A WAVENURE e a tutti i suoi ragazzi, per la fiducia e il supporto.

A chi ritrovo a Zena, per tutto il resto.

Table of Contents

1. Introduction

Following Henry Markowitz's groundbreaking "Portfolio Selection" seminal paper published in 1952, Modern Portfolio Theory has been an active research area in the world of finance, and multiple alternatives and further developments of the core concept have been proposed ever since. However, hardware and algorithmic improvements, coupled with a rapidly increasing availability of financial and non-financial data have also shed light on a more data-driven approaches to the creation of profitable portfolios, rooted in the recognition of patterns in historical data exploiting methods from the fields of Machine Learning (ML) and Artificial Intelligence (AI). This research aims to explore the potential of Reinforcement Learning (RL) in this domain, a subfield of AI where an agent is trained by trial and error to behave intelligently within an environment. In this context, it translates to developing a model that autonomously makes investment decisions with minimal supervision by a human required. This thesis specifically examines the applications of this technology within the stock market, investigating whether the AI can learn to construct and manage portfolios that generalize to different compositions. After highlighting some of the shortcomings of previously adopted methodologies, a novel logic is proposed to train an agent to solve the problem of Generalized Portfolio Management. First, a comparisons between different deep learning architectures and reward systems is carried out, to highlight with approach displays the best learning behavior. Then, trained agents are benchmarked against some notable strategies. A steep improvement is observed as training progresses, and the out-of-sample tests provide evidence that some of the trained agents indeed over perform the alternative strategies. These findings highlight that the agents are able to autonomously learn key aspects of the task at hand, including the negative effect of transaction costs. As a consequence, this research showcases the promises of RL for new investment-strategy discovery and fully data-driven, quantitative asset management.

This thesis will be structured as follows:

Chapter 2 will outline the premises of the thesis, briefly discussing the role of ML in different areas of finance and introducing the reasoning that led this research towards the field of RL.

In Chapter 3 a formal introduction to RL will be delineated as well as some of its successful applications outside the scopes of finance. Specifically, this chapter will focus on the area of game simulations which have been greatly influential molding research in this field.

Chapter 4 will delve deeper in finance-related implementations of RL, and its role in the literature of asset management will be reviewed. This chapter will also present the FinRL open-source project that provided a very useful framework for the following experiments.

Chapter 5 will lay out the experimental setup with the aim of creating a deep RL asset manager. Contrarily to the reviewed literature, the ultimate objective of this work is training a stock-agnostic trading agent that can generalize what it learns to a variety of different assets, without overfitting to a limited pool of stocks. This chapter will include a description of the data preprocessing process, the main trading environment and the different agents trained.

Chapter 6 will then discuss the experimental results, comparing the different architectures and reward systems employed.

Finally, Chapter 7 will conclude this thesis, delineating the implications of this work and possible future directions for research in this area.

2. The Artificially Intelligent Investor

Machine Learning (ML)[1] has proven to be a transformative, even disruptive, force across diverse fields. Content recommendation systems currently shape our entertainment industry, image recognition software detects tumors with remarkable precision, and several automotive companies have introduced early versions of autonomous vehicles, to highlight just a few examples. At the foundation, ML algorithms aim to identify patterns within historical data, leveraging them to make predictions.

The efficacy of ML in finance becomes evident if one acknowledges that historical data retains valuable information for forecasting future behaviors of financial markets. Consequently, research in this domain becomes crucial to democratize such information, ensuring market efficiency on exchanges accessible to the wider public. Additionally, unlike discretionary portfolio managers who often base their decisions on personal judgement and intuition, ML-based techniques are deterministic and rooted strictly on data. Hence, given the right precautions, their outcomes are objective and can be consistently reproduced.

2.a Asset Pricing

Asset pricing, that is the study and determination of the correct or "fair" price of an asset, has been a focal point of research in finance for a long time, and continues to be extensively studied today. Since the proposal of the Capital Asset Pricing Model by William Sharpe (1964), there have been many theoretical advancements to explain what determines the price of stocks. Most notable among these is the three-factor model by Fama and French (1993) and the subsequent extensions to their work. While these results are grounded in transparent and interpretable theory, their

[1] Machine learning is a subset of artificial intelligence that involves the development of algorithms allowing computers to learn from and make decisions based on data.

reliance on simple linear structures raises questions about their ability to comprehensively capture the intricate drivers of stock prices.

2.b ML for empirical Asset Pricing

Gu et al. (2018) provided empirical evidence showcasing the potential benefits of ML in asset pricing. They demonstrated that state-of-the-art algorithms, including artificial neural networks[2] and regression trees[3], can model stock prices more efficiently than classical methodologies prevalent in much of the existing financial literature. The authors' experiments spanned a wide range of ML techniques, gradually moving away from the simple linear regression models, such as OLS, that have traditionally dominated research in this field. They found that a basic trading strategy based on the predictions derived from neural networks resulted in an out-of-sample Sharpe Ratio[4] (Sharpe, 1998) of 2.45. In contrast, the strategy based on OLS predictions yielded a Sharpe Ratio of 0.83. This difference underscores the capability of advanced algorithms to capture nonlinearities and intricate interactions amongst predictors. Such algorithms can identify more sophisticated relationships and relax the limiting assumptions inherent to basic linear models.

From a purely practical point of view, formulating a trading strategy based on the prediction of an asset’s future price presents an additional challenge for the investor: determining how to translate these predictions into tangible investment actions in the

---

[2] An artificial neural network (ANN) is a computational model inspired by the structure and functional aspects of biological neural networks. ANNs are generally composed of interconnected nodes or 'neurons' that process information. They are designed to recognize patterns and are commonly used in machine learning and artificial intelligence applications.

[3] Decision trees are a supervised machine learning algorithm used for both classification and regression tasks. They split the data into subsets based on feature values, making decisions at each node until a prediction is made.

[4] The Sharpe Ratio is a measure used in finance to assess the risk-adjusted performance of an investment. It is calculated by dividing the difference between the investment's return and the risk-free rate by the investment's standard deviation. A higher Sharpe Ratio indicates better risk-adjusted performance.

market. Moreover, independent forecasts are made for all considered assets, overlooking interrelations and co-movements among stocks. In the likely scenario a multi-asset portfolio needs to be built and actively managed, these decisions are left to the investor's discretion. As a result, a prediction-based framework may lead to decisions that are ultimately affected by the investor's personal expectations, believes or emotions, which are fundamentally not rooted in the data and hence potentially biased. Therefore, it becomes clear that asset management should not be reduced to a simple sequence of forecasts, but it is a more complex and delicate process that requires a more all-encompassing solution.
An answer to some of these problems is presented in the following section.

2.c Asset Allocation

Asset allocation refers to the process of distributing investments across various asset classes such as stocks, bonds, real estate, cash, and other investment vehicles, in a manner that reflects an investor's specific financial goals, risk tolerance, and investment horizon. Usually, the emphasis does not fall explicitly on a prediction task, but rather on how to make use of past data to design a profitable multi-asset portfolio, thus making the outcome of the models in this field closer to the general needs of asset managers. Henry Markowitz was notably the first to propose a mathematical framework to the concept of diversification through the notion of the efficient frontier in his pioneering paper "Portfolio Selection" (1952), laying the groundbreaking foundations for what would later become known as Modern Portfolio Theory (MPT), and providing a formal risk-return framework for investment decision-making. In practice, Markowitz demonstrated that an investor could in theory exploit historical correlations between a pool of stocks and their own historical returns to construct a diversified portfolio to maximize the expected return given a defined level of risk, expressed as the variance of such portfolio's returns. This process would be later formalized as canceling out assets' idiosyncratic risk, unique to individual stocks

and hence diversifiable, leaving the portfolio exposed to the non-diversifiable market risk only. The resultant portfolio is often referred as the mean-variance portfolio (MVP), given that the first 2 moments of returns are used as the sole measures of profitability and risk respectively.

Despite the undeniable significance of Markowitz's contribution to the field of finance, such method comes with a few remarkable shortcomings (e.g., Fabozzi, Huang & Zhou, 2010). Among the most eminent, stock returns and covariances need to be estimated from an unknown distribution with limited data, which are key for the optimization problem. In practice, returns are typically assumed to follow a Gaussian distribution and are often naively estimated using historical means. This assumption can be overly simplistic given the complexities of real-world financial data. Moreover, the model demonstrates excessive sensitivity to these input parameters, especially the expected returns (e.g., Merton, 1980; Maillard, Roncalli & Teïletche, 2010). This has led investors in practice to prefer more heuristic solutions that don't rely on expected returns, hence deemed more robust. One such heuristic is the minimum-variance portfolio (MINP), that is the only portfolio lying on the mean-variance efficient frontier that does not incorporate information about the expected returns. As the name suggests, it aims to find the allocation minimizing the expected portfolio variance. However, a limit of this approach is that it might result in highly concentrated portfolios. Therefore, a pragmatic alternative to this is an equally-weighted portfolio (EWP) which assigns identical weights to all considered assets. Albeit extremely simple, EWP has demonstrated noteworthy out-of-sample performance. In fact DeMiguel, Garlappi & Uppal (2009) conducted extensive backtesting across varied markets, sectors, and time frames from 1963 to 2004. Their findings showed that EWP strategy is never outperformed by other standard methods, including MVP and MINP. For all these reasons, MINP and EWP constitute useful baselines when evaluating alternative allocation strategies.

A second issue of the classical MVP framework is the model assumes a single-period investment horizon, meaning that investors need to optimize their portfolios for a specific, fixed period.

From a portfolio management perspective the consequences are twofold: a robust way of estimating the parameters must be defined, which might not be representative of future expectations, and secondly the asset manager needs to define in advance a common investment horizon for all the assets considered.

Given these premises and limitations, ML can be an valuable tool in this setting as well.

2.d ML for Asset Allocation

Zhang et al. (2020) leveraged deep learning[5] techniques to determine dynamic portfolio weights for a set of four US ETFs: the US total stock index (VTI), US aggregate bond index (AGG), US commodity index (DBC), and the Volatility Index (VIX). They employed an artificial neural network trained on daily prices and returns spanning from 2006 to 2020, with periodic retraining every two years. The network architecture was kept relatively simple, consisting of an input layer, a single Long Short Term Memory (LSTM)[6] (Hochreiter & Schmidhuber, 1997) hidden layer, and an output layer. This design aimed to directly optimize the portfolio's Sharpe Ratio using gradient ascent. As a result, the model generated a new portfolio allocation daily. The performance of this dynamic allocation was benchmarked against various fixed allocations and established baselines, notably the mean-variance portfolio. Zhang et al. demonstrated that their Deep Learning Strategy (DLS) consistently surpassed

[5] Deep learning is a subset of machine learning that utilizes neural networks with many layers (hence "deep") to analyze various forms of data. It excels in tasks like image and speech recognition, and has been pivotal in advancing artificial intelligence applications.

[6] LSTMs are a neural architecture specific for the processing of sequences which has been employed in multiple domains including natural language processing, video captioning, time series forecasting and more.

other strategies in terms of the Sharpe Ratio, even when transaction costs were factored in. Specifically, under the most rigorous transaction cost conditions, DLS achieved a Sharpe Ratio of 1.403, significantly outpacing MVP's 0.191 during the test period from 2011 to April 2020. Notably, the DLS exhibited exceptional adaptability during the COVID-19 market turmoil, swiftly shifting from a diversified holding to a predominantly bond-focused portfolio following the market downturn in February 2020.

Interestingly, this approach is quite different from more traditional methods in that it does not explicitly predict future asset movements as a preliminary step for portfolio optimization, but rather directly tackles the allocation task. As a result, DLS leverages the intricate function-approximation capabilities of deep learning, aligning closely to the asset manager’s end-goal. However, a limitation arises as a new allocation is proposed at each new day regardless of the current holding. It fails to keep in direct consideration the potential frictions linked to such rebalancing, primarily transaction costs. In fact, the model simply aims to find portfolio weights that maximize the daily risk-adjusted portfolio return, but can’t develop a long-term strategy that could ultimately lead to a higher Sharpe Ratio at the end of the considered period. In other words, the learning task overlooks the essence of asset management, which isn't merely a sequence of isolated actions, but is instead a sustained, cohesive task where decisions taken today will have consequences far into the future. While DLS holds promise, it misses these nuances, and thus, yet again, the problem calls for alternative solutions.

2.e Towards ML end-to-end for Asset Management

As ML methodologies become more prevalent in finance, the primary responsibilities of asset managers undergo a transformative shift. Traditionally, their role centered on continuously monitoring the market to identify lucrative investment opportunities and adjusting portfolio compositions in response to changing trading conditions.

Nonetheless, with the integration of ML, their focus pivots to a more model-centric approach. This entails data collection, searching informative predictors, defining learning objective, verifying the robustness of results, and ultimately deploying the refined systems into a live environment.

Nonetheless, for ML to be successfully employed end-to-end, it is crucial that the learning objective is comprehensively outlined and effectively framed as a sequential decision-making process. The task is further complicated if transaction costs are factored in and if the asset manager wants to revise its allocation at every time step. Indeed, as discussed by Neuneier (1995), portfolio management can be formalized as a Markov Decision Process, a theory that models control problems within stochastic environments and which will be properly illustrated in the next chapter. The key intuition laid out by Neuneier is that asset allocation is a multi-stage decision problem and may not be reduced to pure prediction. In fact, while there are components that are independent of the portfolio decisions, for example asset prices and other market factors, the overall value of the portfolio and its returns are clearly not. As noted by Filos (2019), decisions taken in the past will have a direct impact on the optimality of the future decisions. Therefore competent asset managers must not focus on maximizing immediate profits, but have to consider potentially negative effects of present decisions, as myopic optimal actions can lead to sub-optimal results in the longer term. In other words, the primary aim of portfolio management is not prediction or immediate profit maximization. Instead, portfolios are designed and refined with a clear, overarching, long-term goal, for example that of generating a high risk-adjusted return.

Luckily, Reinforcement Learning is the subset of ML that provides the framework to optimize Markov Decision Processes, even when no explicit model of the task is available.

The next chapter will provide an introduction to Reinforcement Learning, its key elements, recent evolution and some major applications.

3. Introduction to Reinforcement Learning

Reinforcement Learning (RL) is a subfield of AI where an agent is trained by trial and error to behave intelligently within an environment. The nature of the environment is not of direct interest for the definition of the RL framework. It could be a chess board, an ATARI gaming simulator or a stock trading environment, just to name a few possibilities. The key is that an agent will have to learn how to successfully traverse such environment directly by interacting with it. In fact, RL is fundamentally different from Supervised Learning[7], as there are no labeled ground truths that the model must learn to predict. Instead, the outputs of the trained model are actions for the agents to take given the current state of the environment. At each time step the agent observes the environment, takes an action and receives feedback from the world. Specifically, it is rewarded when a desirable result is obtained and punished when its actions lead to an undesirable outcome. For example, training a RL agent to play chess, a negative reward might be received when the agent loses a game, and a positive reward is produced when it wins. The agent's main objective is maximizing the total amount of rewards received.

RL is especially interesting in the cases where it is not clear what a good strategy might be, but it is still easy to judge when an agent is performing well and when it is not. While it is certainly not trivial to design an effective stock trading strategy, it is relatively straightforward to assess its performance, for example if it is making money or producing a high risk-adjusted return.

In the early stages of learning, the agent will start exploring, taking completely random actions and occasionally receiving a positive reward. This signal will drive the learning, as the weights of the model that maps observed states to actions are

[7] Supervised learning is a type of machine learning where an algorithm is trained on labeled data, meaning the algorithm is provided with input-output pairs. The goal is to learn a mapping from inputs to outputs and make predictions on new, unseen data based on this learned mapping.

updated with the goal of maximizing the cumulative rewards, aiming to ultimately develop a successful policy to solve the assigned job.

A good policy calls for a sequence of actions to be made and for the agent to rely on the fact that it will keep on exploiting the learned strategy in the following time steps. At the same time, it is central that new policies are also searched, so that the agent doesn't get stuck acting according to a suboptimal strategy. This paradigm is often referred to as the "exploration-exploitation" tradeoff.

Additionally, the learner must solve the "temporal credit assignment" issue, that is determining how to allocate praise and responsibility among the inputs and outputs that culminate in the final reward signal, which is often noisy and delayed (e.g., Tesauro, 1994).

This "awareness" of the continuous nature of the problem makes RL a perfect candidate for the task of active asset management, and well-suitable to overcome the limitations connected to previously discussed approaches.

The following sections will delve deeper properly formalizing the RL framework[8], highlighting why it holds great promise in this domain and how it has already proven to be a potential solution worth researching further.

3.a Markov Decision Processes

The concept of a Markov Decision Process (MDP) is closely related to RL. More precisely, MDP is a mathematical framework used to describe an environment for RL. It provides a formal description of the decision-making situation an agent faces, especially when outcomes are partly random and partly under the control of the agent.

[8] The contents of the following sections are highly inspired by the excellent overview present in chapter 3 of Sutton and Barto's "Reinforcement Learning: An Introduction", 2018, MIT Press.

More precisely, the agent constitutes the decision maker and the learner, while everything outside the agent is referred to as the environment. These are essential constituents of the framework of MDP and RL, and interact continually.

At each time step t, the environment is in some state S_t. The agent receives some representation of the state and chooses an action A_t that is available. At the following time-step, the process responds by moving into a new state S_{t+1}, and giving the agent a scalar reward R_t. This gives rise to a trajectory of states, actions and rewards as time steps progress, starting like the following:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2 \ldots \tag{3.1}$$

This thesis will focus on cases where the agent-environment interaction breaks naturally into subsequences, called episodes. In a chess environment an episode might represent a full game, whereas in a portfolio management application an episode may end if the portfolio value goes to zero, or after a predetermined time horizon.

Figure 3.1 provides a simple visual representation of the agent-environment interaction in a MDP.

It is crucial to stress that new state S_{t+1} is partially dependent on the agent's action and partially random. Moreover, the state is assumed to possess the Markov property. That is, the probability of each possible value for S_t and R_t depends only on the immediately preceding state and action, S_{t-1} and A_{t-1}, and, conditionally on them, it is independent on earlier states and actions. Also, this thesis will focus on finite MDP, where the sets of states, actions, and rewards all have a finite number of

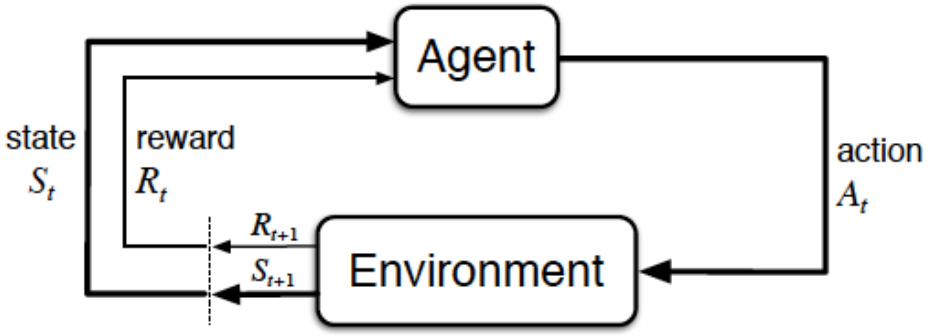

Figure 3.1: Agent-environment interaction in a Markov decision process (source: Sutton & Barto, 2018, page 48)

elements. This means that the random variables R_t and S_t have well defined discrete probability distributions as shown below[9]:

$$p(s', r \mid s, a) = \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{3.2}$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$. This is known as dynamics of the MDP.

Therefore, an MDP is a tuple of 4 elements: a State Space, an Action Space, a State Transition Function and a Reward. Each of these components plays a pivotal role in shaping RL applications, as detailed in the subsequent sections.

3.b States, Actions and Rewards

A State Space ($\mathcal{S}$) encompasses the entirety of potential states the system can assume. States can be thought of the different scenarios or situations the agent can encounter. For instance, in chess, a state could correspond to a unique board configuration, while in finance, it might depict prevailing market conditions. From a modeling standpoint, crafting a suitable representation of the state of the

[9] Please note that Random Variables (RVs) are denoted using uppercase letters, specific realizations of these variables are represented in lowercase and the sets containing all possible realizations are indicated with capital cursive characters. As an illustration, S_t signifies the Random Variable for the state at time t, s denotes a particular realization of that state, and finally $\mathcal{S}$ connotes the set of all possible states.

environment is paramount. In the context of the game of chess one could deploy a matrix to represent the board, where each matrix element corresponds to a board cell, and distinct numerical values signify different chess pieces. However, real-world scenarios often present additional complexities, and for example the state of the environment might be only partially observable. Consider a stock-trading application: the agent lacks comprehensive insight of all the relevant factors determining the current dynamics of the market and driving trading prices. These may include, but not be limited to historical prices, trading volumes, market sentiment, macroeconomic and geopolitical circumstances, but also undisclosed company-specific news or unpredictable global events. Instead, it would have to base its actions only on an mere subset of available data, such as past price movements. This extension has led to the definition of Partially Observable Markov Decision Process (POMDP), a generalization of the classical MDP.

An Action Space ($\mathcal{A}$) comprises the set of all possible actions available to the agent. Again, crafting an RL experiment requires a clear definition of the viable actions in any given state and what form they take. In the chess example, given there are 64 cells on the board, ignoring special moves like castling or en passant there are $64 \times 64 = 4069$ possible “from-to" move combinations. An action could be encoded as a probability distribution over the 4096 possible moves. Naturally, not all of them would be feasible in any situation, therefore additional mechanisms would have to be introduced, but this representation still constitutes a simple starting solution to the problem. This scenario exemplifies a discrete action space, as there is a finite number of distinct actions the agent can take. By contrast, in a portfolio management context the agent might be required to decide how to distribute available funds across different assets. Therefore, for a long-only agent, each action can be represented as a vector of real-valued numbers, all of which are positive and sum up to 1. Hence, the action space in this example is clearly continuous. Certain RL algorithms, like the acclaimed Q-Learning (Watkins & Dayan, 1992), are tailored for

discrete action spaces. Others, such as Deterministic Policy Gradient (Silver *et al.*, 2014), Trust Region Policy Optimization (Schulman *et al*., 2015), and Proximity Policy Optimization (Schulman *et al.*, 2017) (elaborated upon in subsequent sections), are versatile enough to handle both discrete and continuous spaces. Consequently, algorithm selection emerges as a crucial aspect of RL experiment design.

A State Transition Function is the probability that action a taken in state s will lead to state s':

$$p(s' \mid s, a) = \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \qquad (3.3)$$

In practice, a significant portion of RL research is focused on model-free RL. In this framework, the agent doesn't have access to a model of the environment and doesn't attempt to explicitly learn its dynamics (refer to Equation 3.2) nor the State Transition Function directly. As a consequence, the agent doesn't aim to estimate transition probabilities and use them to plan future actions. Instead, it learns directly via interaction with the environment. In other words, the agent is not explicitly trying to predict its opponent's next move or future price fluctuations in the contexts of chess and finance respectively. Instead, it searches for the optimal action directly for the purpose of reaching its ultimate goals.

A Reward (R) refers to the immediate numerical feedback received after moving from state s to state s' after taking action a. The process of meticulously crafting this feedback system is termed "reward engineering" or "reward shaping". This step is pivotal in the formulation of an RL experiment, as the agent's goal is to maximize the total amount of reward it receives. More precisely, the agent seeks to maximize the expected value of the discounted sum of rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \text{ , } with\ \gamma \in [0,1] \qquad (3.4)$$

where γ is a parameter called the discount rate. G_t is often called "return".

The discount rate implies that rewards obtained today are more desirable than rewards received ahead in the future, but the closer γ gets to 1 the more future rewards must be taken into account, making the agent more farsighted (common values are 0.95, 0.99 or 0.999). This highlights how the agent is not focused on maximizing the immediate reward, but rather is oriented towards optimizing cumulative rewards over the long term.

It is clear that the nature and frequency of rewards can significantly influence an agent's learning. In the game of chess, a straightforward reward system might offer a positive reward for a win and a negative one for a loss. This, however, results in a sparse feedback: the agent must wait until the game's conclusion to receive any signal. This delayed feedback can sometimes significantly slow the learning process. Therefore, one may think to reward and punish the agent based on the capture or loss of individual pieces. While this would introduce more frequent feedback, the reward signal should always communicate to the agent *what* it needs to achieve, and not *how* that needs to be achieved. The agent might in fact find ways to take the opponent’s pieces even at the cost of ultimately losing the game. Thus, one should not provide the agent with prior knowledge about the task, as the researcher’s own personal biases might hinder the learning process. The agent should be able to find its own solutions to the problem based solely on its interaction with the environment. In the context of portfolio management a simple and frequent reward mechanism then could involve a signal that is proportional to the change in portfolio value, rewarding an increase and punishing a decline in the value of assets under management. This direct relationship ensures that the agent's objective aligns with maximizing the portfolio's value, without biasing it towards specific potentially suboptimal strategies.

3.c Optimal Policies and Policy Gradient Methods

Maximizing cumulative rewards requires finding an optimal policy. Formally, a policy maps states to the probabilities of choosing each potential action. It is generally denoted by $\pi(a \mid s)$, which is the probability that an agent following policy π selects action a when in state s. Furthermore, the value function of a state s under a policy π, indicated $v_\pi(s)$, is the expected return when beginning in s and subsequently following π:

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S} \qquad (3.5)$$

For completeness, the action-value function $q_\pi(s, a)$ expresses the expected return for taking action a in state s and then behaving according to policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \qquad (3.6)$$

An optimal policy, symbolized by π_*, maximizes the value function for every state, and there is always at least one policy that is better than or equal to all other policies. Finding π_* is the ultimate objective of RL methods.

It is worth specifying that a consistent portion of RL literature focuses on value function estimation (approximating a parametrized function for Equation 3.5) or action-value function estimation (approximating a parametrized function for Equation 3.6). Q-learning and Deep Q-networks (Mnih *et al.*, 2013), briefly discussed in section 3.e, are an example of the latter. These approaches are termed value based methods. However, despite the different take on the task, a more comprehensive explanation for Policy Gradient methods and the Proximal Policy Optimization algorithm will provided given their relevance today and because they form the basis for the experiments discussed in chapters 5 and 6.

Policy Gradient (PG) algorithms belong to the category of RL schemes that focus on approximating the stochastic policy directly, setting them apart from value based methods. Then, a representation of the state is given as input to the policy and the output is a probability distribution over possible actions. This family of algorithms is termed policy based algorithms. Like value based methods, policy based algorithms usually make use of a function approximator with its own learnable parameters, often an artificial neural network, where the weights of the neural network constitute the policy parameters. During training, these weights are updated in line with the gradient of the expected return (refer to Equation 3.4) with respect to the policy parameters. Sutton *et al.* (1999) provided a clear theoretical foundation for policy gradient methods, deriving a formulation for the gradient that can be expressed in a form suitable for estimation from experience, augmented by an approximate action-value or advantage function, and proving that this method is convergent to a locally optimal policy.

3.d Proximal Policy Optimization

One of the most prominent algorithms in the domain of continuous control is Proximal Policy Optimization (PPO), introduced by Schulman *et al.* (2017), from the OpenAI research laboratory. Its acclaim stems from to its relative ease of implementation, robustness to hyper parameters choice, capability to manage continuous action spaces and sizable state spaces, and its impressive performance on a variety of benchmark tasks for continuous control, such as several MuJoCo environments (Todorov, Erez & Tassa, 2012). PPO is presented as a new family of policy gradient methods that builds upon Trust Region Policy Optimization (TRPO) (Schulman *et al*., 2015), a learning algorithm which maximizes a “surrogate” objective function subject to a constraint on the size of the policy update. The constraint is expressed in terms of Kullback-Leibler divergence (e.g., Cover & Thomas, 1991) between the current and updated policy. The main intuition is that very large updates to the policy can

lead to catastrophic drops in performance, and recovery from such a bad update can be challenging. This is because the landscape of the objective is notably non-convex and irregular, presenting multiple local optima. Schulman *et al.* proposed PPO as an attempt to develop an alternative algorithm that emulated TRPO's monotonic improvement, but optimizing a simpler objective and using first-order optimization only. They achieved this introducing an unconstrained clipped surrogate objective (refer to equation 3.7[10]) and demonstrating that PPO outperforms its counterparts, including TRPO, in various RL settings.

$$L(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{3.7}$$

PPO is an actor-critic method, which in practice means that two independent networks are trained. The actor refers to the policy part of the method, and is responsible for selecting an action based on a given state. For continuous action spaces it produces a mean and a standard deviation (or a vector of the two when an action requires multiple values), and then the action is sampled from a normal distribution parametrized by these network outputs. The critic, on the other hand, refers to the value function part of the method and evaluates the action taken by the actor, crucial for stabilizing the policy update. In practice, this part of the network estimates the expected cumulative reward from state s when following the current policy. In a sense, actor-critic methods can be seen as combining the advantages of value based and vanilla policy based algorithms. Moreover, often a feature extractor is shared by the actor and the critic, which processes the state before feeding a common representation to 2 separate networks.

Interestingly, subsequent studies, such as Engstrom *et al.* (2020), have risen

---

[10] In the presented equation $r_t(\theta) = \dfrac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$, $\hat{A}_t$ is the estimator of the advantage function at time t and ϵ is a hyperparameter. For an in-depth understanding of the objective, readers are encouraged to consult the original paper.

questions regarding the factors underpinning PPO's efficacy, hinting that its success might be more attributable to code-level optimizations rather than the core algorithm presented in the original research paper. Nonetheless, PPO remains one of the state-of-the-art and most widely used RL algorithms today.

3.e Artificial Neural Networks, Deep Learning and Games

Early RL research concentrated on tabular solution methods, applicable when state spaces are small enough where value functions could be represented as tables with each entry representing the expected return for a distinct state. Naturally, as state spaces grow larger this approach quickly becomes infeasible, which firstly led researchers to resolve to manually crafted features that aimed to retain relevant information about the task at hand, often seeking insights from experts intimately familiar with the task itself, such as chess masters for a chess-playing agent. However, there were some meaningful exceptions to this approach. Notably, in 1994 Gerald Tesauro presented TD-Gammon, a groundbreaking RL program that learned how to play the game of backgammon from little to no prior knowledge about the game, reaching a level near to that of grandmasters at the time. The game is played on a board divided into four quadrants, with each player having 15 pieces that move between 24 triangular points according to the roll of two dice. The primary goal of backgammon is to move all of one's own pieces into their home board and then bear them off (remove them from the board) before the opponent does. The combination of chance and strategy made it a perfect test bed for RL. TD-Gammon was an impressive achievement, given that backgammon is a complex table game with a secular history dating back to 17th-century England (O'Bryan, 2019), and is still highly competitive today just like it was at the time the paper was published. Most importantly however, backgammon has an enormous number of distinct possible states ranging around 10^{20}, and a very large average number of possible moves per turn which is significantly greater than chess and checkers. TD-Gammon cleverly

tackled this obstacle by having an original representation of the board processed through a multilayer artificial neural network (ANN) estimating the probability of winning the game in any given state. At each time-step, the agent compared all possible moves, selecting the one resulting in the state yielding the maximum probability of winning. The agent was trained by self-play using the TD(λ) learning algorithm (Sutton, 1988), effectively deploying the ANN as a non linear function approximator. The success of TD-Gammon provided early evidence that RL combined with neural networks could be a powerful approach for solving complex problems, and paved the way for further research and advancements in the field.

However, more complex state spaces often require representations with an exceptionally high number of dimensions to be thoroughly represented, like images which can be encoded as large matrices of pixels. Traditional neural networks struggled to handle such large spaces, and for this reason handcrafted features arbitrarily restricting the focus of the agent remained widely used in many subsequent applications, despite the success of TD-Gammon's approach. Much of the renewed interest in RL however can be credited to the influential work of Mnih *et al.* (2013): the researchers from Google's AI research lab DeepMind demonstrated that advances in supervised deep learning, specifically computer vision, could be adeptly leveraged in RL to learn control policies directly from high-dimensional sensory input. In practice, they trained RL agents to play a diverse range of classic ATARI 2600 games directly from raw pixel inputs. ATARI 2600 is a home video game console developed and produced by Atari, Inc. which was released in September 1977 and led to the widespread popularization of iconic early videogames such as Pong, Breakout Space Invaders and many more. In this work, agents learned to play a variety of games available in an ATARI 2600 simulator using a novel algorithm coined Deep Q-Network (DQN). DQN combined the aforementioned Q-learning

algorithm and Convolutional Neural Networks (CNNs)[11] (LeCun *et al.*, 1998) for automatic feature extraction, eliminating the need for handcrafted features. In fact, CNNs had already shown to be highly effective for image processing in the context of supervised ML, and had been successfully applied to the tasks of image classification (e.g., Alex Krizhevsky, Sutskever & Hinton, 2012) and object detection (e.g., Sermanet *et al.*, 2013) without the need of heavy feature engineering. However, RL research still had not experimented with these progresses before. In the work by Mnih *et al.*, the reward signal provided to the agent consisted in the simple change in game score at any given moment and each state was represented as a sequence of the most recent frames. These images were processed by a series of convolutional and fully connected layers[12] trained to approximate the optimal action-value function (refer to Equation 3.6) through the DQN algorithm. Therefore, in each time-step the agent would select the action associated with the highest expected cumulative future reward. For a detailed discussion on the algorithm, readers are referred to the original paper.

In practice, the agent only observed what was directly visible on the screen and had to learn to generalize from its training to handle a wide variety of game situations without having access to the emulator internal state, just like a human would do. In a subsequent study, Mnih *et al.* (2015) showed that their agents reached superhuman level across multiple games (including Pong, Breakout and Space Invaders), significantly outperforming alternative ML approaches. This work underscored the striking potential of integrating deep learning architectures with RL, effectively sparking increased enthusiasm in this field, and is even credited as one of the pioneers of deep reinforcement learning.

---

[11] Convolutional Neural Networks (CNNs) are a class of deep learning algorithms primarily used for image processing and recognition. They employ convolutional layers to automatically and adaptively learn spatial hierarchies of features from input images.

[12] In the context of deep learning, a fully connected layer is a neural network layer where each neuron is connected to every neuron in the previous layer.

From the examples above it is clear that games and simulations have been among the most influential and well-documented domains of application of RL. In fact, they serve as ideal proving ground for the capabilities of the framework, due to their well-defined boundaries and the ease of their evaluation and control, yet posing unique challenges and allowing for gradual increase in complexity. For the sake of completeness, the following section will include other significant RL gaming implementations pioneered by the DeepMind research lab that have been influential in shaping subsequent research and have demonstrated ambitious objectives.

3.f DeepMind's Alpha Agents

AlphaGo (Silver *et al.*, 2016) and its successor, AlphaGo Zero (Silver *et al.*, 2017), were developed by Google DeepMind to master the ancient board game of Go. Originating in China over 2,500 years ago, Go is played on a 19x19 grid where the aim is to dominate more territory than the opponent. Players take turns placing the stones on vacant points of the board. Once placed, stones do not move, but they can be captured if entirely surrounded by the opponent. When there are no more beneficial moves left the game ends and captured stones and controlled areas are counted to determine the winner.

Despite its simple rules, the game is extremely complex and presents an exceptional strategic depth. The number of unique possible states greatly surpasses those of backgammon and chess, being estimated to be in the orders of 10^{170} (for reference, there are approximately 10^{80} atoms in the whole observable universe - Wikipedia, 2023). Given this complexity, traditional brute-force search strategies, which had been successful for games like chess as demonstrated by IBM's Deep Blue (Campbell, Hoane Jr & Hsu, 2002), are not viable for Go.

The first model to reach professional-level was AlphaGo. It was first exposed to numerous amateur games to gain an understanding of the main rules, and was

subsequently trained via a combination of Monte Carlo Tree Search[13] and an actor-critic RL approach. In October 2015 AlphaGo beat the reigning three-time European Champion and in March 2016, it won against eighteen-time world champion Mr Lee Sedol in Seoul, South Korea, in a series of matches watched by over 200 million people worldwide. This victory earned AlphaGo a 9 dan professional ranking, the highest in Go. Notably, Mr. Lee Sedol described AlphaGo's playing style as "creative." Intriguingly, a particularly unconventional move (move 37) in its second match against Sedol challenged centuries of established strategy and ultimately led to AlphaGo's victory.

In 2017, DeepMind introduced AlphaGo Zero, an even more advanced model. Trained entirely from scratch using RL end-to-end, without any human-derived knowledge, AlphaGo Zero surpassed even AlphaGo, underscoring the immense potential of the approach (Google Deepmind, n.d.).

DeepMind's innovation did not stop with AlphaGo Zero. In late 2018, they unveiled AlphaStar (Vinyals *et al.*, 2019), the first AI to defeat a top professional player in StarCraft II. StarCraft is a real-time strategy video game introduced by Blizzard Entertainment in 1998, with its sequel, StarCraft II, launching in 2010. As one of the most enduring esports, players have been competing in StarCraft tournaments for over two decades.

The complexity of StarCraft, both in terms of rules and gameplay, poses unique challenges for AI. Players must gather resources, build units and strategically update their infrastructure. This demands both macro-management of the game's economy and micro-management of individual units, requiring a delicate balance between immediate and long-term objectives. Unlike in games like chess or Go, the agent must deal with imperfect information as players need to "scout" and explore the map

[13] Monte Carlo Tree Search is a heuristic search algorithm used primarily for decision-making in games. It builds a search tree by repeatedly simulating random paths (or play-outs) from the current state to a terminal state, then uses the results of these simulations to make statistically informed decisions about the most promising moves.

to gather crucial information that is otherwise hidden The real-time nature of the game, with no subdivision in turns, calls for instantaneous decision-making. Finally, the extremely large action space, with hundreds of elements being controlled at once and approximately 10^{26} legal actions possible at every time-step, further amplifies the challenge.

Like AlphaGo, training consisted of a supervised preliminary phase where agents learnt by imitation observing anonymised human games. This was followed by a multi-agent reinforcement learning process inspired by the previously mentioned PPO algorithm. Here, a training league was created, with multiple agents competing and learning from each encounter. As training progressed, new agents were dynamically added branching from existing competitors, allowing for a continuous exploration of the huge strategic space. After two weeks of rigorous training, with agents learning from many thousands of parallel instances of the game, AlphaStar was tested against Mana, one of the world's strongest players. The AI won by 5 games to 0, a crushing victory that further pushed forward the boundaries of AI research and what was previously considered possible in the sphere of RL (Google Deepmind, 2019).

4. Reinforcement Learning for Portfolio Management

The RL paradigm adequately addresses the concerns described throughout chapter 2, and could greatly complement the research in finance and especially asset management modeled as a sequential decision-making problem. RL allows to train an agent to handle all relevant decisions concerning the management of assets, namely where to trade, at what price and at what quantity, even if a model of the market is unavailable. In fact, it overcomes the common misconception that Supervised Learning encompasses the entirety of ML, and that every task must be framed necessarily as a prediction task. Instead, RL provides a clear framework for sequential decision making that is specific to training a goal-seeking AI from the ground up, and it is also sufficiently flexible to leave plenty of freedom for researchers to set up their own experiments in multiple different ways, even when sharing a common objective.

Naturally, given the generality of the approach, finance is only one of the areas RL has been implemented in. There are numerous applications in domains such as healthcare (e.g., Yu et al., 2021), robotics (e.g., Kober, Bagnell & Peters, 2013; Singh, Kumar & Singh, 2022), computer vision (e.g., Le *et al*.,2022), intelligent transportation systems (e.g., Haydari & Yılmaz, 2020) and many more (e.g., Li, 2017).

The subsequent sections will briefly discuss some of the main obstacles of research in finance and then will dive into the use of Deep RL in this domain in the existing literature. More precisely, they will provide a review of some of the most significant model-free approaches for asset trading and control of diverse investment portfolios, with the aim to present the current state of the discipline in the field of active portfolio management. Finally, the FinRL open-source project and its value for research in this area will be examined.

4.a The Research - Profit Dilemma

In a cutting-edge field like ML transparency is key. This is especially true as new algorithms and models frequently emerge, rendering old approaches obsolete. This could be due to advances in algorithms, more efficient models proposed by the literature, new empirical findings or improvements in hardware and computing capabilities. This makes it crucial in theory to keep track of previous methodologies to allow for easy comparisons between novel and previous approaches. This is particularly important given that setting up experiments often requires extensive coding skills and can be an extremely tedious and time-consuming process. In addition, access to the specific datasets is crucial to replicate experiments.

However, the financial sector presents unique challenges to this ideal of transparency. In fact, it is usually not in the interest of firms like hedge funds or asset management companies to openly disclose their research or data, as that might undermine the effectiveness of their adopted methods. This can be linked to the concept of financial markets not being perfectly efficient in practice (Fama, 1970). Markets may not be efficient in a semi-strong sense, meaning that publicly available information could still be exploited to produce profits, for example through an innovative ML model. This of course would discourage firms from publishing their findings and software, as, once these models are made entirely public, it is fair to assume that they would be immediately deployed by competitors as well, further challenging their capability to consistently beat the market. A similar argument could be made regarding efficiency in a strong sense and the reluctance to disclose proprietary or nonetheless difficult-to-obtain data.

All the aforementioned considerations are especially relevant for financial RL applications, which are notably challenging to implement from scratch. This is exemplified by the fact that, contrarily from a supervised ML problem, data collection and preprocessing are necessary but not sufficient preliminary steps to train a RL model, as the logics of the simulated environment also need to be programmed, thus

introducing an additional layer of complexity.

For all these reasons, accessible data and code are particularly valuable in this area of research, as they allow to replicate and build on previous works, creating new knowledge when research papers are not enough. Together with the aim of democratizing information in financial markets, these are some of the shared motivations of the following works, including the open-source project Financial Reinforcement Learning (FinRL), which constitutes a representative embodiment of this renewed philosophy.

4.b Deep RL in the Cryptocurrency Space

An appreciable portion of research in this domain is carried out in cryptocurrency markets. Jiang, Xu & Liang (2017) highlighted two distinctive features of cryptocurrencies that differentiate them from traditional financial assets: decentralization and openness. These characteristics make the cryptocurrency market an convenient platform for testing algorithmic portfolio management strategies due to its increased accessibility.

This section will delve into two prominent applications of Deep RL in the domain of portfolio construction for cryptocurrencies. The first application approaches portfolio management with actions in a continuous space, while the second one adopts a discretized action methodology. For readers keen on delving deeper into this subject, Millea (2021) offered a comprehensive review of Deep RL applications in trading, with a notable focus on the cryptocurrency market, including a detailed segment on portfolio management.

The work by Jiang, Xu & Liang constituted a fundamental contribution in this area of AI applied to quantitative finance. The authors introduced their research discussing the limitations of price-prediction-based trading rules and presented a general deep RL framework for portfolio management as a fully machine learning solution. Finally, they tested their RL-based strategy on a portfolio of twelve cryptocurrencies. More

precisely, they employed the Deterministic Policy Gradient (DPG) (Silver *et al.*, 2014) learning algorithm to tackle portfolio management as a continuous control challenge. This approach sidesteps the pitfalls of action space discretization, notably the unpredictable risks from arbitrary discretization and scalability issues.

The core of the proposed framework is the Ensemble of Identical Independent Evaluators (EIIE) topology. In this setting, each IIE is a neural network which observes the past price movements of an individual asset and evaluates its potential growth for the immediate future. This evaluation score is discounted by the size of the necessary weight change for the asset in the portfolio and, together with the evaluation score of all other assets considered, it is input to a softmax[14] layer, termed the "voting" layer. The outcomes of the voting layer are the new portfolio weights for the subsequent trading period. The concept of EIIEs allows for easy scalability and increase in the number of assets under management: all IIEs share the same network weights and are simply related via the voting layer, meaning that introducing additional assets just requires increasing the number of IIEs.

The reward mechanism employed consisted in the logarithmic rate of return of the portfolio, thus the agent's job was equivalent to maximizing the average portfolio logarithmic cumulated return.

Another key element introduced was the Portfolio Vector Memory (PVM) mechanism. PVM stored previous portfolio weights and made them part of the observable state, in order to allow the RL agent to consider the effect of transaction costs to the portfolio value. In other words, at any new time-step the agent could access information about the recent history of the assets under management, but also regarding current portfolio composition and available liquidity. As a result, agents could exploit this knowledge and select the desired allocation also keeping into account the impact of

[14] The softmax operator is a mathematical function that transforms a vector of real numbers into a probability distribution summing to 1. It takes the following form:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \text{ for } i = 1,2,...,n.$$

transaction costs on portfolio profitability. This is because an optimal allocation might lead to suboptimal portfolio performance if the cost of rebalancing is not accurately accounted for.

Additionally, the authors presented an Online Stochastic Batch Learning scheme for both offline and online efficient training of the new agents.

In their empirical tests, three neural architectures were evaluated for the IIEs: Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) (Werbos, 1988), and Long Short Term Memory (LSTM). At each time-step, an IIE was fed the last 50 half-hourly normalized prices of a specific asset. These inputs underwent processing either by two convolutional layers or a recurrent layer with 20 hidden units. The final “voting” layer received both the current portfolio allocation and the outputs from the preceding layers.

The authors' experiments spanned data from July 2014 to March 2017. Each of the three backtests involved 12 assets, including Bitcoin (BTC) and the top 11 non-BTC assets by volume preceding each backtest. The IIEs were tested against several benchmarks and trading strategies, including the classical buy and hold strategy (which for small periods can be considered equivalent to the equally weighted portfolio). The EIIEs consistently outperformed, with the CNN resulting in the highest Sharpe Ratio in the first two backtests (0.087 and 0.059 respectively) and the RNN in the third, last one (0.082). For context, the buy-and-hold strategy yielded Sharpe Ratios of -0.015, 0.004, and 0.036 across the three tests.

Although these performances remain impressive to this day, the most remarkable contribution stems from the generality and elegance of the proposed framework, flexible but simple enough to be conveniently transposed to different markets and asset classes without loss of generality.

There are also some works in the cryptocurrency space that experiment with the Deep Q-network (DQN) introduced in section 3.e (e.g., Bu & Cho, 2018). Although DQNs require a discretization of the action space, and thus are exposed to the

pitfalls highlighted by Jiang, Xu & Liang, they still represent interesting takes on the task.

Notably, Lucarelli & Borrotti (2020) introduced a deep Q-learning framework tailored for cryptocurrency portfolio management. Their innovative approach employed a multi-agent system, where each asset in the portfolio was overseen by a dedicated local agent trained via deep Q-learning. The authors experimented with three distinct deep Q-learning techniques for these agents: Deep Q-Networks (DQNs) (Mnih *et al.*, 2013), Double Deep Q-Networks (D-DQNs) (Van Hasselt, Guez & Silver, 2016), and Dueling Double Deep Q-Networks (DD-DQNs) (Wang *et al.*, 2016).

In this setup, each agent focused on a state specific to its assigned asset and decided on an action: whether to retain the current asset amount or determine a buy/ sell quantity. For their experiments, agents were fed the last 24 hourly closing prices of a given cryptocurrency. Due to the challenges posed by continuous action spaces for Q-learning algorithms, the action space was discretized into 61 distinct actions: holding, 30 buy quantities, and 30 sell quantities.

Local agents earned rewards based on either the asset's nominal return or its Sharpe Ratio. These individual rewards were then aggregated to produce a global reward. In particular, authors experimented with both a simple sum of nominal net returns (R1) and a linear combination of Sharpe Ratio and portfolio net return (R2). Notably, this global reward was shared across all local agents and was used to update their network parameters. The deep Q-learning techniques largely shared a common network architecture: three convolutional layers succeeded by a 150-neuron fully connected (FC) layer. While DQN and D-DQN culminated in a 61-neuron output layer, DD-DQN diverged by incorporating two separate 75-neuron FC streams, one for value function estimation and the other for advantage function.

The authors conducted their experiments using hourly closing prices of four major cryptocurrencies: Bitcoin, Litecoin, Ethereum, and Ripple from July 2017 to December 2018. The performance of all deep Q-learning frameworks was assessed

over ten consecutive test periods spanning July to December 2018. Among the tested models, the DQN with the R2 global reward metric emerged as the top performer, yielding the highest average daily return of 4.67%. Furthermore, when tested against an equally weighted portfolio (EWP) from November to December 2018, the DQN-R2 model showcased a superior Sharpe Ratio of 0.202, compared to EWP's 0.043.

4.c Deep RL in Equity Markets

Deep RL methods have also been rigorously examined in the context of the stock market. Meng & Khushi (2019) and Mosavi *et al.* (2020) each offered overarching reviews of RL applications in finance and economics, incorporating various examples from stock trading and equity portfolio management. Meanwhile, Sato (2019) delivered a more specialized examination, focusing on model-free RL strategies specifically tailored for financial portfolios, with a primary emphasis on the equity market.

This section will describe some of the state-of-the-art Deep RL implementations to the task of managing portfolios of stocks, underscoring the novel design choices of the experiments. Again, an example for discrete and one for continuous action spaces will be provided.

In their study, Jang & Seong (2023) sought to bridge the gap between deep learning and traditional financial theory. They observed a notable oversight: while stock return covariances play a foundational role in Markowitz's modern portfolio theory, most of previous deep RL works in this area failed to consider this type of data. Thus, they proposed a Deep DPG agent observing both information regarding recent price

movements in the form of technical indicators[15] and the co-movement across assets through estimated correlation matrix between stock returns. At each new step, the technical indicators and the correlations were combined via matrix multiplication to create a novel tensor to efficiently model complex relationships between assets. The fused tensor was then processed by a 3D convolutional layer, and the resulting high-dimensional output was condensed using Tucker decomposition[16], obtaining a core tensor. Finally the core tensor went through a stacked deep neural network providing actions as an output.

The agent was trained to manage a portfolio of 29 stocks from the Dow Jones index[17] (with a recent addition excluded) and cash. The price data for the stocks from January 1st 2008 to December 31st, 2016 was used for training, while data from January 1st 2017 to December 31st, 2019 formed the testing set.

To gauge the efficacy of their strategy, Jang & Seong benchmarked their agent against multiple standards. These included the equally-weighted portfolio (EWP) and advanced Deep RL strategies like the EIIE agent from Jiang, Xu & Liang (2017), which was discussed in the previous section, and the ensemble approach by Yang *et al.* (2020), detailed in subsequent section. The authors showed that their agent displayed more adaptive behavior to the changing market conditions, resulting in the

---

[15] In finance, technical indicators are mathematical calculations based on historical price, volume, or open interest information that aim to forecast financial market direction. Technical indicators are used extensively in technical analysis to help traders predict future price movements. Jang & Seong (2023) used some of the most well known technical indicators, that is moving averages (MA), the relative strength index (RSI), and the moving average convergence divergence (MACD).

[16] The Tucker decomposition is a method that decomposes a tensor into a core tensor and several matrices, representing different modes or dimensions. It simplifies the original tensor by capturing its essential features in the core tensor, while the matrices provide the necessary transformations to approximate the original data. It is widely used for compression tasks.

[17] The Dow Jones Industrial Average (DJIA) is a stock market index representing the performance of 30 major U.S.-based publicly traded companies during a typical trading session.

highest Sharpe Ratio of 2 on the backtests. By comparison, the Sharpe Ratios for the benchmarks settled at 1.66 (EWP), 1.6 (Jiang, Xu & Liang), and 0.7 (Yang *et al.*).

Gao *et al.* (2021) transposed the core principles of the work by Jiang, Xu & Liang onto the Chinese stock market, but applying significant variations to the methodology. The most radical changes concerned the learning algorithm and overall neural architecture. In fact, the authors employed an original Hierarchical Deep Q-network (H-DQN) and empirically showed that it outperformed a more standard DQN agent developed in their previous work (Gao *et al.*, 2020). The authors' main contribution stemmed from the intuition that DQN is not able to handle very large action spaces effectively. Thus, they adopted a hierarchical structure with multiple DQNs where each of them was assigned only a subset of the assets, hence reducing their respective number of actions. The structural unit of the H-DQN was composed by a series of DQNs and their controllers. Each DQN was responsible for three assets, namely 2 stocks and cash, and interacted directly with the market. Every 2 market DQNs were assigned to a higher-level DQN, called the controller. Then, this structural unit could be hierarchically extended letting an even higher-level controller be in control of two lower-level controller DQNs, and so on. The general architecture of the DQNs was the following: firstly 2 convolutional layers processed normalized recent prices of the assets under management by the DQN, then the weight vector indicating the proportion of portfolio value assigned by the DQN to each of these assets was inserted in the feature maps[18] output by the previous layer, and finally the newly obtained feature maps were processed by two fully connected layers. The final layer had a dueling Q-Network (Wang *et al.*, 2016) structure, meaning that the value of the state and value of the action were estimated separately.

The H-DQN agent was trained to trade four stocks in Chinese A-share market, with various training and testing periods from January 2011 to December 2017. Finally,

[18] In the context of Convolutional Neural Networks (CNNs), a feature map refers to the output of a filter, an intermediate step in CNNs. As the network goes deeper, these feature maps can capture increasingly complex and abstract representations of the input.

three backtests were evaluated, each lasting two days in 2013, 2016 and 2017 respectively. H-DQN was compared to some of the benchmark strategies referenced in Jiang, Xu & Liang, with the addition of a more traditional DQN trading agent as well. In all the performed backtests, H-DQN outperformed the other strategies in terms of risk-adjusted return, yielding the highest average Sharpe Ratio of 10.33. For reference DQN and the buy and hold strategy had an average Sharpe Ratio of 2.72 and 1.97 respectively.

4.d Steps Towards an Open-source Framework

The remaining sections in this chapter will elaborate on a portion of the work and publications by the AI4Finance Foundation, responsible for the development and maintenance of the aforementioned FinRL open-source project. Therefore, the main objective will not be discussing the current state-of-the-art models, but rather tracing the evolution and inception of FinRL.

In one of the earlier works Liu *et al.* (2018) trained a Deep DPG (Silver *et al.*, 2014) agent to trade the 30 stocks constituting the Dow Jones Industrial Average index (DJIA) in January 2016. This is the publication that set the premises for FinRL and the subsequent work, and hence merits deepening.

In this research, stock trading was modeled as an MDP specified as follows:

- The state was modeled as $s = [p, h, b]$, where $p \in \mathbb{R}_+^D$ contained the information of the prices of stocks, $h \in \mathbb{Z}_+^d$ represented the amount of holdings of each stock and $b \in \mathbb{Z}_+$ was the remaining balance, while D denoted the total number of stocks.
- Actions $a \in \mathbb{Z}^D$ could perform selling, buying, and holding of each individual stock. Letting the subscript t denote time, the available actions on any given stock were denoted k, and could result in decreasing, increasing, and no change of the holdings h, in other words $h_{t+1} = h_t + k$.

- The reward $r(s, a, s') \in \mathbb{R}$ was the change in portfolio value in each time-step, where portfolio value could be computed as $p^T h + b$. Letting t_f denote a target time, maximizing the cumulative sum of rewards $\sum_{t=1}^{t_f - 1} r(s_t, a_t, s_{t+1})$ was therefore equivalent to maximizing the end portfolio value $p_{t_f}^T h_t + b_{t_f}$.
- The policy $\pi(s)$ represented the trading strategy, and in practice was the probability distribution of actions a in state s. It is approximated using fully connected neural networks.

An additional imposed restriction was that all bought stocks could not result in a negative balance, that is $b \geq 0$.

The simulations proceeded as usual. At each time-step the agent selected the number of stocks to buy or sell, if any, and the portfolio composition was updated. The agent then observed the new trading prices, and the resulting balance and allocation, based on which it received a reward. Then, the process repeated itself until a stopping condition was reached.

The authors trained the agent using data from January 1st, 2009 to December 31st, 2014. Then, they tuned key hyper parameters of the DPG algorithm, namely learning rate and number of episodes, on the validation period from January 1st, 2015 to January 1st, 2016. Finally they tested the agent's performance on the trading period from January 1st, 2016 to September 30th, 2018, benchmarking it to DJIA and the minimum variance portfolio (MINP, see section 2.c). It was shown that the Deep DPG strategy significantly outperformed DJIA and MINP on the considered backtesting period. Deep DPG reached a Sharpe Ratio of 1.79, against 1.45 for MINP and 1.27 for DJIA, indicating that the RL agent was substantially more robust in balancing risk and returns.

In a subsequent publication, Yang *et al.* (2020) extended on the previous work including a series of notable improvements. In fact, they trained three RL agents and proposed an ensemble strategy to select the most appropriate agent based on recent market conditions. Agents were trained with Deep DPG, Advantage Actor Critic (A2C) (Mnih *et al.*, 2016) and Proximal Policy Optimization (PPO) respectively.

Secondly, the authors went more into depth formalizing additional assumptions and constraints of their environment. As far as market liquidity was concerned they assumed orders could be rapidly executed at closing price, and that the market would not be affected by the decisions of the agents. Next, they reiterated that permitted actions should not result in a negative balance. Then they introduced a 0.1% transaction cost on the value of each trade (either buy or sell). Finally they added a notion of risk-aversion for market crash, through a turbulence index inspired by the work of Kritzman & Li (2010), which was recomputed in each time step. When this measure of extreme and abnormal price movements exceeded a predetermined threshold they simply halted buying, forcing the agent to sell all owned shares. This was done to limit the risk an agent could undertake during particularly hazardous market phases.

While authors did not alter the reward mechanism and action space, another noteworthy change concerned the state definition. A state was defined as a 181-dimensional vector consisting of seven distinct elements of information $[b_t, p_t, h_t, M_t, R_t, C_t, X_t]$ where:

- b_t, p_t, h_t remained unchanged from the previous state definition.
- M_t, R_t, C_t, X_t constituted technical indicators, respectively Moving Average Convergence Divergence, Relative Strength Index, Commodity Channel Index and Average Directional Index. This way agents also had access to information regarding past stock behavior.

Finally, the ensemble strategy was described. All agents were trained using price data for Dow Jones 30 constituents from January 1st, 2009 to September 30th, 2015,

then data from October 1st 2015 to December 31st 2015 was used for validation and hyper parameter tuning, and finally the agent's performance was tested on out-of-sample data from January 1st 2016 to May 8th 2020. To exploit the adaptive capabilities of the RL approach, agents kept on training on the trading data as well. Every new quarter the performance of the 3 agents on the previous three months was compared, and the agent yielding the highest Sharpe Ratio was selected to trade for the upcoming period. The authors demonstrated that this ensemble strategy beat DJIA and MINP on the testing period in terms of Sharpe Ratio (1.30 against 0.87 and 0.45 respectively), actually improving over the performance of all the individual agents, effectively proving that the novel approach inherited and integrated the best features of the three algorithms. In addition, this approach also allowed to analyze how the three agents behaved differently in different market conditions. For example, A2C proved to be more adaptive to risk, PPO was especially good in bullish markets phases and culminated in the highest cumulative return, while Deep DPG resulted to be a weaker alternative to PPO.

After demonstrating the effectiveness of their framework, researchers and programmers from the AI4Finance Foundation opted to make their environments and code completely accessible to the online community, releasing their work on GitHub[19]

4.e Financial Reinforcement Learning: FinRL and Research Democratization

FinRL is an open-source framework for financial reinforcement learning in Python. It was formally introduced in Liu *et al.* (2020), subsequently revised in Liu *et al.* (2021), and it is currently available on the AI4Finance Foundation official Github page. Most importantly, it comes with openly available code to replicate the published experiments discussed in the previous section and explore original research

[19] GitHub is a web-based platform that serves as a hub for software development projects using the version control system called Git. It provides tools for collaboration, code sharing, and project management.

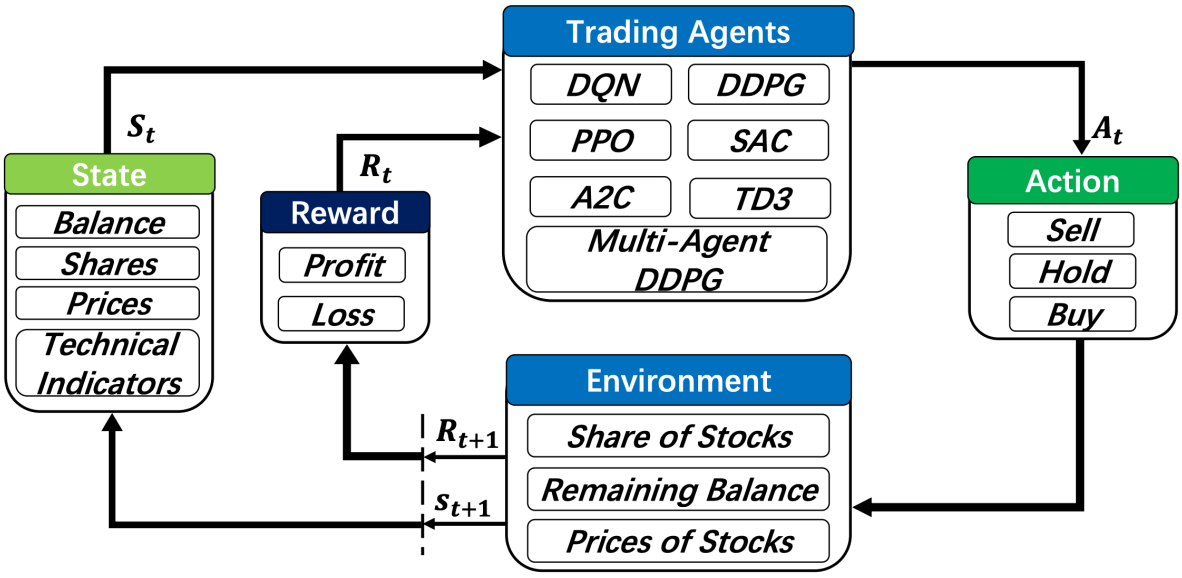

Figure 4.1: Overview of the global FinRL framework (source: Liu *et al.*, 2021, FinRL: Deep reinforcement learning framework to automate trading in quantitative finance, page 2)

questions. In fact, due to its transparency, it is flexible and extendible. While it offers pre-configured components, including data downloaders, stock trading environments, fine-tuned state-of-the-art RL algorithms and network architectures, its open-source nature encourages users to customize these elements. This adaptability ensures that FinRL can be tailored to address specific investigative needs effectively. Broadly speaking, it allows researchers and industry practitioners to work on the foundations of the studies mentioned in the previous section. In fact, the authors even included code from their previous research as starting tutorials. A brief overview of the global framework is described in Figure 4.1.

The FinRL framework was presented as combination of three layers: application layer, agent layer and environment layer. This subdivision was introduced to ensure transparency and simplify extensibility. Figure 4.2 offers a simple representation of the 3 layers.

With respect to the applications layer, key components remain mostly unchanged from the afore described works and are summarized in Table 4.1. The authors

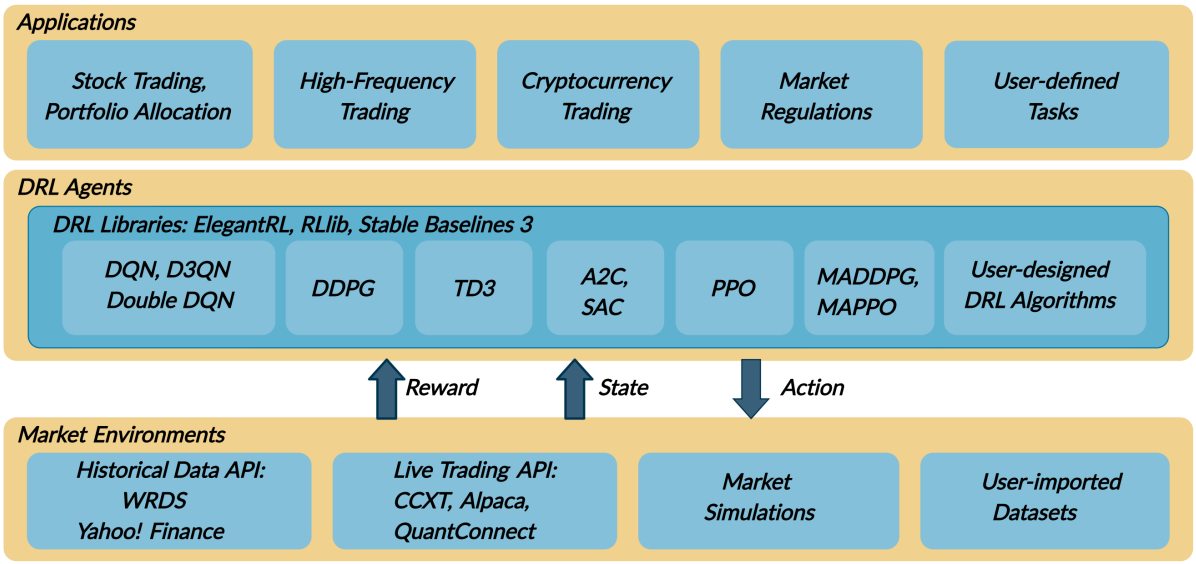

Figure 4.2: Application layer at the top, agent layer in the middle and environment layer at the bottom (source: Liu *et al.*, 2021, FinRL: Deep reinforcement learning framework to automate trading in quantitative finance, page 3)

specified that FinRL can be deployed for a variety of tasks including stock trading as previously discussed, but also portfolio allocation, high-frequency trading, cryptocurrency trading and naturally other user-defined tasks. Given the objective of this thesis, portfolio allocation represents an interesting differentiation from stock trading as previously intended: instead of letting the actions be the quantity of each asset to buy or sell, actions can also be framed as the distribution of the available wealth across assets, similarly to as in Jiang, Xu & Liang (2017) (refer to section 4.b). In practice this was done simply adding a softmax activation function to the last layer of the policy network.

As far a the state definition is concerned, FinRL allows for great modularity. Authors mentioned allowing agents to access more refined features such as open, high and low prices in addition to previously observed closing prices, fundamental indicators on top of technical indicators among others.

New reward mechanisms were made easily testable, including portfolio logarithmic

Key components	Attributes
State	Balance $b_t \in \mathbb{R}_+$; Shares $\boldsymbol{k}_t \in \mathbb{Z}_+^n$
	OHLCV data $\boldsymbol{o}_t, \boldsymbol{h}_t, \boldsymbol{l}_t, \boldsymbol{p}_t, \boldsymbol{v}_t \in \mathbb{R}_+^n$
	Technical indicators; Fundamental indicators
	Smart beta
	NLP market sentiment features
Action	Buy/Sell/Hold; Short/Long
	Portfolio weights
Rewards	Change of portfolio value
	Portfolio log-return
	Shape ratio
Environment	Dow-30, NASDAQ-100, S&P-500
	Cryptocurrencies
	Foreign currency and exchange
	Futures and options
	Living trading

Table 4.1: FinRL Key Components (source: Liu *et al.*, 2021, FinRL: Deep reinforcement learning framework to automate trading in quantitative finance, page 4)

return and Sharpe Ratio, other than simple change in portfolio value. In addition, user-defined reward systems are easily implementable.

Regarding the agent layer, FinRL incorporates three renowned open-source Deep RL libraries, namely Stable Baseline 3 (Raffin *et al.*, 2021), RLlib (Liang *et al.*, 2018) and ElegantRL (Liu *et al.*, 2021), supporting state-of-the-art algorithms including aforementioned DPG, PPO, A2C and more.

At the highest level, environments for trading and portfolio allocation integrate the previous layers and allow to run simulations and train agents. They are based on OpenAI Gym (Brockman et *al.*, 2016), a popular toolkit for reinforcement learning research. These environments consolidate the assumptions and constraints of Yang *et al.* (2020) discussed in the previous section, including specifiable transactions costs and the turbulence risk-aversion mechanism. Although standard datasets are available (e.g. the DJIA index constituents and relative prices and indicators), user-

imported data are also easily supported, both in terms of assets and observed features.

FinRL provides automatic backtesting and comparison against some notable baselines including the equally weighted strategy, mean-variance and minimum-variance strategy. Additionally, it even supports live trading APIs to test agents and place trades live.

To conclude their work, Liu *et al.* (2021) successfully replicated the findings of Jiang, Xu & Liang. This not only highlighted the elegance and simplicity of their open-source project but also emphasized its potential to drive further research in the domain.

5. A.T.E.N.A.: Motivations and Experimental Setup

The deep RL agents described so far have proven to be highly effective at constructing and controlling diverse portfolios of investments, mostly outperforming the alternative strategies they have been benchmarked with by a significant amount. At the same time, one of the main missions of research is to probe the boundaries of technology, stretching the state-of-the-art to discern its utmost potential. For this reason, setting realistic yet ambitious objectives is fundamental to progress further in these forefront domains.

Approaches depicted so far show how to create skilled trading agents that are specialized to predetermined sets of assets, which is perfectly in line with an asset manager's needs. However, this may raise some concerns as on one hand it might expose the learning process to the risk of overfitting, while on the other it is actually missing the opportunity to take advantage of far vaster sources of data that could be employed to train more knowledgeable agents.

This chapter will outline the limits of the discussed works, and will aim to propose a solution in the form of a more general agent, A.T.E.N.A.. Then, the experimented learning process will be described, including the types of data employed, the dynamics of the environment and the neural architectures employed.

5.a Generalized Portfolio Management and Asset-Overfitting

Strategies like the mean-variance (MVP) and minimum-variance (MINP) portfolios require to reestimate the covariance matrix (and the mean vector as well for the former) whenever a new allocation is desired, but the underlying algorithm and logic, that is the sequence of computations necessary to do so, remain the same regardless of the identity of the assets. In contrast, Deep RL agents as discussed so far need to be retrained from scratch every time a different set of assets must be managed, and might also require periodical retraining as unfolding market conditions

call for updated strategies. Of course, the adaptability of Deep RL approaches in practice represents the main element of strength that allowed them to surpass the traditional approaches in the experimental research illustrated in the previous chapter. However, it comes natural to question to what extent the agents specialize to the assets they trade during training, and by contrast how much they can learn about the more general problem of actively managing a portfolio. The task of ably handling a portfolio of investments irrespectively of its specific components will be referred as Generalized Portfolio Management (GPM).

From a different perspective, an agent trained on a predetermined set of securities may lead to an issue of asset-overfitting. Borrowing the term from Supervised ML, a model overfits to training data when it fails to generalize what is has learnt to new out-of-sample observations, and instead it remains excessively tailored to training instances. In this context, an agent that is unable to manage a diverse range of portfolios is overfitting and thus does not find a solution to GPM.

Generally speaking, the problem of overfitting is tightly linked to an excessive model complexity relative to the quality and quantity of training data. Indeed, deep learning models notoriously estimate a large number of weights, and consequently Deep RL agents have considerably more parameters than MVP and MINP, despite being estimated on similar types of data, that is the history of the managed assets. Furthermore, deep learning models are extremely flexible and can process virtually any kind of data. This is not limited to historical returns, but can include past Open-High-Low-Close prices, technical indicators and fundamental data as previously discussed, as well as other sources of information like macroeconomic factors or market sentiment to name a few. While this can enhance the expressive power of the models, it also further increases their complexity, making them particularly vulnerable to the risk of overfitting, including asset-overfitting.

5.b All-Trading Equity Neural Agent

As hinted in the previous section, a solution to the aforementioned shortcomings and limitations is simple. To tackle the GPM problem and contain the risk of overfitting an agent would need to observe a sufficiently large and diverse group of securities during training. Assets would need to span different sectors, markets and phases, thus providing the most manifold training sample possible. While the resulting agent may not represent the most profitable solution in every single moment in time or with every possible collection of securities, it would be exposed a much more complete and variegated dataset, and could therefore incorporate a more exhaustive knowledge of the market's history and behaviors. Additionally, this process could possibly act as a pre-training phase, and the trained agent could then be fine-tuned to more recent data and narrowed groups of assets, exploiting the advantages of transfer learning[20] widely used in modern deep learning.

This thesis proposes a novel Deep RL agent, termed All-Trading Equity Neural Agent (A.T.E.N.A.), which applies this philosophy, although restricting itself to the general stock market. Thus, cryptocurrencies, bonds ETFs or other types of securities are not considered in the experiments. The name is inspired by the Italian translation of Athena, the ancient greek goddess of wisdom and war. On a lighter note, in fact, wisdom and strategic warfare represent the most indispensable skills required to successfully traverse the turbulent odyssey that is the stock market.

The remaining of this chapter will delve into the steps undertaken to train A.T.E.N.A. using the PPO algorithm introduced in section 3.d.

[20] Transfer learning in machine learning is a technique where a model developed for a particular task is reused as the starting point for a model on a second task. It aims to leverage the knowledge gained from the initial task to improve learning in a new, related task.

5.c Data and Preprocessing

The dataset utilized for the training of A.T.E.N.A. is the property of WAVENURE SRL[21] and was generously provided for research purposes.

It encompasses trading prices of 1495 stocks spanning from January 2000 to May 2023, subsequently processed by WAVENURE's proprietary data quality algorithm. Although the precise steps of the algorithm are exclusive and not openly disclosed, a general outlook of the process is provided.

For each individual stock the algorithm receives as input:

- Intraday trading data, which includes opening price, highest price, lowest price, closing price and volume registered every available day.
- Dates and amounts of issued dividends.
- Dates and ratios of registered stock splits[22].

Then, three main steps follow in the pipeline:

- Firstly the algorithm scans the prices of each stock to detect abnormal, abrupt fluctuations in the series. The values of the ambiguous days are doublechecked on a difference source and either corrected with data from the secondary source or interpolated between the nearest unambiguous data points.
- Secondly, each price series is adjusted for dividends and stock splits. These events influence trading prices (Fama *et al.*, 1969), and standard methods are available to incorporate this information. This ensures that stock prices accurately reflect the share's true intrinsic value. Norton (2011) provided a very practical guide for the task.

---

[21] WAVENURE is an independent fintech company that employs Artificial Intelligence and Quantitative Algorithms to optimize investment decisions in financial markets.

[22] A stock split is a corporate action in which a company increases the number of its outstanding shares by issuing more shares to current shareholders. This results in a decrease in the individual share price, while the total market capitalization of the company remains the same.

- Finally, the updated series are scanned one last time to ensure there are no unexpected behaviors that might be due to missed dividends or splits. In the case a series still experiences unusual, unjustified peaks or drops, the series prior to the last extreme price movement is deleted.

The dataset provides a comprehensive snapshot of medium and large capital stocks over of the last 23 years. Figures 5.1 (a) and 5.1 (b) display the distribution of the collected stocks across sectors and continents respectively, confirming the dataset represents a diverse and representative basket of companies spanning all industries and markets. Additionally, Figure 5.1 (c) shows the number of data points per year, demonstrating that the dataset does not exhibit excessively overrepresented market phases. Moreover, there is a clear upwards trend, and while it is fair to assume that more data may have been collected in more recent years due to its increased availability, it may also raise concerns regarding the presence of a survivorship bias (Brown *et al.*, 1992), which, however, is deferred to a subsequent study.

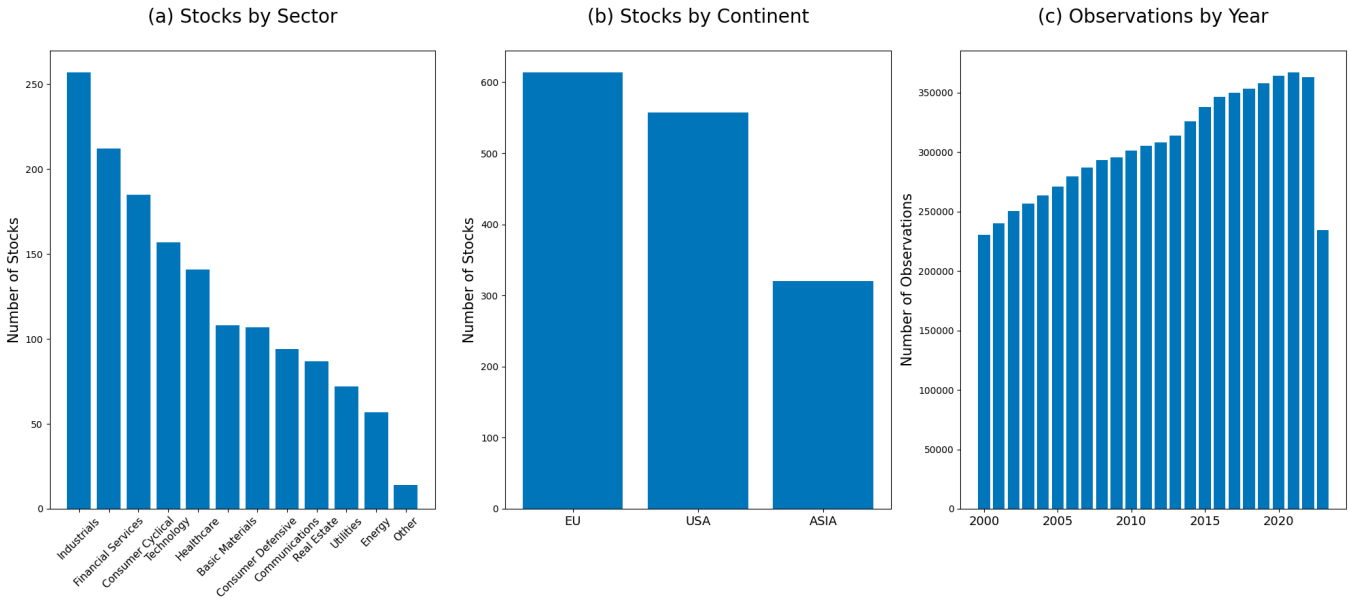

Figure 5.1: Distribution of stocks across sectors (left) and continents (middle), and number of observations per year (right) in the described dataset

5.d Environment Assumptions and Constraints

The environment used for training and evaluation was largely inspired by FinRL's StockTradingEnv (AI4Finance Foundation, 2022), albeit with significant modifications. The foundational assumptions and constraints relative to the behavior of the environment mostly align with those outlined by Yang *et al.* (2020) detailed in section 4.d, with a few notable variations. More precisely:

- Orders are assumed to be executed at closing prices, and the actions of the agent do not influence the market.
- The agent's actions cannot result in a negative balance.
- Short trades[23] are not allowed. While it is not explicitly mentioned in the original paper, an agent can only invest a positive fraction of its wealth in a determined asset, thereby acting as a "long-only" investor.
- Each trade, either purchase or sale, incurs in a transaction fee of 0.2%. This cost is increased from the original 0.1% to better account for unconsidered market frictions such as liquidity slippage and bid-ask spread[24].

The main deviation from the assumptions presented by Yang *et al.* is the elimination of the risk-aversion mechanism tied to the turbulence index. The underlying rationale is that the agent should adapt independently to volatile market conditions, rather than being explicitly directed to liquidate all its positions.

---

[23]Short selling, in the context of financial trades, refers to the practice of selling a security that the seller does not currently own, with the intention of later purchasing the security back at a lower price, thus profiting from a decline in its price.

[24]Liquidity-slippage refers to the difference between the expected price of a trade and the price at which the trade is actually executed, due to a lack of sufficient liquidity in the market. The bid-ask spread is the difference between the highest price a buyer is willing to pay for a security (the bid) and the lowest price a seller is willing to accept (the ask).

5.e Episodes

Perhaps the biggest change to the environment concerns the sampling of new episodes. Every episode lasts a total of T time-steps, where every time-step t corresponds to a simulated trading day. For training each episode is fixed at five business years, equivalent to approximately 1260 time-steps. Firstly, a random time window of length T is uniformly chosen from a timeline set between a designated minimum and maximum date. Subsequently, a batch of D stocks which were trading concurrently within the time window is selected. At the time of training the minimum date is fixed at the earliest date in the dataset, January 3rd, 2000, while the maximum date is capped at May 1st, 2020. This configuration reserves the subsequent three years for out-of-sample evaluations. The quantity of stocks under management D is set at 30 and they are assumed to trade in dollars. At the beginning of each episode, the agent is allocated a liquidity of $10,000,000.

As a result, the total number of unique training episodes is approximately 1.84^{66} [25].

This mechanism is pivotal for its ability to expose the agent to varied market trends and economic cycles from multiple viewpoints. As a result, training episodes encompass diverse market scenarios, from general market expansion to notable downturns such as the Dot-com bubble crash, the 2007–2008 financial crisis, and the Covid-19-induced recession. These events can have varied effects on assets, forcing the agent to find management tactics that are not specialized to individual stocks.

The main revision to the original environment relatively to the unfolding of each episode regards the definition of the agent's actions. In fact, while in the original formulation they specified buying and selling orders, in the revised environment

---

[25] This can be computed as number of possible sets of 30 assets from a total of 1495, multiplied by the number of 5-years periods from a total of 20 years:

$$\text{Total Combinations} = \binom{1495}{30} \cdot 252 \cdot (20 - 5) \approx 1.84^{66}$$

Naturally, some episodes will be partially overlapping.

actions indicate the desired portfolio allocation. Every day the agent observes a new state s_t and the policy network produces an action vector $a_t \in [0,1]^{D+1}$. A softmax function is applied to the last layer of the policy network so that a_t is a distribution summing up to one where each element $a_{t,i}$ for i in $1, 2, \ldots, D+1$ represents the desired percentage of wealth allocated in the i_{th} asset at time t, with the last element corresponding to liquidity. Once a new target allocation has been submitted to the environment, it is juxtaposed to the currently held portfolio and current market prices. The action vector is then translated into specific stock purchase and sale orders to achieve the desired configuration. Hence, the agent can update its allocation on a daily basis.

Then, at time-step $t+1$ the new market closing prices are used to compute the updated portfolio value V_{t+1}, and a reward signal is produced. Finally, the new state s_{t+1} is observed by the agent and the process is repeated for t in $1, 2, \ldots, T$.

5.f Rewards

Two different types of reward are tested.

The first reward mechanism is the daily logarithmic return of the agent's portfolio, which is the same reward utilized in the majority of the reviewed works. More precisely, letting V_t denote the value of the portfolio at the beginning of day t:

$$r_t = \ln\left(\frac{V_{t+1}}{V_t}\right) \quad \text{for } t \text{ in } 1, 2, \ldots, T-1 \tag{5.1}$$

Because the optimal policy at each time-step seeks to maximize the expected future return (refer to Equation 3.4), in episodic tasks this translates to the agent aiming to maximize the total sum of rewards received per episode. Then, taking advantage of the additive property of logarithms:

$$\sum_{t=1}^{T-1} r_t = \sum_{t=1}^{T-1} \ln\left(\frac{V_{t+1}}{V_t}\right) = \ln\left(\frac{V_T}{V_1}\right) \tag{5.2}$$

Hence, the agent's goal coincides with that of maximizing the growth of the portfolio value by the end of each episode. A similar result would be achieved using the daily change of portfolio value in absolute terms, as in Liu *et al.* (2018) or Yang *et al*. (2020) and outlined in section 4.d. However, the use of a measure of relative change is more suited to reward and penalize the agent. This is because the impact of a $1000 loss varies depending on the total value of the portfolio. For instance, such a loss is considerably more detrimental for a $10,000 portfolio than for one worth $1,000,000. Logarithmic returns effectively capture these proportional differences. This reward will be denoted as Simple Log-return Reward (SLR).

The second type of reward is a novel mechanism based on the following intuition: while a growth in portfolio value is certainly a desired outcome, the real objective of new strategies is actually over performing more established benchmarks. From a slightly different perspective, the success of a capital management firm does not stem from its mere capability to make money, but in that of making more money than its competitors. Thus, incorporating knowledge regarding the benchmark to beat directly into the reward signal allows to produce more informative feedback to the agent, providing insights regarding its performance relatively to the alternative strategies. This can be achieved with a simple change to the previous reward. Letting B_t denote the value at the beginning of day t of a portfolio managed with a benchmark strategy:

$$r_t = \ln\left(\frac{V_{t+1}}{V_t}\right) - \ln\left(\frac{B_{t+1}}{B_t}\right) \tag{5.3}$$

As each episode progresses, the performance of a portfolio managed with some predetermined baseline strategy is recorded. Naturally, the baseline portfolio and the

agent's portfolio must share all trading conditions, including pool of stocks, market frictions and starting liquidity. Similarly to Equation 5.2, from Equation 5.3 it derives that the episodic cumulative reward is equivalent to:

$$\sum_{t=1}^{T-1} r_t = \sum_{t=1}^{T-1} \ln\left(\frac{V_{t+1}}{V_t}\right) - \ln\left(\frac{B_{t+1}}{B_t}\right) = \ln\left(\frac{V_T}{V_1}\right) - \ln\left(\frac{B_T}{B_1}\right) \qquad (5.4)$$

Then, the agent's objective is simply that of over performing the benchmark in terms of portfolio logarithmic return.

This reward mechanism will be denoted as Beat the Baseline Reward (BBR).

In the experiments carried out the chosen benchmark strategy is the equally weighted portfolio with a yearly rebalancing, which has proven to be a simple yet particularly effective strategy, as outlined in section 2.c.

As a closing remark, an interesting aspect concerning the difference between the two mechanisms can be noted. In a period of general growth of the agent's portfolio SLR will generate mostly positive rewards, while in a downside phase it will produce negative rewards. On the other hand, rather counterintuitively, employing BBR a period of portfolio drawdown may be associated to positive rewards if the agent manages to contain the losses more effectively than the benchmark, while a stage of portfolio growth may generate negative rewards if the benchmark is more successful at capitalizing on the advantageous market phase. As a consequence, signals can widely differ and provide different types of feedback to the agent.

5.g Observable State and Network Architectures

As mentioned earlier, the training algorithm utilized is PPO. Consequently, the network architecture features a shared extractor for both the actor and the critic, and subsequently it diverges into two distinct pathways.

Each time-step's observed state comprises two primary components. The network's design ensures tailored processing for each.

Firstly, the agent observes a history component capturing recent movements of managed stocks, a reference benchmark, and its own portfolio. This is represented as an $L \times (D + 2)$ matrix H_t, where each l_{th} row denotes the daily logarithmic return of the stocks, the benchmark and the portfolio at time $t - l + 1$. Initially, the agent lacks information about its portfolio's performance and the benchmark. Hence, the last two columns of H_t are initialized to zero and populated as the simulation advances. This configuration equips the agent with the historical context in which it has to operate, offering insights into market trends and its recent performance relative to the benchmark. This aids in formulating a more effective risk management strategy.

In the conducted experiments the equally weighted portfolio with yearly rebalancing is chosen as the benchmark, and number of observed lags L is set to 32.

Three simple architectures are explored to process H_t, acting as history embedders:

(i) Dense Embedder: A straightforward approach where H_t is flattened and passed through a single fully connected (FC) layer with 316 hidden nodes, followed by a nonlinear activation. The rationale for this node count will be elaborated upon momentarily.

(ii) Convolutional Embedder: each column of H_t undergoes independent processing via two 1-dimensional convolutional layers, a nonlinear activation, and a max pooling layer, followed by a FC layer with 32 hidden nodes. This is succeeded by an FC layer with 32 hidden nodes. The resulting embedded series are concatenated and processed by another FC layer with 311 hidden nodes and a nonlinear activation.

(iii) Long Short Term Memory (LSTM) Embedder: each column of H_t is independently processed by two LSTM layers, followed by an FC layer with 32 hidden nodes. The resulting embedded series are concatenated and processed by an FC layer

Component	Details
Input Size	(1,32)
Conv1	Out Channels: 16, Kernel Size: 4, Stride: 1, Activation: GELU, MaxPool Size: 2
Conv2	Out Channels: 32, Kernel Size: 4, Stride: 1, Activation: GELU, MaxPool Size: 2
Linear Layer	Output Size: 32

(a) Convolutional Embedder

Component	Details
Input Size	(1,32)
LSTM	Hidden Size: 16 Number of Layers: 2
Linear Layer	Output Size: 32

(b) LSTM Embedder

Table 5.1: Details of the Convolutional (left) and LSTM (right) history embedder architectures

with 313 hidden nodes and a nonlinear activation.

For all architectures, the Gaussian Error Linear Units (Hendryck & Gimpel, 2016) is chosen as nonlinear activation function. Table 5.1 provides details of the convolutional and LSTM layers for designs (ii) and (iii). The size of the hidden nodes is selected to ensure that all architectures have approximately the same number of parameters. This ensures that performance differences arise from design choices rather than a simple disparity in model complexity.

Convolutional Neural Networks and LSTMs are prevalent for time series data feature extraction, as also confirmed by their adoption in most of the reviewed literature. Hence, comparing them is logical. Conversely, design (i) offers a useful baseline to evaluate the efficacy of the more intricate architectures.

In essence, this first part of the architecture acts as a feature extractor, distilling relevant information from the individual time series, producing an embedding of each sequence and ultimately providing a lower dimensional representation of the embedded history e_t. This vector likely incorporates insights about the relationships between the inputs and the expected growth of each stock.

The second ingredient of the observed state is termed the “Current Snapshot” c_t. This vector of size $D + 3$ encapsulates the agent's present condition. Drawing parallels with the "Portfolio Vector Memory" concept introduced by Jiang, Xu & Liang (2017) and detailed in section 4.b, the initial $D + 1$ elements of the snapshot represent the current holdings of each asset. These are expressed as a percentage of the total wealth, calculated using the market's closing prices at time t. This design enables the agent to adjust its decisions based on its existing allocation. This is crucial, given the costs associated with portfolio rebalancing and that all trades require the payment of a transaction fee.

The last two elements of c_t offer insights into the benchmark's and the agent's capabilities to generate wealth. They are the simple ratios of the portfolio value to the initial available liquidity at the beginning of the episode. They serve to contextualize the information derived from the history embedding, providing a holistic view of the agent's performance.

Following this, e_t and c_t are concatenated and this combined vector is independently processed by the actor and critic networks. Both networks comprise two identical FC layers, with 128 and 256 hidden nodes, respectively. All hidden layers are followed by the hyperbolic tangent activation function[26] commonly used in PPO applications. Then, the critic network produces an estimate of the value function, while the actor network produces a vector of size $D + 1$. Finally, after applying a softmax function the output represents the desired allocation across assets.

---

[26] The hyperbolic tangent function, often abbreviated as "tanh", is an activation function used in neural networks and deep learning models. The formula for the tanh function is the following:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Hyperparameter	Value
Learning Rate	0.0001 (With linear annealing)
N Steps	512
Batch Size	1024
N Epochs	3
Gamma	0.99
GAE Lambda	0.95
Clip Range	0.2
Clip Range VF	0.2
VF Coef	0.5
Ent Coef	0.01
Normalize Advantage	True
Max Grad Norm	0.5
Target KL	None

Table 5.2: Selected hyper parameters for the PPO algorithm

5.h Training A.T.E.N.A

A total of 6 PPO agents are trained, one for each possible combination of network architectures and reward mechanisms. 16 environments are handled in parallel and each agent is trained for a total of 10,000,000 steps, for a total of approximately 144 hours of training. Unfortunately, due to time and computational constraints it is not possible to train every single agent until a clear plateau is observed in the monitored learning curves, and additional training might be required to reach the best agent performance overall. However, 10,000,000 steps represent an adequate compromise to carry out an suitable comparison between the considered implementations and their learning potential.

The main hyper parameters of the PPO algorithm are reported in Table 5.2, and their choice is based on the work of Andrychowicz *et al.* (2020).

Additionally, states are centered and scaled using running means and standard deviations, while the rewards are simply scaled using a running standard deviation.

6. Experimental Results

In this chapter the main experimental results will be outlined. First, the learning curves observed during training will be compared across the different models, in order to understand which architectures and reward mechanisms lead to more stable and effective learning. Then, trained agents will be evaluated on out-of-sample data and compared to relevant benchmarks.

6.a In-sample Learning with SLR

The level of progress made during training visibly differs across the various tested implementations. This becomes evident monitoring the mean reward obtained by each agent as more episodes unravel and more experience is accumulated.

To avoid confusion a further remark is necessary regarding the terms used in the proposed visualizations. The total number of individual episodes generated during training is roughly 7500. However, as mentioned in section 5.g, 16 environments are run in parallel which means that each agent during training collects experiences from 16 different episodes at the same time. Therefore, the term “episode” used in the visualizations is an average of all the simulations running in parallel to ensure that the agent in episode $i + 1$ has indeed received more updates than it had in episode i, and therefore it can be expected to be better at its task and to receive more reward on average.

The Simple Log-return Reward (SLR) mechanism is analyzed first, hence three agents are compared. Figures 6.1, 6.2 and 6.3 on the right display the mean reward per step (MRS) obtained by each of the three agents. An exponential weighted moving average in also showed to simplify visualization given the high volatility of MRS. An increasing trend represents a progressive improvement and is the intended outcome. Additionally, the steeper the climb, the faster the agent is learning, which is also desirable.

The visualization proposed on the left of each image shows the cumulative reward collected up until any given time-step in every episode. The plot of a cumulative reward is termed a “trajectory”. When the SLR mechanism is employed, trajectories can be interpreted as the portfolio logarithmic return since the beginning of training. Trajectories of episodes that happened early in training are represented with the color blue, and they gradually fade to red as training continues and the agent matures, providing an alternative way of visualizing progress. In fact, initial trajectories will likely plummet given that the agent still needs to learn that each action is costly due to the presence of transaction fees and will accumulate mostly negative rewards. One would hope that later trajectories will ascend progressively higher indicating that the agent is becoming more skilled at obtaining positive rewards.

Figure 6.1 displays the progress of the Dense Embedder Agent (DEA). The trajectory plot on the left doesn’t highlight visible improvements and most of the trajectories tend to decline, with only a minority of them ending up in the positive zone. In turn, a

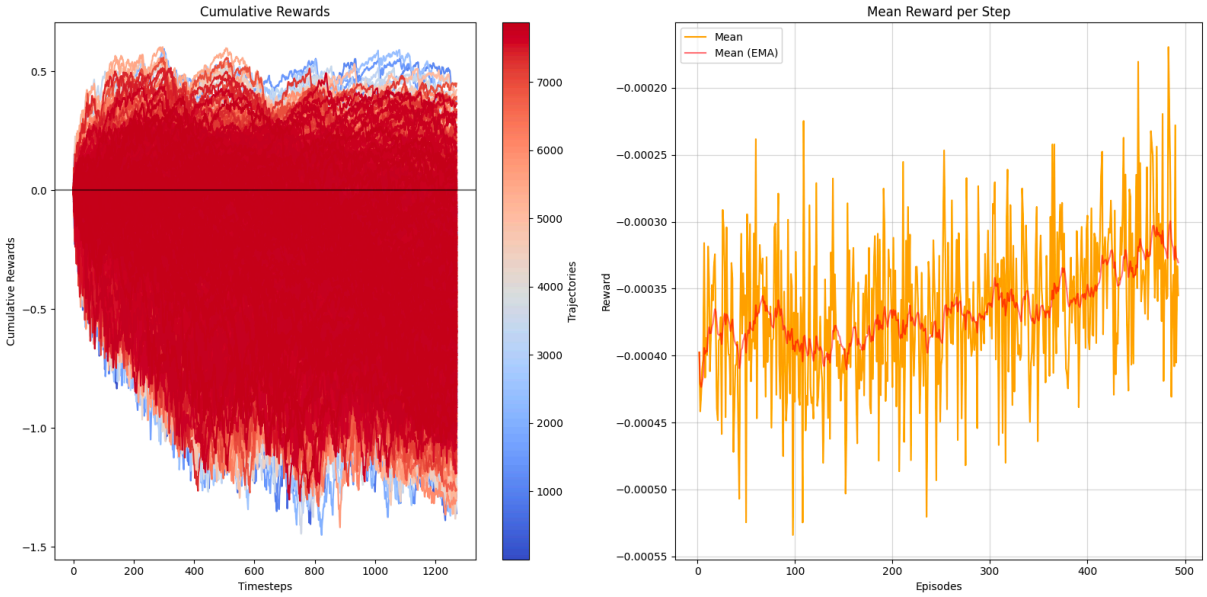

Figure 6.1: Trajectory plot (left) and MRS plot (right) for the Dense Embedder Agent using Simple Log-return Rewards

slow increase in the MRS can be appreciated on the right.

Figure 6.2 on the other hand exhibits the learning progress for the Convolutional Embedder Agent (CEA). While the trajectory plot is fairly similar to the previous one, the image highlights a clearer growing trend in MRS.

Finally, Figure 6.3 displays the progress of the LSTM Embedder Agent (LEA). The MRS plot features the steepest increment yet, and also the trajectories plot shows a visible advancement, albeit small, with later trajectories securing more cumulative reward than in the earlier ones, and a more pronounced positive inclination overall.

The three agents can be easily compared in Figure 6.4, confirming that LEA demonstrates the best learning behavior, followed by CEA. It appears clear that LSTM and Convolutional architectures provide useful structures to process time series in this context. However, even after 10,000,000 time-steps of training MRS is mostly negative and so is the average cumulative reward per episode, implying that during training the agents on average fail to increase the value of their assets and lose money.

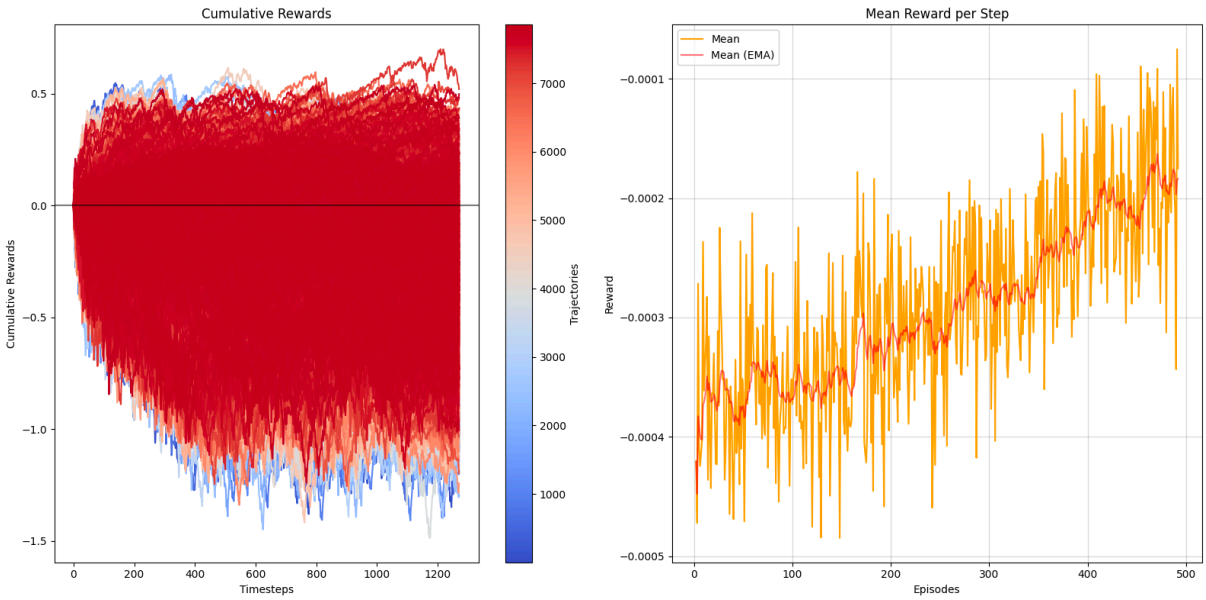

Figure 6.2: Trajectory plot (left) and MRS plot (right) for the Convolutional Embedder Agent using Simple Log-return Rewards

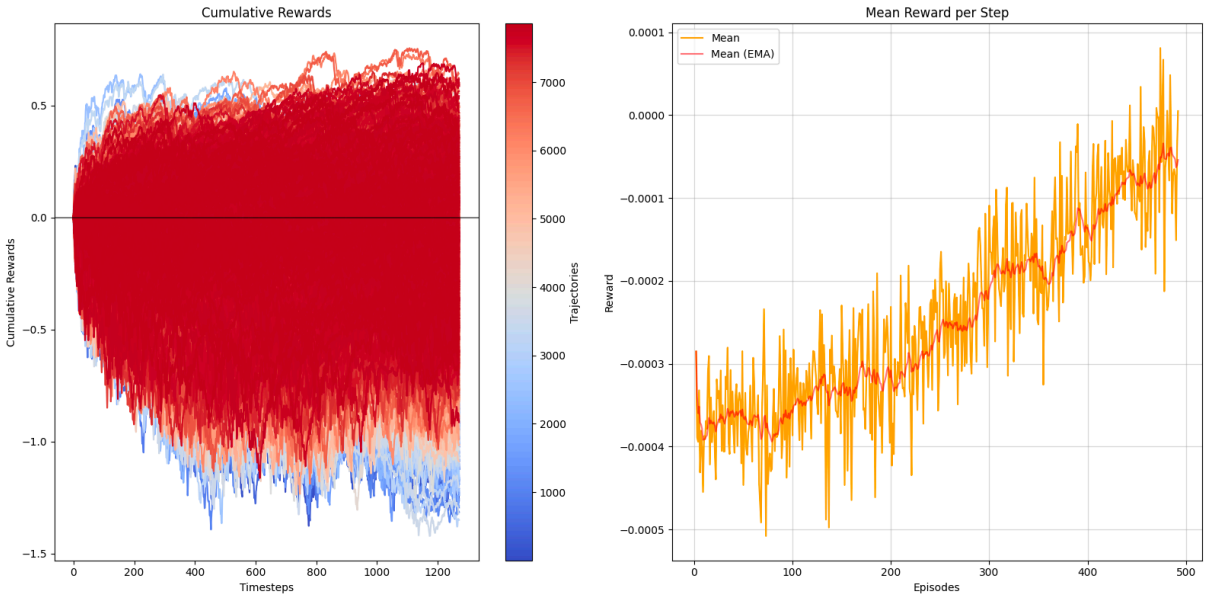

Figure 6.3: Trajectory plot (left) and MRS plot (right) for the LSTM Embedder Agent using Simple Log-return Rewards

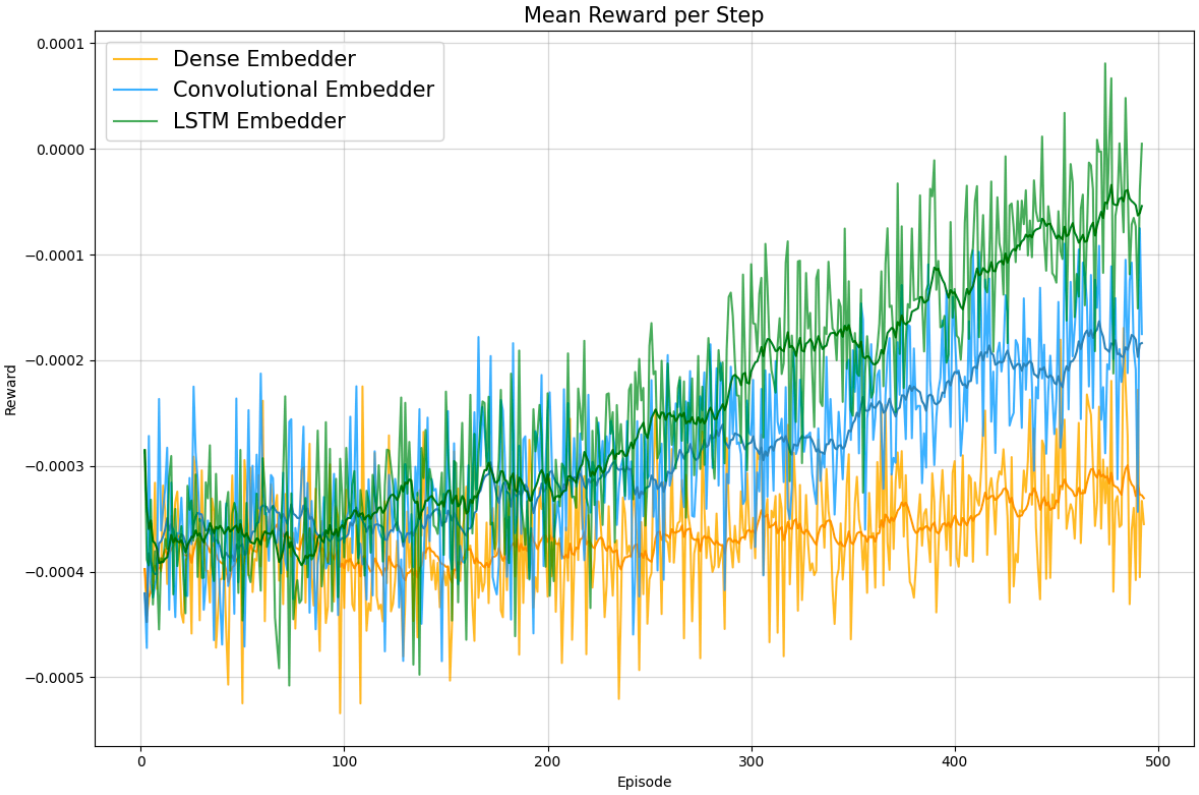

Figure 6.4: MRS of the three agents using Simple Log-return Rewards

Figure 6.5 and Figure 6.6 show a random episode experienced by LEA at the beginning and at the end of training respectively. Specifically, in the first panel the images exhibit equity lines for the agent and the Equally Weighted portfolio with yearly rebalancing (EWP), while the following three panels provide useful diagnostics and insight into the agent's behavior and environment. They include a representation of LEA's expositions as a percentage of its total wealth, the prices of the stocks under management normalized by their value at time-step 0 and the cumulative rewards. While the two episodes are comparable in terms of overall growth and volatility of the

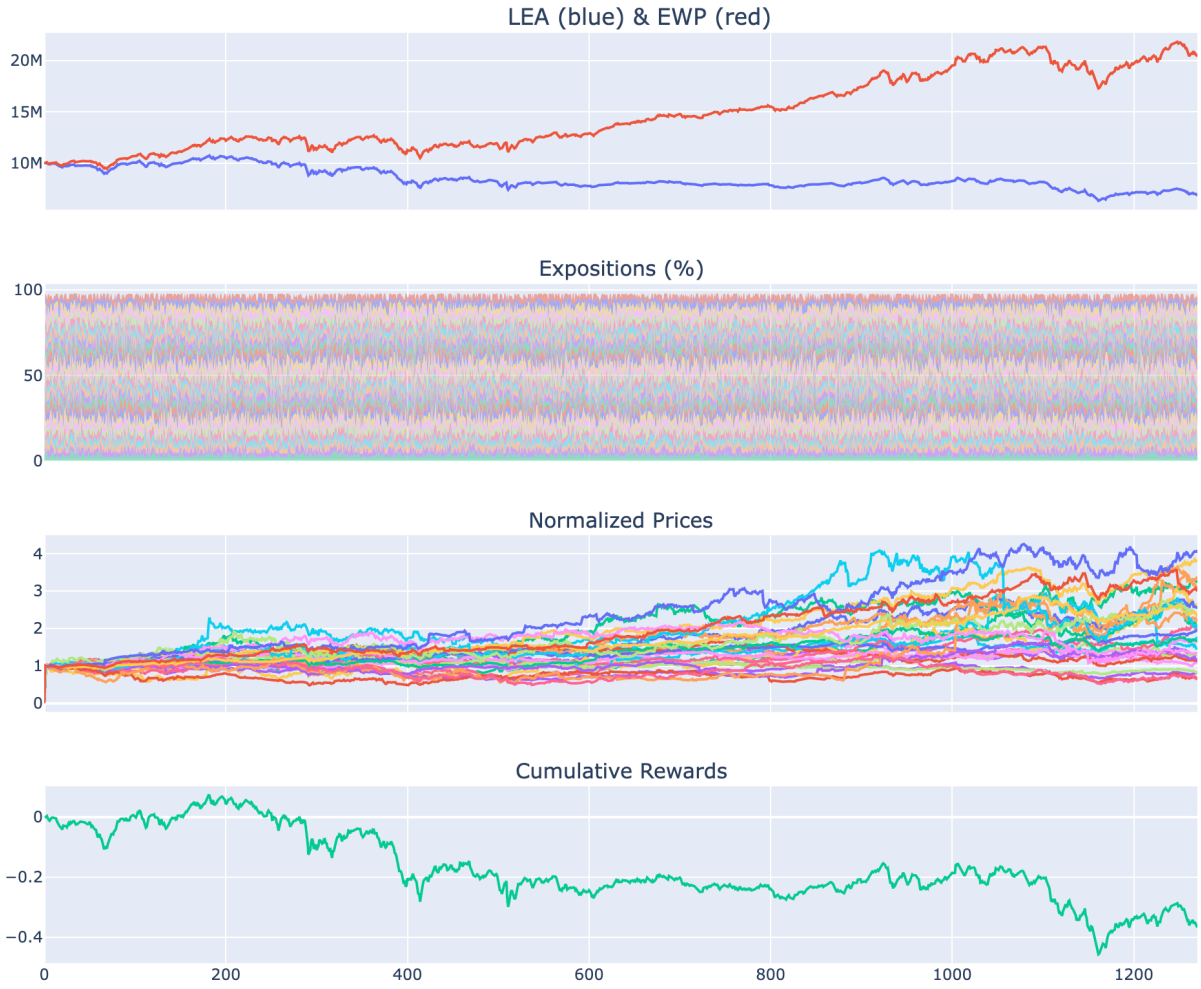

Figure 6.6: Diagnostics for a random episode experienced by LEA (SLR) at the beginning of training

assets under management and the EWP benchmark, Figure 6.6 shows that after training LEA is more adept at stabilizing the value of its own portfolio, as confirmed by the recovery in cumulative rewards. Additionally, while expositions in Figure 6.5 are extremely erratic and inconsistent, they become more steady and permanent by the end of training. This is likely due to the fact that LEA has learnt the negative effect of unwarranted rebalancing and the advantages of maintaining a more constant allocation.

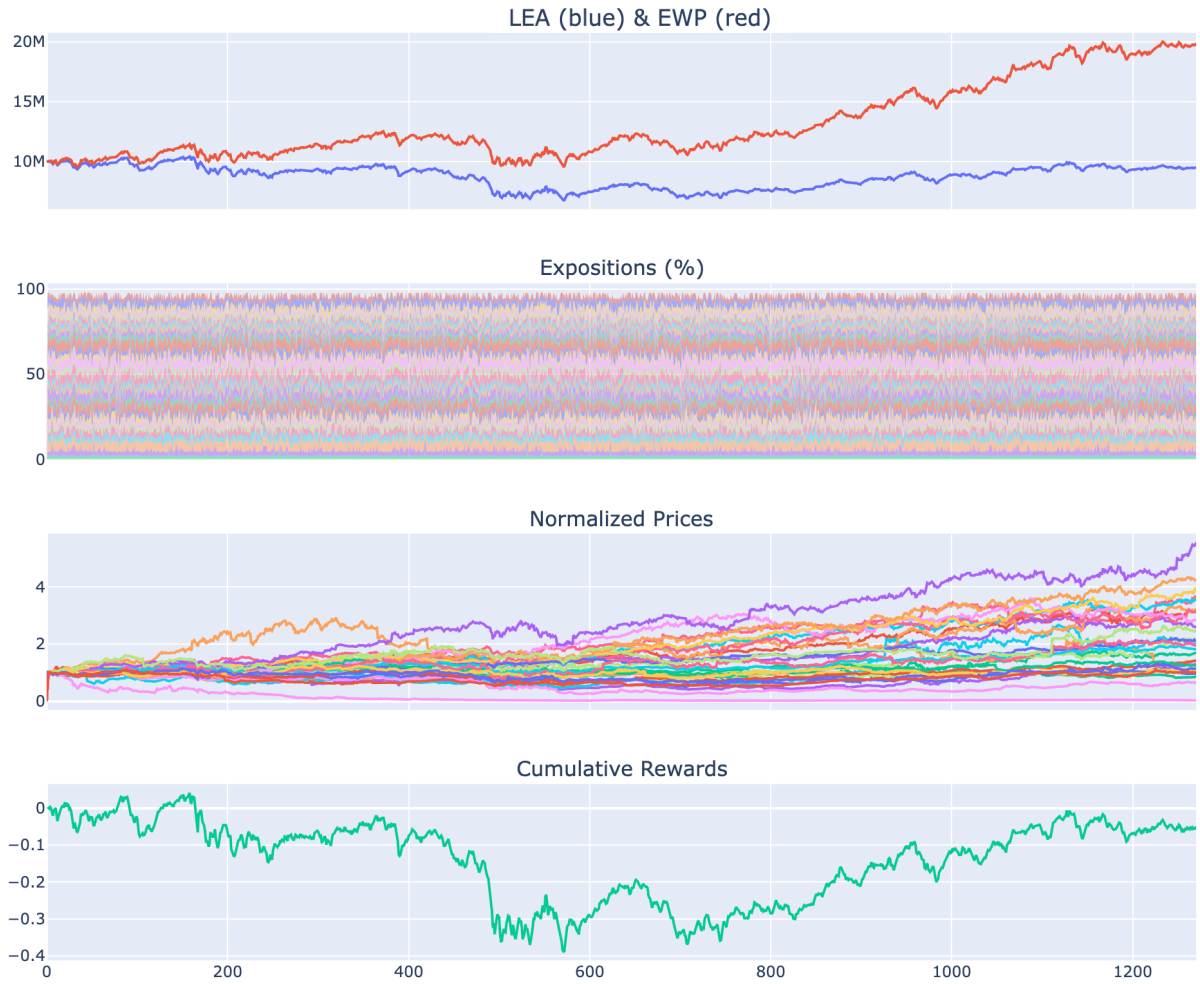

Figure 6.6: Diagnostics for a random episode experienced by LEA (SLR) at the end of training

6.b In-sample Learning with BBR

The same diagnostics are provided for the three agents trained via the Beat the Benchmark Reward (BBR) mechanism, with Equally Weighted Portfolio (EWP) chosen as designated benchmark.

Figures 6.7, 6.8 and 6.9 showcase trajectory and MRS plots for DEA, CEA and LEA respectively. When the BBR mechanism is employed, trajectories can be interpreted as the agent's over performance on the benchmark in terms of portfolio logarithmic return.

BBR proves to be an effective reward mechanism for two reasons. First, all agents display a steep and clear improvement during training, substantial enough to be discernible also in the trajectory plot. Secondly, the reward is computed comparing the agent's performance to that of a benchmark which has already proven to be an effective strategy (see DeMiguel, Garlappi & Uppal, 2009 and section 2.c). In other words, contrarily to when SLR is employed, a negative reward does not necessarily translate to a loss of money, but it highlights an underperformance relatively to the

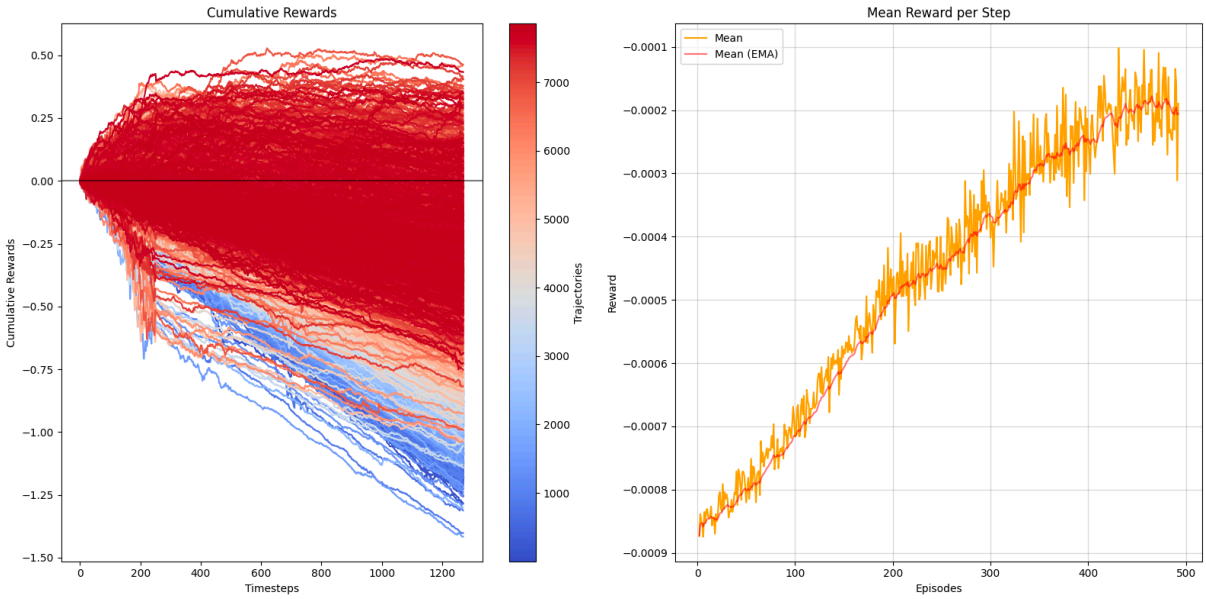

Figure 6.7: Trajectory plot (left) and MRS plot (right) for the Dense Embedder Agent using Beat the Benchmark Rewards

benchmark which has been shown to be profitable on average in most market conditions.

As a result, episodes play out significantly differently at the end of training compared to the initial iterations, as emphasized by the trajectories progressively surging and becoming more centered around zero.

Convolutions and LSTM are confirmed better alternatives to the simple Dense Embedder, showcasing significantly lower volatility in the trajectory plots. Moreover, as made clear in Figure 6.10, they both reach higher plateaus in MRS. Contrarily to the SLR mechanism, there is not apparent evidence that LEA over performs CEA, as the latter experiences a sharper increment in the earlier phases of training, but it is readily recovered by LEA and the two end up with a comparable MRS.

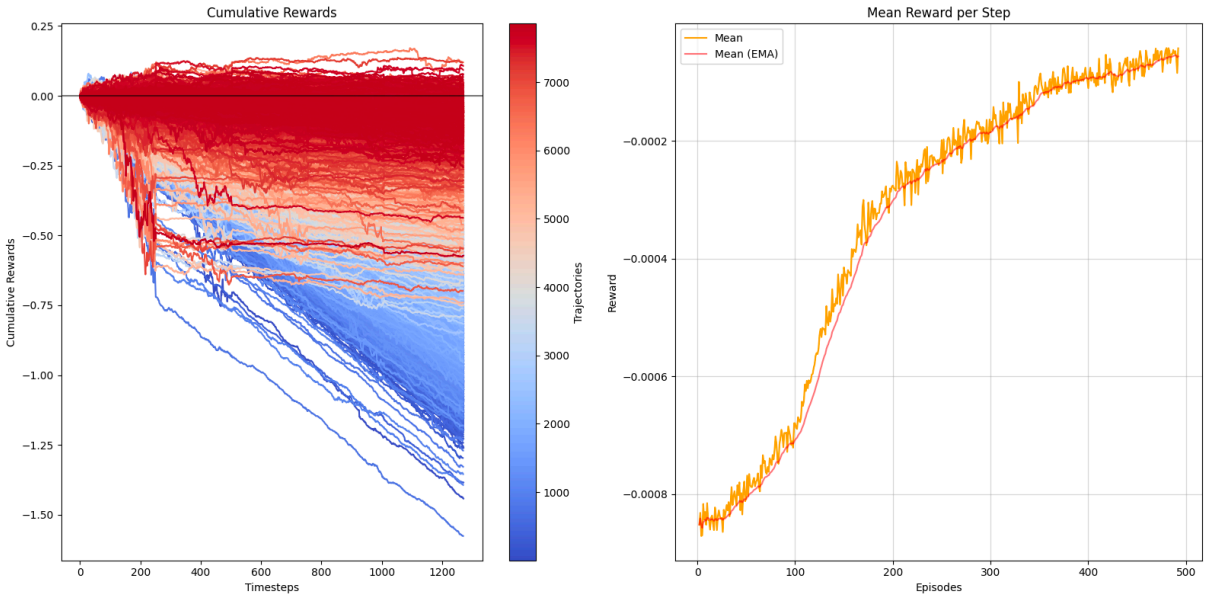

Figure 6.8: Trajectory plot (left) and MRS plot (right) for the Convolutional Embedder Agent using Beat the Benchmark Rewards

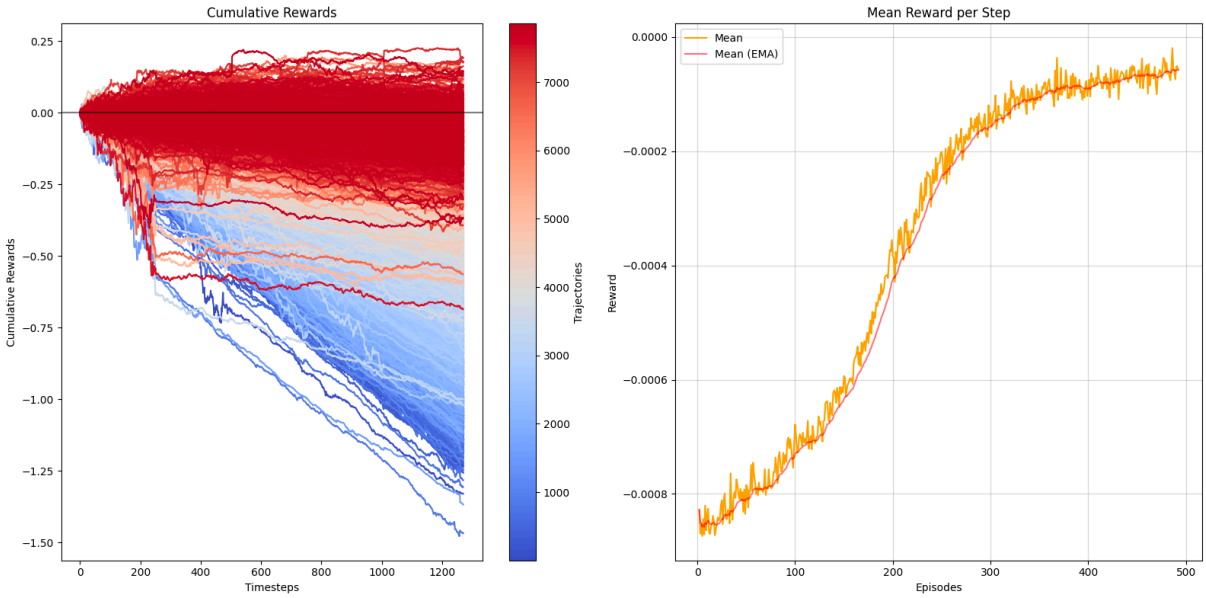

Figure 6.9: Trajectory plot (left) and MRS plot (right) for the LSTM Embedder Agent using Beat the Benchmark Rewards

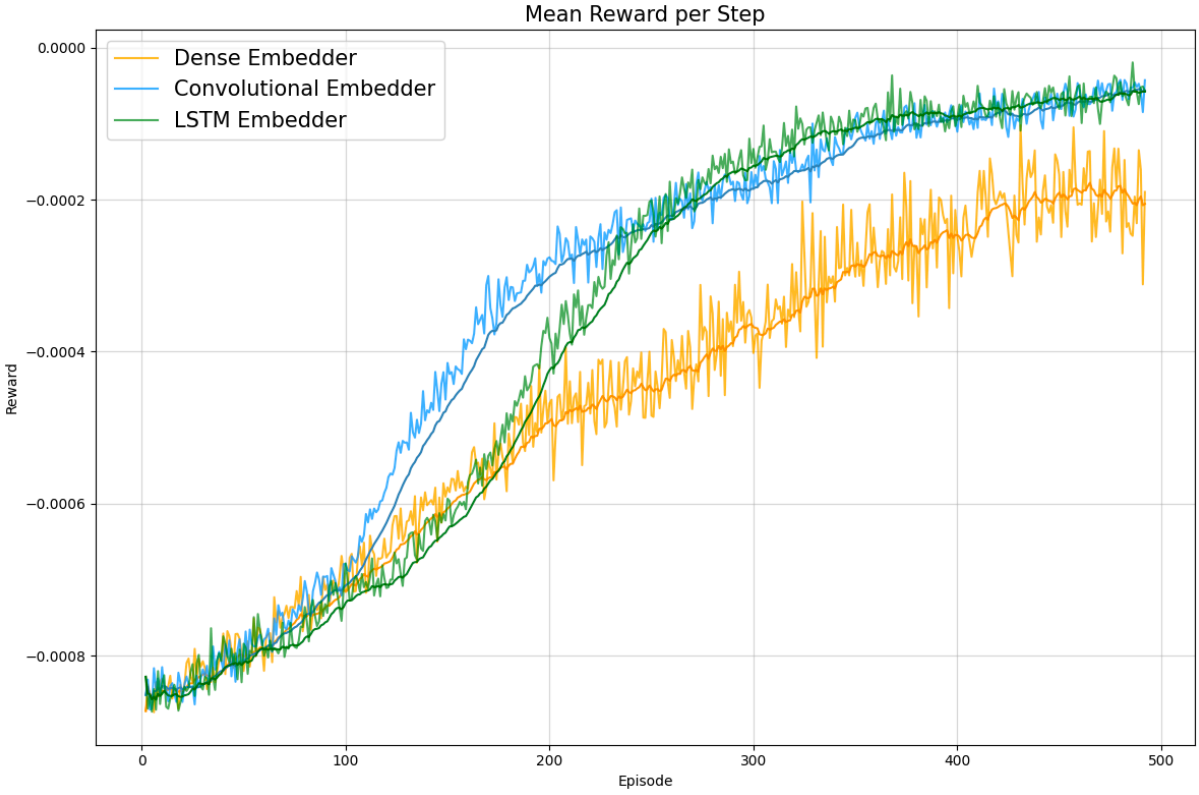

Figure 6.10: MRS of the three agents using Beat the Benchmark Reward

As in the previous section, two episodes sampled at the beginning and towards the end of training respectively are analyzed. Given that LEA and CEA reach akin performances, for consistency the behavior of LEA is examined again. In the early episode displayed in Figure 6.11 cumulative rewards immediately crash down, implying that the EWP benchmark is producing significantly higher returns. Moreover, similarly to the behavior of early LEA with SLR, the portfolio allocations are random and unstable, which is likely the main reason of this striking underperformance. The value of LEA's portfolio gradually decreases, leading to a consistent loss by the end of the period and a final value amounting to approximately one third of EWP's.

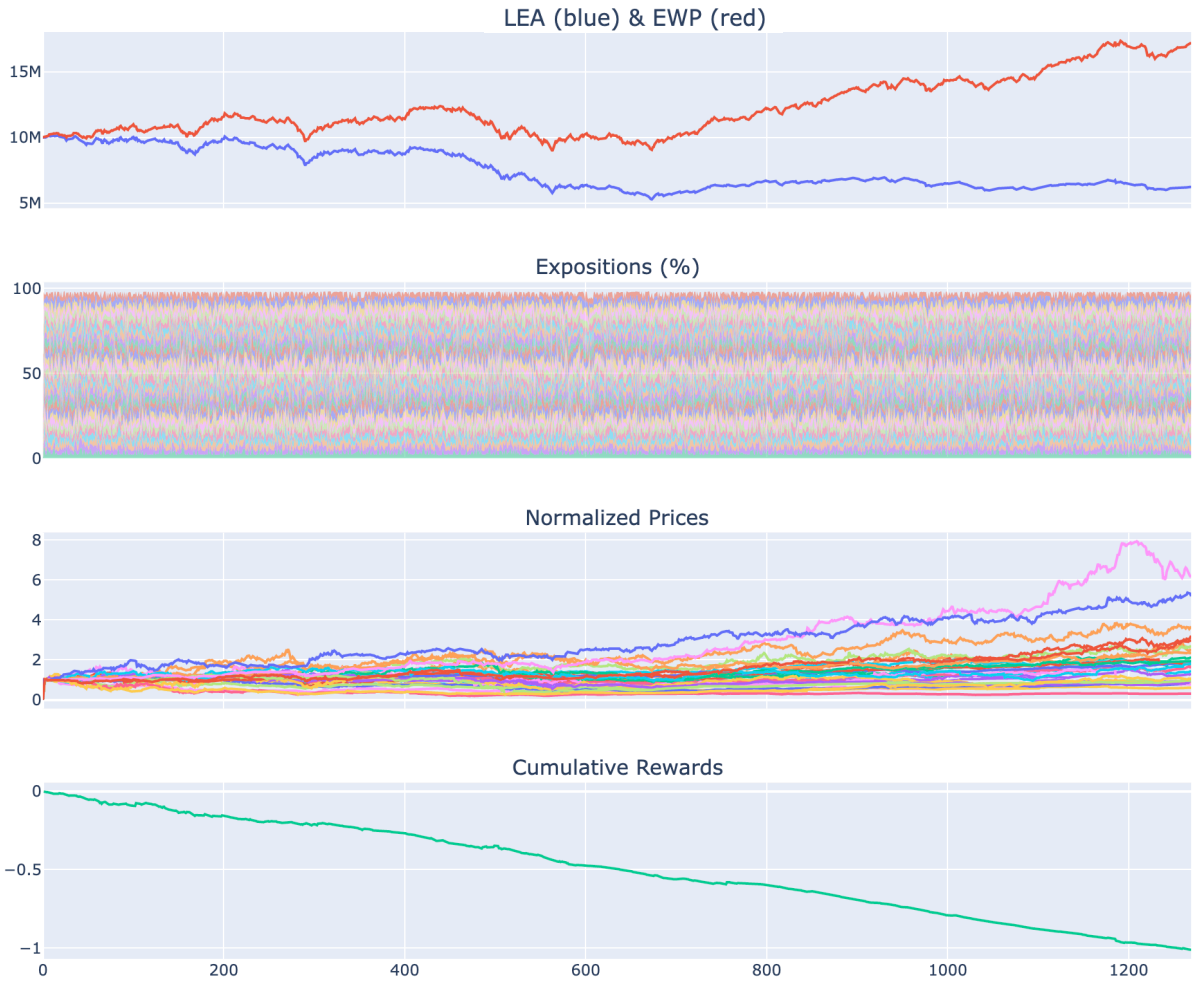

Figure 6.11: Diagnostics for a random episode experienced by LEA (BBR) at the beginning of training

On the other hand, in Figure 6.12 one of the latest episodes experienced by LEA is displayed. The movements of LEA and EWP are closely related, and the two end up with a similar portfolio value. In addition, LEA's expositions are substantially more polished. In fact, the agent maintains a mostly constant allocation, with the exception of a rebalancing during a market downturn and subsequently again in correspondence of its recovery. Accordingly, rewards are more balanced and drops are contained, demonstrating that LEA has become more skilled and experienced.

Figure 6.12: Diagnostics for a random episode experienced by LEA (BBR) at the end of training

6.c Out-of-sample Evaluation

The in-sample analysis suggests that more complex neural architectures may be more efficient compared to the simpler Dense Embedder structure. Moreover, BBR seems to lead to faster learning and more capable agents. However, in-sample analyses should be carried out with caution.

In fact, despite the exceptionally large number of unique, possible training episodes, a substantial portion of them share considerable overlap. Moreover, distinct equities can often be influenced by the same market factors. As a consequence, even if the agent encounters an episode comprising a entirely new set of stocks, in theory it might still deduce information about the prevalent market phase, potentially gaining an unintended and impractical advantage. Therefore, it is logical to test the performance of the trained agents on out-of-sample data that remained untouched and unseen during the training process.

From a different perspective, during training the agents actions are intentionally disrupted to enhance exploration. In the long-term, this allows agents to find new, potentially more effective strategies, but it also means that during training their performance is undermined. Thus, to assess the agents true capabilities it is crucial that these disruptions are eliminated during evaluation so that the highest possible level of success in ensured.

During evaluation, 500 episodes are generated by sampling 500 distinct starting dates from the available timeline. From each of these dates, 30 stocks trading concurrently are chosen in line with the approach detailed in section 5.e. Adjustments are made only to the date range and episode length. Specifically, the earliest date is set to May 1st, 2020, which marks the end of the training period, and the latest is set to May 1st, 2023. This ensures the agents encounter entirely novel episodes during testing. Moreover, each episode is fixed at a duration of two business years, equivalent to approximately 504 steps. For each episode, all the trained agents are tested and two measures are recorded, namely Sharpe Ratio and Maximum

	Avg. Sharpe	Std. Sharpe	Avg. MDD	Std. MDD
DEA w/ SLR	0.4237	0.4423	0.1921	0.0479
DEA w/ BBR	0.4920	0.4541	0.1856	0.0494
CEA w/ SLR	0.4772	0.4463	0.1909	0.0484
CEA w/ BBR	0.4852	0.4421	0.1866	0.0471
LEA w/ SLR	**0.5033**	0.4495	0.1841	0.0473
LEA w/ BBR	0.4968	0.4553	**0.1821**	0.0476
EWP	0.4751	0.3936	0.1985	0.0462
MINP	0.1485	0.4397	0.1990	0.0578

Table 6.1: Average (Avg.) and standard deviation (Std.) of Sharpe Ratio (Sharpe) and Maximum Drawdown (MDD) across 500 testing episodes

Drawdown[27] (e.g. Reilly & Brown, 2011). In addition, also the performance of two benchmarks is collected, namely the equally weighted portfolio (EWP) and the minimum variance portfolio (MINP), both rebalanced yearly.

Table 6.1 summarizes the results. The LSTM Embedder is confirmed to be the best performing architecture, as LEA trained with SLR (LEA w/ SRL) and with BBR (LEA w/ BBR) obtain the highest Sharpe Ratio and the lowest Maximum Drawdown respectively. Moreover, BBR leads to higher Sharpe Ratios and lower Maximum Drawdowns on average, with the sole exception of LEA where BRR leads to a slightly smaller Average Sharpe Ratio compared to SLR. This offers empirical evidence supporting the effectiveness of the BBR mechanism, likely for the reasons elaborated in section 5.f. In addition, employing SLR the in-sample considerations are confirmed, with LEA emerging as the best performing agent across both metrics, followed by CEA. By contrast, when BBR is deployed DEA surprisingly outperforms CEA, while LEA remains the superior agent.

[27] The Maximum Drawdown (MDD) is a measure used in finance to quantify the largest single drop from peak to trough in the value of a portfolio or an asset, before a new peak is attained. It is an indicator of downside risk over a specified time period. MDD is commonly expressed as a percentage and represents the maximum loss one could have experienced if investing at the highest point before the largest drop and selling at the lowest point.

All agents except DEA w/ SLR surpass the benchmarks across both performance measures. To determine if the agents' outperformance is statistically significant, paired t-tests[28] are employed. In this analysis, for each testing episode, the difference between the Sharpe Ratio produced by two strategies is documented. The null hypothesis tested posits that this difference is equal to zero. The same procedure is applied to the Maximum Drawdown as well, and for all conducted tests the significance level is set to 5%. For clarity and focus, only the top-performing agents, specifically the two LEA variants, are considered. First, the two agents are compared to determine if one best agent can be identified. However, both null hypothesis fail to be rejected with p-values equal to 0.374 and 0.063 respectively, suggesting that there is not sufficient proof that the two agents perform significantly differently in

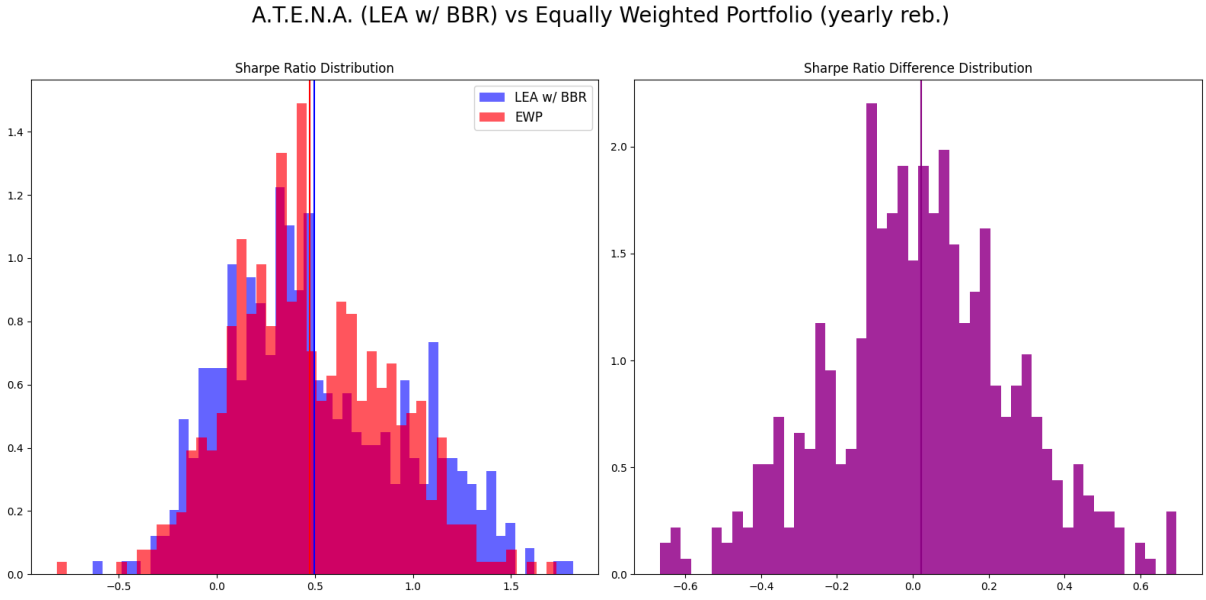

Figure 6.13: Empirical distributions of Sharpe Ratios produced by LEA w/ BBR and EWP (left) and their differences (right) across the 500 testing episodes

[28] A paired t-test is a statistical procedure used to determine whether the mean difference between two sets of paired observations is significantly different from zero. It is typically applied when the same subjects are used in both conditions being compared, ensuring that the two samples being compared are related or matched in some way.

terms of the considered metrics. Thus, LEA w/ BBR is analyzed for simplicity and given the novelty of the reward.

First, the afore described procedure is applied to compare the agent with EWP. The null hypothesis that the difference between the Sharpe Ratios of the two strategies is equal to zero is rejected with a p-value of 0.0491, confirming that the agent produces a higher Sharpe Ratio on average. Similarly, the null hypothesis that the difference between the Maximum Drawdowns of the two strategies is equal to zero is also rejected with a p-value of $3.8 \cdot 10^{-40}$, meaning that the average Maximum Drawdown of the agent's portfolio is significantly lower. The empirical distributions are displayed in Figures 6.13 and 6.14.

Likewise, the procedure is repeated to compare the agent with the MINP strategy. The null hypotheses are rejected with p-values of $2.6 \cdot 10^{-50}$ and $6.3 \cdot 10^{-13}$ for the Sharpe Ratio and the Maximum Drawdown respectively, implying that LEA w/ BBR significantly outperforms MINP as well. The empirical distributions are displayed in Figures 6.15 and 6.16.

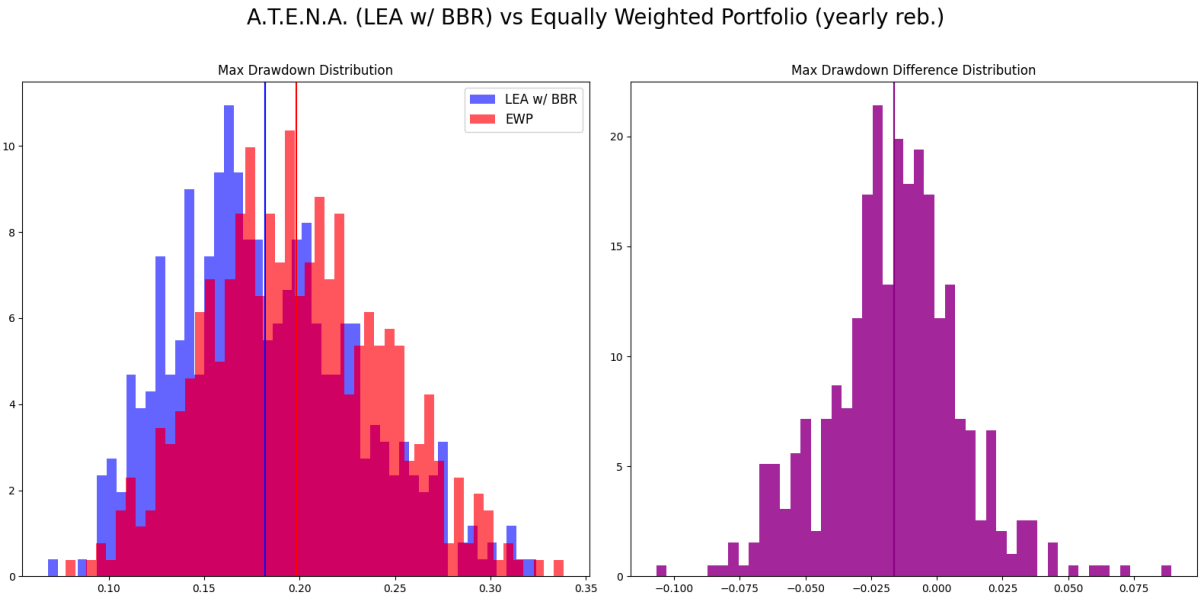

Figure 6.14: Empirical distributions of Maximum Drawdowns produced by LEA w/ BBR and EWP (left) and their differences (right) across the 500 testing episodes

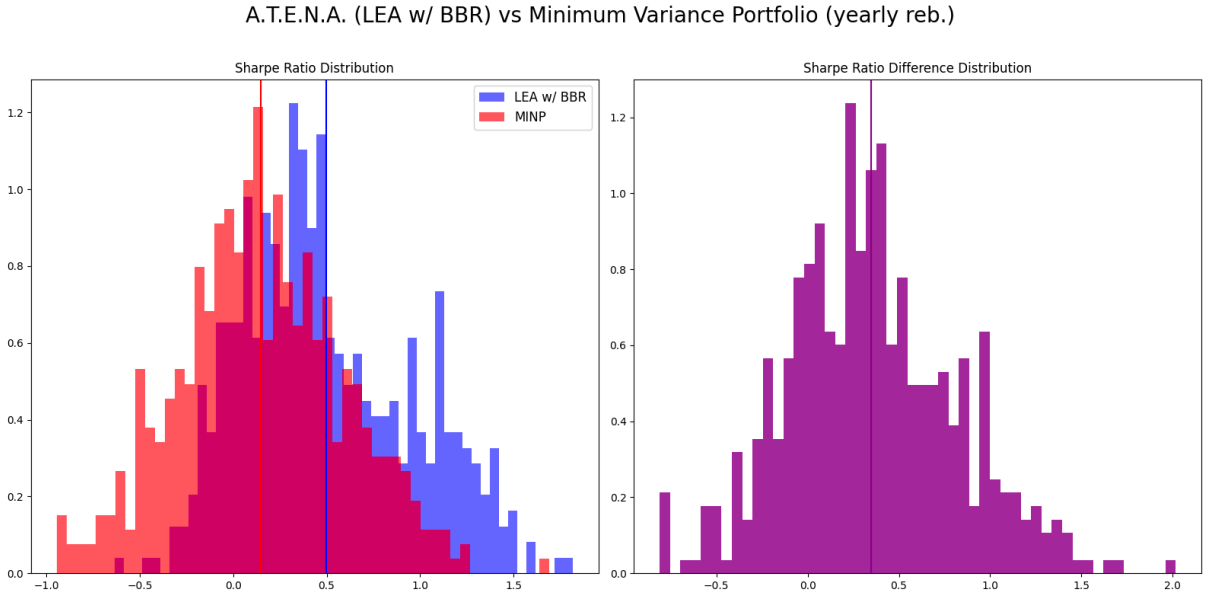

Figure 6.15: Empirical distributions of Sharpe Ratios produced by LEA w/ BBR and MINP (left) and their differences (right) across the 500 testing episodes

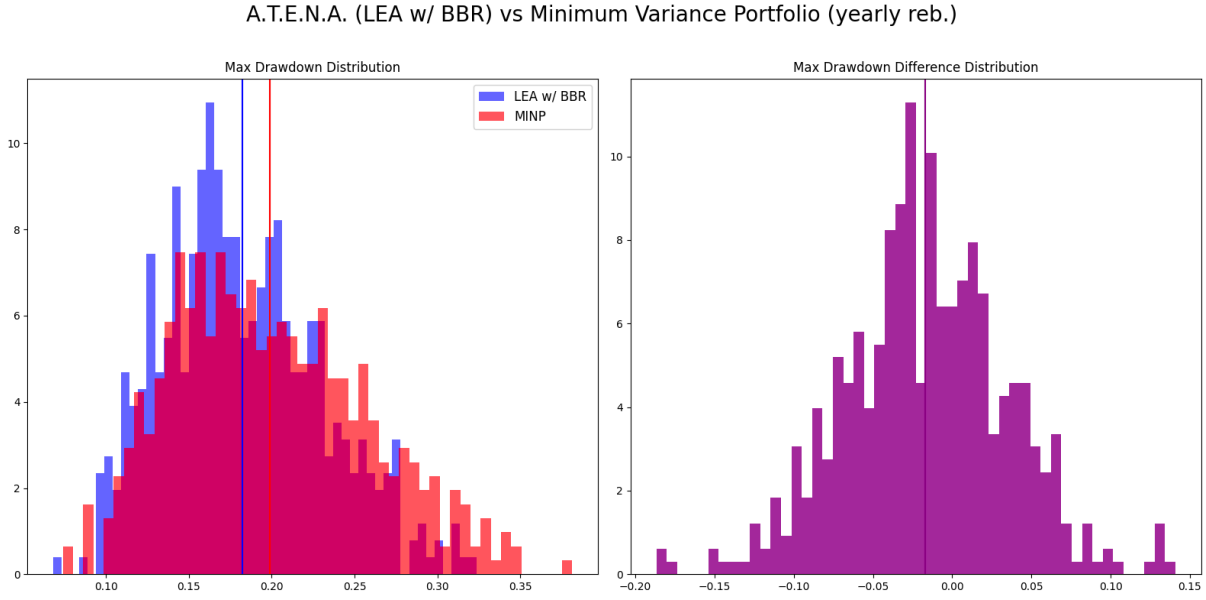

Figure 6.16: Empirical distributions of Maximum Drawdowns produced by LEA w/ BBR and MINP (left) and their differences (right) across the 500 testing episodes

7. Conclusions and Future Work

The presented experimental work provides early evidence regarding the potential of Deep Reinforcement Learning for the task of Generalized Portfolio Management. In fact, a novel agent is proposed, A.T.E.N.A., which tackles the problem through a new training routine exploiting historical data from multiple assets, and thus without overfitting to a predetermined set of stocks. A comparative study shows the behavior of different neural network architectures for the processing of the observed states across a classic reward mechanism based on the portfolio logarithmic return and an original reward system that incorporates information from a benchmark. In-sample and out-of-sample evaluations agree on the capability of Convolutional and Long Short Term Memory layers to process the time series for this task, with a particular emphasis on the latter. Additionally, the out-of-sample backtests highlight the efficacy of the novel reward to improve upon the more established alternative mechanism in terms of Sharpe Ratio and Maximum Drawdown. Finally, A.T.E.N.A. significantly outperforms classic heuristic portfolio management strategies, namely the Equally Weighted Portfolio and the Minimum Variance Portfolio.

The remaining of this chapter will elaborate on the potential role of Deep RL in the definition of a new market equilibrium and will conclude outlining viable directions for future work in this field.

7.a New Equilibrium

Successfully addressing Generalized Portfolio Management has countless implications. Among these, understanding the potential effects of such an agent at market level requires special attention. Supposing that in the future a refined version of A.T.E.N.A. is released and universally acknowledged as the paramount strategy for optimizing the portfolio Sharpe Ratio, it becomes reasonable to believe that every investor will adopt it. Under the assumption that all market participants are rational,

that is seeking optimal risk-return trade-offs, and operate with equivalent access to information and same shared beliefs, the introduction of a superior portfolio strategy could significantly influence collective investment behaviors. This could, in turn, adjust the capitalization of various market assets, resulting in a new market portfolio composition. Drawing parallels with the foundational ideas behind the Capital Asset Pricing Model (CAPM) (Sharpe, 1964), this shift would advance a fresh market equilibrium. This equilibrium would not be constrained by the assumptions of the Markowitz framework (Markowitz, 1952), especially concerning return distributions and the single-period investment horizon, as discussed in section 2.c. In fact, the Deep RL agent underlying the new allocation strategy would be capable to model more complex relationships between its inputs and dynamically respond and adapt based on evolving market scenarios and incoming data. To continue on the line traced by CAPM, the market equilibrium would be guaranteed by arising arbitrage opportunities in the moment an asset is not correctly priced relatively to its systematic risk, i.e. the exposition to the market risk. In theory, this would momentarily allow some investors to generate risk-free extra returns until these opportunities are exploited and cease to exist, effectively bringing the market back to its equilibrium.
Broadly speaking, further work in this area or research could then lead to markets that operate with enhanced efficiency and thereby ensure more stable investment conditions. This would be characterized by asset prices that faithfully incorporate all accessible information, negating any undue edge to a select group of sophisticated market players.

7.b Future Work

This thesis delves into both the theoretical and empirical facets of Deep RL in the realm of Generalized Portfolio Management. However, given the findings presented, numerous avenues remain uncharted and warrant further investigation.
A primary area of interest is the exploration of advanced deep learning architectures.

Considering the evident advantages of intricate deep learning structures in this context, it becomes imperative to consider the adoption of contemporary supervised deep learning architectures, notably the transformer as outlined by Vaswani *et al.*, (2017). The self-attention mechanism within transformers may provide a fresh perspective in understanding inter-asset relationships, worth deepening further.

Furthermore, the evolution of reward systems should be taken into account. This includes addressing more intricate facets of the task at hand, such as introducing a n explicit notion of risk aversion. A more nuanced reward system might, for example, factor in portfolio drawdowns to present a more comprehensive understanding of risk and return.

An enhancement in the dataset used is also a significant area for development. Utilizing a broader dataset, encompassing a more extensive range of stocks and spanning varied time frames, would not only enrich the training process but also enable more rigorous backtesting, affirming the robustness of the results.

In conclusion, this thesis lays the foundational groundwork for the evolution of A.T.E.N.A., shedding light on its significance and offering an initial contribution to the domain. It is hoped that this work sparks further research and exploration in the field.

Bibliography

AI4Finance Foundation. Official GitHub Page. <https://github.com/AI4Finance-Foundation> [last accessed: 10/09/2023].

Andrychowicz, M. *et al.* (2020). “What matters in on-policy reinforcement learning? a large-scale empirical study”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.2006.05990> [last accessed: 10/09/2023].

Brockman, G. *et al.* (2016). “Openai gym”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1606.01540> [last accessed: 10/09/2023].

Brown, S. J. *et al.* (1992). “Survivorship bias in performance studies”. *The Review of Financial Studies*, 5(4): 553-580.

Bu, S. J., & Cho, S. B. (2018). “Learning optimal Q-function using deep Boltzmann machine for reliable trading of cryptocurrency”. In *Intelligent Data Engineering and Automated Learning–IDEAL 2018: 19th International Conference*, Part I 19, Madrid, Springer International Publishing, pp. 468-480.

Campbell, M., Hoane Jr, A. J., & Hsu, F. H. (2002). “Deep blue”. *Artificial intelligence*, 134(1-2): 57-83.

Cover, T.M. & Thomas, J.A. (1991). Elements of information theory. New York, Wiley.

DeMiguel V., Garlappi L. & Uppal R. (2009), “Optimal Versus Naive Diversication: How Inefficient is the 1/N Portfolio Strategy?”. *Review of Financial Studies*, 22(5): 1915-1953.

Engstrom, L. *et al.* (2020). “Implementation matters in deep policy gradients: A case study on ppo and trpo”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.2005.12729> [last accessed 10/09/2023].

Fabozzi, F. J., Huang, D., & Zhou, G. (2010). “Robust portfolios: contributions from operations research and finance”. *Annals of operations research*, 176(1): 191-220.

Fama, E. F. (1970). “Efficient capital markets: A review of theory and empirical work”. *The journal of Finance*, 25(2), 383-417.

Fama, E. F. *et al.* (1969). “The adjustment of stock prices to new information”. *International economic review*, 10(1), 1-21.

Fama, E. F., & French, K. R. (1993). “Common risk factors in the returns on stocks and bonds”. J*ournal of financial economics*, 33(1): 3-56.

Filos, A. (2019). “Reinforcement learning for portfolio management”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1909.09571> [last accessed: 10/09/2023].

Gao, Y. *et al.* (2021). “A Framework of Hierarchical Deep Q-Network for Portfolio Management”. In *13th International Conference on Agents and Artificial Intelligence*, 2, ICAART, pp. 132-140.

Gao, Z. *et al.* (2020). “Application of deep q-network in portfolio management”. In *5th IEEE International Conference on Big Data Analytics*, IEEE, pp. 268-275.

Google Deepmind (2019). “AlphaStar: Mastering the real-time strategy game StarCraft II”. <https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii> [last accessed: 25/08/2023].

Google Deepmind (n.d.). AlphaGo Official Page. <https://www.deepmind.com/research/highlighted-research/alphago> [last accessed: 25/08/2023]

Gu, S., Kelly, B. T., & Xiu, D. (2019). “Empirical asset pricing via machine learning”. Working Paper. Chicago Booth Research Paper.

Haydari, A., & Yılmaz, Y. (2020). “Deep reinforcement learning for intelligent transportation systems: A survey”. *IEEE Transactions on Intelligent Transportation Systems*, 23(1): 11-32.

Hendrycks, D., & Gimpel, K. (2016). “Gaussian error linear units (gelus)”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1606.08415> [last accessed: 10/09/2023].

Hochreiter, S., & Schmidhuber, J. (1997). “Long short-term memory”. *Neural computation*, 9(8): 1735-1780.

Jang, J., & Seong, N. (2023). “Deep reinforcement learning for stock portfolio optimization by connecting with modern portfolio theory”. *Expert Systems with Applications*, 218, 119556.

Jiang, Z., Xu, D., & Liang, J. (2017). “A deep reinforcement learning framework for the financial portfolio management problem”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1706.10059> [last accessed: 10/09/2023].

Kober, J., Bagnell, J. A., & Peters, J. (2013). “Reinforcement learning in robotics: A survey”. *The International Journal of Robotics Research*, 32(11): 1238-1274.

Kritzman, M., & Li, Y. (2010). “Skulls, financial turbulence, and risk management”. *Financial Analysts Journal*, 66(5), 30-41.

Krizhevsky, A., Sutskever I., & Hinton, G. (2012). "Imagenet classification with deep convolutional neural networks”. In *Advances in Neural Information Processing Systems* 25, Nevada, NeurIPS, pp. 1106–1114.

Le, N. *et al.* (2022). “Deep reinforcement learning in computer vision: a comprehensive survey”. *Artificial Intelligence Review*, 55(4): 2733–2819.

LeCun, Y. *et al.* (1998). “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE*, 86(11): 2278-2324.

Li, Y. (2017). “Deep reinforcement learning: An overview”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1701.07274> [last accessed: 10/09/2023]

Liang, E. *et al.* (2018). "RLlib: Abstractions for distributed reinforcement learning”. In *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, PMLR, pp. 3053-3062.

Liu, X. Y. *et al.* (2020). “FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.2011.09607> [last accessed: 10/09/2023].

Liu, X. Y. *et al.* (2018). “Practical deep reinforcement learning approach for stock trading”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1811.07522> [last accessed: 10/09/2023]

Liu, X. Y. *et al.* (2021). “FinRL: Deep reinforcement learning framework to automate trading in quantitative finance”. In I*CAIF '21: Proceedings of the Second ACM International Conference on AI in Finance*, 1, New York, ACM, pp. 1-9.

Liu, X.Y. et al. (2021). “ElegantRL: A scalable and elastic deep reinforcement learning library”. *GitHub Repository*. <https://github.com/ AI4Finance-Foundation/ElegantRL> [last accessed: 10/09/2023].

Lucarelli, G., & Borrotti, M. (2020). “A deep Q-learning portfolio management framework for the cryptocurrency market”. *Neural Computing and Applications*, 32: 17229-17244.

Maillard, S., Roncalli, T., & Teïletche, J. (2010). “The properties of equally weighted risk contribution portfolios". *The Journal of Portfolio Management*, 36(4): 60-70.

Markowitz, H. (1952). "Portfolio Selection”. *The Journal of Finance*, 7(1): 77-91.

Meng, T. L., & Khushi, M. (2019). "Reinforcement learning in financial markets”. *Data*, 4(3), 110. <https://www.mdpi.com/2306-5729/4/3/110> [last accessed: 10/09/2023].

Merton, R. C. (1980). “On estimating the expected return on the market: An exploratory investigation”. *Journal of financial economics*, 8(4): 323-361.

Millea, A. (2021). “Deep reinforcement learning for trading—A critical survey”. *Data*, 6(11), 119. <https://www.mdpi.com/2306-5729/6/11/119> [last accessed: 10/09/2023].

Mnih, V. et al. (2013). “Playing atari with deep reinforcement learning”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1312.5602> [last accessed 10/09/2023].

Mnih, V. *et al.* (2015). “Human-level control through deep reinforcement learning”. *Nature*, 518(7540): 529-533.

Mnih, V. *et al.* (2016). "Asynchronous methods for deep reinforcement learning”. In *ICML'16: Proceedings of the 33rd International Conference on International Conference on Machine Learning*, 48, New York, JMLR, pp. 1928-1937.

Mosavi, A. *et al.* (2020). “Comprehensive review of deep reinforcement learning methods and applications in economics”. *Mathematics*, 8(10), 1640. <https://www.mdpi.com/2227-7390/8/10/1640> [last accessed: 10/09/2023].

Neuneier, R. (1995). “Optimal asset allocation using adaptive dynamic programming”. In *NIPS'95: Proceedings of the 8th International Conference on Neural Information Processing Systems*, Cambridge, MIT Press, pp. 952–958.

Norton, V. (2011). “Adjusted closing prices”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1105.2956> [last accessed: 10/09/2023].

O'Bryan, R. (2019). “Games and Game Playing in European Art and Literature, 16th-17th Centuries”. Amsterdam, Amsterdam University Press.

Raffin, A. *et al.* (2021). “Stable-baselines3: Reliable reinforcement learning implementations”. *The Journal of Machine Learning Research*, 22(1): 12348-12355.

Reilly, F. K., & Brown, K. C. (2011). “Investment analysis and portfolio management”. 10. Boston, Cengage Learning.

Sato, Y. (2019). “Model-free reinforcement learning for financial portfolios: a brief survey”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1904.04973> [last accessed: 10/09/2023].

Schulman, J. *et al.* (2015). “Trust region policy optimization”. In *ICML’15: Proceedings of the 32nd International Conference on Machine Learning*, 37, Lille, Journal of Machine Learning Research, pp. 1889-1897.

Schulman, J. *et al.* (2017). “Proximal policy optimization algorithms”. *arXiv preprint*. <https://doi.org/10.48550/arXiv.1707.06347> [last accessed: 10/09/2023]

Sermanet, P. *et al.* (2013). “Pedestrian detection with unsupervised multi-stage feature learning”. In *Proceedings of the IEEE conference on computer vision and pattern recognition,* Portland Oregon, IEEE, pp. 3626-3633.

Sharpe, W. F. (1964). “Capital asset prices: A theory of market equilibrium under conditions of risk”. *The journal of finance*, 19(3): 425-442.

Sharpe, W. F. (1998). “The sharpe ratio”. In: Bernstein, P. L. (edited by) *Streetwise: the Best of the Journal of Portfolio Management,* Princeton, Princeton University Press, pp. 169-85.

Silver, D. *et al.* (2014). “Deterministic policy gradient algorithms”. In *ICML’14: Proceedings of the 31st International Conference on International Conference on Machine Learning*, 32, Beijing, Journal of Machine Learning Research, pp. I-387–I-395.

Silver, D. *et al.* (2016). “Mastering the game of Go with deep neural networks and tree search”. *Nature*, 529(7587): 484-489.

Silver, D. *et al.* (2017). “Mastering the game of go without human knowledge”. *Nature*, 550(7676): 354-359.

Singh, B., Kumar, R., & Singh, V. P. (2022). “Reinforcement learning in robotic applications: a comprehensive survey”. *Artificial Intelligence Review*, 55(2): 945–990.

Sutton, R. S. & Barto, A. G. (2018). Reinforcement learning: An introduction. 2. Cambridge, MIT press.

Sutton, R. S. (1988). “Learning to predict by the methods of temporal differences”. *Machine learning*, 3: 9-44.

Sutton, R.S. *et al.* (1999). “Policy gradient methods for reinforcement learning with function approximation”. In *Advances in neural information processing systems 12, Cambridge*, MIT Press, pp. 1057-1063.

Tesauro, G. (1994). "TD-Gammon, a self-teaching backgammon program, achieves master-level play". *Neural computation*, 6(2): 215-219.

Todorov, E., Erez, T. & Tassa, Y. (2012). "MuJoCo: A physics engine for model-based control". In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, IEEE, pp. 5026-5033.

Van Hasselt, H., Guez, A., & Silver, D. (2016). "Deep reinforcement learning with double q-learning". In *Proceedings of the AAAI conference on artificial intelligence*, 30(1), Phoenix Arizona, AAAI Press, pp. 2094–2100.

Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). "A review on the long short-term memory model". *Artificial Intelligence Review*, 53(8): 5929-5955.

Vaswani, A. *et al.* (2017). "Attention is all you need". In *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, 30, Red Hook NY, Curran Associates Inc., pp. 6000–6010.

Vinyals, O. *et al.* (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". *Nature*, 575(7782): 350-354.

Wang, Z. *et al.* (2016). "Dueling network architectures for deep reinforcement learning". In *ICML'16: Proceedings of the 33rd International Conference on International Conference on Machine Learning*, 48, New York, Journal of Machine Learning Research, pp. 1995-2003.

Watkins, C. J., & Dayan, P. (1992). "Q-learning". *Machine learning*, 8, 279-292.

WAVENURE. Home page. <https://www.wavenure.com/> [last accessed: 10/09/2023].

Werbos, P. J. (1988). “Generalization of backpropagation with application to a recurrent gas market model”. *Neural networks*, 1(4): 339-356.

Wikipedia (2023). “Observable Universe” <https://en.wikipedia.org/wiki/Observable_universe#> [last accessed: 21/08/2023].

Yang, H. *et al.* (2020). “Deep reinforcement learning for automated stock trading: An ensemble strategy”. In *ICAIF '20: Proceedings of the First ACM International Conference on AI in Finance*, 31, New York, ACM, pp. 1-8.

Yu, C. *et al.* (2021). “Reinforcement learning in healthcare: A survey”. *ACM Computing Surveys (CSUR)*, 55(1): 1-36.

Zhang, Z., Zohren, S., & Roberts, S. (2020). “Deep learning for portfolio optimization”. *The Journal of Financial Data Science*, 2(4): 8-20.