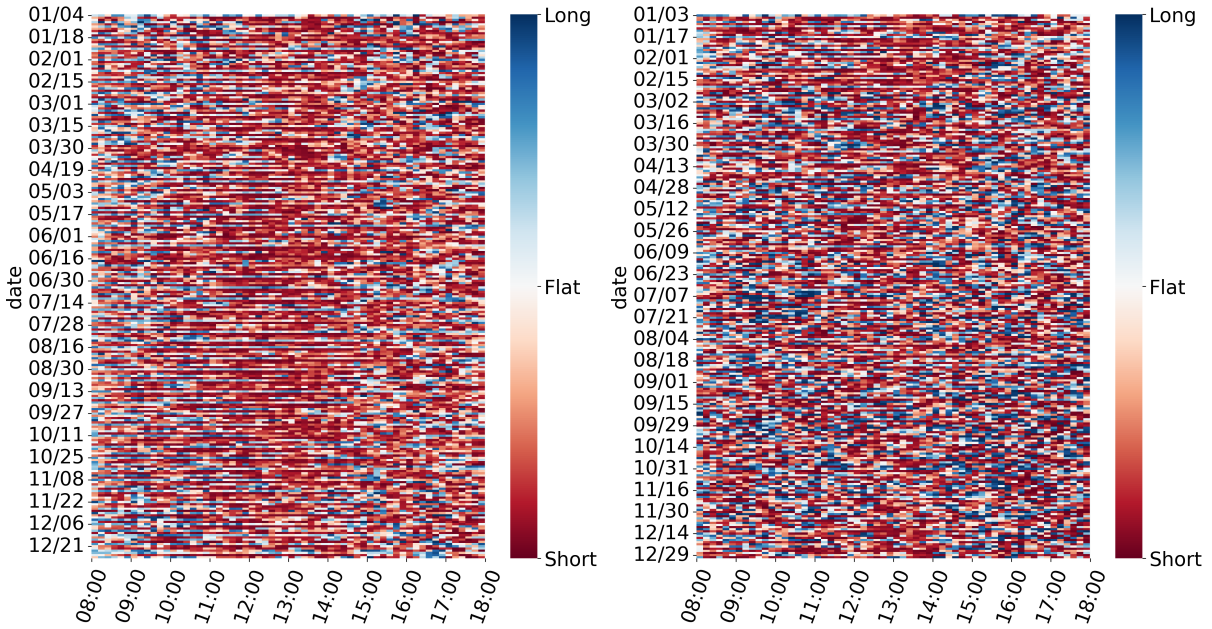


Figure 6.7: Cumulative return over the training process in 2018-2019 years (training set) and in the year 2021 (validation set) using a model with persistence 10 and continuous actions.



(a) Allocations of the validation year 2021.

(b) Allocations of the test year 2022.

Figure 6.8: Heatmaps of the allocations chosen by the best model with persistence 10 identified in the validation year 2021.

As can be seen by the heatmaps in Figure 6.8 and the histogram regarding the test year 2022 in Figure 6.9, the chosen allocations tend more towards the extreme possible allocation of the Long action and, to a lower extent, toward the maximum value of the Short action, indicating that the agent has a tendency to choose actions for keeping in the portfolio the largest possible amount.

Another useful analysis for evaluating the risk attitude of the model and its predisposition to change the allocation, with the consequence of paying the fee derived from the spread, is to understand how the agent behaves with respect to the allocation present in its portfolio: The histogram presented in Figure 6.10 shows the allocation changes between subsequent timestep, highlighting the presence of actions that completely change the current portfolio going from a long position to a short position. Of course, these types of actions are potentially the riskiest, both because they incur high transaction costs due to the spread and because they indicate that the agent completely changes its prediction with respect to the market trend.

6.3.2. Step Fee Function Setting

In this section, we integrate the use of the step function defined in Equation 5.7 to modify the reward formulation. As previously anticipated, this function models the spread

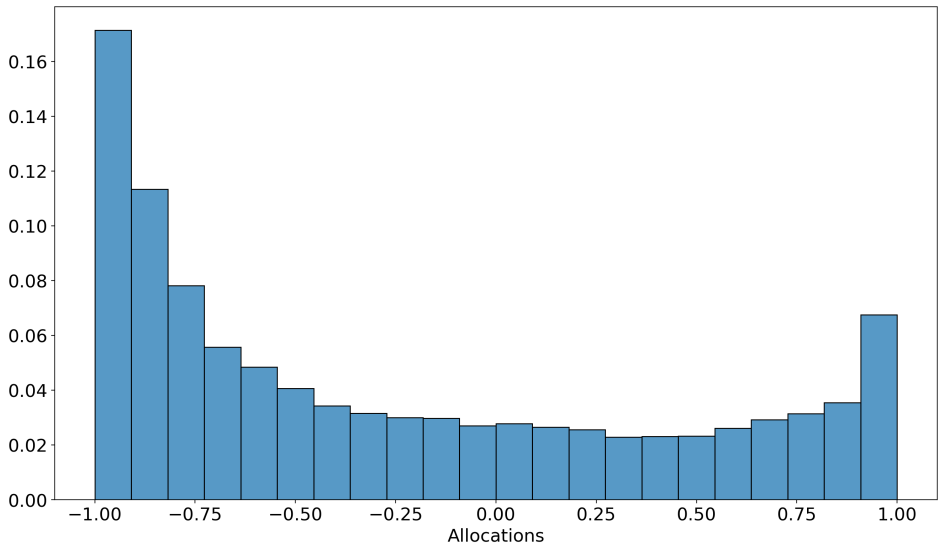


Figure 6.9: Histogram of the allocations chosen by the best seed in the test year 2022 (excluding the final hold action performed at each end of the day), wherein the x-axis is the action and the y-axis is the proportion of the action to the total.

transaction cost by taking as input the amount of the allocation changes, permitting to consider in a more realistic way the cost encountered in the market when the agent chooses to place large orders.

Years	**% P&L (Mean ± Std)**	**Sharpe Ratio**
2018-2019	31.42 ± 2.61	4.48
2020	2.30 ± 3.53	0.58
2021	10.21 ± 0.56	2.58
2022	3.16 ± 1.31	0.49

Table 6.4: Performance obtained with 3 different seeds, employing persistence 10 and the step function in 5.7, by using the best model selected in the validation years 2021.

Observing the performance in Table 6.4, the models trained in this formulation demonstrate good performance in the year 2021, however, they fail to perform well in the year 2022, primarily due to a drop in performance towards the end of the year (as shown in Figure 6.11). This drop was already observed in the base setting and seems to be caused by the change in some market conditions, specifically, looking at the mid-price trend in 5.1 it appears that the market is undergoing a bearish trend of the quote currency which the agent cannot manage efficiently, especially due to the lack of similar market behaviour in the training set.

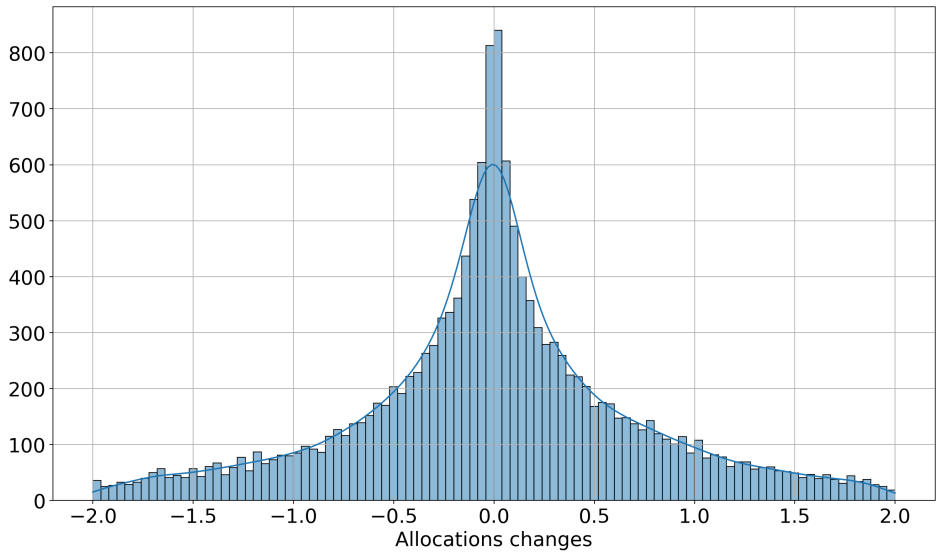


Figure 6.10: Histogram of the allocations changes in the test year 2022, with the estimated distribution using a kernel density estimation.

As mentioned earlier, the introduction of the step function increases the sensitivity of the model to the order size and consequently to changes in allocation. This holds particularly for higher trade amounts, since transaction costs increase as the amount of allocation changes increases. To demonstrate whether the models have successfully learned to handle this setting, we can examine the heatmaps showing the allocations chosen by the agent.

Observing the heatmaps in Figure 6.12, it is evident that there are continuous colour stripes, representing patterns where the agent gradually changes allocations to not encounter high transaction costs.

Similar to the previous base setting, in Figure 6.13 is shown the histogram of the allocation changes, highlighting that the agent limits the volume of the order to mitigate the impact of spread costs.

6.3.3. Risk-Averse Setting

In this subsection, we show the results obtained by the integration of the two previously risk-averse objectives mentioned in Section 5.4. Since the optimization objective is no longer to maximize the expected reward, the best models selected will be those that maximize the risk-averse objective function adopted.

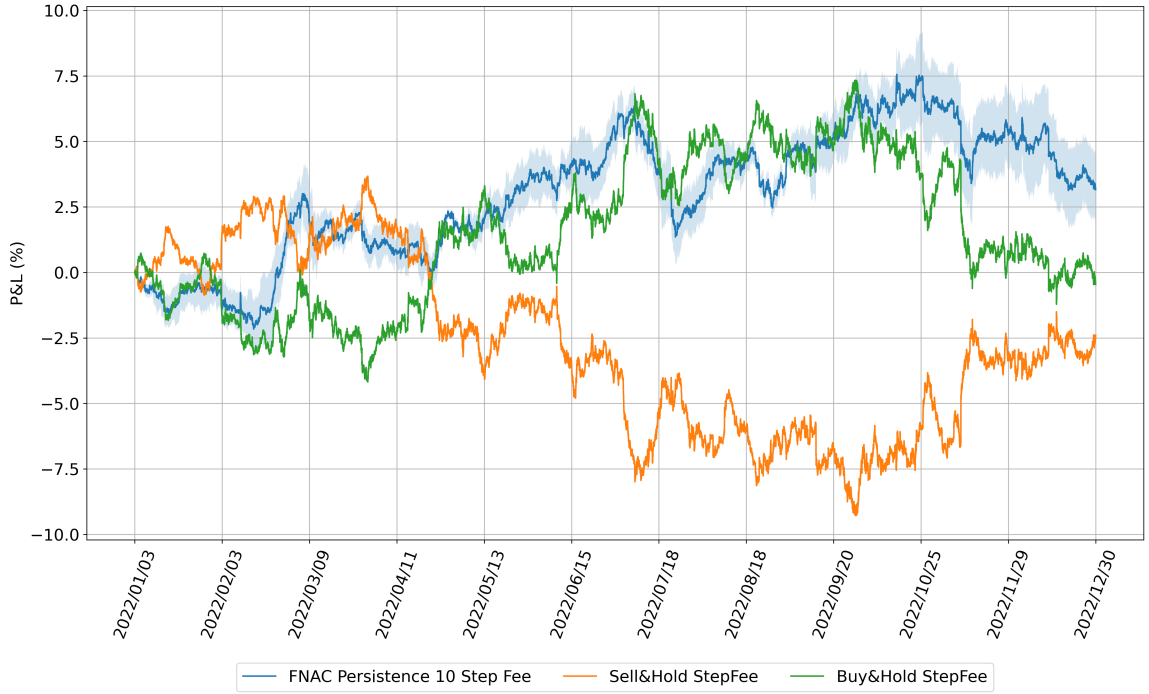
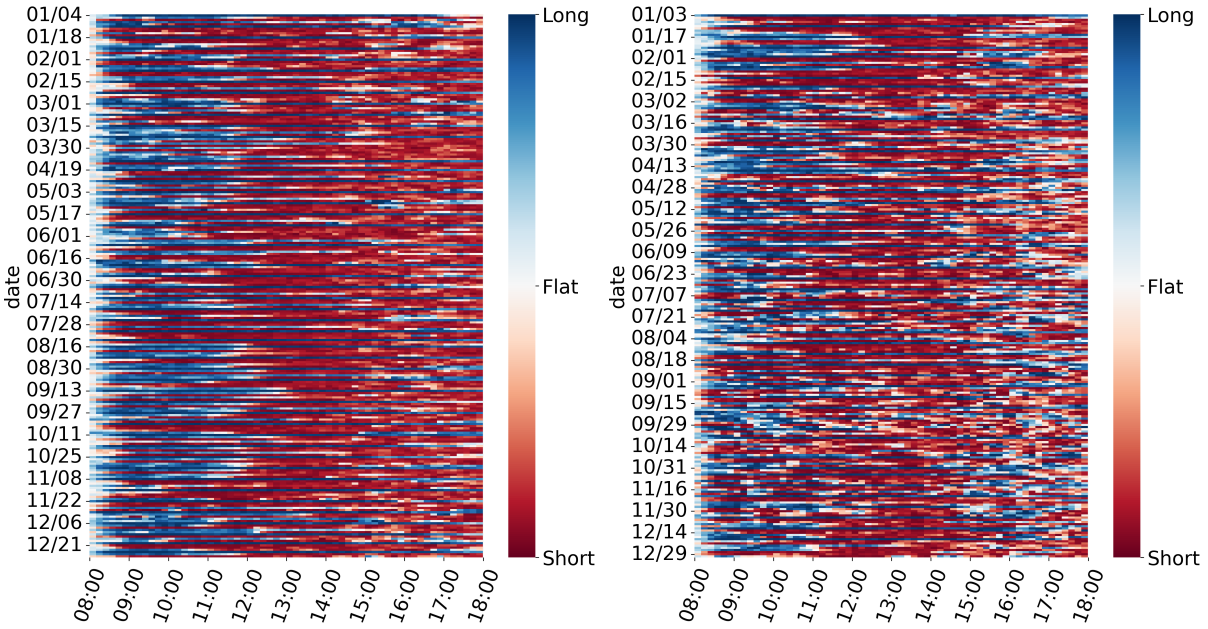


Figure 6.11: Plot of the percentage P&L in the test year 2022, comparing the best models in 6.4 with the new step fee baselines.



(a) Allocations of the validation year 2021

(b) Allocations of the test year 2022

Figure 6.12: Heatmaps of the allocations chosen by the best model using the step fee function and persistence 10 identified in the validation year 2021

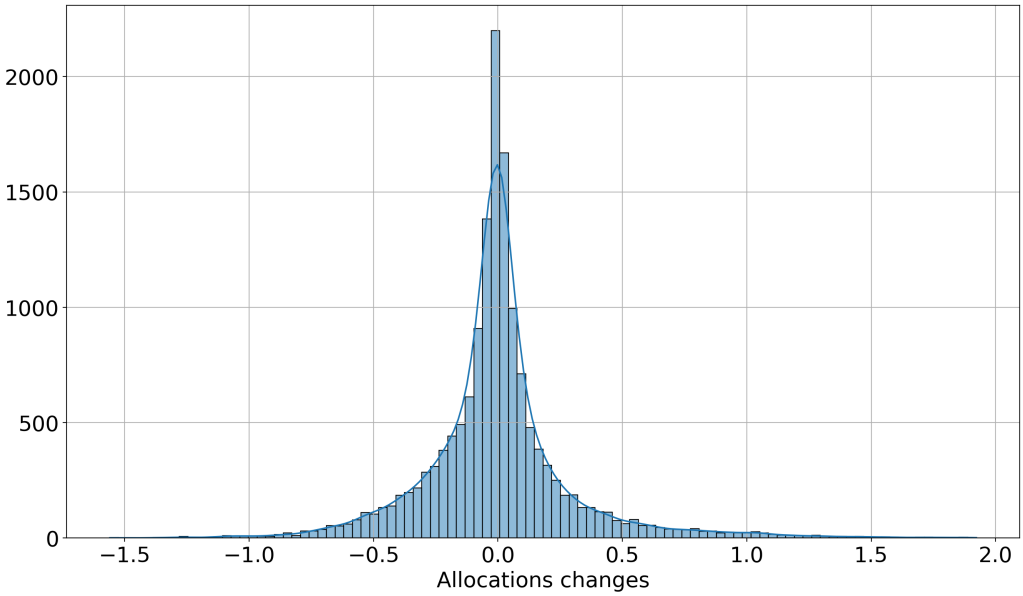
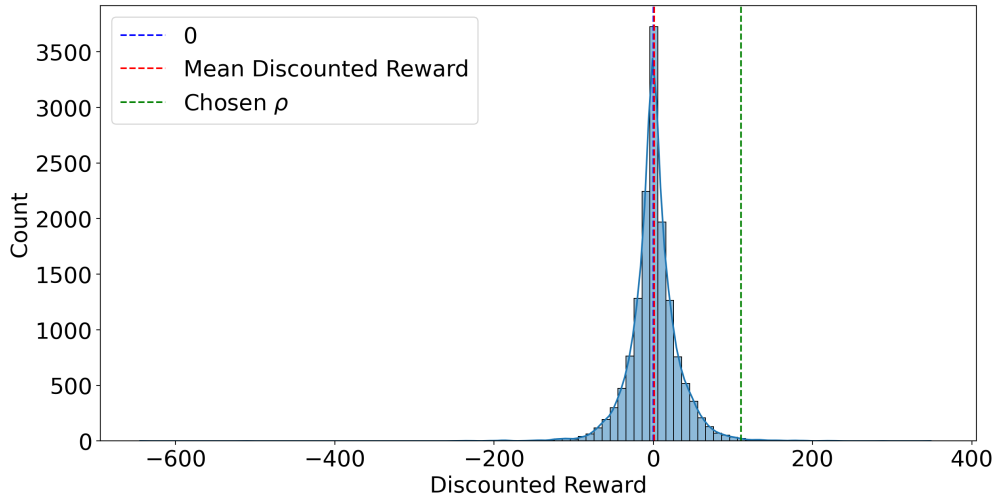


Figure 6.13: Histogram of the allocations changes in the test year 2022, with the estimated distribution using a kernel density estimation

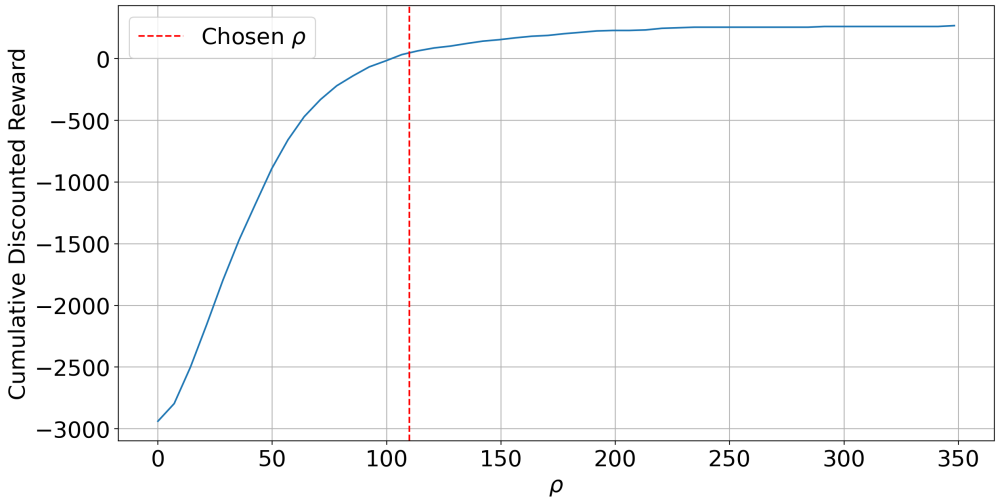
RCVaR Objective Setting

To optimize the RCVaR objective (Equation 5.10), we use the reward transformation described in Equation 5.12. In particular, treating ρ as a hyperparameter and taking into account the computational resources available, we select a few values of ρ for the learning process of our models.

In order to determine a suitable range of values for the hyperparameter ρ that can induce a risk-averse behaviour, we initially analyzed the distribution of discounted rewards (Figure 6.14a) and the cumulated distribution function of the risk-neutral setting (Figure 6.14b). Afterwards, we utilized the RCVaR estimator introduced in [58] to identify a value a proper value of RVaR (i.e., our hyperparameter ρ) in the risk-neutral models. Using this procedure on the results obtained from the risk-neutral model in the validation year 2021, we have found that an average empirical value for induce a risk-averse behaviour could be around the threshold $\rho = 110$. Finally, to test for different levels of risk aversion, we selected two more ρ values to generate higher risk-aversion behaviour in our models.



(a) Distribution of the discounted reward.



(b) Cumulative Distribution of the discounted rewards.

Figure 6.14: Examples of Distribution and Cumulative Distribution of the discounted rewards obtained by one of the previous risk-neutral models in the validation year 2021.

In the following, we present a summary of the performances of the obtained risk-averse models in terms of P&L (although maximizing the cumulative reward is not the main objective of such models), the Sharpe Ratio and the expected return obtained using the transformed rewards (Equation 5.12).

ρ	% P&L (Mean ± Std)			Sharpe Ratio	J
	2018-2019	2020	2021	2021	2021
50	16.98 ± 2.32	2.58 ± 0.20	4.41 ± 0.27	2.67	0.19
80	24.46 ± 4.65	3.02 ± 0.04	8.88 ± 0.93	3.19	0.41
110	37.73 ± 4.16	3.49 ± 0.24	10.71 ± 0.04	2.83	0.52

Table 6.5: Performance obtained with 3 different seeds using the best model obtained that maximizes the RCVaR in the validation years 2021, measured by the cumulative percentage P&L (mean ± std), Sharpe Ratio and the expected return with the transformed rewards.

ρ	% P&L (Mean ± Std)	Sharpe Ratio	J
50	1.93 ± 0.32	0.52	-0.49
80	4.25 ± 1.90	0.86	-0.89
110	6.09 ± 2.32	0.85	-0.89

Table 6.6: Performance obtained in the test year 2022 using the models obtained in the validation year 2021, measured by the cumulative percentage P&L (mean ± std), the Sharpe Ratio and the expected return with the transformed rewards.

In general, as we can see in Tables 6.5 and 6.6, lower values of ρ lead to producing more conservative policies: in fact, we can notice that lower values of ρ bring a deterioration in the P&L performance. The reason behind this is simply because ρ controls how many risky rewards we want to discard in the training process, therefore regulating the risk-aversion level of the trained model. Moreover, the performance obtained in the test year 2022 (Figure 6.15) does not fully reflect the performance obtained in the validation year 2021, suffering the usual losses already seen in previous settings.

To get a better understanding of the learning process of risk-averse models, it is beneficial to observe the curve of the expected return with the transformed rewards calculated during the training iterations. Additionally, it is interesting to observe the percentage of rewards $\widehat{\alpha}$ obtained by the agent that are lower than the selected ρ used to train the model. More specifically, $\widehat{\alpha}$ is a measure for understanding how much the model has learned to perform actions able of obtaining rewards lower than the ρ risk hyperparameter.

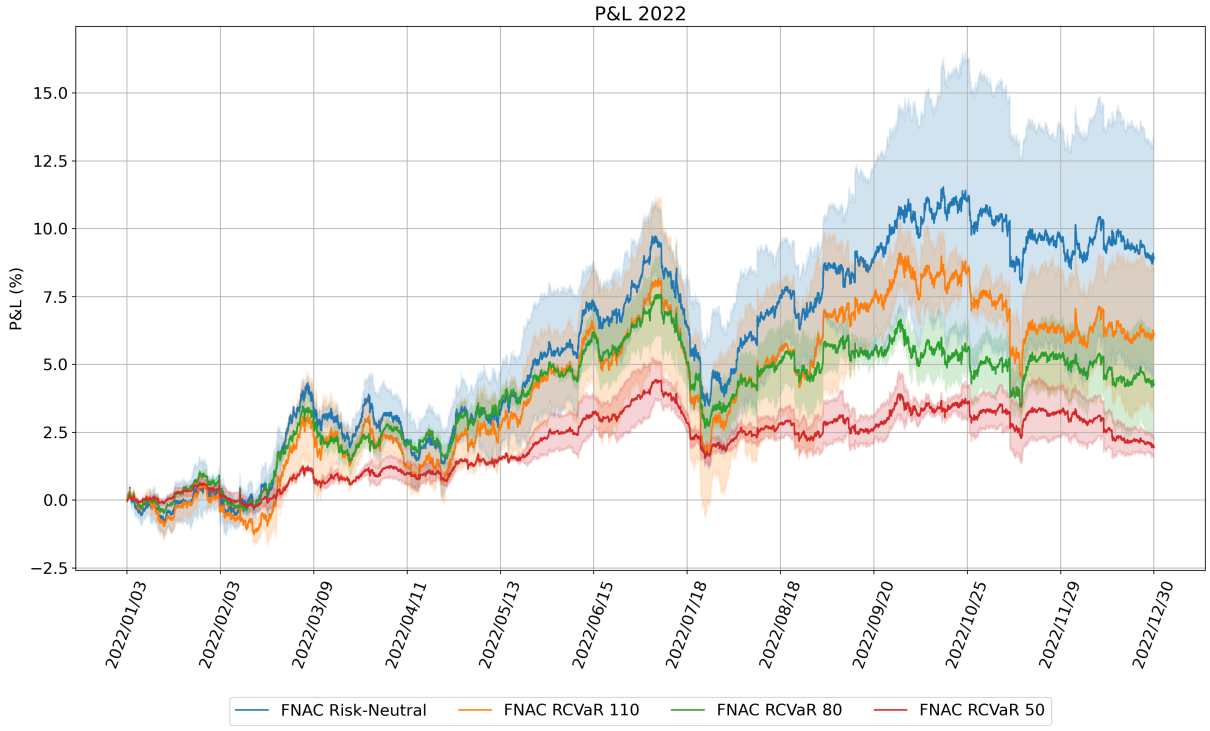
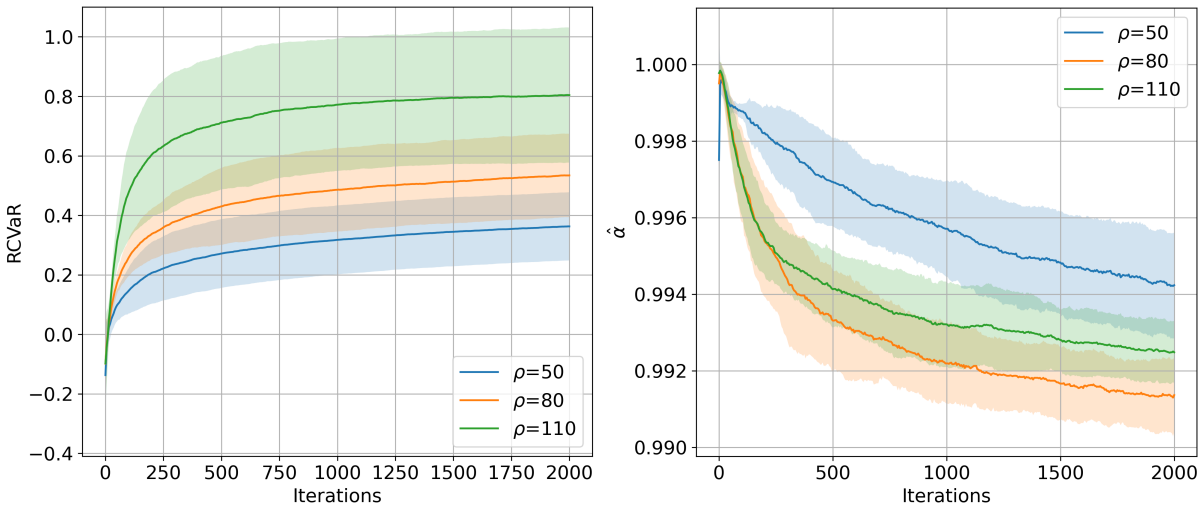


Figure 6.15: Plot of the percentage P&L in the test year 2022 using the models trained with the RCVaR objective.



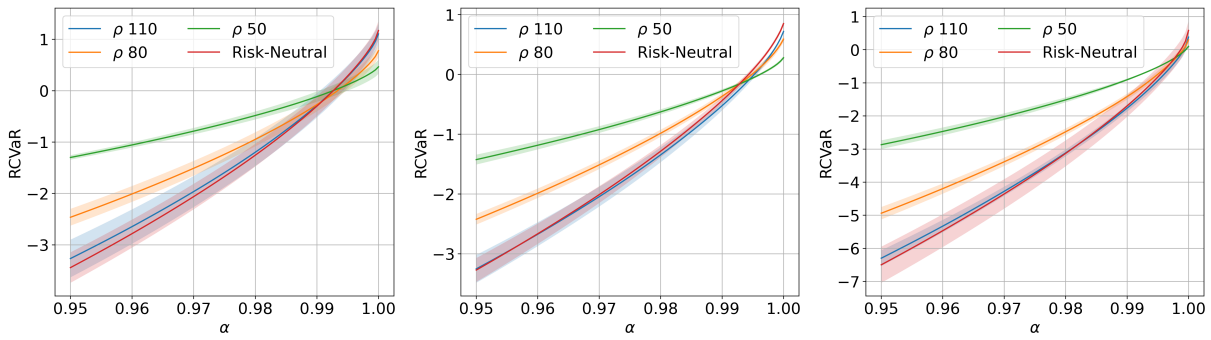
(a) J learning curve.

(b) $\widehat{\alpha}$ estimated through the learning process.

Figure 6.16: Expected return with the transformed rewards and $\hat{\alpha}$ during the learning process in the train years 2018-2019.

As we could expect, we can see in Figure 6.16 that as the learning iterations proceed, the model maximizes the expected return of the transformed rewards. Similarly, the level

of $\widehat{\alpha}$ tends to converge, indicating that the model has learned to select better actions to obtain a lower reward than ρ.



(a) Training years 2018-2019 (b) Validation year 2021 (c) Test year 2022

Figure 6.17: Estimation of the RCVaR using the estimator in [58] using different values of α.

The plots in Figure 6.17 show the estimate of the RCVaR using the estimator in [58] through different values of α. From the obtained RCVaR values, we can conclude the approximation used is not very effective in optimizing the RCVaR measure in the tested α range. In fact, the curves of the various policies obtained are able to obtain positive RCVaR values only in some cases and for a very narrow final range of α, crossing each other only for very low RCVaR values, meaning that they are probably dominated by a flat policy. This is probably also due to the fact that the fully convergence of the objective function has not yet been achieved. In any case, by observing the trend of the various curves, it is possible to notice how models with a lower ρ initially dominate the models with higher ρ, to then reverse the trend for higher α values. This indicates that the optimization imposed by ρ, it is still effective for obtaining models with different levels of risk aversion.

To conclude, we present Table 6.7, which shows the summary statistics of the reward distribution obtained by selecting the best model for each value of ρ in the test year 2022. From these statistics, we can observe that, as the risk aversion parameter ρ decreases, the models tend to take actions that produce rewards with a lower standard deviation, indicating that the agent follows a policy that leads a more conservative behaviour.

Mean-Volatility Objective Setting

In this final setting, we exploit the concept of the Mean-Volatility with the optimization procedure described in Section 5.4.2.

Model	Rewards			
	Mean	Std	Min	Max
$\rho = 50$	0.14	25.40	-406.86	377.40
$\rho = 80$	0.40	41.99	-965.10	706.02
$\rho = 110$	0.54	56.78	-1263.35	850.37
Risk-Neutral	0.85	55.66	-1140.85	1134.21

Table 6.7: Statistics of the rewards of models trained by the RCVaR objective function and of risk-neutral models that obtain the maximum P&L in the test year 2022.

λ	% P&L (Mean ± Std)			Sharpe Ratio	MV	RV
	2018-2019	2020	2021	2021	2021	2021
10^{-3}	21.70 ± 5.90	3.71 ± 2.08	4.76 ± 0.38	2.79	0.09	0.00015
$5 \cdot 10^{-4}$	32.40 ± 7.54	4.55 ± 2.35	9.28 ± 0.40	3.05	0.29	0.00025
10^{-4}	41.90 ± 11.14	7.11 ± 3.19	11.19 ± 1.64	2.74	0.63	0.00033

Table 6.8: Performance obtained employing the Mean-Volatility (MV) objective function with different λ hyperparameters, with the square root of the Reward Volatility (scaled using maximum order amount), using the best models obtained in the validation year 2021.

As previously explained in Section 2.5.1, through this approach it is possible to apply a trade-off between the expected return and the reward volatility, using a hyperparameter λ that identifies the risk-aversion level of the model. To determine a suitable range of values for the hyperparameter λ, we first study the results obtained from the risk-neutral models. This analysis provides us with a range of values for λ around 10^{-3}, potentially able to induce a risk-averse behaviour.

From Table 6.8, we can see how λ affects the learning process of the model: for higher values of lambda, a worse P&L is obtained. However, this is due to the fact that the model learns a more conservative behaviour, obtaining in this way a lower reward-volatility. Instead, for smaller values of λ, the model seems to maintain a lower risk approach, thus maximizing the final P&L but increasing the reward-volatility.

By looking instead at Table 6.9, we can see that excessively high λ values adversely affect the P&L and the Sharpe Ratio obtained by the model in the test year 2022. In addition, the models with the lowest λ suffer from a marked variability in the P&L, caused by particularly negative performances of one of the models trained by the different initializations. As before, we can notice the performance drop present at the end of 2022 (Figure 6.18), which causes the model to fail to fully handle the non-stationarity of the data.

λ	P&L (%) (Mean ± Std)	Sharpe Ratio	MV	RV
10^{-3}	-0.62 ± 2.97	-0.12	-0.04	0.00031
$5 \cdot 10^{-4}$	4.25 ± 0.97	0.75	-0.85	0.00047
10^{-4}	4.35 ± 4.30	0.60	-0.06	0.00059

Table 6.9: Performance obtained in the test year 2022 employing the Mean-Volatility (MV) objective function with different λ hyperparameters, with the square root of the Reward Volatility (scaled using maximum order amount), using the best models obtained in the validation year 2021.

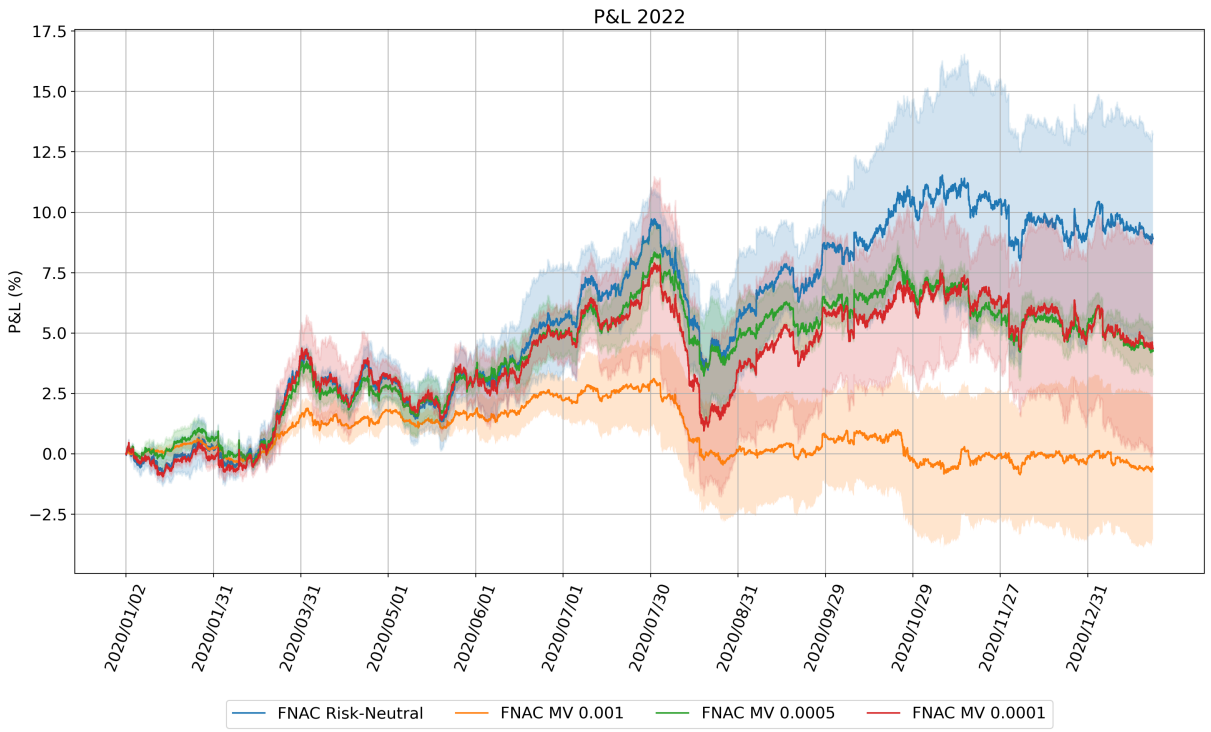
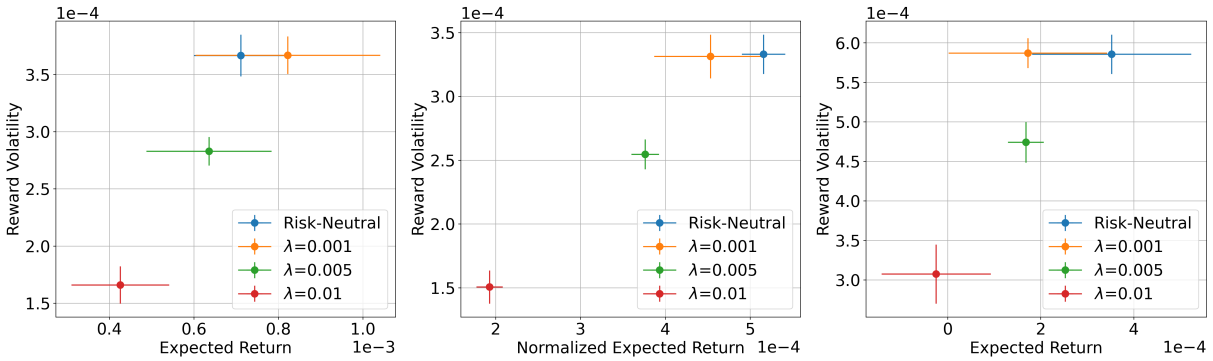


Figure 6.18: Plot of the percentage P&L in the test year 2022 using the models trained with the Mean-Volatility objective.

Figure 6.19 shows the frontier obtained by comparing the normalized expected return and the reward volatility using the previous models and the risk-neutral model. As we can see in the plots relating to the years of training and validation, by varying the λ value it is possible to effectively control the level of risk of the model. In the case of the test year, the previous consideration fails, this is mainly due to the value of λ=0.001 which causes a worsening of the expected reward obtained. However, the standard deviation of the expected return and the higher reward volatility of the same λ, still indicates how the



(a) Training Years 2018-2019. (b) Validation Year 2021. (c) Test Year 2022.

Figure 6.19: Scatter Plots of the frontier between the normalized expected return and the reward volatility, representing the mean and standard deviation of each measurement.

level of risk imposed can be effective for controlling the behaviour of the policy.

Finally, as similarly done for the RCVaR, we present a summary (Table 6.10) regarding the statistics of the reward distribution generated by the best models obtained in the test year 2022 using the Mean-Volatility objective. We compare these results with the best risk-neutral model obtained. Through these statistics, we can see that by increasing the risk-aversion parameter λ, the standard variation of the rewards tends to diminish, causing the model to acquire a more conservative behaviour.

Model	**Rewards**			
	Mean	**Std**	**Min**	**Max**
$\lambda = 10^{-3}$	0.16	25.52	-522.98	403.96
$\lambda = 5 \cdot 10^{-4}$	0.36	44.13	-649.34	1078.40
$\lambda = 10^{-4}$	0.62	60.49	-1358.12	1026.92
Risk-Neutral	0.85	55.66	-1140.85	1134.21

Table 6.10: Statistics of the rewards of models trained by the Mean-Volatility objective function and of risk-neutral models in the test year 2022.

7 | Conclusions

Reinforcement learning provides a significant opportunity to develop decision-making agents, capable of learning patterns in market trends that are often imperceptible to the human eye. The Forex market, due to its enormous liquidity, is one of the ideal places to apply automated trading techniques. Unfortunately, it also presents great challenges due to its non-stationary nature and volatile market data.

In this work, we modelled the Forex environment using the last 45 mid-price differences and the spread value, using the latter to determine the transaction costs for each executed order. For our formulation, we use the market data regarding the EUR/USD currencies pair between 2018 and 2022. We implemented the Fitted Natural Actor-Critic algorithm, exploiting its characteristic for creating an agent that supports the continuous action space. Furthermore, in order to obtain a better signal-to-noise ratio, we modify the control frequency using the concept of action-persistence. Through the use of continuous actions, we then modified the reward, making transaction costs depend on the volume of the order executed by the agent. Lastly, in order to obtain a risk-averse agent, we integrate into the learning process the RCVaR and Mean-Volatility risk-averse objective.

We first tested FNAC using a discrete action space with different persistence values. The results showed how persistence helps the generalization capabilities of the model: indeed, while obtaining positive performances in the validation year 2021, models with lower persistence performed much worse in the test year 2022. In this discrete setting, the model trained with the value of persistence 10 outperforms the baselines and the FQI algorithm trained in the same setting. Unfortunately, especially after the second half of the year 2022, the performance shows a significant variability.

In a second instance, we started to exploit the continuous action space settings, focusing on the models trained with a frequency equal to 10 minutes per trade. As we expected, due to the difficulty to handle a continuous actions space, we generally obtained slightly worse performances, especially during the test year 2022. Precisely, in the last 4 months of 2022, we noticed a significant rise in the variability and a drop in the cumulative return.

As a further contribution, we took leverage of the continuous-action setting to adopt

a more realistic, non-linear transaction cost function. In a similar fashion as before, the performances obtained in 2022 are subject to the trends observed in the base setting, resulting in some worse performances due to higher transaction costs. However, observing the distribution of the changed allocation between two subsequent steps, the model seems to have learned how to manage the amount of orders to minimize transaction costs.

Finally, we integrated risk-aversion in the form of the RCVaR and the Mean-Volatility objective functions to promote more careful behaviours in the trained agents. While obtaining worse results in the test year 2022 than in the validation year 2021, through both risk measures we managed to induce a more conservative behaviour in the model training process, able to minimize the reward variance obtained by the risk-averse policies.

A common aspect among all formulations that use continuous actions, is the drop in performance and higher standard deviation in the final months of the test year 2022. This seems to be caused mainly by two factors:

- Due to the need for a differentiable policy and the consequent use of an FFNN for its parametrization, the initialization of the weights seems to play an important role in achieving an optimal policy. In fact, while trying to obtain a stable initialization of the weights using the He methodology [60], initializations generated by different independent instances cause the creation of policies that obtain very different performances.
- The non-stationarity of market data makes the generalization capability a paramount factor for obtaining positive and stable performance. Market periods with different trends from those used in the training process in fact perform worse. Indeed the last 4 months of the year 2022 seem to have a remarkably different trend from the years used to train the models. As can be seen from the mid-price trend in Figure 5.1, the end of the year 2022 seems to have a marked upward trend (called bullish trend). Instead, the years 2018-2019 used for train seem to have more a downward trend (called bearish trend). The use of these years for the training process has probably led to having an agent able to better manage rising market trends.

7.1. Future Developments

Our work, while obtaining positive performances using both discrete and continuous actions, shows multiple needs for further study to improve and obtain more stable performances due to the non-stationarity of market data.

A possible approach to limit the impact of non-stationarity is to use a multi-expert learn-

ing algorithm, ad described in [61]. More precisely, a set of agents trained with multiple market settings and different levels of complexity are weighted by an online learning algorithm using the performance obtained the previous day. Through this approach it is possible to handle the market regime drifts in a better way, taking into account the experience of the different models used.

Moreover, as described in Section 5.3, FNAC uses three main components, where the first critic is in charge of approximating the value function, the second critic is used for the calculation of the natural gradient and the third for the approximation of the policy of the actor. This number of components leads the overall algorithm to have a high computational requirement and hyperparameters, making it difficult to perform rigorous hyperparameter optimization. Further research investigations might aim to optimize the training process by reducing the required time. This involves exploring techniques that can enhance the speed of both value function approximation and, more importantly, the computation of the natural gradient. Additionally, the possibility of using GPU to speed up and improve the efficiency of parallel operations could be explored.

Furthermore, models with higher persistences seem to achieve a greater capacity for generalization: an approach that could be investigated is the use of persistence values that depend directly on market conditions and the agent's predisposition to risk, allowing for greater profit opportunities in situations of low volatility.

Regarding the use of risk measures, to improve the RCVaR optimization process, the use of the meta-algorithms for policy optimization described in [15] and [58] could be evaluated. In particular, these meta-algorithms allow the use of a risk-neutral algorithm by transforming the MDP through an external loop, modifying the states of the trajectories or the reward used in the policy optimization process in order to induce a risk-averse behaviour.

Finally, the differentiable nature of the policy opens up possibilities for investigating transfer learning techniques on currencies other than EUR/USD. This is especially relevant for currencies with low liquidity, where finding successful strategies can be more challenging.

Bibliography

[1] Otc foreign exchange turnover in april 2022. `https://www.bis.org/statistics/rpfx22_fx.htm`. Accessed: 2023-06-01.

[2] Francisco S Melo and Manuel Lopes. Fitted natural actor-critic: A new algorithm for continuous state-action mdps. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II 19*, pages 66–81. Springer, 2008.

[3] Antonio Riva, Lorenzo Bisi, Pierre Liotet, Luca Sabbioni, Edoardo Vittori, Marco Pinciroli, Michele Trapletti, and Marcello Restelli. Learning fx trading strategies with fqi and persistent actions. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9, 2021.

[4] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[5] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 2005.

[6] Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine learning*, 49:161–178, 2002.

[7] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[10] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[11] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[12] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[13] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32, 1992.

[14] Matteo Papini. Safe policy optimization. 2021.

[15] Lorenzo Bisi. Algorithms for risk-averse reinforcement learning. 2022.

[16] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

[17] Dotan Di Castro, Aviv Tamar, and Shie Mannor. Policy gradients with variance related risk criteria. *arXiv preprint arXiv:1206.6404*, 2012.

[18] Matthew J Sobel. The variance of discounted markov decision processes. *Journal of Applied Probability*, 19(4):794–802, 1982.

[19] Harry M Markowitz and G Peter Todd. *Mean-variance analysis in portfolio choice and capital markets*, volume 66. John Wiley & Sons, 2000.

[20] R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[21] Lorenzo Bisi, Luca Sabbioni, Edoardo Vittori, Matteo Papini, and Marcello Restelli. Risk-averse trust region optimization for reward-volatility reduction. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 4583–4589, 2021.

[22] Aviv Tamar, Dotan Di Castro, and Shie Mannor. Policy gradients with variance related risk criteria. In *Proceedings of the twenty-ninth international conference on machine learning*, pages 387–396, 2012.

[23] Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the cvar via sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[24] Stephen J Wright. Coordinate descent algorithms. *Mathematical programming*, 151 (1):3–34, 2015.

[25] Corentin Tallec, Léonard Blier, and Yann Ollivier. Making deep q-learning methods

robust to time discretization. In *International Conference on Machine Learning*, pages 6096–6104. PMLR, 2019.

[26] Luca Sabbioni, Luca Al Daire, Lorenzo Bisi, Alberto Maria Metelli, and Marcello Restelli. Simultaneously updating all persistence values in reinforcement learning. 37:9668–9676, Jun. 2023. doi: 10.1609/aaai.v37i8.26156.

[27] Alberto Maria Metelli, Flavio Mazzolini, Lorenzo Bisi, Luca Sabbioni, and Marcello Restelli. Control frequency adaptation via action persistence in batch reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119, pages 6862–6873. PMLR, 2020. URL `http://proceedings.mlr.press/v119/metelli20a.html`.

[28] Reinforcement learning: An introduction. `http://incompleteideas.net/book/ebook/the-book.html`. Accessed: 2023-06-01.

[29] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[30] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[31] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[32] Yann Ollivier, Ludovic Arnold, Anne Auger, and Nikolaus Hansen. Information-geometric optimization algorithms: A unifying picture via invariance principles. *The Journal of Machine Learning Research*, 18(1):564–628, 2017.

[33] Justin A Boyan. Least-squares temporal difference learning. In *ICML*, pages 49–56, 1999.

[34] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[35] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[36] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4):875–889, 2001.

[37] Carl Gold. Fx trading via recurrent reinforcement learning. In *2003 IEEE Inter-*

national Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings., pages 363–370. IEEE, 2003.

[38] Michael AH Dempster, Tom W Payne, Yazann Romahi, and Giles WP Thompson. Computational learning techniques for intraday fx trading using popular technical indicators. *IEEE Transactions on neural networks*, 12(4):744–754, 2001.

[39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[40] João Carapuço, Rui Neves, and Nuno Horta. Reinforcement learning applied to forex trading. *Applied Soft Computing*, 73:783–794, 2018.

[41] Chien Yi Huang. Financial trading as a game: A deep reinforcement learning approach. *arXiv preprint arXiv:1807.02787*, 2018.

[42] Sutta Sornmayura. Robust forex trading with deep q network (dqn). *ABAC Journal*, 39(1), 2019.

[43] Antonio Briola, Jeremy Turiel, Riccardo Marcaccioli, and Tomaso Aste. Deep reinforcement learning for active high frequency trading. *arXiv preprint arXiv:2101.07107*, 2021.

[44] Lorenzo Bisi, Pierre Liotet, Luca Sabbioni, Gianmarco Reho, Nico Montali, Marcello Restelli, and Cristiana Corno. Foreign exchange trading: A risk-averse batch reinforcement learning approach. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

[45] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[46] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63:3–42, 2006.

[47] Xiao-Yang Liu, Zhuoran Xiong, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522*, 2018.

[48] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[49] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[50] Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. Deep reinforcement learning for automated stock trading: An ensemble strategy. In *Proceedings of the first ACM international conference on AI in finance*, pages 1–8, 2020.

[51] Mark Kritzman and Yuanzhen Li. Skulls, financial turbulence, and risk management. *Financial Analysts Journal*, 66(5):30–41, 2010.

[52] Histdata.com. `https://www.histdata.com/`. Accessed: 2023-06-02.

[53] Antonio Riva. Multi-asset fx trading with fqi and action persistence. 2021.

[54] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[56] John Burkardt. The truncated normal distribution. *Department of Scientific Computing Website, Florida State University*, 1:35, 2014.

[57] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

[58] Massimiliano Bonetti, Lorenzo Bisi, and Marcello Restelli. Risk-averse optimization of reward-based coherent risk measures. *Artificial Intelligence*, page 103845, 2023.

[59] Nicole Bäuerle and Jonathan Ott. Markov decision processes with average-value-at-risk criteria. *Mathematical Methods of Operations Research*, 74:361–379, 2011.

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[61] Antonio Riva, Lorenzo Bisi, Pierre Liotet, Luca Sabbioni, Edoardo Vittori, Marco Pinciroli, Michele Trapletti, and Marcello Restelli. Addressing non-stationarity in fx

trading with online model selection of offline rl experts. In *Proceedings of the Third ACM International Conference on AI in Finance*, pages 394–402, 2022.

List of Figures

List of Tables