eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

FACULTY OF COMPUTER SCIENCE
UNIVERSITY OF THE BASQUE COUNTRY

Master Thesis

Computational Engineering and Intelligent Systems Master's Degree

Approach to Financial Portfolio Management via Deep Reinforcement Learning with DDPG Algorithm

Mikel Salvoch Vilches

Advisors
Josu Ceberio Uribe
Juan Diego Rodríguez Fernández

July 3, 2025

Abstract

This work addresses the portfolio management problem using Deep Reinforcement Learning, specifically the Deep Deterministic Policy Gradient (DDPG) algorithm, applied to a set of assets traded on the New York Stock Exchange (NYSE). The portfolio management problem consists of dynamically reallocating capital across a set of stocks with the goal of maximizing returns while minimizing risk. In order to achieve the objective, first, the problem is formalized as a Markov Decision Process, defining appropriate state and action spaces together with reward functions aligned with financial objectives. Afterward, an extensive analysis of algorithm configurations, hyperparameters, and performance evaluation is conducted. Then, the results contribute to the understanding of how Reinforcement Learning techniques can be effectively applied to dynamic portfolio optimization problems. Particularly, the results show that the model is able to outperform the proposed benchmark, although the gains are not entirely stable and depend on the specific configuration. To further improve the approach, we propose future work directions to enhance model robustness and performance. The work has been carried out jointly between *Euskal Herriko Unibertsitatea* (UPV/EHU) and Interstock Development. In this context, Interstock Development has contributed its know-how for addressing the problem while UPV/EHU has provided the theoretical insights.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, there are more and more fields of society where innovation and technology are gaining importance. In this context, and following the current trend, new ways of operating are also appearing in the financial sector. Within the broad scope of finance, in this work we focus on investment portfolios. An investment portfolio is a collection of financial assets that an individual or entity owns. The assets can be of different nature but in this case we are considering company shares, which can also be referred as stocks, therefore throughout this document we will use both terms interchangeably.

In order to understand what shares are, the National Securities Market Commission (CNMV) has many publications that explain in a simple but precise way various economic concepts on which we rely [1]. *Shares* are participations in companies that are sold on the *stock market*. By issuing shares, companies obtain funds to finance their business, and investors become shareholders, owning a proportionate part of the company. Additionally, shares have a return, which can be in the form of dividends paid by the company to shareholders, or capital gains and losses generated by the evolution of the market price. The latter do not become effective until the sale.

This buying-selling activity is formalized in what is known as the stock market or stock exchange. When companies offer their shares to the public through an Initial Public Offering, this is known as the *primary market*. Later, in the *secondary market*, shareholders can sell their shares to other investors, without the issuing companies receiving any funds. Most of the trading takes place in this market. In this case we focus on the secondary market and particularly on the New York Stock Exchange (NYSE), the world's largest and most valuable stock exchange, located on Wall Street. It handles billions of dollars in daily transactions and hosts many of the world's leading companies. As of February 2025, and at the time of this writing, the NYSE is ranked as the largest market capitalization stock market. In figures, this translates to 31,664,649.32 million USD for the specified month [2].

The assets that compose a portfolio have an associated risk, the uncertainty (or lack) of a share's return is, for example, a source of risk. Another element of risk is liquidity, which translates into the time horizon in which an asset can be sold to recover liquid money. Later on, we discuss with more detail about risk sources and how to quantify them. Basically, portfolio management consists on periodically reallocating the available capital among the assets in the portfolio. Nonetheless, given the existence of risk, the challenge of portfolio management lies in how to reallocate it to maximize returns while minimizing risk.

We have mentioned that there are new ways of operating in these financial environments, in particular, in this work we focus on Artificial Intelligence based solutions by proposing a new approach. Decision-making algorithms and specially Reinforcement Learning (RL) are valuable tools in order to solve dynamic problems, such as portfolio management. RL offers a natural framework for problems in which an agent must make sequential decisions under uncertainty [3]. In the context of portfolio management, actions taken at each step affect the evolution of the environment, and rewards are often delayed and noisy. These characteristics match well the structure of RL, where agents interact with the environment and optimize a policy based on feedback received. Moreover, RL allows for the direct optimization of financial objectives, such as return or risk-adjusted metrics, without the need for labeled data.

In order to apply RL to portfolio management, we first model the problem as a Markov Decision Process. Moreover, in this work we propose the use of the Deep Deterministic Policy Gradient (DDPG) algorithm. DDPG is an off-policy actor-critic algorithm designed for continuous action spaces. It was introduced by Lillicrap et al. [4] and combines the deterministic policy gradient with experience replay introduced in DQN.

The contribution of the present work lies in the approximation of the portfolio management problem to an RL setting using DDPG. This contribution implies key aspects such as problem formalization, definition of state and actions spaces as well as appropriate reward functions, hyperparameter tuning or performance evaluation. For all this, an exhaustive analysis of the impact and suitability of each configuration is performed. In conclusion, the aim of this work is to propose an approach to portfolio management via Deep Reinforcement Learning, and, in particular, using DDPG algorithm.

Finally, it is important to note that this work is developed together with Interstock Development, a company specialized in the development of AI solutions for financial markets, which is the one who poses the problem from a finance perspective. Therefore, the work is carried out jointly and in coordination between university and company.

The remainder of the document is organized as follows: in Chapter 2, a background on Deep Neural Networks, RL and more precisely RL applied to finance is introduced. Next in Chapter 3, the approach is formalized by introducing the problem, the algorithm to be used and, finally, the designed learning scenario. Afterward, in Chapter 4, the experimental study is presented by explaining the experimental context and posing questions to be addressed through experimentation. The results of the experimentation are shown in Chapter 5. Finally, conclusions and future work lines are discussed to conclude the document in Chapter 6.

Chapter 2

Background

2.1 Reinforcement Learning

Formally, a Markov Decision Process (MDP) is formed by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ denotes the space of states, $\mathcal{A}$ is the action space and $\mathcal{A}(s)$ determines the possible actions to take in state S. $P(S' \mid S, A)$ represents the transition probability from state S to state S' after taking action A under policy π. Finally, R corresponds to the reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ providing a scalar feedback and γ controls the importance of immediate rewards with respect future ones ($\gamma \in [0, 1)$). In addition to the above, there is also a time step t for which the agent takes actions. Moreover, the main property of MDPs lies in the assumption that given the present, the future is independent of the past and the current state is sufficient statistics for the next step.

For illustration, consider an MDP with two states, S_0 and S_1. Figure 2.1 depicts an MDP where we define some rewards and transition probabilities for these two states.

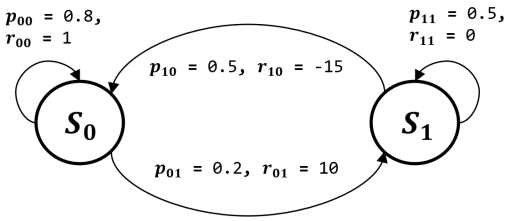


Figure 2.1: Graphic representation of an MDP where r_{ij} denotes the reward associated to transitioning to state S_j under state S_i. Similarly, the probability of transitioning to state S_j under state S_i is denoted as p_{ij}.

RL is a machine learning paradigm that is defined under an MDP. Specifically, we are dealing with an MDP where the transition probabilities and rewards are unknown. The environment is unknown and the agent has no prior knowledge of the outcomes of its actions [5]. Thus, the agent learns as it interacts with the environment, being this the basis of RL.

Within RL, there are several distinct approaches, each with its strengths and weaknesses. The main categories being: value based methods (critic only), policy based methods (actor

only), and actor-critic methods. Value-based methods estimate the Q-value function, which represents the expected cumulative reward of taking a specific action in a given state and following the optimal policy thereafter. This function is then used to guide the decision-making process by choosing the action with the highest value, i.e., defining the policy. These methods typically involve updating a value function through techniques such as temporal difference learning, where the critic evaluates the quality of actions taken by the agent based on past experiences. On the other hand, policy based methods focus directly on optimizing the policy itself, typically by parameterizing it and applying policy gradient techniques. This approach avoids the need to explicitly compute value functions but often suffers from high variance [6].

In a natural way, actor-critic methods are born, which try to combine the strengths of both of the above methods. These actor-critic methods use two components: the actor, which is responsible for deciding which action to take based on the policy, and the critic, which evaluates the action taken by estimating its quality. The critic provides feedback to the actor, helping to update the policy properly. This combination leads to more stable learning and faster convergence compared to only policy based or value based methods [6].

For the sake of clarity, we focus on value-based methods, since this is where the first Deep Reinforcement Learning model was born. Classical RL approaches such as Q-learning [7] or SARSA [8] belong to value-based group. These algorithms store the expected utility of taking an action in a given state, i.e., a value for each (state, action) pair in what is known as Q-table. This value is denoted as Q-value and is given by the Bellman equation [9]

$$Q^{\pi}(S, A) = \mathbb{E}_{\pi}\left[r_{t+1} + \gamma\, Q^{\pi}(S_{t+1}, A_{t+1})\right]$$

and in the optimal case, when the agent follows the optimal policy π^* from policy space Π, the function $Q^*(S, A)$ is defined as

$$Q^*(S, A) = \max_{\pi \in \Pi} Q^{\pi}(S, A)$$

hence the optimal policy is given by

$$\pi^*(S) = \arg\max_{A \in \mathcal{A}} Q^*(S, A)$$

If one looks closely at these expressions, the term P from the MDP tuple modeling the transition probability is not present. This is because we are considering a model-free scenario in contrast to the most primitive scenario presented in Figure 2.1. In the model-free context, the agent has access to neither the transition probabilities nor the associated rewards. It learns directly from interacting with the environment. This simplification is intentional, as the problem we address in this work fits naturally into the model-free framework, as will be seen later.

Continuing with these model-free algorithms, it is worth highlighting a key distinction in the way they update their Q-values. The Q-values are updated continuously to incorporate new information, allowing the agent to refine its estimate of the expected return for each state-action pair. While the Q-learning method updates its Q-table using the best possible action in the next state, i.e., by taking the maximum over all estimated future Q-values

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[r_{t+1} + \gamma \max_{A \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

SARSA updates this table using the action that actually is taken, i.e., following the policy applied

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[r_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

In particular, Q-learning is an example of off-policy learning, as it updates its estimates of the Q-values using a policy that has not necessarily been followed by the agent. Conversely, SARSA is on-policy, since it learns from the actions the agent actually takes.

In 2013, DeepMind Technologies introduced the Deep Q-Network (DQN) algorithm [10], which represented a breakthrough in RL. DQN was designed for solving Atari games and was able to outperform traditional RL algorithms on a series of benchmark environments on this task. The key innovation of DQN was the use of a deep neural network to approximate the Q-values, replacing the traditional Q-table. This allowed the algorithm to handle much larger state spaces. From this point on, the term Deep Reinforcement Learning (DRL) begins to appear.

Bringing neural networks into the classical approach to RL was in line with the trend that still exists today: the widespread use of neural models. In this sense, there exist different methods that vary in their application of these network. In general, the networks are used to estimate the Q-value by means of the critic network, which approximates the expected return for each state-action pair, or the policy by means of the actor network, which outputs the probability distribution over actions. In some cases, both the actor and the critic are approximated using deep networks, leading to the development of actor-critic methods in DRL. This allows the agent to handle more complex environments, particularly those with high-dimensional or continuous state spaces, where traditional methods struggle. The most popular actor-critic DRL algorithms are DDPG introduced by Lillicrap et al. [4], A2C by Mnih et al. [11] SAC by Haarnoja et al. [12] or PPO by Schulman et al. [13] among others.

2.2 Deep Neural Networks

The most basic element of a neural network, whether deep or not, is the perceptron. A perceptron is a structure that receives an input (x) and returns an output (y). Figure 2.2 shows the classical representation of perceptrons.

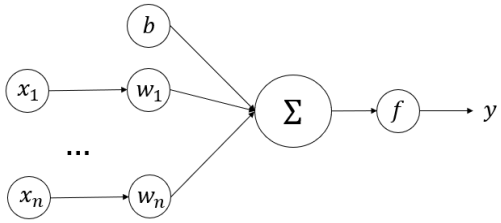


Figure 2.2: A perceptron where x is the input W and b are the weight vector and the bias term respectively, f is the activation function and it outputs y.

In this process, the perceptron applies a linear mathematical operation of the form $z = x_1 \cdot w_1 + \cdots + x_n \cdot w_n + b$, then, a function f known as activation function, is applied,

usually, to break the linearity: $y = f(z)$ [14]. Figure 2.3 shows some of the most used activation functions.

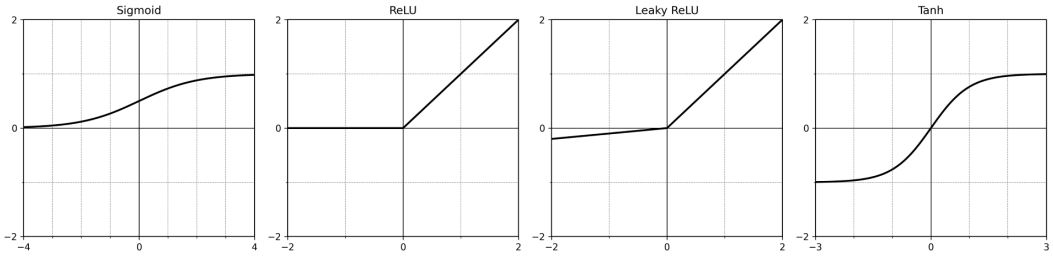


Figure 2.3: Sigmoid, ReLU, Leaky ReLU and tanh activation functions.

By concatenating different perceptrons in layers, neural networks can be constructed, as shown in Figure 2.4. Within this concatenation, the first layer is usually referred as input layer and the last one as output layer, the intermediate ones are known as the hidden layers. Actually, a neural network is a set of layers composed of some neurons at each one, so they do not have to be extremely complex structures. Therefore, the term “deep" can be slightly arbitrary, and although it seems obvious that a network of a single layer is not deep and a network of thousands of layers is deep, it is still an ambiguous term.

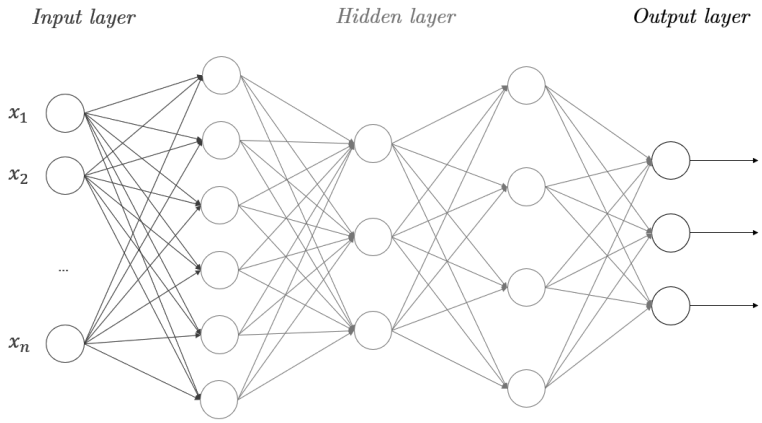


Figure 2.4: A neural network, specifically a Multi-Layer Perceptron presenting the layered structure.

With respect to neural network structures, there exist a wide variety proposals to solve problems of different nature. Starting from the base, we have the Multi-Layer Perceptrons (MLPs), here the objective is to learn a mapping between inputs x and outputs y represented by a neural network μ defined by some parameters θ^μ. This results in the expression $y = \mu(x|\theta^\mu)$. Starting from this basic structure, other structures such as Convolutional Neural Networks (CNNs) can be built. CNNs have proven to be very useful in tasks that require working with images, among others. There are other alternatives such as Recurrent Neural Networks (RNNs) where this feedback connections do exist [1].

[1] A good reference for consulting more about this topic is *Deep Learning: Foundations and Concepts* by Bishop and Bishop [15] or the previously cited *Deep Learning* book by Goodfellow et al. [16], which is considered essential reading on this subject.

Actually, the most interesting point of all these structures is that they are trainable, as data passes through the neurons, the weights W together with the bias term b are adjusted so that it is able to learn complex characteristics of the data. To adjust them, the network is shown all the data for which the outputs and their errors are calculated. W and b are then adjusted following the opposite direction of the gradient so that this difference is minimized (loss function). Generally, this process is known as gradient descent although there are different adaptations that optimize this process such as Stochastic Gradient Descent (SGD) and its variants, which in practice, are commonly preferred [17].

For the learning process, it is important to understand the backpropagation algorithm introduced by Rumelhart et al. [18], which is responsible for updating the weights of the network to learn from the training data. It works by applying the chain rule to propagate the error from the output to the previous layers and calculate the gradients of the loss function with respect to each parameter. For a network with L layers, $x^{[l-1]}$ inputs, $W^{[l]}$ weights, $z^{[l]} = W^{[l]}x^{[l-1]} + b^{[l]}$ linear output and activation $x^{[l]} = f(z^{[l]})$, the objective is to compute the gradients

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b^{[l]}}$$

It starts from the output by computing

$$\delta^{[L]} = \frac{\partial \mathcal{L}}{\partial x^{[L]}} \cdot f'(z^{[L]})$$

Intuitively, $\delta^{[l]}$ represents how sensitive the loss $\mathcal{L}$ is to changes in the linear output $z[l]$ of each neuron. It guides how the network should adjust the corresponding weights and biases to reduce the error. Then, it is back-propagated by

$$\delta^{[l]} = \left(W^{[l+1]}\right)^T \delta^{[l+1]} \cdot f'(z^{[l]})$$

so the gradients are computed as

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} = \delta^{[l]} \cdot \left(x^{[l-1]}\right)^T$$

$$\frac{\partial \mathcal{L}}{\partial b^{[l]}} = \delta^{[l]}$$

However, it is usual that in deep networks the problem of vanishing and exploding gradients appears. Sometimes, gradients can take extremely small values making it difficult for information to propagate through the network, resulting in learning problems. The same can happen in the reverse case, where it can take extreme values leading again to learning instability. There are several techniques to address these gradient problems such as changing the initialization of the weights [19], clipping the gradients [20] or introducing regularization [16].

Another problematic scenario when learning is the overfitting which implies that the model is adjusted excessively to the training data and fails to generalize to unseen data. To avoid this, it is common to apply regularization techniques in order to reduce the generalization error. The first technique is ridge regression [21], also known as L2 regularization or weight decay. It consists of adding an additional term to the loss function

$$\mathcal{L}_{\text{L2}} = \mathcal{L}_{\text{original}} + \lambda \sum_i w_i^2$$

This term added to the loss function is controlled by the λ parameter and penalizes the extreme weights in the network facilitating its generalization capacity [16].

Another form of regularization is batch normalization. This technique consists of normalizing each batch $B = \{x_1, \cdots, x_M\}$ of activations as follows. First, the mean μ_B and the variance σ_B^2 of the batch is calculated.

$$\mu_B = \frac{1}{M}\sum_{i=1}^{M} x_i$$

$$\sigma_B^2 = \frac{1}{M}\sum_{i=1}^{M}(x_i - \mu_B)^2$$

So now the normalized activations are scaled and shifted with two parameters γ and β learned by the network

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma\hat{x}_i + \beta$$

this transformation resulting on the normalized batch [22].

Along these lines, there is another popular technique known as dropout [23] which pursues overfitting avoidance by setting some inactive neurons in the network during the training process. The dropping out consists of setting some neurons to zero so it eliminates the co-adaptation problem where some neurons learn to compensate for the error of other ones.

2.3 RL in finance

As occurred in many other fields, many research works have tried to apply RL techniques in finance, and more specifically on portfolio management, the topic of study in this work. In the following, some relevant works will be revised. Focusing on existing portfolio management work using RL techniques, we mainly see the application of the previously mentioned algorithms that combine RL with neural networks, i.e., DRL algorithms. For clarity, we select what we consider to be the key characteristics of an RL scenario applied to portfolio management. These aspects are the algorithm or type of model used, the action space, the representation of the state, and the reward function. Following this reasoning, Table 2.1 summarizes the main characteristics of the selected works on portfolio management applying RL.

As far as particular methods are concerned, we can find mainly works on actor-critic algorithms such as DDPG applied to portfolio optimization in [25, 26, 28], A2C present in [26] or PPO used in works like [27, 26]. One of the most recurrent challenges when applying neural networks is designing their architecture. In this regard, Aritonang et al. [26] present a study on the impact of hidden layers in actor-critic algorithms (such as those described above) and specifically applied to portfolio management [26]. In the paper, the authors pointed out that adding more hidden layers in network architectures does not lead to better results and may lead to overfitting. However, it warns that one should be cautious in generalizing these conclusions to markets other than the one tested in the study.

Work	Year	Model Type	Action Space	State Representation	Reward Function
[24]	2017	Policy-based (DPG) with CNN/RNN/LSTM variants	Continuous (Portfolio weight vector)	$s = (X_t, w_{t-1})$ with X_t price tensor and w_{t-1} previous portfolio weights	Average of logarithmic return
[25]	2019	Actor-Critic (DDPG)	Continuous (Portfolio weight vector)	$S_t = (X_t, w_{t-1}, \hat{h}_{t+1}, I_t)$ with X_t = price tensor), w_{t-1} = previous weights, $\hat{h}_{t+1}$ = predicted returns, I_t = market index value	Log-return: $r_t = \ln(c_t \cdot u_t \cdot w_{t-1})$ with c_t transaction costs
[26]	2025	Actor-Critic (A2C, DDPG, PPO, TD3)	Discrete (buy, hold, sell)	OHLCV (open, high, low, close, volume)	Profit-based
[27]	2021	Actor-Critic (PPO)	Continuous (Portfolio weight vector)	Composition of prices, volume, quote-volume, and current portfolio weights processed per asset via GRU	Period return or Differential Sharpe Ratio
[28]	2020	GDQN (value-based, DQN) / GDPG (actor-critic, DDPG)	Discrete (buy, hold, sell)	Historical prices and technical indicators processed with GRU	Sortino Ratio (risk-adjusted return)
[29]	2016	Value-based (DQN)	Discrete $A_t \in \{-0.25, ..., 0.25\}$, representing trade ratios between two stocks	Historical prices, shares owned per stock, portfolio value, and cash	Penalized return: $\psi_t - \lambda \cdot \text{std}(\psi_t)$ and Sharpe ratio
[30]	2017	Recurrent Reinforcement Learning	Continuous (Portfolio weight vector)	$S_t = x_t = [1, \psi_t, \ldots, \psi_{t-M}, F_{t-1}]$ with $\psi_t = \log(p_t) - \log(p_{t-1})$, representing recent log-returns and F_{t-1} previous trading signal	Differential Calmar Ratio (based on Expected Maximum Drawdown)
[31]	2001	Recurrent Reinforcement Learning	Continuous, real valued trading signal $A_t \in [-1, 1]$ representing position exposure (long/short/neutral)	$S_t = x_t = [1, \psi_t, \ldots, r_{t-M}, F_{t-1}]$ with ψ_t return and F_{t-1} previous trading signal	Differential Sharpe Ratio and Differential Downside Deviation Ratio
[32]	2020	Policy-based (PG)	Continuous (Portfolio weight vector)	$s = (X_t, \hat{w}_{t-1})$ with X_t representing historical prices and $\hat{w}_{t-1}$ being the adjusted previous portfolio weights	Cost-sensitive: log return minus variance minus transaction cost
[33]	2022	Value-based (DQN)	Discrete: $-1, 0, 1$ (sell, hold, buy)	$s = (st_{p_{t-1}}, DNN(st_{p_{t+1}}))$ with $st_{p_{t-1}}$ historical prices and $DNN(st_{p_{t+1}})$ predicted future prices via GRU	Profit if trade successful and 0 (penalty) if loss
[34]	2020	Policy-based (DPG), compatible with PPO/PG	Continuous (asset reallocation vector)	$s = (s^*, \delta)$ with s^* = historical prices and δ = predicted movement	Log return penalized by transaction cost (as in [24])
[35]	2019	Policy-based (PG)	Continuous (buy/sell/hold weights)	Composition of prices, volume, sentiment, indices, and current portfolio weights processed via GRU	Differential Sharpe Ratio, Downside Deviation Ratio (DDRT)

Table 2.1: Summary of RL Portfolio Management Work

Note that, in order to apply RL, the portfolio management problem must be formulated as an MDP using the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where the key elements state space, action space, and reward function need to be formalized. To define the state space $\mathcal{S}$, the most common in portfolio management is to find a set of technical indicators with information on past market behavior. To these indicators, the current distribution of the portfolio is added, for instance, represented as a set of weights assigned to some assets. In the literature it is possible to find some implementations that include the use of Gated Recurrent Units (GRU) [36]. GRU is used to model the time dependencies in the time series nature of the stock market [33] and extract informative features of stock data [28]. Other proposals, such as [34], use extended states that incorporate both price-derived predictions and external information. They apply an LSTM to forecast asset movements from historical prices and a hierarchical attention network (HAN) to extract features from financial news [34]. These predictions are then used to augment the state, helping to address the non-stationary nature of stock price time series.

In the context of portfolio management in RL, the types of action spaces $\mathcal{A}$ that are defined can be divided in two types: discrete and continuous spaces. Thus, the choice of

action space and the particular algorithm being used are completely linked. On the one hand, algorithms such as DQN require discrete actions. A widely used discrete action space is the sell-hold-buy scheme, defined as

$$\mathcal{A} = \{(a^1, a^2, \ldots, a^N) \mid a^i \in \{-1, 0, 1\}\ \forall i \in \{1, \ldots, N\}\},$$

where each a^i denotes the discrete action taken over asset i, corresponding to sell (-1), hold (0), or buy (1).

On the other hand, in the case of continuous actions spaces, it is common to find work that focuses on directly predicting the distribution of capital to be made over the different stocks, defined by an action space

$$\mathcal{A} = \left\{ (a^1, a^2, \ldots, a^N) \in \mathbb{R}^N \;\middle|\; \sum_{i=1}^{N} a^i = 1,\ a^i \geq 0\ \forall i \in \{1, \ldots, N\} \right\}$$

where each a^i represents the proportion of capital assigned to asset i. Algorithms such as DDPG, A2C or PPO work with continuous action spaces.

In the finance world, it is easy to find something that evokes this concept of reward function: the return of the portfolio. Intuitively, a transaction in the portfolio is only as good as the money it generates. Therefore, in state-of-the-art reward functions, it is common to find this return involved in the calculations. However, it is important to consider the risk associated with achieving these returns, and there are different ways to do so. For instance, some works use the Sortino ratio [28], the Sharpe ratio with penalty terms [29], or the Calmar ratio [30]. These are classical measures that include risk and are widely used in finance. Along these lines, other approaches propose the use of the Differential Sharpe Ratio and the Sterling ratio [31], or subtract transaction costs and variance from the return to directly incorporate risk [32]. However, these ratios are sometimes used purely as evaluation metrics, and some works do not explicitly account for risk, using instead the net return or the average logarithmic cumulated return as reward [24]. We will define the most relevant ratios for this work later on in Section 3.1.

Chapter 3

Approach

To outline the approach we have taken in this project, it is useful to define three elements (see Figure 3.1). First, portfolio management problem as defined by Interstock Development, which is specific to the financial world (Section 3.1). Second, the algorithm from the literature chosen to optimize the existing problem, DDPG (Section 3.2). Finally, and where the greatest contribution lies, we present the learning scenario where we formulate the problem as a RL task and adapt it to the DDPG framework: we present the designed reward functions, the action to be taken by the agent, and the definition of the environment in which the agent finds itself (Section 3.3). Most of these studies focus on Asian stock markets, with limited attention to the NYSE.

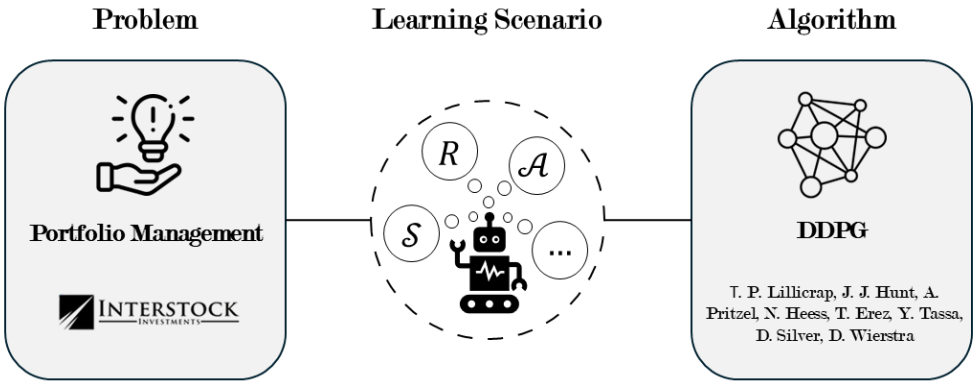


Figure 3.1: Overview of the work.

3.1 Problem

We have defined the portfolio management problem as the distribution of capital across a set of assets that must be periodically reallocated. In order to formalize the portfolio management problem, we consider the portfolio has a limited number of companies that we will denote as N. Since the capital must be distributed among these assets, the natural way to do so is through a weight vector that sums to 1. Then, we assign a vector of weights

of size N representing the proportion of money invested to each company

$$W_t = \{w_t^1, w_t^2, \ldots, w_t^N\}, \quad w_t^i \in [0,1] \quad \forall i \in \{1, \ldots, N\}$$

where t represents the time step. Moreover, $\sum_{i=1}^{N} w_t^i = 1$ is satisfied. For instance assuming a portfolio of 2 companies, the vector $W_0 = \{0.4, 0.6\}$ would represent an investment of 40% of the portfolio money in the first asset and 60% in the second one as an initial configuration ($t = 0$). Also, it would be valid because it meets the restriction of positivity and sum one. In the financial context, each asset has several values that evolve over time. Typically, we can find the acronym OHLC which comes from the words *Open, High, Low and Close*. With these four values, one can determine a complete representation of price movements for an asset in time t. These values form the well-known candlesticks of the financial charts as shown in Figures 3.2 and 3.3.

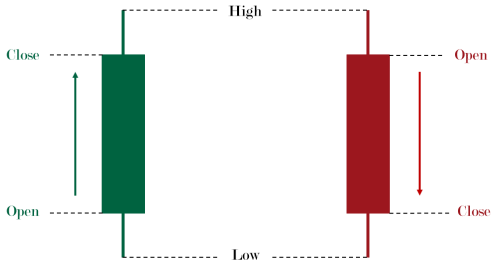


Figure 3.2: Graphical explanation of the candlestick: *Open*, *High*, *Low* and *Close* values

Figure 3.3: Market performance over six months for a given asset.

In this case, to determine the value of the portfolio, it is enough to look at the close price as it allows us to know the evolution of an asset between a time step from t to $t + 1$. Finally, there are some indicators that can be computed from these values by observing different market periods, which can help interpret trends in the market.

Since we are working on a problem of a sequential nature, there is a temporal evolution of both the portfolio and the companies it is composed of. The evolution of the portfolio is modeled by the vector W_t, as above mentioned, but also by the value of the portfolio itself. We can therefore express this value V as a function of the weights and the prices of the

assets at time t. Thus, having

$$V_t = V_{t-1} \cdot \sum_{i=1}^{N} w_t^i \cdot \frac{C_t^i}{C_{t-1}^i}$$

where C_t^i represents the closing value of the assets at time t; thus, $\frac{C_t^i}{C_{t-1}^i}$ gives the rise or fall rate for that asset.

We have also mentioned the existence of a risk associated with these assets. Quantifying risk is a natural challenge in the world of finance, fortunately, it is a well-studied field and there are metrics and strategies that we can use for this risk analysis. The basic approach to measure risk is through the volatility of the portfolio value. In particular, we can define this volatility-based risk measure using the standard deviation of the portfolio values. So given the average value of the portfolio $\bar{V}$ for T observed time steps

$$\sigma_t = \sqrt{\frac{1}{T-1} \sum_{t=1}^{T} (V_t - \bar{V})^2}$$

In fact, this is the basis of the Sharpe ratio, one of the most popular risk-adjusted metrics used in finance introduced by Sharpe [37]

$$Sharpe_t = \frac{\psi_t - \psi_f}{\sigma_t}$$

Note that ψ_f represents the risk-free rate, which in many tasks is ignored. It is subtracted to account for the return you could get from a risk-free asset, like cash or government bonds (further information on risk-free rate can be found here). ψ_t is the return of the portfolio at time t, which should not be misinterpreted as the value of the portfolio V_t, moreover, the return at time t is the relative change in the value V with respect $t-1$

$$\psi_t = \frac{V_t - V_{t-1}}{V_{t-1}} = \sum_{i=1}^{N} w_t^i \cdot \left(\frac{C_t^i}{C_{t-1}^i} - 1 \right)$$

3.2 Algorithm

Among the methods mentioned, the most common one in the works consulted, and also the one with the best results in most cases, is the DDPG. This idea is also reflected in other studies, such as [38]. The determinism of DDPG provides the precision needed for continuous decision-making in portfolio management, making it more suitable than SAC, which adds variability through stochastic policies that prioritize exploration over consistency. It is for all these reasons that we opt for DDPG in this work.

As stated earlier, the portfolio management task is modeled as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. The agent follows a deterministic policy $\mu : \mathcal{S} \rightarrow \mathcal{A}$ parameterized by θ^μ, which maps the current state S_t to an action A_t ($\mu(S_t|\theta^\mu)$). The objective is to optimize the policy by maximizing

$$J(\mu(S|\theta^\mu)) = \mathbb{E}_{S \sim \rho^\mu} \left[R(S, \mu(S|\theta^\mu) \right]$$

This objective computes the expected reward obtained by following the policy μ where ρ^{μ} denotes the distribution of visited states S under policy μ [39].

Given the continuous nature of the action space, common value-based RL methods such as Q-learning are not applicable. Instead, we employ DDPG algorithm, which is specifically designed for environments with high-dimensional continuous action spaces.

DDPG is an actor-critic algorithm where the actor network $\mu(S|\theta^{\mu})$ outputs the action to be executed in a given state, while the critic network $Q(S, A|\theta^{Q})$ estimates the expected return of taking action A in state S, under the current policy. Additionally, a target network is defined for both the actor and the critic, denoted as $\mu'(S|\theta^{\mu'})$ and $Q'(S, A|\theta^{Q'})$. The policy parameters θ^{μ} are updated through the deterministic policy gradient

$$\nabla_{\theta^{\mu}} J = \mathbb{E}_{S\sim\mathcal{R}} \left[\nabla_A Q(S, A|\theta^{Q})|_{S=S_i, A=\mu(S|\theta^{\mu})} \nabla_{\theta^{\mu}} \mu(S|\theta^{\mu})|_{S=S_i}\right]$$

where $\mathcal{R}$ refers to the replay buffer used to store past transitions.

Actor and Critic are defined as two neural networks. The first one is in charge of learning the policy that determines what action is to be taken in a given state. The second one, on the other hand, is in charge of measuring the quality of the policy that the actor is following during this learning process. The critic network is approximated via temporal difference learning, while the actor is updated through the deterministic policy gradient using the critic's feedback. To stabilize training, DDPG employs both a replay buffer, which breaks temporal correlations by using random experiences during training, and soft target updates, which slowly track the learned networks. This latter technique helps prevent target from rapidly changing value estimates, improving convergence stability [40].

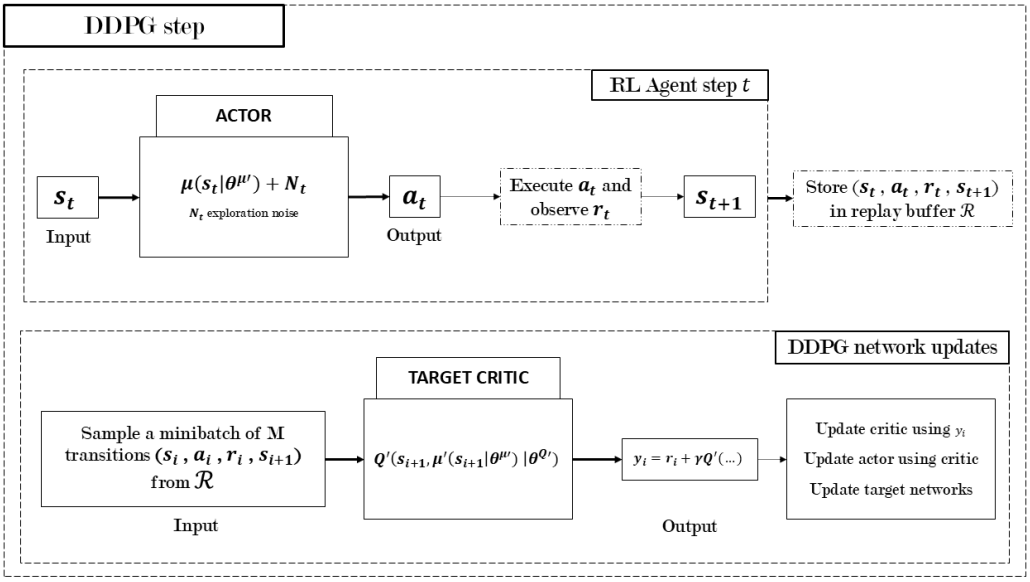


Figure 3.4: Illustration of a single step in the DDPG algorithm including action selection and actor-critic network update.

Figure 3.4 shows a step in the DDPG algorithm. First, we define an initial state S_0. For this state, the actor network produces an action A_t which is executed in order to observe its associated reward r_t. Once this is done, the experience/transition (S_t, A_t, r_t, S_{t+1}) is

stored into the replay buffer $\mathcal{R}$. After this experience is collected, the networks are updated. So once the agent has executed a transition and stored it in the buffer, it is time to train the networks. For this, a sample of M elements $(S_i, A_i, r_i, S_{i+1}) \quad \forall i \in \{1, \ldots, M\}$ from $\mathcal{R}$ is sampled. Note that when working with high-dimensional action spaces, exploring the entire environment becomes unfeasible. For this reason, exploration is introduced by adding to the action a random noise process implemented as Ornstein–Uhlenbeck (OU) noise.

Then, the critic is trained to minimize the Bellman error using target networks and temporal-difference learning. y_i is computed for each sample as

$$y_i = r_i + \gamma Q'(S_{i+1}, \mu'(S_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

and the critic is updated by minimizing the loss

$$\mathcal{L} = \frac{1}{M}\sum_{i=1}^{M}(y_i - Q(S_i, A_i|\theta^Q)^2$$

Note that the previously defined deterministic policy gradient, in practice, is computed using the sampled batch as

$$\nabla_{\theta^\mu} J \approx \frac{1}{M}\sum_{i=1}^{M} \nabla_A Q(S, A|\theta^Q)|_{S=S_i, A=\mu(S|\theta^\mu)} \nabla_{\theta^\mu}\mu(S|\theta^\mu)|_{S=S_i}$$

Finally, the target networks are soft updated via hyperparameter τ

$$\theta^Q \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^\mu \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

Although in the original algorithm this is done for each step, it is usual to find in the existing implementations a parameter that controls after how many steps (new experiences collected) networks should learn. This is because sampling a batch of M samples from the buffer when only one is renewed may not be ideal in some scenarios.

3.3 Learning Scenario

Both the algorithm and the problem are introduced so far, in this section we detail the different strategic decisions that are taken in order to adapt the algorithm to the specific problem we are facing. For the sake of clarity, it is convenient to separate these decisions into two blocks, the first one covering the matters related to RL and the second one the specifics of DDPG. The latter is discussed further on in Section 4.2 and not here.

As anticipated in the introduction, the portfolio management task is modeled as an MDP. Actually, this is a decision that is made in order to solve the problem in terms of RL, so it is now that we define the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Here, it is important to recall that P is a function of the form $P(S_{t+1}|S_t, A_t)$, it is not explicitly modeled as the environment is based on historical data and transitions are directly observed following the model-free RL scenario.

For the space of states, ideally, a state must represent as faithfully and accurately as possible the current situation in which the agent is. Actually, in the state-of-the-art approaches of the existing literature, the current weight configuration of the portfolio is always represented in the state. Along with this, descriptive variables are introduced for each of the assets, providing relevant information about the market context

$$\mathbf{X}_t = \{X_t^1, X_t^2, \ldots, X_t^N\}, \quad X_t^i \in \mathbb{R}^d \quad \forall i \in \{1, \ldots, N\}$$

where each X_t^i is a d-dimensional feature vector representing the characteristics of asset i at time t. More details on these features are given in Section 4.2. In this way, we can define the set of states as a tuple of weights and features

$$\mathcal{S} = \left\{ (W, \mathbf{X}) \,\middle|\, W \in \mathbb{R}^N,\ \sum_{i=1}^{N} w^i = 1,\ w^i \geq 0\ \forall i, \quad \mathbf{X} \in \mathbb{R}^{N \times d} \right\}$$

Note that this structure defines the state as an element of the combined domain $\mathbb{R}^N \times \mathbb{R}^{N \times d} = \mathbb{R}^{N \times (d+1)}$, being a particular state S_t from this space defined as

$$S_t = \{W_t, \mathbf{X}_t\} = \{\{w_t^1, \cdots, w_t^N\}, \{X_t^1, \cdots, X_t^N\}\} \in \mathbb{R}^{N \times (d+1)}$$

In the case of the action space, the problem definition itself tells us how we should represent it. Since the result of applying an action to the current state of the portfolio has to lead to a new configuration of weights (or maintain it). In MDP terms, the transition from W_t to W_{t+1} induced by an action A_t is given by the transition function $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ i.e., $W_{t+1} = f(W_t, A_t)$

Intuitively two ideas appear. The first is to define the action as the adjustment of weights to be performed on each asset, such that

$$A_t = \{\Delta w_t^1, \Delta w_t^2, \ldots, \Delta w_t^N\} \in \mathbb{R}^N,$$

where each component Δw_t^i represents the change in assignment for asset i at time t which can be denoted as

$$\Delta w_t^i = w_{t+1}^i - w_t^i, \quad \forall i \in \{1, \ldots, N\}.$$

Thus, the new weight vector after applying the action is given by

$$W_{t+1} = W_t + A_t,$$

these new weights W_{t+1} must satisfy the same restriction defined earlier $\sum_{i=1}^{N} w_t^i = 1$. These actions allow us to define a constraint giving some limits for this weights' changes so there are not big fluctuations. Given this, the action space is defined as

$$\mathcal{A} = \left\{ (a^1, a^2, \ldots, a^N) \in \mathbb{R}^N \,\middle|\, \sum_{i=1}^{N} a^i = 0,\ -0.1 \leq a^i \leq 0.1\ \ \forall i \in \{1, \ldots, N\} \right\}$$

Considering this proposal, it is easy to define through the output of the Actor network a vector of N elements, also limiting the range in which the weight changes for each asset should be. Actually, this approach fits perfectly with a financial environment where one does not want to have abrupt changes between days. Nevertheless, once implemented,

it does not work properly and is therefore not considered in the experimentation. This approach requires post-processing of the actions to meet the constraints imposed by the very definition of the problem. In particular, since the actions are determined by the output of the Actor network, enforcing positivity constraint on all elements of the vector W_{t+1} is not trivial after applying action A_t to W_t. While it is possible to make adjustments to the actor's output to obtain an action that fulfills these conditions. Yet, this makes the feedback in the form of reward that the DDPG agent receives not faithful to the output the network has presented, thus breaking with the learning process of the algorithm.

The second option, which is adopted given the limitations of the previous one, is to define the action directly as the distribution of weights in the portfolio for the next step $t+1$

$$A_t = \{w_{t+1}^1, w_{t+1}^2, \ldots, w_{t+1}^N\} \in \mathbb{R}^N,$$

such that

$$W_{t+1} = A_t$$

this way it is straightforward to fulfill the constraints from the network output, for instance, by applying a softmax activation in the output layer. Given this, the action space is defined as

$$\mathcal{A} = \left\{ (a^1, a^2, \ldots, a^N) \in \mathbb{R}^N \;\middle|\; \sum_{i=1}^{N} a^i = 1,\ a^i \geq 0 \ \ \forall i \in \{1, \ldots, N\} \right\}$$

Nevertheless, this approach also has drawbacks since we no longer control the magnitude with which each asset fluctuates. Moreover, we eliminate any time dependence between states, since in a single step the agent can change the configuration of the portfolio in an extreme way. We are not interested on this from a financial point of view due to transaction costs, among other things.

Fortunately, we still have more tools available outside the state and action space, and that is the reward function. Through the reward, we can encourage the agent not to take certain policies, for example, by penalizing extreme changes in asset weights. This technique is known as reward shaping and fits perfectly into the RL foundations. More precisely, and given a reward function R from the MDP, we can redefine R to $R' = R + F$ where $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a bounded real-valued function called the shaping reward function [41]. In particular, this penalty in the form of function F is modeled as

$$F = -\lambda \sum_{i=1}^{N} |\Delta w_t^i|$$

Further details on reward formulation for particular experiments are given in Section 4.3.

Chapter 4

Experimental Study

The purpose of this chapter is to present an experimental study that enables analysis of the suitability and performance of the proposed approach, that is, to use DDPG for optimizing the portfolio management problem. To that end, the chapter is organized in three parts. First, the dataset is defined, detailing the variables it contains, the covered time period and its dimensions, among other aspects. Then, the experimental setup is presented, including more specific elements such as the algorithm's hyperparameters and network architectures. Finally, the research questions are posed: (1) How can a reward function be designed to accurately reflect the goals of the task? and (2) How can we incorporate portfolio constraints and diversification objectives into the Actor's output?

4.1 Data Set

In this scenario, each asset in the portfolio corresponds to the shares of a single company. For simplicity, we may refer to these assets interchangeably as "companies" or "assets" throughout the text. More precisely, we consider 10 of the most capitalized companies on the NYSE, listed in Table 4.1. The historical data covers the period between 03/01/2018 and 06/05/2025. This period is split into two sets of 80% and 20%, as shown in Table 4.2, the former used for training and the latter for backtesting process, which is detailed later on.

Symbol	Company Name
AAPL	Apple Inc.
AMZN	Amazon.com Inc.
AVGO	Broadcom Inc.
BRK.B	Berkshire Hathaway Inc.
GOOGL	Alphabet Inc.
JPM	JPMorgan Chase & Co.
META	Meta Platforms Inc.
MSFT	Microsoft Corp.
NVDA	NVIDIA Corp.
TSLA	Tesla, Inc.

Table 4.1: List of the 10 most capitalized companies on the NYSE selected fot the proposed appraoch.

Set	Proportion	Period	Market days
Training	80%	03/01/2018 to 13/11/2023	1476
Backtesting	20%	14/11/2023 to 06/05/2025	369
Total	100%	03/01/2018 to 06/05/2025	1845

Table 4.2: Data split between training and backtesting periods with corresponding size and date interval.

Additionally, the data set contains features for each of the ten companies at each time step tt, following the structure previously defined

$$\mathbf{X}_t = \{X_t^1, X_t^2, \ldots, X_t^N\}, \quad X_t^i \in \mathbb{R}^d \quad \forall i \in \{1, \ldots, N\}$$

where each X_t^i is a d-dimensional feature vector representing the characteristics of asset ii at time t. These features provide a rich representation of the market state. These features are organized into several groups, including price and volume indicators, moving averages, volatility and momentum metrics, as well as proprietary indicators provided by Interstock. Table 4.3 lists all features used, grouped by type, with a brief description to provide some intuition about their role. Some of the most relevant examples include the Bollinger Bands and Keltner Channels, which can be intuitively visualized alongside price in candlestick charts. In particular, Figure 4.1 shows the price movement of an asset together with its Bollinger Bands. Another illustrative case is the Moving Average Convergence Divergence (MACD), a popular momentum indicator that captures trend changes by comparing exponential moving averages. This is visualized in Figure 4.2. For more technical detail on these indicators and others, we refer the reader to [42].

Figure 4.1: Bollinger Bands for a given asset together with its candlesticks during nine months. Additional information: Investopedia. Source: Yahoo Finance.

4.2 Model Setup

We have already defined in Section 3.3 the state S from the MDP as a structure of the form

$$S_t = \{W_t, \mathbf{X}_t\} = \{\{w_t^1, \cdots, w_t^N\}, \{X_t^1, \cdots, X_t^N\}\} \in \mathbb{R}^{N \times (d+1)}$$

[2]These values indicate the time horizon (days) for which the value is computed.

Group	Feature	Description
Price	Open	Price at market open
	High	Highest price of the session
	Low	Lowest price of the session
	Close	Price at market close
Volume	Volume	Number of shares traded
	OBV	On-Balance Volume: links price and volume
Moving Averages	EMA (5,10,20) [2]	Exponential moving averages over 5, 10, and 20 days
Bollinger Bands	BB Middle (5,10)	Typically Simple Moving Average (SMA)
	BB Upper (5,10)	Upper band: SMA + $k \cdot \sigma$
	BB Lower (5,10)	Lower band: SMA – $k \cdot \sigma$
	BB Percent B (5,10)	Relative location of price in band
	BB Width (5,10)	Distance between upper and lower bands
Keltner Channels	KC Middle (5,10)	Exponential moving average (EMA) of price
	KC Upper	Upper band: EMA + $k\cdot$ ATR
	KC Lower (5,10)	EMA - $k\cdot$ ATR
	KC Percent B	Relative position in channel
	KC Width	Width between bands
Volatility	ATR (5,10)	Average True Range
	PPO (5, 10) and (10, 20)	Percentage Price Oscillator: measures relative difference between two EMAs
	MACD (5, 10)	Difference between a short-term EMA (5) and a long-term EMA (10)
	MACD Signal (5, 10, 3)	3 days EMA of the MACD line
	MACD Hist, (5, 10, 3)	Difference between the MACD line and its signal line
Momentum	RSI (5,10)	Relative Strength Index
	MFI (5,10)	Money Flow Index (volume-weighted RSI)
	ADX (5,10)	Average Directional Index
	CMF (5,10)	Chaikin Money Flow
Interstock Indicators	IMMA (5,10)	Custom designed proprietary indicators created by Interstock Development
	IMLREG (5,10)	
	SPX_CORR	
	SPX_DIFF	

Table 4.3: Grouped list of features in $\mathbf{X}_t$ with brief descriptions.

where W_t is the weight vector representing the capital distribution in the portfolio at time t and $\mathbf{X}_t\}$ is the set of feature vectors describing the characteristics of each asset at time t. Here, d represents the number of features we have for each of the assets in every time step. In total, 52 features are considered, hence $d = 52$.

As far as the algorithm itself is concerned, we have seen the details of DDPG in Section 3.2. In particular, when using DDPG, exploration is originally introduced by an OU noise. Nevertheless, some recent works point out that there is no need to implement OU as it is done in the original paper. In contrast, a Gaussian noise, among others, is preferred for its simplicity and similar results with respect to OU noise [43, 44]. Therefore, in this work, we consider Gaussian noise for exploration.

Another important element to define is the architecture of the Actor and Critic neural network of the DDPG algorithm. In this sense, we opt for introducing the regularization techniques already mentioned. Also, we try to adjust the complexity of the network to the available data. To define the architectures, we have considered the regularization elements introduced above. The activation function we use is the hyperbolic tangent (tanh) due

Figure 4.2: MACD and its signal and histogram for a given asset together with its candlesticks during nine months. Additional information: Investopedia. Source: Yahoo Finance.

to the range of the features, which often include negative numbers. With respect to the dimensions of the networks, the Actor network receives as an input the state S_t which is composed of the 10 weights of the portfolio and the 52 features that are being used for each asset

$$\mu(S_t|\theta^\mu)_{\text{input}} = S_t \in \mathbb{R}^{N\times(d+1)}$$

specifically with $n = 10$ and $d = 52$, and gives as an output the action to be taken

$$\mu(S_t|\theta^\mu)_{\text{output}} = A_t \in \mathbb{R}^N$$

A very important point to consider is how to define the activation function of the last layer of the Actor so that it outputs a valid vector. Remember that a particular action A_t is defined as the distribution of weights W_{t+1} in the portfolio for the next step $t+1$. Moreover, this distribution must satisfy the restrictions of positivity and sum one. Intuitively, the idea of applying a softmax function arises, as it presents a very direct way of meeting the constraints. However, this has some drawbacks which we will examine later, proposing some solutions. In the case of Critic, the input is both the state S_t and the taken action A_t

$$Q(S_t, A_t|\theta^Q)_{\text{input}} = \{S_t, A_t\} \in \mathbb{R}^{N\times(d+2)}$$

Recall that the critic represents a quantitative measure of the quality of the action A_t when in state S_t, so the output is

$$Q(S_t, A_t|\theta^Q)_{\text{output}} = \text{Q-estimate} \in \mathbb{R}$$

More precisely, Table 4.4 shows the specific architecture defined for both the Actor and Critic networks, noting that the target networks are copies of these ones.

It is also important to adjust the size of the replay buffer $\mathcal{R}$. As we have seen, this buffer constitutes a fundamental element. It is from here that the experiences are taken so the networks can learn an optimal policy. A too large buffer can lead to the storage of redundant experiences, while too small one can lead to insufficient diversity. The latter leading to learning problems such as stalling in non-optimal policies. Similarly, the size of the batch that is sampled from this buffer $\mathcal{R}$ should be adjusted in proportion to the total size of the buffer itself. Note that if the batch is too large, there is a risk of defeating the purpose of the

Component	Structure	Details
Actor	Input Layer	$N \times (d+1)$
	Hidden Layer 1	Linear(512) + BatchNorm + Tanh
	Hidden Layer 2	Linear(512) + BatchNorm + Tanh
	Hidden Layer 3	Linear(256) + BatchNorm + Tanh
	Hidden Layer 4	Linear(128) + BatchNorm + Tanh
	Output Layer	Linear(N) with final scaling
	Regularization	Dropout (p = 0.3)
	Action noise	Gaussian noise added to logits
	Optimizer	Adam, learning rate $\eta = 10^{-5}$, weight decay 10^{-5}
	Total parameters	702,858 trainable
Critic	Input Layer	Concatenation of state ($N \times (d+1)$) and action (N)
	Hidden Layer 1	Linear(512) + BatchNorm + Tanh
	Hidden Layer 2	Linear(512) + BatchNorm + Tanh
	Hidden Layer 3	Linear(256) + BatchNorm + Tanh
	Hidden Layer 4	Linear(128) + BatchNorm + Tanh
	Output Layer	Linear(1) – returns scalar Q-value
	Regularization	Dropout (p = 0.3)
	Optimizer	Adam, learning rate $\eta = 10^{-5}$, weight decay 10^{-5}
	Total parameters	706,817 trainable

Table 4.4: Architecture and complexity of the Actor and Critic neural networks.

replay buffer which is, in part, to provide varied and disordered experiences. Nonetheless, if it is too small, a reliable estimate of the gradient cannot be obtained, which is precisely why batches are introduced rather than individual samples. In general terms, Table 4.5 shows the hyperparameters used during the training process. Most of the hyperparameters have been adjusted through experimentation. For the sake of clarity, they are not shown in the results so we can focus on the most relevant contents of the approach.

Hyperparameter	Value / Description
Discount factor γ	0.99
Soft update coefficient τ	5×10^{-4}
Batch size	64
Episode size	60
Replay buffer size	60×10
Learning starts	60×10 steps
Total time steps	1,500,000 steps
Action noise	Gaussian, $\mathcal{N}(0,\, \sigma^2)$ with $\sigma = 0.1$
Target update frequency	Every 100 steps

Table 4.5: DDPG hyperparameters used during training process.

Finally, we note that at the beginning of each episode, the initial state is defined as an equally weighted portfolio with an initial value of 1,000,000 USD. In addition, the

replay buffer is initially filled with random actions before the agent starts predicting and learning. This is a common practice in DDPG, used to prevent biased learning from a non-representative set of experiences. Each episode spans a period of approximately three months, equivalent to 60 market days, while reallocations are performed every day (step).

4.3 Questions

Once the learning scenario and experimental setup have been defined, the research questions to be answered are posed.

- **Question 1: Reward Function Design**. How can a reward function be designed to accurately reflect the objectives of the problem?

  This question aims to evaluate whether the reward function effectively aligns with the main objective of the portfolio management problem, which is to maximize benefit while reducing risk. By testing different reward formulations, we can assess which one better reflects this objective. In particular, we experiment with four different reward functions, each reward formulation is assigned a specific name to facilitate its identification during the experiments.

  Return. The first reward is simply the portfolio return at time t, denoted as ψ_t. This formulation is used as a basic approximation and does not include any risk component. Recall that it is calculated as

  $$r_t = \psi_t = \frac{V_t - V_{t-1}}{V_{t-1}} = \sum_{i=1}^{N} w_t^i \cdot \left(\frac{C_t^i}{C_{t-1}^i} - 1 \right)$$

  being r_t the reward at time t. This reward function aims to directly maximize portfolio returns, offering a straightforward and intuitive objective. We will evaluate how well it maximizes benefits in practice and compare it with other reward formulations that explicitly incorporate risk, to understand the trade off between benefit and risk. Here, it may encourage the agent to take excessive risks, resulting in unstable or unrealistic portfolio allocations.

  Return-Dev. The second reward introduces risk by computing the ratio between the return and the standard deviation over the episode:

  $$r_t = \frac{\psi_t}{\sigma_k}$$

  where σ_k represents the standard deviation of the portfolio return during the episode ($k = 60$). By including the standard deviation of returns, this formulation introduces a basic risk adjustment that penalizes volatility. This encourages more balanced portfolios that seek consistent performance. However, this reward presents a very different scale of values for the numerator and denominator, which may present a problem

  Adjusted Return-Dev. In the third case, we adapt the previous formulation by applying a logarithmic transformation in the numerator

  $$r_t = \frac{\log(\psi_t + 1)}{\sigma_k}$$

here, the log is used to mitigate the imbalance between the magnitude of the numerator and denominator. This way we avoid any of them from dominating the reward.

Reward Shaping. Finally, we apply reward shaping over the previous definition, as explained earlier, by introducing a penalty term that discourages abrupt portfolio reallocations

$$r'_t = \frac{\log(\psi_t + 1)}{\sigma_k} - \lambda \sum_{i=1}^{N} |\Delta w_t^i|$$

where Δw_t^i is the change in weight for asset i, and λ is a penalty coefficient set to 0.1. This shaping guides the agent toward smoother transitions.

- **Question 2: Actor Output Transformation**. How can we incorporate portfolio constraints and diversification objectives into the Actor's output?

  This question focuses on ensuring that the Actor's output respects the portfolio management constraints, which, despite having several options, must also encourage diversification. To address this question, we explore several strategies applied to the final layer of the Actor network, whose pre-activation output is denoted as $z_t^{[L]}$ at time t. The final action is obtained by applying a transformation f to $z_t^{[L]}$, i.e.,

$$A_t = f(z_t^{[L]})$$

  In particular, we explore several ways to define the final transformation f. Each transformation is also given a name to clearly reference it in the experimental analysis.

  Softmax$_1$ The first approach consists of applying a softmax function [45]

$$\text{Softmax}(z_t^{[L]}) = \frac{e^{z_t^{[L]}}}{\sum_{j=1}^{N} e^{z_{t,j}^{[L]}}}$$

  to the Actor's output. This ensures that the resulting weight vector is positive and sums to one. However, due to its exponential nature, softmax tends to concentrate most of the weight on a few assets, which goes against the goal of diversification. Despite this, it remains as the most intuitive solution.

  Softmax$_T$ To mitigate the concentration effect, we introduce a temperature parameter T

$$\text{Softmax}_T(z_t^{[L]}) = \frac{e^{z_t^{[L]}/T}}{\sum_{j=1}^{N} e^{z_{t,j}^{[L]}/T}}$$

  By adjusting the temperature, we can control how distributed the output is. We set $T = 20$, a higher temperature leads to more uniform weights, which favors diversification. As an alternative temperature of $T = 5$ is applied. Note that the first softmax base case is equivalent to applying $T = 1$.

  NormSigmoid Finally, we apply a sigmoid function (ensures positivity)

$$\text{Sigmoid}\big(z_t^{[L]}\big) = \frac{1}{1 + \exp(-z_t^{[L]})}$$

followed by normalization (ensures sum one)

$$\text{NormSigmoid}(z_t^{[L]}) = \frac{\text{Sigmoid}(z_t^{[L]})}{\sum \text{Sigmoid}(z_t^{[L]})}$$

This approach is promising avoiding the strong concentration tendencies of softmax.

As a final note, and before moving on to the results, it is important to point out that due to the limited time, only one run has been considered for each experiment. Ideally, several should be done in order to obtain results that are more faithful to reality. This limitation restricts the statistical significance of the findings, as a more rigorous evaluation would require multiple independent runs to account for variability and ensure the robustness and reliability of the conclusions.

Chapter 5

Results

In this chapter, we present the results of the experiments designed to address each of the previously posed questions. Each of the experiments produces a trained agent capable of making portfolio reallocation decisions every market day. The individual agents are evaluated on the test set described in Table 4.2.

5.1 Quantitative Performance Metrics

This form of evaluation is called out-of-sample backtesting, which involves assessing the agent's performance on historical data that was not seen during training The aim is to estimate its generalization ability before real-time deployment. In the financial domain, backtesting results are typically presented through return curves, which offer a clear and intuitive depiction of the portfolio's performance over time. As a simple illustrative example, Figure 5.1 shows the return over a period of time for two different trading strategies. The chart helps us visualize the different agents performance against a benchmark. In particular, we define a benchmark using the most naive approach, which consists of assigning equal weights to all assets in the portfolio. In addition to the return curves, we compute several financial quantitative performance metrics during the backtesting. Particularly, we consider the following:

- The annualized return is calculated from the average daily return as

$$\text{Annual Return} = \left(\left(\bar{\psi} + 1 \right)^{252} - 1 \right)$$

  where $\bar{\psi}$ is the average daily return given by

$$\bar{\psi} = \frac{1}{252} \sum_{t=1}^{252} \psi_t$$

  and 252 is the approximate number of trading days in a year.

- The annualized volatility which measures the variability of the portfolio returns using the standard deviation

$$\text{Annual Volatility} = \sigma_{\text{annual}} = \sigma_{\text{daily}} \times \sqrt{252}$$

where σ_{daily} is the standard deviation of the daily returns

$$\sigma_{\text{daily}} = \sqrt{\frac{1}{252-1}\sum_{t=1}^{252}(\psi_t - \bar{\psi})^2}$$

.

- The Sharpe ratio which combines both return and risk

$$\text{Sharpe Ratio} = \frac{\text{Annual Return}}{\text{Annual Volatility}}$$

Note that the risk-free rate is set to zero, a common practice introduced earlier. The Sharpe ratio measures the return an asset provides relative to its risk.

- The Maximum Drawdown (MDD) which represents the largest loss between a local maximum and the subsequent minimum within a time frame

$$\text{MDD} = \max_{t\in[0,T]}\left(\frac{V_t - \max_{\tau\in[0,t]} V_\tau}{\max_{\tau\in[0,t]} V_\tau}\right)$$

where V_t is the cumulative value of the portfolio at time t and T is the total length of the evaluation period. Note that $\max_{\tau\in[0,t]} V_\tau$ represents the highest value up to time t. A maximum drawdown highlights an investment's most severe historical loss (for further details, refer to here).

Figure 5.1: Return over time of two simple investment strategies: investing in a broad US stock market fund (S&P 500) and investing in a tech-focused stock fund (Nasdaq-100). This illustration shows how an initial investment would have grown in each case for the same time period.

Regarding the learning process itself, all the presented experiments have shown signs of proper learning as all models eventually converged, although some configurations may

have required more time. Here, convergence refers to the agent learning a consistent policy. In particular, during training, it can be observed that for the same time periods, the agent applies the same policy. This is also reflected in the convergence of the values estimated by the critic network, the Q-estimates. To illustrate the behavior during training, a representative example is shown in Figure 5.2. This visualization is not included for all experiments, as its interpretation can be sensitive to various factors such as reward scale which may make difficult direct comparisons.

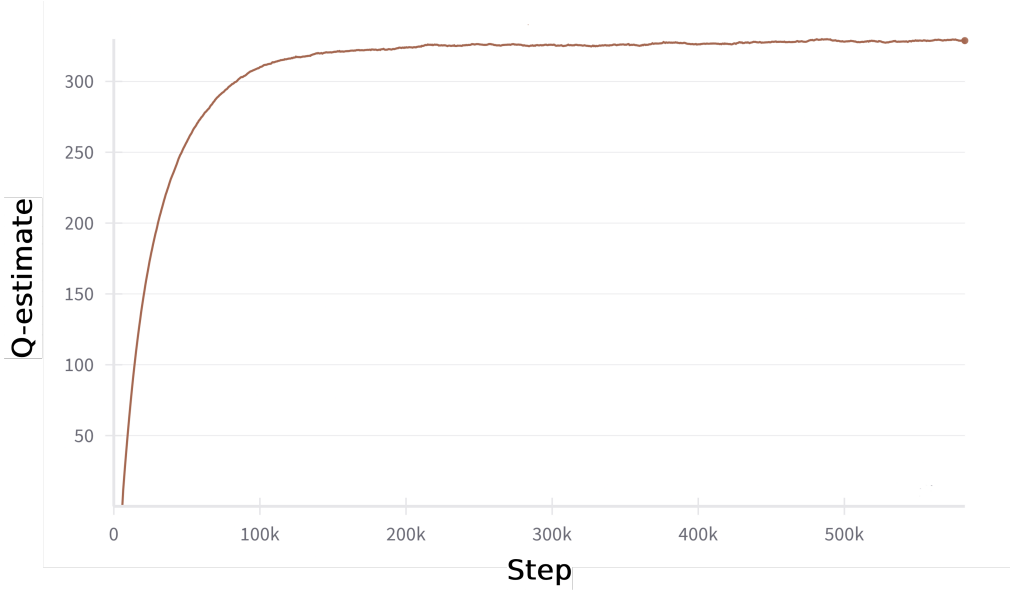


Figure 5.2: Representative example of the evolution of the Q-value estimated by the Critic network during training.

5.2 Reward Function Design

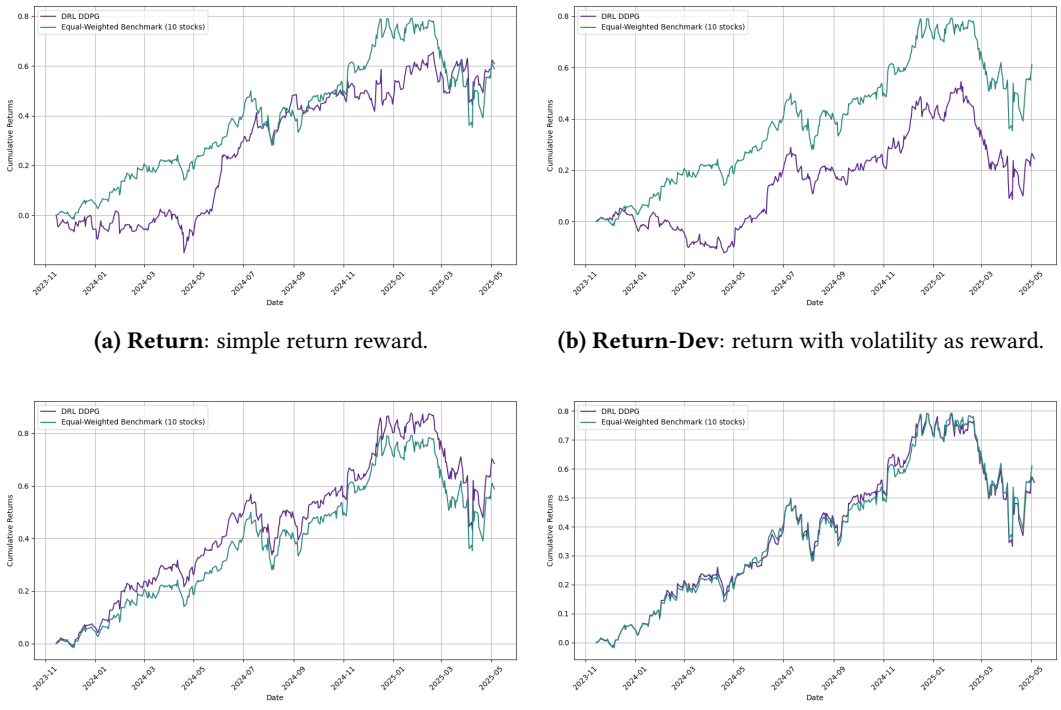
Table 5.1 shows the performance metrics for the four experiments conducted to answer the first question in Section 4.3: how can a reward function be designed to accurately reflect the objectives of the problem? Starting with the Return approach, it achieves the highest annual return than the benchmark, which is expected given that its reward is only focused on maximizing benefit. The Return approach also manages to achieve the lowest maximum drawdown, suggesting that directly maximizing returns helps to avoid a major drop that the other approaches do experience.

Moving on, when risk is introduced in the form of standard deviation, volatility decreases significantly, making this the best-performing approach in that regard, while returns are also notably reduced, as volatility has greater magnitude than return in the reward function. This gain in stability does not compensate for the loss in benefit, as reflected in its Sharpe ratio. By adjusting the scale of risk relative to return using the logarithm, a better balance between both values is achieved, leading to a very good annual return very close to the Return approach. More importantly, its Sharpe ratio is the best among all, which, as we recall, is the metric that captures the return–risk trade-off. Lastly, this approach also improves the benchmark in terms of maximum drawdown, although the values are nearly identical and both experience the same drop.

Metric	Benchmark	Return	Return-Dev	Adjusted Return-Dev	Reward Shaping
Annual Return	0.364560	**0.384990**	0.158232	0.382103	0.343878
Annual Volatility	0.265330	0.277485	**0.240655**	0.249943	0.255976
Sharpe Ratio	1.373987	1.387426	0.657506	**1.528760**	1.343399
Maximum Drawdown	-0.245179	**-0.167816**	-0.297314	-0.237151	-0.256207

Table 5.1: Performance metrics comparison for the benchmark and different experiments for the Reward Function Design.

It is also worth noting that introducing the penalty for abrupt changes in the reward did not lead to better results compared to the case without it. This was unexpected, however, it is important to consider that the effect of this penalty would likely be more evident if transaction costs were included in the returns. As explained, that is not the case in this scenario, and it remains to be explored whether penalizing large changes truly brings an advantage.



(a) Return: simple return reward.

(b) Return-Dev: return with volatility as reward.

(c) Adjusted Return-Dev: reward as adjusted return with volatility.

(d) Reward Shaping: reward as adjusted return with volatility penalized by abrupt changes.

Figure 5.3: Cumulative return of different experiments during the backtest period for the Reward Function Design.

In general terms, all approaches exhibit a similar maximum drawdown and experience a significant drop at the same point, as clearly shown in Figure 5.3. However, the first approach (Figure 5.3a), which focuses only on maximizing return, manages to avoid this drop and achieves by far the best value in this metric. Although it initially struggles to match the benchmark's returns, it eventually outperforms it by avoiding the final drop. A similar pattern is observed with the non-adjusted Return-Dev reward (Figure 5.3b), it starts slowly but ends up being affected by the final drop, as it happens to the benchmark,

resulting in the lowest annual return. In the third case (Figure 5.3c), where this reward is adjusted, the model consistently improves over the benchmark across all metrics, as previously discussed. In the last case (Figure 5.3d), where reward shaping is applied, the performance evolves almost identically to the adjusted reward, but with slightly worse values in every metric compared to the benchmark.

Finally, answering to the first question posed in Section 4.3, we can say that the proposed rewards capture the objective of the problem, though to different degrees. With the exception of the second approach where the reward is dominated by volatility, which did not succeed in maintaining benefit while reducing risks, the others achieve the desired outcome. Specifically, the Return-Dev approach is the one that obtains the best results, presenting values of annual return and volatility close to the best, this is reflected in its Sharpe ratio, which, as mentioned above, improves all the other approaches including the benchmark.

5.3 Actor Output Transformation

Table 5.2 shows all the metrics resulting from the experimentation conducted to answer the second question: how can we incorporate portfolio constraints and diversification objectives into the Actor's output? For this scenario, we choose the reward function that performed best in the previous experimentation shown in Section 5.2, which is defined as

$$r_t = \frac{\log(\psi_t + 1)}{\sigma_k}$$

Using the standard softmax ($T = 1$), we observe that it leads to poor results in every metric. It records the lowest annual return and the worst Sharpe ratio by a significant margin. This confirms the expected effect of softmax in its default form. It results in a concentration of the allocation on a limited number of assets, thereby increasing exposure to individual fluctuations and reducing the overall stability of the portfolio. Moreover, it presents the highest maximum drawdown among all tested methods.

Metric	Benchmark	Softmax_1	Softmax_{20}	Softmax_5	NormSigmoid
Annual Return	0.364560	0.122088	0.349794	**0.433871**	0.382103
Annual Volatility	0.265330	0.279972	**0.233620**	0.281061	0.249943
Sharpe Ratio	1.373987	0.436072	1.497277	**1.543690**	1.528760
Maximum Drawdown	-0.245179	-0.325441	**-0.221199**	-0.243449	-0.237151

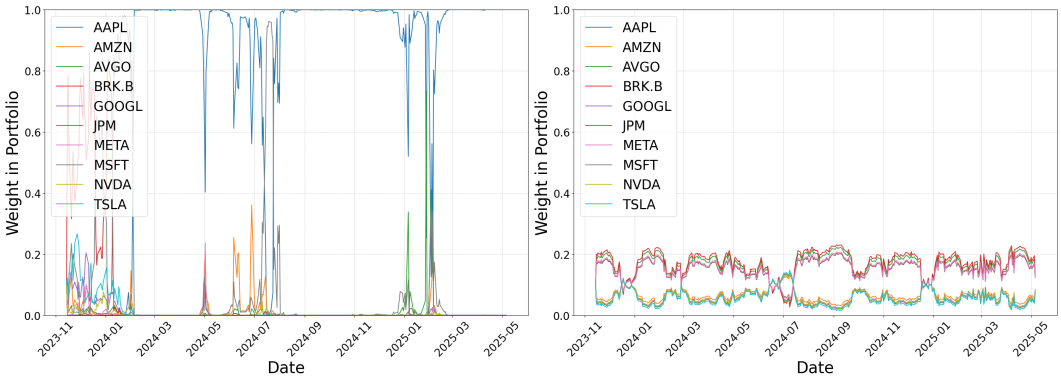
Table 5.2: Performance metrics comparison for the benchmark and different experiments for the Actor Output Transformation.

When increasing the temperature to $T = 20$, the effect of softmax becomes more moderate. The resulting allocations are more uniform, leading to the lowest annual volatility. This outcome is consistent with higher diversification, as a more even distribution tends to reduce fluctuations. However, this benefit comes at the cost of a reduced annual return, slightly below the benchmark. Despite this, Sharpe ratio improves significantly compared to the base case, indicating a more efficient return–risk balance.

In the third case, softmax with $T = 5$ achieves the best results overall. It reaches the highest annual return and the best Sharpe ratio, while maintaining a volatility level

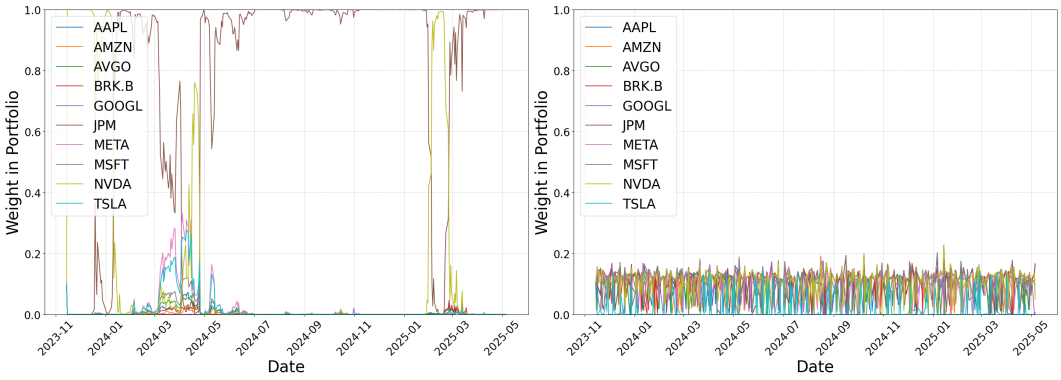
comparable to the benchmark. This suggests that this temperature presents a good balance since it avoids the excessive concentration of the default softmax while still allowing the model to focus selectively on promising assets. In addition, the maximum drawdown is slightly lower than the benchmark, which suggests that this configuration provides a more robust behavior.

The NormSigmoid approach also produces very competitive results. It offers the second best annual return and Sharpe ratio, as well as a lower volatility than both the benchmark and most other experiments. Its performance is similar to that of softmax with $T = 5$, but with the added advantage of not requiring the tuning of a temperature parameter. This makes it a simple yet effective choice, as it ensures positivity and normalization while avoiding excessive concentration.



(a) Softmax$_1$: softmax with $T = 1$.

(b) Softmax$_{20}$: softmax with $T = 20$.



(c) Softmax$_5$: softmax with $T = 5$.

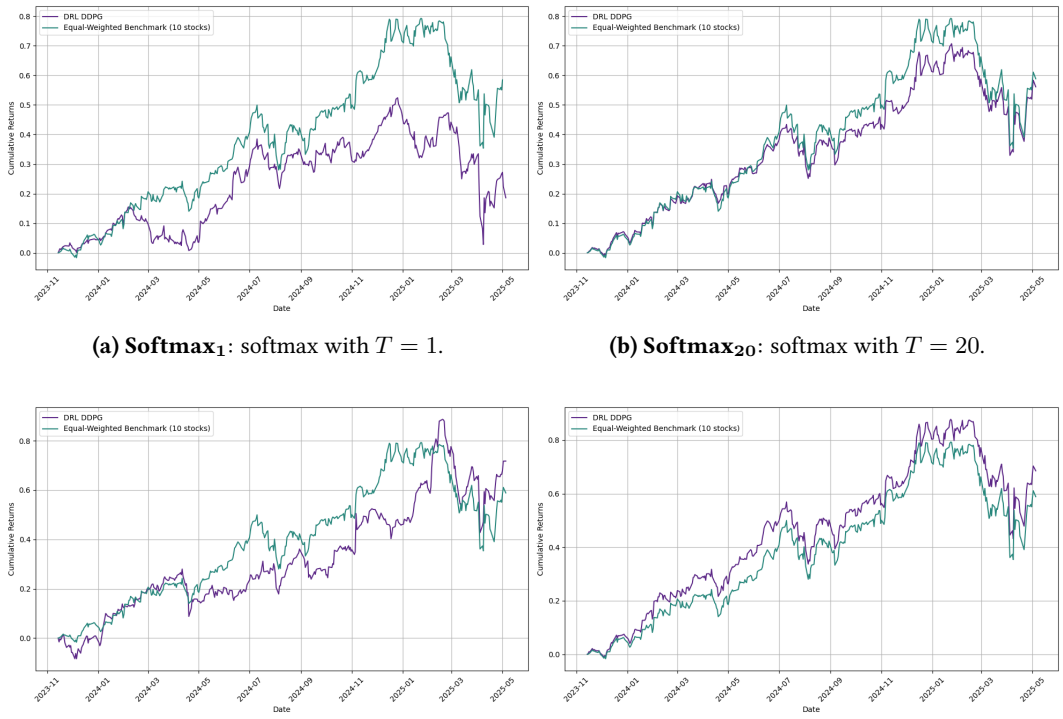
(d) NormSigmoid: sigmoid and normalization.

Figure 5.4: Actions taken by the agent during the backtesting period. The distribution of the portfolio weights for each time step is shown.

The metrics presented in Table 5.2 are consistent with the expected outcomes in terms of portfolio diversity. However, to better understand how this diversity is expressed over time, we introduce Figure 5.4, which illustrates the evolution of asset weights throughout the evaluation period. This visualization provides a more direct and intuitive representation of the portfolio's diversification, one of the key objectives of this question. In Figure 5.4 each line (color) represents a company whose weight fluctuates between 0 and 1 on the y-axis.

In the low temperature softmax approaches, the weights tend to be extreme, with some assets taking up to 100% of the portfolio, which is undesirable as it contradicts the diversification principle discussed earlier. In the case of softmax with $T = 20$, introduced to avoid this behavior, this extreme concentration is not present, but the capital is distributed mainly between two groups of assets with little change over time. This pattern might indicate that the agent has learned correlations among assets, although it results in a rigid and non-natural allocation strategy from a financial perspective. Despite this, the agent seems to have learned something, since in general terms it improves on the benchmark. Conversely, the NormSigmoid approach displays a more natural behaviour avoiding excessive capital concentration in a single asset, here, individual assets rarely exceed 20% of the portfolio, allowing any asset to gain prominence without being permanently excluded. However, changes in weights are quite abrupt, showing no gradual transitions or smoothness.

In Figure 5.5, the cumulative returns illustrate that only the experiment using NormSigmoid and **Softmax$_5$** surpass the benchmark, showing how crucial the choice of output transformation is even when the reward function remains constant. In summary, despite all experiments using the same reward function, their outcomes differ significantly depending on the output transformation applied. Both NormSigmoid and softmax with $T = 5$ show strong performance in all metrics, clearly outperforming the benchmark. In contrast, the default softmax seems not to be the best strategy due to its strong tendency to focus on a few assets. The softmax with $T = 20$ reduces risk effectively, but its conservative behavior limits the potential for returns. Therefore, selecting an appropriate output transformation is essential for guiding the agent toward stable and profitable strategies.



(a) Softmax$_1$: softmax with $T = 1$.

(b) Softmax$_{20}$: softmax with $T = 20$.

(c) Softmax$_5$: softmax with $T = 5$.

(d) NormSigmoid: sigmoid and normalization.

Figure 5.5: Cumulative return of different experiments during the backtest period analyzing the different approaches for Actor Output Transformation.

Finally, in the other cases where softmax is applied, the exponential nature of the function causes the weights to tend to concentrate more on a single asset. Despite this behavior being controllable through temperature, it has been found not to be very suitable. The need to introduce another parameter T to be adjusted also produces a disadvantage compared to sigmoid with normalization. However, the temperature can be useful to regulate the aggressiveness or conservatism with which the agent operates, producing more uniform or non-uniform weight distributions. Nevertheless, this is just *making a virtue of necessity* [46], and more elegant strategies such as reward shaping with entropy regularization could provide better control over portfolio diversity.

Chapter 6

Conclusion and Future Work

In this work, we have presented an approach on how to apply DDPG to the portfolio management problem. To do so, we first proposed a learning scenario where we defined the problem formalization under an MDP. Along with this, we carried out an experimentation to determine if this approach made sense. In particular, we posed two questions that guided our experimentation, and from this, we concluded that although the model is able to outperform a naive benchmark and seems to learn relevant patterns, the results are not robust and vary considerably depending on the configuration and other aspects discussed below.

During the training process of the different agents there have been no technical learning problems. Among other things, this means that the convergence of the networks and the algorithm's distinctive features, such as buffer management, have worked correctly. However, as we have seen in backtesting section, the results do not significantly improve the proposed benchmark. In general terms, it has been shown that the model lacks robustness, at least in the scenario in which it has been configured. Despite the attempts to give stability to the agent, the lack of robustness is observable in the sensitivity to any change in the scenario. One positive aspect of the results is that, when examining the different proposals, in most of the cases models respond appropriately. For instance, when greater diversification is introduced, volatility decreases, or when more weight is given to returns, the annual return improves, which seems logical. However, while individual changes result in some improvement and collectively they provide better results compared to a naive approach like a uniform distribution, the results are not stable enough to claim a big significance.

Building on the previous analysis, the results presented in this work are based on a specific dataset, and as such, may be influenced by the characteristics of the data. Although it can be expected that using a different dataset might lead to different results, the challenges and limitations associated with this approach are not likely to be heavily influenced by the data used. However, incorporating additional and more varied data could help stabilize certain aspects, as will be discussed later. In this regard, this should not be seen as a problem but rather as an opportunity for improvement. Overall, using RL for portfolio management appears to be a reasonable and promising approach. However, it is important to recognize that its application in this context is inherently complex and demands a high level of precision.

In order to mitigate these limitations, this work presents several directions in which future work can be carried out. First, it could be worth considering expanding the number of assets being employed. Having only 10 companies, all among the most capitalized on the NYSE, is a good starting point, but there may be issues related to correlation. In addition to the small amount, many of them come from the tech sector, which could lead to excessive correlation, making the signals received by the agent not diverse enough. One approach to mitigate this issue is to introduce more companies into the portfolio. However, this comes with the downside of adding complexity, which may slow down learning and even reduce the interpretability of the actions. An alternative would be to introduce correlation metrics in the state to try to control this phenomenon. Building upon this, introducing more data could also be explored, particularly through sentiment analysis. There are studies that suggest using large language models (LLMs) for sentiment analysis can improve traditional DRL approaches in portfolio optimization, compared to using price-based models alone [47].

Another promising idea is to try to predict an ordering of best assets in which to invest. In this sense, the action would be defined as a permutation, opening up new possibilities and challenges. New reward metrics would need to be introduced, with permutation comparison methods such as Kendall's τ serving as a tool for evaluating asset rankings. This allows to have a ground truth since we know the best ordering at each step. In addition, there would be no need to adjust generated outputs to the constraints, thus simplifying the approach.

As far as the proposed approach is concerned, some strategies could be considered to make it more robust. In particular, there are some fronts that are not addressed in this work. First, new structures could be defined for the Actor and Critic networks. In fact, DDPG exhibits a certain form of self-adjustment, as it approximates both the actions and their quality through neural networks. Nonetheless, it is known to function effectively despite this. However, designing the networks in a more refined way could help improve performance. Second, other representations of actions as well as states could be considered. In this sense, we have already seen that there are some proposals in the literature whereby the state is processed to obtain a more informative representation of the environment. Additionally, new reward functions could be further explored, as has been done in this work.

One of the areas that has not been fully addressed in this work is the statistical evaluation of the agent's performance. To improve the validity of the results obtained, it would be useful to perform more runs per experiment. This would allow performance to be evaluated under a broader set of conditions and would help to obtain more representative and generalizable results. With a larger number of runs, appropriate statistical assessment (such as hypothesis testing, Bayesian analysis or bootstrapping) could be applied to more rigorously compare the performance of different configurations and approaches. Additionally, hyperparameter tuning strategies could be pursued. In our case, the parameters were adjusted more manually, observing how the agent behaved. For example, we know that the buffer cannot be set to 100,000 because it's too large, but we are unsure about which other values such as 500, 600, or 800 are better. This is also the case for other parameters.

Another aspect that was not considered in terms of constraints is the application of legal restrictions, such as European Union regulations that establish rules for investment funds to protect investors and ensure transparency. While we have tried to focus on diversification, there are more direct constraints that could affect the agent's decision-making process.

However, this would be more of a step toward creating a product than improving the agent's behavior itself. The priority should be refining the agent first, and then implementing these restrictions. In fact, these legal constraints could be introduced externally, manipulating the agent's output slightly to comply with legal requirements before applying the actions.

Nevertheless, we do not believe there is much room for improvement in these regards. Actually, we may be encountering limitations of DDPG itself. In particular, there is a well-identified problem consisting of the critic network overestimating the quality of the actions. In order to address some of the identified problems of DDPG, Fujimoto et al. [48] presented Twin Delayed DDPG (TD3). The main novelty in TD3 with respect to DDPG is the use of two target networks to avoid the overestimating problem. Additionally, it introduces some other techniques to give stability, such as delayed updates for the Actor, which helps to prevent the Actor from being updated too frequently based on noisy Q-value estimates. Given these limitations, TD3 as well as other algorithms within DRL can be explored.

Bibliography

[1] Comisión Nacional del Mercado de Valores (CNMV). (2022) Cómo invertir en bolsa - Guía. [Online]. Available: https://www.cnmv.es/DocPortal/Publicaciones/Guias/Guia_Como_Invertir_en_bolsa.pdf See page 1.

[2] T. W. F. of Exchanges. (2025) Market statistics - focus | the world federation of exchanges. [Online]. Available: https://focus.world-exchanges.org/issue/april-2025/market-statistics See page 1.

[3] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1. See page 2.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1509.02971 See pages 2, 5.

[5] M. Wiering and M. Otterlo, *Reinforcement Learning: State-Of-The-Art*. Springer Berlin, Heidelberg, 01 2012, vol. 12. See page 3.

[6] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012. See page 4.

[7] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992. See page 4.

[8] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, UK, 1994, vol. 37. See page 4.

[9] R. Bellman, "Dynamic programming," *science*, vol. 153, no. 3731, pp. 34–37, 1966. See page 4.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602 See page 5.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016. [Online]. Available: https://arxiv.org/abs/1602.01783 See page 5.

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 1861–1870. [Online]. Available: https://proceedings.mlr.press/v80/haarnoja18b.html See page 5.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347 See page 5.

[14] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958. See page 6.

[15] C. M. Bishop and H. Bishop, *Deep learning: Foundations and concepts*. Springer Nature, 2023. See page 6.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org. See pages 6, 7, and 8.

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015. See page 7.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986. See page 7.

[19] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256. See page 7.

[20] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1310–1318. [Online]. Available: https://proceedings.mlr.press/v28/pascanu13.html See page 7.

[21] A. E. Hoerl and R. W. Kennard, "Ridge regression: Applications to nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 69–82, 1970. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00401706.1970.10488635 See page 7.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456. [Online]. Available: https://proceedings.mlr.press/v37/ioffe15.html See page 8.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html See page 8.

[24] Z. Jiang, D. Xu, and J. Liang, "A deep reinforcement learning framework for the financial portfolio management problem," 2017. [Online]. Available: https://arxiv.org/abs/1706.10059 See pages 9, 10.

[25] P. Yu, J. S. Lee, I. Kulyatin, Z. Shi, and S. Dasgupta, "Model-based deep reinforcement learning for dynamic portfolio optimization," 2019. [Online]. Available: https://arxiv.org/abs/1901.08740 See pages 8, 9.

[26] P. K. Aritonang, S. K. Wiryono, and T. Faturohman, "Hidden-layer configurations in reinforcement learning models for stock portfolio optimization," *Intelligent Systems with Applications*, vol. 25, p. 200467, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667305324001418 See pages 8, 9.

[27] C. Betancourt and W.-H. Chen, "Deep reinforcement learning for portfolio management of markets with a dynamic number of assets," *Expert Systems with Applications*, vol. 164, p. 114002, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420307776 See pages 8, 9.

[28] X. Wu, H. Chen, J. Wang, L. Troiano, V. Loia, and H. Fujita, "Adaptive stock trading strategies with deep reinforcement learning methods," *Information Sciences*, vol. 538, pp. 142–158, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520304692 See pages 8, 9, and 10.

[29] O. Jin and H. El-Saawy, "Portfolio management using reinforcement learning," *Stanford University*, 2016. See pages 9, 10.

[30] S. Almahdi and S. Y. Yang, "An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown," *Expert Systems with Applications*, vol. 87, pp. 267–279, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417417304402 See pages 9, 10.

[31] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, no. 4, pp. 875–889, 2001. See pages 9, 10.

[32] Y. Zhang, P. Zhao, Q. Wu, B. Li, J. Huang, and M. Tan, "Cost-sensitive portfolio selection via deep reinforcement learning," 2020. [Online]. Available: https://arxiv.org/abs/2003.03051 See pages 9, 10.

[33] Y. Ansari, S. Yasmin, S. Naz, H. Zaffar, Z. Ali, J. Moon, and S. Rho, "A deep reinforcement learning-based decision support system for automated stock market trading," *IEEE Access*, vol. 10, pp. 127 469–127 501, 2022. See page 9.

[34] Y. Ye, H. Pei, B. Wang, P.-Y. Chen, Y. Zhu, J. Xiao, and B. Li, "Reinforcement-learning based portfolio management with augmented asset movement prediction states," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, pp. 1112–1119, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/5462 See page 9.

[35] Y.-J. Hu and S.-J. Lin, "Deep reinforcement learning for optimizing finance portfolio management," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, 2019, pp. 14–20. See page 9.

[36] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078 See page 9.

[37] W. F. Sharpe, "The sharpe ratio," *Journal of portfolio management*, vol. 21, no. 1, pp. 49–58, 1994. See page 13.

[38] M. Velay, B.-L. Doan, A. Rimmel, F. Popineau, and F. Daniel, "Benchmarking robustness of deep reinforcement learning approaches to online portfolio management," in *2023 International Conference on Innovations in Intelligent Systems and Applications (INISTA)*. IEEE, Sep. 2023, p. 1–6. [Online]. Available: http://dx.doi.org/10.1109/INISTA59065.2023.10310402 See page 13.

[39] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. Pmlr, 2014, pp. 387–395. See page 14.

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015. See page 14.

[41] H. Müller and D. Kudenko, "Improving the effectiveness of potential-based reward shaping in reinforcement learning," in *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '25. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2025, p. 2684–2686. See page 17.

[42] J. Murphy, *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*, ser. New York Institute of Finance Series. Penguin Publishing Group, 1999. See page 20.

[43] O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius, "Pink noise is all you need: Colored noise exploration in deep reinforcement learning," in *The Eleventh International Conference on Learning Representations*, 2023. See page 21.

[44] J. Hollenstein, S. Auddy, M. Saveriano, E. Renaudo, and J. Piater, "Action noise in off-policy deep reinforcement learning: Impact on exploration and performance," *Transactions on Machine Learning Research*, 2022. [Online]. Available: https://arxiv.org/abs/2206.03787 See page 21.

[45] J. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf See page 25.

[46] G. Chaucer, *The Canterbury Tales*, 1387-1400. See page 34.

[47] K. Kirtac and G. Germano, "Leveraging LLM-based sentiment analysis for portfolio allocation with proximal policy optimization," in *ICLR 2025 Workshop on Machine Learning Multiscale Processes*, 2025. [Online]. Available: https://openreview.net/forum?id=n64e2mKzTg See page 36.

[48] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 1587–1596. [Online]. Available: https://proceedings.mlr.press/v80/fujimoto18a.html See page 37.