Department of Economics and Management

Bachelor's Degree in Management and Computer Science

Course of Artificial Intelligence and Machine Learning

"Efficient Portfolio Allocation: Reinforcement Learning methods applied to modern finance"

Candidate: Federico Del Nobile

ID number: 229451

Professor: Querini Marco

Academic Year: 2020/2021

SUMMARY

1 Introduction:…………………………………………………………………………..5

| 1.1 Reinforcement Learning and Finance……………………….…………….....5

| 1.2 The Efficient Porfolio Allocation Problem......…………..……………....7

2 Reinforcement Learning Applications:……………………………………….....8

| 2.1 RL General Applications:…………………………………………………8

| | 2.1.1 RL as a Controller……………………………………………...8

| | 2.1.2 RL as an Optimizer…………………………………………….....10

| 2.2 RL Finance Applications:…………………………………………………11

| | 2.2.1 Single/Multiple Stocks Trader……………………………………11

| | 2.2.2 American Option Pricing…………………………………………12

| | 2.2.3 Order Book Execution……………………………………………12

| | 2.2.3 Optimal Hedging of Derivatives…………………………………13

3: RL Definitions and Models………………………………………………………14

| 3.1 General RL Definitions:……………………………………………………..14

| | 3.1.1 Agent-Environment Interactions…………………………………14

| | 3.1.2 Policy, Trajectories and Rewards……………………………....16

| | 3.1.3 The Optimization Problem………………………………………18

| | 3.1.4 Value Functions, Advantage Function and Bellman Equations….19

| 3.2 Models Applied:……………………………………………………………..22

| | 3.2.1 Taxonomy of RL Approaches……………………………………23

| | 3.2.2 Basic Policy Optimization………………………………………..25

| | 3.2.3 A2C ……………………………………………………………...28

| | 3.2.4 PPO………………………………………………………………30

| | 3.2.5 DDPG……………………………………………………….32

| | 3.2.6 SAC…………………………………………………………35

4 Efficient Portfolio Allocation:……………………………………………..39

| 4.1 Problem Formulation……………………………………………...40

| 4.2 An Introduction to Practice……………………………………….42

5 Implementation:……………………………………………………………43

| 5.1 The Stock Market:………………………………………………...43

| | 5.1.1Environment Setup……………………………………….43

| | 5.1.2Training and Backtest……………………………………..45

| | 5.1.3Hyperparameter Tuning…………………………………..48

| 5.2 The Bonds Market:………………………………………………..50

| | 5.2.1 Environment Setup, Training, and Backtest……………...51

| | 5.2.2 Hyperparameter Tuning………………………………….53

| 5.3 The Cryptocurrencies Market……………………………………..54

| | 5.3.1 Environment Setup, Training, and Backtest……………...55

| | 5.3.2 Hyperparameter Tuning………………………………….57

| 5.4 Results Discussion………………………………………………..58

6 Web App:…………………………………………………………………..60

| 6.1 Introduction and Motivations……………………………………..60

| 6.2 User Experience and Inner Functioning…………………………..61

| 6.3 Further Developments…………………………………………….64

7 General Conclusions………………………………………………………65

Bibliography………………………………………………………………...66

Appendix 1…………………………………………………………………..67

Abstract

This work explores the potential of Reinforcement Learning methods applied to financial markets and, more specifically, on the Efficient Portfolio Allocation problem. Four different Reinforcement Learning algorithms, A2C, PPO, DDPG, and SAC, are tested on three different financial markets, Stocks, Bonds, and Cryptocurrencies, adopting the implementations of the Finrl Library(1). The results of the analysis showed the superiority of Policy Optimization algorithm with respect to Mixed ones and, generally, that different algorithms perform better in different markets. Following the results of the analysis, a web app is then developed to provide proof of how these results can be exploited practically.

SECTION 1: INTRODUCTION

1.1

Reinforcement Learning and Finance

Finance has always been a science about data. Since its earliest days, investors from all around the world have constantly tried to read, interpret and foresee fluctuations of the market, in order to understand and speculate off of them.

The more time passes and the more sophisticated the methods to do such work become.

Nowadays, the plethora of different tools, helpers, and courses that are available online are making finance more accessible to the large public than ever before and, as a result, the interest in machine learning and artificial intelligence methods applied to the subject has raised significantly over the last decade.

Among all the possible approaches, Reinforcement Learning (RL) is surely the one that seems to have the more potential to outclass the others in the long run, due to its intrinsic nature.

The specific characteristics of this vast class of algorithms are, in fact, especially suited for problems that require sequential trial and error, evaluation of a strategy, and a constant update of the plan given the new information acquired.

RL works in a much similar way to the one of the human being: given an environment that is responsive to the algorithm actions, the model is capable of repeatedly taking decisions based on past experience, evaluate each time the response of the environment and adapt its strategy in light of the ambient reaction.

Finance seems, therefore, a perfect field of application as every aspect of it is easily encodable in a way that can fit the general assumptions of RL.

Given its iterative nature, for example, each day in which the market opens and closes can be seen as what will be later called an “episode”, the stock market is easily taken as the “environment” and the selling of the stocks itself can be encoded as the actions that the “agent”, our model, can take. Furthermore, the returns we get from our strategy can represent the “rewards” that our model gets, and the way and rationale with which the model decides what and when to sell can be assumed to be its strategy or “policy”.

In addition, with respect to other common fields of application of RL, finance has some peculiarities that make it particularly easy to explore.

First of all, the incredible quantity and ease of to access the data. Thanks to software like Yahoo Finance and their API, retrieving financial data of basically any sort is nowadays extremely easy and fast. Furthermore, financial data are usually structured and formatted in a way shared among all different markets around the world increasing the interoperability of the whole ecosystem around them.

Secondly, the time delay between an action and the related reward, for which a high value can cause real troubles training the models, is pretty low in this case: since the maximum time you have to wait for the response is, usually, the closing of the market, you have a fixed and predictable time horizon for your rewards and you can easily match them with the causing action.

Finally, the high interpretability and precision of the results is another plus that finance gets over other fields.

Differently, for example, from RL applied to self-driving cars, in which is difficult to rank the performance of the models with granularity, in finance the existence of simple, interpretable, and continuous measures, such as the returns and the Sharpe ratio, substantially help in the selection phase in understanding the real differences between similar models.

Given all that, it is easy to understand why always more people are beginning to apply RL to quantitative finance and it is foreseeable that this number will almost surely continue to increase.

1.2

The Efficient Portfolio Allocation Problem

The Efficient Portfolio Allocation problem is defined as finding the optimal combination of securities that, given a portfolio containing these securities, maximizes the value of the portfolio itself over time,

Efficient Portfolio Allocation has always been one of the primary objects of study for financial researchers through the 20th century and it is still today.

Academic approaches tried to find a theoretical form of an efficient portfolio, as one, for example, that lied on the Security Market Line, while RL takes a bit of a more practical direction.

Benefitting from all the points described above and from an almost natural fit of the problem with their structure, RL models can look at an incredible amount of historical data, recognize and extract valuable patterns between them, and learn the best way to maximize the profits of the given portfolio by altering the composition of the portfolio itself.

On top of all of that, they can do this in a highly automated manner opening the doors for a wide range of possibilities. Given the recent spreading of algorithmic trading, is not impossible to foresee that such models could be commercialized through a web app, as a SAS (software as service), for investors who are willing to pay for such services.

To conclude, EPA is a classical finance problem on which RL is trowing new light and is, therefore, interesting to explore it to understand to which extent results produced by automated models can lie above the ones obtained by non-algorithmic approaches.

•

The work will be structured as follows: in chapter 2 will be given a general overview of the various applications of Reinforcement Learning methods across different fields and, more specifically, in finance; in chapter 3 will be explained the theory behind the Reinforcement Learning methods adopted in the analysis, including the basic definitions and the mathematical formulation of the algorithms utilized; in chapter 4 an introduction and a precise formulation will be provided for the Efficient Portfolio Allocation problem; in chapter 5 the practical application of the algorithms and the results obtained will be described; in chapter 6 the development of a web app implementing the obtained models will be showed and, finally, in chapter 7 a brief conclusion and some thoughts on further improvements to the work will be given.

SECTION 2: REINFORCEMENT LEARNING APPLICATIONS

2.1

RL General Applications

Before starting to deal with Efficient Portfolio Allocation itself, it feels necessary to mention the numerous fields in which Reinforcement Learning methods found application, to get a more general understanding of the subject and give a more defined picture of what these algorithms are capable of.

Given the assumptions of such models, RL algorithms are best suited in situation which require a complex and repeated decision-making process, in which passed decision can have an influence on future states of the environment and in which the wide variety of actions allowed in the environment itself makes finding the best possible sequence a highly nontrivial problem.

Furthermore, being substantially different from both supervised and unsupervised learning, RL can often offer a different angle to look at a problem or completely unlock solutions that could not have been possible without it.

For this reasons, RL, in its admittedly short history, has already found a large number of useful applications ranging from the Medical sector to the music composition and it is likely to find even more as new and powerful RL algorithms continue to be developed every day.

2.1.1 RL as a Controller

One of the classes of problems in which RL is most widely adopted is the one of controllers. By a controller, it is intended a problem in which an algorithm needs to control an entity, in a physical or simulated environment, and make it perform some actions in the latter, in order to reach an objective.

Examples of these can be pretty simple, as the classic “Cart Pole Balancing” problem, also supported on the Open AI Gym platform, but can also reach a pretty high complexity, especially once the environments start to be refined to resemble more and more the real world.

One of the most famous papers treating the argument is “Playing Atari with Deep Reinforcement Learning” (2) which for the first time introduced Deep Reinforcement Learning, a method in which Deep Learning is applied to RL. In the paper, the authors train a DQN (Deep Q Network) capable of

playing a vast variety of Atari games, reaching results for the most comparable if not superior to the ones achieved by the best human players of the time.

Applications of this kind can also easily be found in robotics, where the adoption of RL seems almost a natural development of the field. RL in fact, fits perfectly with the needs of the researcher, often acting as the brain that controls the robot.

Studies inspecting the capability of RL models as movement controllers have thrived in these recent years, for both four-legged robot, as described in “Learning to Walk via Deep Reinforcement Learning” (3), and for two-legged ones, as depicted in “A Simple Reinforcement Learning Algorithm For Biped Walking” (4). In both of these papers Reinforcement Learning methods are applied to regulate the inclination, angular velocity and speed of the various parts of the robot legs, in order to make it stand, stabilize it, and finally walk.

Another field of robotic that utilizes RL algorithms is robot dexterity, a branch that aims to make robots able to cope with a variety of objects and actions. An example that comes to mind can be a robot in a factory, able to interact and rightfully assemble the pieces of the object it is building or, as described in “Learning dexterous in-hand manipulation system overview” (5) a robotic hands that, exploiting a Proximal Policy Optimization (PPO) algorithm is capable to rotate a solid cube in order to display the desired face to the viewer.

Finally, Self-Driving cars are another very famous example of RL utilized as a controller. In this case, the algorithm is used to regulate anything related to the car, from its speed to the direction, and to analyze and react to the surrounding environment in order to constantly update the previous commands.

As described in “Deep Reinforcement Learning framework for Autonomous Driving” (6), supervised learning, especially Deep Neural Network, can also be adopted to solve the same problem but, as the simulated environment becomes more similar to the real world, this class of methods begins to stumble. RL models substantially improve the reliability of the controllers, since they can better manage the need of *“...strong interactions with the environment, including other pedestrians, vehicles, and roadworks…”* that the implementation requires.

2.1.2 RL as an Optimizer

In many other cases, RL methods are implied as optimizers.

Differently from the previous situations, this kind of problem requires a certain quantity to be optimized, i.e. maximized or minimized, by finding an optimal strategy, or the sequence of action that yields the best results. The edge that RL methods gain over more traditional approaches this time is due to the fact that they do not adopt a greedy strategy to solve the problem, but instead look to maximize the discounted sum of rewards in the long run. Basically, since RL algorithms are not looking for, as traditional regressors, predicting time after time a single value from a set of data, but instead to find the optimal sequence of actions that results in the best possible outcome over time, this class of algorithm makes for a perfect fit for such kind of problems, and its wide adoption in these use cases makes clear evidence of that.

A field that is much benefitting from the introduction of RL algorithms is, rather surprisingly, the online advertisement one. The way in which the selling of online advertisement space works in fact, as a series of real-time bids, makes the finding of an optimal bidding strategy an essential problem to solve for each vendor. As shown in "Budget Constrained Bidding by Model-free Reinforcement Learning in Display Advertising" (7), RL algorithms, particularly DRL ones, can be successfully applied to optimize the amount of money to be placed on each bid, depending on the budget and on the characteristic of each page, to maximize the responsiveness of the public to the campaign.

Furthermore, RL found applications also in the transportation sector, where such models are being utilized by services like Uber, for example, to efficiently match riders and travelers in order to minimize the waiting time of each of the two groups. As shown in the work presented at the KDD Tutorial London 2018 (8), again RL algorithms' capabilities are incredibly useful in the situation since each matching decision, affecting future supply, needs to be evaluated carefully, taking into consideration the rewards in the long run and not the immediate results.

Finally, RL made its way also to the public sector, where interesting applications are being developed in the traffic reduction field. The work of Arel et Al. (9) for example shows how RL methods can produce interesting results on the matter, optimizing the scheduling of light signal in order to *"... minimize the average delay, congestion, and likelihood of intersection cross-blocking..."*.

Again the high complexity of the problem and the presence of correlation between past actions and future environment states would make the problem almost unsolvable with traditional deep learning approaches and gives a perfect outline on how RL algorithms are unlocking solutions that were not possible before.

2.2

RL Finance Applications

Following the previous section, it feels now dutiful to complete the general overview of Reinforcement Learning applications with some examples and implementations of RL models in the world of finance.

Finance is a field particularly well suited for machine learning in general and, even more, for RL approaches, since many problems proper of the subject can be seen as sequential or Markov decision processes.

In addition, it is important to remember that also factors more related to practice, as the abundance, precision, interoperability, and intrinsic structure of financial data, the reduced time delay between actions and rewards, and the ease of ranking different models provided by the existence of a high number of different financial indicators, make finance an extremely easy field for researchers to enter and, due to this fact, an extremely explored one.

2.2.1 Single/Multiple Stock Trader

A factor that also contributes to the appeal of financial-related RL is that even problems that are easy to formulate can produce results that are in practice valuable for the average investor. A classic example of that is the Single/Multiple stock trader. The problem is, indeed, relatively simple: an investor posses a portfolio of given assets and, at any new time step t, the algorithm has to read the state of the environment and decide which of the assets suggesting to sell and which suggesting to buy at t+1, to achieve a maximum gain in the long run.

It is easy to see how a properly trained model can be exploited as a powerful financial tool, perfect to help crafting a prudent investment strategy, and how meaningful results can be obtained also without particular knowledge of the subject.

A complete formulation of this problem can be found in the documentation of the Finrl library (10), a tool also used for the implementation of the algorithms present in this work.

2.2.2 American Option Pricing

Another interesting financial application resides in the American Option Pricing problem (11). In this case, due to the functioning of American Options, the investor constantly faces the decision to exercise the contract, so to buy or sell the underlying assets for the exercise price, in which case the reward is immediately evaluated as the gain or loss to the initial investment, or to wait for a better time to do it. The objective of the RL algorithm is, therefore, to decide the best time in which to exercise to obtain maximum gain. The formulation of the environment depends case by case, on the underlying asset itself and the factors that can influence its value. In the eventuality of stock as underlying, such factors could be, for example, of economic nature, such the interest rate, of financial one, such the value of the market portfolio, or even external ones, such as important political events. Since time is so deeply related to the options' price, due to their formulation, the former is also usually taken into consideration by the models.

2.2.3 Order Book Execution

A peculiar use case can also be made for the Oder Book Execution problem (11). The objective of the model is, this time, to sell or buy a detrmined volume of a single stock before a given time horizon, trying to maximize the gain when selling, or minimize the expenditures when buying. An additional layer of complexity is added to the problem by the assumption that, in limit order markets, investors involved can not only specify the desired volume but also the desired price, so that can happen the situation in which there are enough stocks to be sold on the market to satisfy the demand but buyers and sellers cannot be matched due to the difference in required prices.

The actions the model can take all refer to modifying the current outstanding limit relative to its ask, the price to sell, or bid, the price to buy, in such a way that there is always just one limit outstanding.

The state instead, can be composed by private variables, as the elapsed time or the remaining volume to trade, or market ones such as bid-ask spread, the difference between mean ask and bid prices in the current book, bid-ask volume imbalance, and immediate market order, or the cost of trading the remaining shares immediately at the market price.

A more specific formulation of the problem and model implementations can be found at (12).

2.2.4 Optimal Hedging of Derivatives

Finally, Optimal Hedging of Derivatives (13) is another example of how RL methods can help find an efficient response to classical financial questions.

The problem arises from the work of Black and Scholes (1973) and the assumptions of their model.

The idea is that, in a complete and frictionless market, there exists a continuously rebalanced dynamic trading strategy in stocks and riskless security that perfectly replicates the position of an option. In the real world though, continuous trading of arbitrarily small fractions of a portfolio end up being prohibitively costly, and therefore perfect replication is impossible. Discrete rebalancing is indeed needed in order to approximate the position of an ideal option holder, but finding the optimal tradeoff between replication error and trading cost, as well as the optimal policy to achieve it is a highly nontrivial task.

The state of a simple instance of this problem can include information about the remaining time before the expiration date and the position of the agent with respect to the n stocks in the portfolio, while the set of action at the agent disposal is identified as the possibility of rebalancing the portfolio at will.

A RL approach is here implied as the position of the option must be replicated during all the time up until the expiration date, and therefore a sequential decision model is required.

A complete formulation of the problem and of the models applied can be found at (13)

SECTION 3: REINFORCEMENT LEARNING DEFINITIONS AND MODELS

3.1

RL Key Concepts and Terminology

After providing a general overview of various applications of Reinforcement Learning methods, it is now time to go a little bit deeper into the functioning of such class of algorithms and describe the basic concepts that all of them have in common.

This section will also set a precise terminology that will be later adopted to describe the processes carried on and the models utilized in this work.

3.1.1 Agent-Environment interactions

There are three main actors involved in every Reinforcement Learning algorithm and these are the Agent, the Environment, and the Rewards.

Despite many different iterations and variations of the possible interactions between these actors, which are proper of different kinds of algorithms, the main cycle and core of every RL process can be summarized as depicted in figure 3.1.

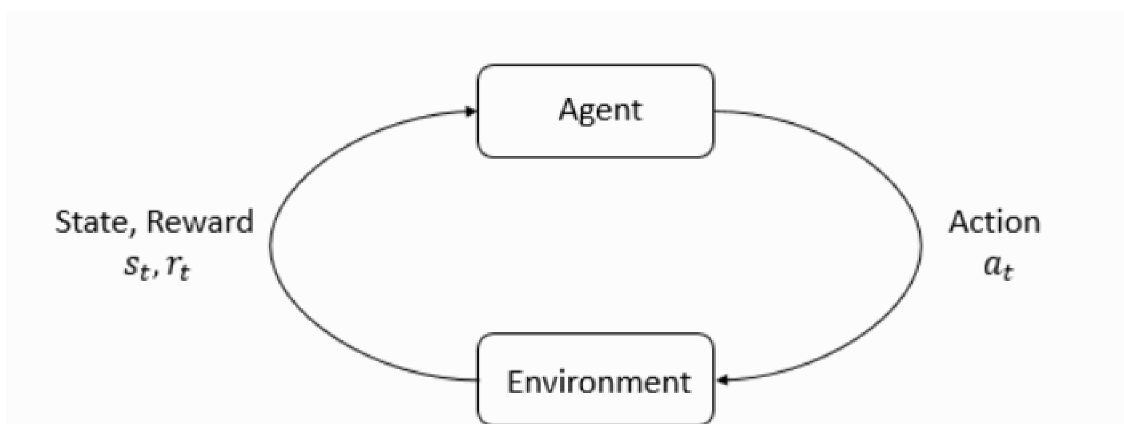


Figure 3.1: Agent-Environment interaction loop

The loop is composed as follows. The agent, usually a piece of software, at each timestep t chooses an action a_t according to a policy π after observing the current state of the environment s_t.

For environment is intended the world the agent lives in. Its specific characteristics depend on the problem the algorithm is trying to solve. For a robot that is learning how to stand, the environment could be a simulation of the real world, where objects are subject to the laws of physic, while for an agent that is learning how to trade stocks, the environment could resemble a stock trading market, and provide real-time data about prices, volumes and so on.

The action to perform at each step is instead chosen from what is called the Action Space, defined as the set of all the actions that are possible to the agent in a given environment. It is important to notice that, depending on the case, the set of actions can be continuous, as the one containing all the possible velocities at which a self-driven car can go, or discrete, as the one containing all the moves a chess-player agent can execute in a given moment of a chess game.

The state of the environment finally, is the complete set of information that describes the environment itself at time t. For the legged robot example, the environment state can comprehend the orientation and angular velocity of the parts composing its legs, as well as the inclination of the ground or the presence of other objects nearby it, while for the chess player agent, the state could be easily described as the position of all the pieces on the chessboard as well as information about the pieces that have already been eaten during the game.

Furthermore, for sake of completeness, is important to draw a distinction between states and observations where the latters are defined as partial observations of the complete state of the environment. In this section, since no real problem is analyzed, complete information about the state of the environment is assumed in every step.

Going back to the loop, after the choice, the agent executes the action a_t in the environment and get a reward r_t and the new state of the envirormet s_{t+1}. The environment can change in response to the actions of the agent but it can also change independently from them.

After the agent receives the reward and the new state, the cycle starts back again.

3.1.2 Policies, Trajectories and Rewards

After describing the basic iteration between the agent and the environment, it is time to inspect further the way in which the agent takes the decisions at each timestep.

In the previous section is stated that the agent chooses actions according to a policy and a policy can be in fact defined as the rule by which the agent decides what actions to take.

The policy is basically the brain of the agent, the strategy it adopts to reach its objectives.

Policies can be of two types, deterministic and stochastic.

For deterministic policies is intended a rule that assigns a given state s_t to an action a_t in a deterministic way, outputting always the same action for the same state. A deterministic policy is usually denoted by μ and the actions taken by such policy can therefore be obtained as:

$$a_t = \mu(s_t)$$

Stochastic policies instead are rules that, for a given state s_t, output the relative probabilities of each action in the action space depending on the state. The agent then samples the action from the action space according to the distribution formed by these probabilities, resulting in the possibility of executing different actions for the same state.

Stochastic policies are denoted by π and the actions taken by an agent following them are obtained as:

$$a_t \sim \pi(\cdot|s_t)$$

Trajectories are another important piece of the picture.

A trajectory, often also called an episode, is defined as a sequence of states and actions in the environment and is denoted as :

$$T=(s_0,a_0,s_1,a_1.....)$$

where the first state s_0 is randomly sampled by the start-state distribution denoted as ρ_0.

The composition of a trajectory depends therefore both on the action taken by the agent, the start-state distribution but also by the so-called state-transition, or the rule that determines how the environment changes between s_t and s_{t+1}. As policies, state-transitions can be deterministic:

$$s_{t+1} = f(s_t, a_t)$$

or stochastic,

$$s_{t+1} \sim P(\cdot|s_t, a_t)$$

Finally, the notion of rewards is another essential piece to the comprehension of RL algorithms.

The reward function R is the function that evaluates the reward of the agent at each step and usually depends on the current state s_t, the current action a_t and the next state of the environment s_{t+1}. For sake of simplicity the reward function is often formulated in order to depend only on the state, $R(s_t)$, or on the state-action pair, $R(s_t,a_t)$. This last form is adopted during this work.

The goal of an RL agent is to maximize the value of cumulative return over a trajectory T.

The cumulative return R(T) is defined as the sum of rewards obtained by the agent following a given trajectory T, and can be of two types. The first one is the finite-horizon undiscounted return computed as

$$R(\tau) = \sum_{t=0}^{T} r_t.$$

where

$$r_t = R(s_t, a_t)$$

It is basically just the sum of rewards computed over a given time horizon or number of steps, where for a step is intended a complete cycle of agent-environment interaction.

The second type of cumulative return is instead called the infinte-horizon discounted return, computed as

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

In this case, the return is evaluated over all the rewards ever obtained by the agent, but each of them is now discounted by a discount factor γ between 0 and 1 elevated to the number of steps occurred from the start of the epiisode to the obtainign of the reward.

The rationale behind the discount factor is twofold. On one hand, as finance teaches us, a dollar today is better than a dollar tomorrow, and therefore rewards acquired sooner should have a higher value with respect to the ones acquired later. On the other hand, the discount factor gives the mathematical certainty that the infinite sum or rewards will eventually converge to a finite number, under reasonable conditions, and it is, therefore, necessary for computations.

3.1.3 The Optimization Problem

What is, therefore, the optimization problem that Reinforcement Learning is trying to solve?

As stated in the previous paragraph, the objective of every RL algorithm is to maximize the sum of rewards, finite or infinite, over a trajectory T. The evolution of a trajectory does not depend only on the agent's actions though, and it is, therefore, not predictable by it. It would not make sense then, to optimize the rewards over a single given trajectory, since the agent would not be able to produce it and it would be probable, instead, that the agent would never follow such trajectory.

The solution to this impasse is to find a policy π that, instead of maximizing returns over a single trajectory, maximizes the *expected* return over all the possible trajectories that the agent could follow. In this way, an optimal solution can be reached in any circumstance, giving to RL the capability to obtain much more meaningful results.

Let's expand on this reasoning.

Supposing that both the policy and the state-transition are stochastic, the probability P to follow a given trajectory, in this case, would be given by

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$$

The expected return, denoted by J(π), can be defined, for whichever return measure chose, as

$$J(\pi) = \int_\tau P(\tau|\pi)R(\tau) = \underset{\tau\sim\pi}{\mathrm{E}}[R(\tau)]$$

The optimal policy, therefore, would be the one for which the value of J(π) is maximized, and can be mathematically expressed by

$$\pi^* = \arg\max_\pi J(\pi).$$

where π^* denotes the optimal policy.

In practice, the research for the optimal policy is performed by an iterative optimization algorithm called gradient ascent, and the optimal policy is approximated using parameterized neural networks as function approximators. Policies obtained in such ways are denoted in a slightly different way, to underline the fact that they depend on a set of parameters θ. For stochastic policies, π_θ is used, while μ_θ is adopted for deterministic ones.

The research of the optimal policy will be better covered in section 3.2.2.

3.1.4 Value Functions, Advantage Function and Bellman Equations

In the research of the optimal policy, it is often useful to be able to evaluate the value of a state or the one of a state-action pair. In this case, the word *value* has to be intended as the expected reward the agent will get by starting from that state/state-action pair and then acting according to a policy π from then on. Value functions serve this exact purpose and they are, more or less, adopted by all the RL algorithms.

There exist four types of value function, depending on whether they are On-policy or Optimal, and whether they are just value or action-value function.

The first one, called the On-Policy Value Function, describes the value of a state s_0, given as a fact that the agent will start from there and that will follow the policy π from then on. It is s defined as:

$$V^{\pi}(s) = \underset{\tau \sim \pi}{\mathrm{E}} \left[R(\tau) \,|s_0 = s \right]$$

The second one, the On-Policy Action-Value Function, describes the expected return of an agent that, starting from a state s_0, performs the action a_0, and then acts accordingly to policy π. It is defined as:

$$Q^{\pi}(s,a) = \underset{\tau \sim \pi}{\mathrm{E}} \left[R(\tau) \,|s_0 = s, a_0 = a \right]$$

The third one, the Optimal Value Function, express the expected reward of an agent that, starting from a state s_0 act only according to the *optimal* policy. It is defined as:

$$V^{*}(s) = \max_{\pi} \underset{\tau \sim \pi}{\mathrm{E}} \left[R(\tau) \,|s_0 = s \right]$$

Finally, the Optimal Action-Value Function expresses the expected return of an agent that starts in the state s_0 performs an action a_0 and then just acts according to the *optimal* policy. Note that a_0 does not have to be given by the optimal policy itself . The Optimal Action-Value Function is defined as:

$$Q^{*}(s,a) = \max_{\pi} \underset{\tau \sim \pi}{\mathrm{E}} \left[R(\tau) \,|s_0 = s, a_0 = a \right]$$

All of the four Value Functions obey special self-consistency equations called the Bellman Equations. The Bellman Equations imply a recursive formulation of the Value Functions and spring from the idea that the value of your starting point could be seen as the value you expect to get from starting from there plus the value you expect to get from wherever you land from there on.

Bellman's equations for the On-Policy Value Functions are defined as:

$$V^{\pi}(s) = \underset{\substack{a\sim\pi \\ s'\sim P}}{\mathrm{E}} \left[r(s,a) + \gamma V^{\pi}(s') \right],$$
$$Q^{\pi}(s,a) = \underset{s'\sim P}{\mathrm{E}} \left[r(s,a) + \gamma \underset{a'\sim\pi}{\mathrm{E}} \left[Q^{\pi}(s',a') \right] \right]$$

while for Optimal Value Functions are defined as:

$$V^{*}(s) = \max_{a} \underset{s'\sim P}{\mathrm{E}} \left[r(s,a) + \gamma V^{*}(s') \right],$$
$$Q^{*}(s,a) = \underset{s'\sim P}{\mathrm{E}} \left[r(s,a) + \gamma \max_{a'} Q^{*}(s',a') \right]$$

As can be inferred from these formulas, each of these equations can be recursively unfolded to obtain a discounted sum of rewards, going back to the original definition of return given in 3.1.2

An important difference between the two sets of equations is the presence of the max over action in both the equations for Optimal Value and the Optimal Action-Value Function. The max operator implies, in fact, that the agent is following the optimal policy by taking the action that, each time, maximizes the discounted sum of rewards.

Finally, in a variety of situations, the possibility of evaluating the goodness of an action relative to the others on average can result extremely useful. This measure is called *advantage* and represents the edge gained by the agent performing a specific action a_t instead of randomly selecting one according to its policy π.

Before introducing the Advantage function though, it is necessary to explain the mathematics behind it and, again, Value Functions are present in the mix.

Given the definitions of the two groups of Value Functions in fact, we can establish two key connections:

$$V^{\pi}(s) = \underset{a\sim\pi}{\mathrm{E}} \left[Q^{\pi}(s,a) \right]$$

for On-Policy and

$$V^{*}(s) = \max_{a} Q^{*}(s,a)$$

for Optimal Value Functions.

The first one is easy to explain: since the Q function represent the value of executing an action in a state, the average value of the Q function computed over all the actions possible in that state will

represent, for the definition of expected value, the value of the relative V function for the same state.

For the second one instead, the value of the optimal V function is just equal to the value of the Optimal Q function that chooses as a_0 the optimal action, or the one that follows the optimal policy.

Given these two relations, we can describe a function that gives us a measure of advantage as:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s).$$

Given a state s a policy π and an action a, the advantage function is able to describe the edge gained by the agent that executes that action over executing one randomly chosen according to policy π. If the advantage is negative, the action is worse than the average.

The advantage function represents a powerful computation tool, as it is able to give a relative measure for the “goodness” of an action and provide a way to compare different strategies. Again, this is a fundamental tool that is adopted by the majority of RL algorithms.

3.2

Models Applied

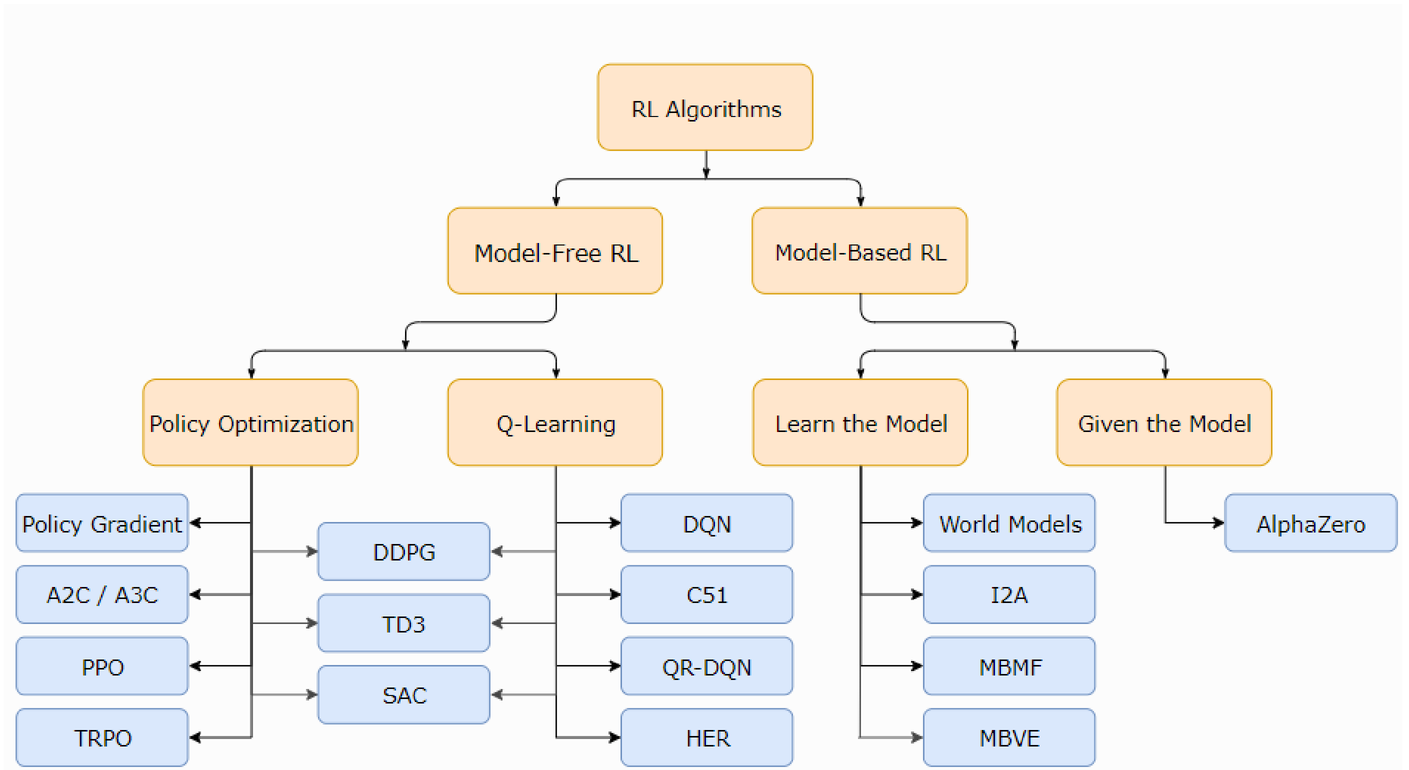


Figure 3.2: Non exaustive taxonomy of RL algorithms

Figure 3.2 depicts a non-exhaustive taxonomy of modern RL approaches.

Before describing the models utilized in this work, it is necessary to provide some information about the differences between broader categories of algorithms present in the RL space, in order to better understand the rationale between the possible choices of such algorithms and the tradeoffs that are present among them. It will be then provided a brief overview of a basic policy optimization approach, anticipating the in-detail description of the models.

3.2.1 Taxonomy of RL Approaches

The first basic distinction that can be traced between RL approaches, separates model-based and model-free algorithms.

What is intended as a model here, is a function that approximates the state transition and the reward function, so that the agent can predict those values at each timestep and evaluate the goodness of each action from those. This can result, when it works, in a much enhanced sample efficiency with respect to model-free algorithms, which instead, do not try to learn a model of the environment, but to learn directly the Optimal Policy or an approximation of the Q function as will be later explained.

The advantage in efficiency of the model-based algorithms is, in some way, limited by the fact that their agent can often exploit bias in the model and, therefore, while having great performances in the training, they tend to perform pretty poorly in practice most of the times.

The greater popularity, at the time of writing, of model-free approaches, is then explained by their often easier implementation and tuning, as well as the fact that they are generally faster to train.

Given that the algorithms adopted in this work are all model-free, a detailed description of model-based approaches is out of the scope of this text, but a more precise description can be found at (14).

Among model-free algorithms a further distinction can be then traced, between models that try to learn the optimal policy and ones that, instead, try to approximate the Optimal Q Function. This second branch is often referred to as Q-Learning.

Policy optimization algorithms try to learn directly an approximator of the Optimal Policy, denoted by $\pi_\theta(a|s)$, that maximises the objective $J(\pi)$. This is often done by adopting a gradient ascent algorithm to update the policy parameters θ. Optimization can be performed directly on the objective, as in A2C/A3C models, or on some local approximation of it as in PPO models. The optimization of the parameters in this family of methods is almost always performed On-Policy, which means that only data gathered by the agent acting according to the latest version of the policy can be utilized to update them.

Often, policy optimization algorithms also learns an approximator $V_\Phi(s)$ of the On-Policy value function, which helps in understanding how to update the parameters of the approximator $\pi_\theta(a|s)$.

Q-Learning algorithms, instead, aim to indirectly derive the optimal policy by learning an approximator $Q_\theta(s,a)$ of the optimal Q function and then choosing each action according to the rule:

$$a(s) = \arg\max_a Q_\theta(s, a)$$

Here, Bellman Equations help to explain the rationale behind this formula. Since the Optimal Q Function, according to Bellman Equations, express the value of a specific action plus the value derived by following the Optimal Policy from then on, choosing each time the action that maximises the value of the Optimal Q Function implies acting according to the Optimal Policy itself.

The optimization of this family of algorithms is almost always performed Off-Policy, which implies the capability of using data gathered in all the training process, regardless of the policy used to obtain these data.

The tradeoffs between these two classes boil down to, on one hand, greater stability for the Policy Optimization models while, on the other, a greater sample efficiency for the Q-Learning ones.

Since Policy Optimization methods directly aim for the best policy, that is the thing you want to find, they are less prone to faliures and this tends to confer them higher reliability. Q-Learning algorithms, instead, only indirectly optimizing for the policy by learning the Optimal Q Function, are subject to a wider variety of fail cases but, when they work, they gain an advantage in sample efficiency since, optimizing Off-Policy, they are able to more efficiently make use of the data at their disposal.

Interpolation between the two methods is also possible and mixed methods can often be utilized to find a midpoint between the tradeoff of these two classes.

A perfect example of this is Deep Deterministic Policy Gradient (DDPG), a method that learns both an approximator for the Optimal Policy and one for the Optimal Q function, using one to optimize the other and the other way around.

3.2.2 Basic Policy Optimization

The last topic that needs to be covered before start diving into the formulation of the models adopted in this work, is a general overview of the derivation of the basic policy gradient.

Starting from scratch, the objective is to find a parametrized policy π_θ that maximizes the expected value of the return $J(\pi_\theta)$ formulated as:

$$J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} [R(\tau)]$$

For this specific example, it is assumed the policy to be stochastic and the R(T) to represent the finite-horizon undiscounted return, but the derivation for the infinite-horizon discounted return is very similar.

The algorithm adopted to optimize the policy is called gradient ascend and consists of iteratively computing the gradient of the objective $J(\pi_\theta)$ and updating the set of parameters θ, following the gradient, to find the parameters that maximises the value of J.

The update of the parameters at each step is executed following the equation:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}$$

where α, also called the learning rate, is a parameter between 0 and 1 that controls the importance that each new update has, relative to the one of the parameters previously computed, in composing the new set θ_{t+1}.

In order to perform an update, it is needed an expression for the policy gradient that can be analytically computed. The process to find it involves 2 steps: deriving the analytical gradient of policy performance, which will have the form of an expected value, and forming a sample to estimate that expected value, which can be done by letting the agent interact with the environment a finite number of times.

Since the derivation of the formula for the policy gradient is particularly complex, it has been preferred to exclude it from this text. Here just the final results of the calculation are reported. More detailed information about the processes adopted to obtain this formula can be found at (15)

The analytical form of policy gradient is therefore reported as:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$$

where |D| represents the cardinality of the sample of trajectories.

The basic rationale behind this formula is that a gradient is first evaluated for each trajectory in the sample, obtained by the agent acting in the environment accordingly to the policy π_θ, and then an average is computed between all the gradients obtained in this way. This average will represent the expected value of the policy gradient and will be then used to update the current parameters θ and obtain a new policy $\pi_{\theta+1}$, which in turn will be utilized to generate a new sample of trajectories and restart the cycle.

This process can be further expanded in two ways, that will be useful later, when describing the models.

The first one is a direct expansion of the analytical form of the policy gradient

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right]$$

This is the exact same equation we used to compute the gradient in the previous example, with the difference that the average over the various trajectories is now expressed as an expected value instead.

A basic intuition that can be inferred by just looking at the formula, is that it is enhancing the log probability of each action by the total sum of return ever obtained, R(T), but this does not make any sense, since the agent should really just reinforce action based on their *consequences*.

A much more reasonable way of evaluating the gradient would then be to enhace the probability of each action just by the rewards obtained after executing it. It can be shown (15) that it actually exist a form of policy gradient that takes into account this rationale and that it can be expressed as:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})\right]$$

This is called the “Reward-to-go policy gradient” because, in this version, an action is only reinforced based on the rewards obtained after that action was taken, and this specific sum of rewards takes the name of “Reward-to-go”.

The second expansion springs from a direct consequence of the EGLP lemma, also described in (15). It can be shown that, for any function *b* that only depends on the state of the environment, this expression holds:

$$\underset{a_t \sim \pi_\theta}{\mathrm{E}} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t) \right] = 0$$

This means that is possible to add or subtract any number of such terms to the policy gradient formula, without changing its expectation. The formula can be therefore rewritten as:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t) \right) \right]$$

Such functions $b(s_t)$ are referred to as *baselines* and are useful to reduce the variance of the policy gradient and, therefore, to make the training process more stable.

A common example of baseline, also useful to explain the intuition behind them, is the On-Policy Value Function $V^\pi(s_t)$. Other than empirically reducing the variance of the gradient, the baseline is, intuitively, also telling the agent to reinforce just the actions that lead to better results than the average and to penalize the ones that lead to worse results.

Other common choices for the baseline are the Q function and the advantage function. The latter will be in fact utilized in the first model described in this work.

3.2.3 Advantage Actor-Critic

The Advantage Actor-Critic (A2C) is one of the four algorithms that will be applied to the main problem of this work: Efficient Portfolio Allocation. It is important to point out that all the following methods are capable of handling continuous action spaces, as the one of EPA is of such a kind. A2C is an On-Policy algorithm and pertains to the class of the Policy Optimization algorithms.

The basic rationale behind A2C is to solve the variance problems proper of the basic derivation of the basic policy gradient by making good use of the baselines.

In order to explain the idea behind this algorithm, a little step back is needed, returning to the formula of policy gradient.

From the previous paragraph, we can derive a general formulation of this expression as:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Phi_t \right]$$

where Φ_t represents the weight for each action.

Φ_t can be substituted by:

$$\Phi_t = R(\tau)$$

by

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$$

or by

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$$

It is possible, though, to substitute Φ_t also with two other already known equations: the Q function and the Advantage function. The passage from the Q to the Advantage function is done by just using the Value function as a baseline, obtaining this final equation:

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t))$$
$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

using the Bellman Equations to perform this passage:

$$Q(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})]$$

This is exactly the formula used by the Advantage Actor-Critic to derive the gradient of its policy. Note that in this way just the Value function has to be estimated in order to obtain the Advantage Function.

The A2C takes its name from the fact that, at each step, it updates both a Critic and an Actor.
The Critic is the Value function, usually approximated using a parameterized neural network, and updated using gradient descend on a mean-squared error loss function. The Actor is, instead, the set of policy parameters, that is updated following the direction of the Critic.

Below the pseudocode for the A2C algorithm is reported :

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, ...$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2,$$

typically via some gradient descent algorithm.
9: **end for**

3.2.4 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is another algorithm pertaining to the Policy Optimization class and it is an On-Policy algorithm.

The rationale behind this method is to try to find a way to update the policy as much as possible at each step, without going too far away and causing the collapse of the process. The solution to this problem is reached through some changes to the objective utilized with respect to the one of A2C.

Before starting to describe the algorithm is necessary to notice that exist two different kinds of PPO, Penalty and Clip, but here just the description of the Clip version is provided since it will be the one used in the actual implementation.

The PPO algorithm updates the policy parameters using the following equation:

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L(s,a,\theta_k,\theta)\right]$$

As anticipated the objective is here no more $J(\pi)$ but has been substituted by a function L which depends on the state action-pair, the current parameters θ_k, and on a set of parameters θ that will be the ones that the algorithm wants to find at each step and which will take the place of the current parameters after each update.

The L function is defined as :

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \quad g(\epsilon, A^{\pi_{\theta_k}}(s,a))\right)$$

where the function g is expressed as:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

and "e" is a very small hyperparameter that is, roughly speaking, setting a limit to how far the new policy is allowed to go from the old one.

To get the intuition behind these equations let's consider a single state-action pair in two different cases. If the advantage of the pair is positive the L function simplifies in:

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right) A^{\pi_{\theta_k}}(s, a)$$

which is basically the advantage of the state-action multiplied by a number. That number, according to the equation, will be the minimum factor between 1+e and the increase, in percentage, of the probability of the action a to be executed in state s according to the new policy, $\pi_\theta(a|s)$, with respect to the same probability according to the old policy, $\pi_{\theta k}(a|s)$.

Since at each update the algorithm wants to find the parameters that maximises the expected value of L, it would be tempted to enhance arbitrarily the new probability $\pi_\theta(a|s)$, which could result in the collapse of the process. Thanks to the hyperparameter e though, this does not happen, since passed the treshold 1+e the value of the L function would not be augmented by increasing $\pi_\theta(a|s)$ and therefore it would not make sense for the model to do it.

For the case in which the advantage is negative the reasoning is the same. The L function becomes:

$$L(s, a, \theta_k, \theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon)\right) A^{\pi_{\theta_k}}(s, a)$$

and again the model could be tempted to arbitrarily diminish the new probability of the action, but the extent to which it could do it is bounded by the combination of 1-e and the max operator.

The one described above is the "clipping" part of the algorithm, intended as a way to remove the incentives for the method to dramatically change the policy at each step and ensuring a smooth convergence. For what concerns the estimation of the Advantage function, it is computed using a parameterized neural network as approximator, trained with a gradient descend algorithm on the mean-square error loss function, exactly as in A2C.

Below, the pseudocode for the PPO clipping algorithm is reported.

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, ...$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

3.2.5 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient is the first model covered not pertaining to the Policy Optimization class. It does not pertain either to the Q-learning one. DDPG is instead, a classic example of a mixed algorithm.

DDPG is an Off-Policy algorithm and is specifically designed for problems with continuous action spaces. In some way, this algorithm can be intended as a version of DQN for continuous action spaces and it is precisely from this point that originates its formulation.

Remember that Q-Learning algorithms usually derive the oprimal policy by the Q function, taking actions according to:

$$a(s) = \arg\max_{a} Q_\theta(s, a)$$

Dealing with discrete action spaces, the arg max operator is not a big problem. To evaluate it, it is sufficient to examine all the possible values of the Q function for that state and take the action that produces the greater one. For continuous action spaces, things are a bit different though since to carry out the procedure just described in such kind of states would result infeasible. To get the absolute best action, in fact, one would have to confront an infinite number of them.

DDPG gets around this problem by assuming the Q function to be differentiable by the action argument. This assumption allows to set up a learning rule for a deterministic policy μ(s) that will be used to approximate the max over action as:

$$\max_a Q(s,a) \approx Q(s,\mu(s))$$

Let's expand on this reasoning.

Recalling the Bellman equation for the Optimal Q Value Function:

$$Q^*(s,a) = \underset{s' \sim P}{\mathrm{E}} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right]$$

This can be used as a starting point to learn an approximator for the Q function. Supposing that the approximator is a parametrized neural network Q_Φ(s,a), we can use a mean-squared error function, or in this case a mean-squared Bellman error function, L(Φ,D), to understand how far the approximator is from the desired value. Here D is to be intended as a set of transitions (s,a,r,s',d) where d, either 1 or 0, indicates whether the state s' is terminal.

The equation for the MSBE is:

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d) \sim \mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) \max_{a'} Q_\phi(s',a') \right) \right)^2 \right]$$

The intuition behind this equation is that we could numerically evaluate the values for the Bellman equation, also called targets, from our set D and then use these values as we would use labels in a supervised learning approach to minimize the computed error via gradient descent.

Here though, is where the problem earlier described, in evaluating the max over action, starts to arise. The solution adopted by the DDPG algorithm in this case is quite convoluted and involves learning another extimator, this time for a deterministic policy, that can be used as a substitute of the max over action to evaluate the Bellman equation value. The new formula looks like that:

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d) \sim \mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s')) \right) \right)^2 \right]$$

It is important to point out that, since the targets could not depend on the same set of parameters the algorithm is trying to learn, as it would make the training phase too unstable, a lagged set of parameter Φ_{targ} is instead used for evaluating their value. Φ_{targ} is computed by Polyak averaging

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi.$$

where ρ is a hyperparameter between 0 and 1 but usually close to 1.

Finally, for the policy learning side of the algorithm, the objective is to find a policy that gives the action which maximizes $Q_\Phi(s,a)$. Given our assumptions, a continuous action space and $Q_\Phi(s,a)$ being differentiable for the action parameter, it is possible to solve this problem just by performing gradient ascend to solve:

$$\max_{\theta} \mathop{\mathrm{E}}_{s\sim\mathcal{D}} \left[Q_\phi(s, \mu_\theta(s))\right]$$

Below, the pseudocode of the algorithm is reported:

1: Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
3: **repeat**
4: Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5: Execute a in the environment
6: Observe next state s', reward r, and done signal d to indicate whether s' is terminal
7: Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
8: If s' is terminal, reset environment state.
9: **if** it's time to update **then**
10: **for** however many updates **do**
11: Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12: Compute targets

$$y(r, s', d) = r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

13: Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_\phi(s,a) - y(r,s',d)\right)^2$$

14: Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s))$$

15: Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$$

16: **end for**
17: **end if**
18: **until** convergence

3.2.6 Soft Actor-Critic

Soft Actor-Critic is the last algorithm that will be applied to solve the Efficient Portfolio Allocation problem in this work. It is another example of mixed method derived for continuous action spaces, and it is by far the most mathematically complex of the bunch. Due to this intrinsic complexity, the objective of this paragraph will not be to provide a detailed description of all the processes and theorems that are at the base of the algorithm but, instead, to depict the intuitions behind them in order to be able to later interpret the results obtained with it in an informed way. A complete and mathematically precise description of Soft Actor-Critic can be found at (16).

The first notable difference between SAC and all the other algorithms previously described, is that the former takes advantage of a slightly different approach to Reinforcement Learning, the Entropy Regularized RL. Entropy Regularized LR is a reinforcement learning setting that makes use of the entropy, a quantity that describes the level of randomness of a random variable. A six-faced dice modified to output always 6, for example, will have a really low entropy, while a fair coin will have a high one.

The entropy H of a random variable P is defined as:

$$H(P) = \underset{x \sim P}{\mathrm{E}}\left[-\log P(x)\right]$$

In Entropy Regularized RL the agent will receive an additional reward at each timestep t, proportional to the entropy of the policy at t. The optimal policy π^*, taking the example of the infinite-time horizon sum of rewards, can be therfore rewritten as:

$$\pi^* = \arg\max_{\pi} \underset{\tau \sim \pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty} \gamma^t \bigg(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \bigg)\right]$$

where α is a tradeoff-coefficient.

The definition of the Value Functions and the relative Bellman Equations are also slightly changed to take into account entropy. Here we report, as the algorithm will make use of these, just the On-Policy Value Function,

$$V^{\pi}(s) = \underset{\tau \sim \pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty} \gamma^t \bigg(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \bigg) \bigg| s_0 = s\right]$$

the On-Policy Action-Value function,

$$Q^{\pi}(s,a) = \underset{\tau \sim \pi}{\mathrm{E}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t H\left(\pi(\cdot|s_t)\right) \middle| s_0 = s, a_0 = a \right]$$

the connection between them, defined as

$$V^{\pi}(s) = \underset{a \sim \pi}{\mathrm{E}} \left[Q^{\pi}(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right)$$

and the Bellman equation for the On-Policy Action-Value function

$$\begin{aligned} Q^{\pi}(s,a) &= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s')\right)\right)\right] \\ &= \underset{s' \sim P}{\mathrm{E}} \left[R(s,a,s') + \gamma V^{\pi}(s')\right]. \end{aligned}$$

After defining the setting, it is time to explain the proper functioning of the algorithm, which is, in many ways, similar to the one of DDPG. Both methods aim to approximate the Optimal Q function Q*(s,a) and use it to decide their actions, both make use of MBSE as a loss function and of gradient descend as the training algorithm and both solve the problem of evaluating max over action approximating a policy that maximizes the value of a Value Function.

The main difference between the two, entropy regularization aside, is that SAC concurrently learns two Q function approximators and then decides which of the two to use in each case, usually taking the one outputting the smaller value, in order to increase the stability of the training phase.

Starting from the definition of Q function and entropy previously given, we can rewrite the formula of the Q function expanding the entropy in this way:

$$\begin{aligned} Q^{\pi}(s,a) &= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s')\right)\right)\right] \\ &= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') - \alpha \log \pi(a'|s')\right)\right] \end{aligned}$$

We can then give the definition of the loss function adopted by the SAC algorithm:

$$L(\phi_i, \mathcal{D}) = \underset{(s,a,r,s',d) \sim \mathcal{D}}{\mathrm{E}} \left[\left(Q_{\phi_i}(s,a) - y(r,s',d) \right)^2 \right]$$

where the targets y(r,s’,d) are defined as:

$$y(r, s', d) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \qquad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

Two important things to point out from this equation are the presence of the min operator, utilized to take the min value between the two Q function approximators, and the slightly new definition of the next action, which is, differently from DDPG, specifically intended to be sampled from the current policy.

For the policy learning aspect of this algorithm, the objective is to learn a policy that maximize both the expected future rewards and the expected future entropy, but this is the exact definition of the On-Policy Value Function, expanded as:

$$\begin{aligned} V^\pi(s) &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^\pi(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right) \\ &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^\pi(s,a) - \alpha \log \pi(a|s)\right] \end{aligned}$$

The research of the policy makes use of what is called the reparametrization trick, a process consisting in computing a deterministic policy function for the actions, which will depend on the state, the policy parameters, and independent noise ξ, defined as:

$$\tilde{a}_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\right), \qquad \xi \sim \mathcal{N}(0, I).$$

in order to get around a pain point of the process: the fact that the distribution of the actions depends on the policy parameters, and transofrm the expectation over the latters in an expectation over the noise.

$$\underset{a\sim\pi_\theta}{\mathrm{E}}\left[Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s)\right] = \underset{\xi\sim\mathcal{N}}{\mathrm{E}}\left[Q^{\pi_\theta}(s, \tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s)\right]$$

Finally, putting all together, gradient ascent is used to maximize the following expectation and find the policy that maximizes the Value Function:

$$\max_\theta \underset{\substack{s\sim\mathcal{D} \\ \xi\sim\mathcal{N}}}{\mathrm{E}}\left[\min_{j=1,2} Q_{\phi_j}(s, \tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s)\right]$$

Below, the pseudocode of the algorithm is provided:

1: Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
3: **repeat**
4: Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
5: Execute a in the environment
6: Observe next state s', reward r, and done signal d to indicate whether s' is terminal
7: Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
8: If s' is terminal, reset environment state.
9: **if** it's time to update **then**
10: **for** j in range(however many updates) **do**
11: Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12: Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

13: Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2 \qquad \text{for } i = 1, 2$$

14: Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\right),$$

where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
15: Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1, 2$$

16: **end for**
17: **end if**
18: **until** convergence

SECTION 4: EFFICIENT PORTFOLIO ALLOCATION

Efficient Portfolio Allocation is a fundamental problem of finance. Its basic formulation comprehends a potrfolio of assets and an investor, who wants to maximize the return of that portfolio, assuming no other securities available on the markets other than those contained in the portfolio itself. The investor, each day, can rebalance at will the proportion of each of the assets he or she owns given an initial budget constraint. The challenge is to find the best possible composition, or series of them, that will yield the maximum portfolio return over time.

Traditionally, EPA was approached with what is called the Modern Portfolio Theory (MPT), one of the cornerstone theories of modern finance. MPT defines an "Efficient Frontier", a line in which lie all the portfolios yielding maximum return for different levels of variance or standard deviation (fig 4.1). It is from here that springs the classic formulation of an efficient portfolio:

$$\min_{w_i} \operatorname{var}(r_p)$$
$$\text{subject to } \mathbb{E}(r_p) = \mu^*, \ \sum w_i = 1,$$

where r_p is the return on the portfolio, w_i are the portfolio weights, and μ^* is the target return. The formula is describing a portfolio that achieves the target return while being subject to the minimum variance among all the others achieving the same return.

Even if MPT provided for decades a fundamental tool for portfolio management strategies and it is still used today as a valuable benchmark, eventually did not perform well in the practice, the cause probably being the fact that it only takes into account variance and returns, leaving aside a great number of important financial indicators like MACD and RSI.

With the arising of machine learning and artificial intelligence in recent times, came also the proof that predicting the complex behavior of the market was a possible thing to do, at least to a certain extent (17) and, as a result, solutions developed for EPA became always more centered around data.

Researches for stock market behavior prediction, adopting machine learning approaches, flourished over the last decades, with Random Forest and Support Vector Machine being commonly utilized methods, as well as Artificial Neural Networks. None of these methods though, took advantage of the time series structure that is proper of financial data and, therefore, the real major breakthrough happened in 2017, with the introduction of Deep Reinforcement Learning (18).

The possibility of approximating the policy by using models specifically designed for time series, like RNN or LSTM did really give a boost to the previous iterations of RL on the problem and it is, therefore, from this point that this work wants to start.

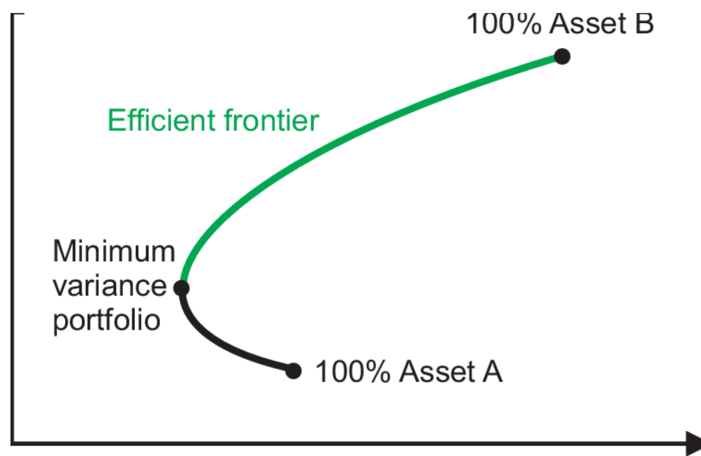


Figure 4.1: Graph Efficient Frontier. Assume return on the Y axis and variance in the X one

4.1

Problem Formulation

Before applying any RL algorithm on EPA, the problem itself needs to be encoded in such a way to be solvable by an RL agent. This means presenting it in terms of an environment, an action space, and a reward function.

First of all, the environment will resemble a financial market. This means that, at each timestep, the agent will be provided with an environment state containing information about all the assets in the portfolio, including previously mentioned high, low, open, and close prices, as well as a list of financial indicators (fig 4.2).

The envivonment state will also include a covariance matrix of the assets, a useful tool by which the agent can evaluate the risk, or standard deviation, proper of the current portfolio.

For what concerns the action space, its definition is related to the one of the weight vector. A weight vector w_t is defined as a vector that contains a weight between 0 and 1 for all the assets in the portfolio. The sum of all the entrances of w_t has to be equal to one, as the weights represent the proportion of each asset contained in the portfolio at that specific time.

The action space of the agent will be, therefore, defined as a set W containing all the possible weight vectors. As previously anticipated, the action space for this problem is, by definition, continuous.

Finally, the reward function of the agent will be represented by the return of the portfolio, but in order to precisely describe it, some previous specifications are needed. In this case, a formulation similar to the one applied in (19) is adopted.

First of all, it is defined the price vector V_t, as a vector containing all the closing price of the assets in the portfolio at the end of a given timestep t. For computations it is then adopted a normalization of the price vector, defined as the element wise division between V_t and V_{t-1}:

$$\mathbf{Y}_t := \mathbf{V}_t \oslash \mathbf{V}_{t-1} = \left[\frac{\mathbf{V}_{1,t}}{\mathbf{V}_{1,t-1}}, \frac{\mathbf{V}_{2,t}}{\mathbf{V}_{2,t-1}}, \ldots, \frac{\mathbf{V}_{m,t}}{\mathbf{V}_{m,t-1}} \right]$$

This form results useful because prices expressed in this way contain information about the daily return of each asset and can therefore be used without further computations to evaluate the total portfolio return.

Then, assuming that when an asset is traded a transaction fee needs to be paid, a shrinking factor μ_t (written in italic to distinguish it from the deterministic policy) is introduced. The value of the portfolio is therefore reduced at each iteration by the μ factor, having a value between 0 and 1, to take into account these fees.

Finally, given the previous definitions, the value of a portfolio at the end of each timestep can be computed as:

$$P_t = \mu_t P_{t-1} \cdot \mathbf{Y}_t \cdot \mathbf{w}_{t-1}$$

and the consequent rate of return r_t over the same timestep as:

$$r_t := \frac{P_t}{P_{t-1}} - 1 = \frac{\mu_t \cdot P'_t}{P_{t-1}} - 1 = \mu_t \mathbf{Y}_t \cdot \mathbf{w}_{t-1} - 1$$

The reward function $r_t(s,a,s')$ will be defined as this rate of return, while $J(\pi)$ will be, in this case, the total cumulative return.

close_30_sma	Simple Moving Average over the last 30 periods
close_60_sma	Simple Moving Average over the last 60 periods
macd	Moving Average Convergence/Divergence
rsi_30	Relative Strenght Index over the last 30 periods
cci_30	Commodity Channel Index over the last 30 periods
dx_30	Directional Movement Index over the last 30 periods
boll_ub	Bollinger Upper Band in the current period
boll_lb	Bollinger Lower Band in the current period

Figure 4.2: A legend of the financial indicators adopted as additional information in the environment state

4.2

An Introduction To the Practice

As the last step before starting describing the applications of the algorithms and the consequent results, it is due to provide some clarifications to explain the rationale behind the execution.

First of all, the models described in section 3.2 have been chosen following the indications of Finrl Library, the Python library utilized to practically train the models. A selection of this sort has two reasons: the first is that all of these models support, and in some cases are specifically thought, for continuous action spaces, and the second is that it provides a good variety across the two main classes of model-free algorithms, Policy Optimization and Q-Learning, and therefore a good trade-off between complexity and reliability.

To obtain a broader intuition of what are, at the moment, the capabilities of RL on financial markets, it has been also decided to apply all the models described to three different portfolios, one composed of stocks, one composed of bonds, and one composed of cryptocurrencies. A more precise description of the assets contained in each portfolio will be provided later in the text.

The rationale behind this diversification is to understand how the models approach securities pertaining to different markets and having different risks, as a bond is probably subject to a lower standard deviation of returns than a cryptocurrency.
In each of these cases, the composition of the portfolio was selected in order to obtain an investiment that followed as much as possible the general pace of its market. This was done for two reasons. At first, it is logical to assume that the average investor would want to diversify its portfolio to reduce the idiosyncratic risk associated with it, and, as a consequence, the more a portfolio is diversified the more it follows the market. Secondly, the idea behind the diversification is to discover what algorithms work best in each market and, therefore, the more a portfolio can track it the better fulfils its purpose.

The hope is, in fact, to understand if the different formulations of these methods will make emerge significant discrepancies in performances between algorithms across different markets. The best performing ones could be then utilized to obtain more refined models through grid search and hyperparameter tuning.

In practice 12 models, 4 algorithms times 3 portfolios, will be trained, and a first step of selection will be carried out, ranking the algorithms mostly on cumulative return, Sharpe ratio and standard deviation. Then, once having selected the best for each market, a step of hyperparameter tuning will be performed to try to optimize them. Finally, a web app will be developed in order to demonstrate how a practical application of the models is easily obtainable.

SECTION 5: IMPLEMENTATION

It is now time to describe the procedures and the results of this work. In each of the following three subsections, one of the previously described markets will be considered and the best algorithm for each one will be selected through a backtest analysis. Then an hyperparameter tuning step will be performed to optimize the best model.

5.1

The Stock Market

For the asset with which to construct the first portfolio, it has been chosen the most widely possessed financial security of it all: stocks. That is because stocks as a whole can be considered as a middle-risk asset, when compared to bonds, which are usually less subject to high variance, or to cryptocurrencies, which are instead a lot more volatile, as the recent turbulences of bitcoin price demonstrated (the currency lost over 35% of its value between 2/05/21 and 2/06/21).

The stock market seemed the perfect one, therefore, to set a kind of baseline for comparisons of methods and results and to give an initial impression on the capabilities of RL algorithm in finance. The composition of the portfolio was then chosen specifically to track the behavior of the market in general, in order to reflect the average risk faced by an investor buying in it. The choice fell on a combination containing the same securities as the Dow Jones 30 index, as a portfolio composed in this way seemed to be subject to a moderate, but not irrelevant amount of risk, while still representing a good indicator for the general pace of the market itself.

5.1.1 Environment Set Up

The first step to train the models was then to set up the environment. This was done exploiting Yahoo Finance APIs to gather a dataset containing, for each of the 30 stocks in the index, information about high, low, open, and closing prices together with the volume traded, in each trading day from 31/12/09 to 31/12/20.

Then, the function preprocess_data of Finrl was utilized to add to the dataset the technical indicators previously described (fig 4.2), together with the covariance matrix. With the complete dataset, rows were then grouped by date, in order to factorize the index with respect to this variable. In this way,

the environment state of each day was easier to isolate in the dataset, as for each state a different index was provided.

The dataset so obtained was then splitted in two subsets, train and test, containing data respectively from 31/12/09 to 1/1/19 and from then on, in order to have a set to use later to evaluate the goodness of the models. It is important to notice that the sets were taken in temporal order, specifically to maintain the information associated with the time series structure of financial data, as the models did learn on the older set and experiment on the more recent.

In addition, some hyperparameters needed to be set up. The basic Finrl configuration for the environment was left untouched for this round of training: hmax, or the maximum number of shares the agent was allowed to sell each episode, was equal to 100; the initial amount of money at the agent disposal was 1 million of US dollars; the transaction cost (μ) was 0.001 of the amount of the transaction; the state space, stock dimension, and action space were all setted up to be equal to the amount of stocks in the porfolio, so 30; the technical indicator list was mainly a functional hyperparameter to tell the algorithm what columns were indicators; and finally, the reward scaling factor for the agent was set at 1^{e-4}.

With these sets and hyperparameters, two environments, both instances of the StockPortfolioEnv of Finrl, were set up: a train environment (fig 5.1), containing states from the train set, and a test one containing states from the test/trade one.

	date	open	high	low	close	volume	tic	day	macd	boll_ub	boll_lb	rsi_30	cci_30	dx_30	close_30_sma	close_60_sma	
0	2009-12-31	7.61179	7.61964	7.52000	6.47169	352410800	AAPL	3	1.94541	138.30612	131.40528	61.57199	53.21176	25.82168	133.09666	124.24616	[[4.56627754e-04 5.35930060
0	2009-12-31	40.90000	41.08000	40.49000	34.06930	4030500	AXP	3	1.92948	138.25145	131.86388	62.32735	66.78058	28.96334	133.36504	124.68939	[[4.56627754e-04 5.35930060
0	2009-12-31	55.00000	55.22000	54.05000	42.18011	2189400	BA	3	1.98064	138.42098	132.09564	63.40799	75.89187	32.68805	133.63772	125.14161	[[4.56627754e-04 5.35930060
0	2009-12-31	57.60000	57.96000	56.99000	41.64734	3859700	CAT	3	1.88444	138.44098	132.10733	61.00424	67.04092	32.68805	133.86638	125.56192	[[4.56627754e-04 5.35930060
0	2009-12-31	24.10000	24.17000	23.94000	17.77566	25208100	CSCO	3	1.81497	137.66940	132.54361	61.36896	76.12280	33.33361	134.06625	125.95893	[[4.56627754e-04 5.35930060
0	2009-12-31	77.72000	77.78000	76.93000	48.88803	4246600	CVX	3	1.62122	137.54752	132.49564	58.89942	35.20034	20.44563	134.23404	126.30478	[[4.56627754e-04 5.35930060
0	2009-12-31	40.32123	40.36393	39.32494	28.42827	2960718	DD	3	1.36950	137.48744	132.31548	57.26316	-25.26811	10.46690	134.38004	126.61596	[[4.56627754e-04 5.35930060
0	2009-12-31	32.27000	32.75000	32.22000	28.09071	19651700	DIS	3	0.91659	137.95688	131.42538	52.79045	-113.33267	7.58407	134.41231	126.87131	[[4.56627754e-04 5.35930060
0	2009-12-31	167.28999	170.13000	166.92999	142.47206	6401800	GS	3	0.54709	138.29791	130.77106	52.71576	-180.18098	17.31716	134.41964	127.11615	[[4.56627754e-04 5.35930060
0	2009-12-31	29.09000	29.37000	28.89000	22.16832	7437100	HD	3	0.29910	138.45937	130.43358	53.48765	-149.81895	14.58267	134.44193	127.37767	[[4.56627754e-04 5.35930060
0	2009-12-31	132.41000	132.85001	130.75000	90.43541	4223400	IBM	3	0.37168	138.38842	130.40773	57.54694	-32.96777	5.23169	134.52906	127.72263	[[4.56627754e-04 5.35930060
0	2009-12-31	20.60000	20.72000	20.40000	14.43387	26429200	INTC	3	0.24037	138.09203	130.22099	54.21598	-80.97033	5.03053	134.53375	128.02960	[[4.56627754e-04 5.35930060
0	2009-12-31	65.12000	65.12000	64.33000	45.81926	6962300	JNJ	3	0.03367	138.02732	129.78319	52.48856	-161.04242	9.41061	134.45953	128.32856	[[4.56627754e-04 5.35930060
0	2009-12-31	41.62000	42.13000	41.45000	31.16688	20143100	JPM	3	-0.08373	137.83060	129.52318	53.17452	-115.29577	6.99135	134.32841	128.65907	[[4.56627754e-04 5.35930060
0	2009-12-31	28.79000	28.87500	28.46500	20.08414	10848800	KO	3	-0.05893	137.71876	129.44229	54.91063	-83.46194	2.20986	134.11192	128.98492	[[4.56627754e-04 5.35930060
0	2009-12-31	62.97000	63.07000	62.39000	44.69355	4495300	MCD	3	0.05875	137.43523	129.51197	56.32201	0.09475	7.34211	134.04093	129.29777	[[4.56627754e-04 5.35930060
0	2009-12-31	83.79000	84.08000	82.54000	60.58413	2049800	MMM	3	0.22330	137.52265	129.50728	57.35516	27.79249	7.34211	134.01101	129.62967	[[4.56627754e-04 5.35930060
0	2009-12-31	36.80000	37.07000	36.50000	24.77870	7064000	MRK	3	0.35388	137.66922	129.55871	57.41527	24.01070	4.29954	134.02303	129.97221	[[4.56627754e-04 5.35930060
0	2009-12-31	30.98000	30.99000	30.48000	23.63019	31929700	MSFT	3	0.51956	137.89480	129.53906	58.38627	69.91184	10.54341	134.10077	130.32447	[[4.56627754e-04 5.35930060
0	2009-12-31	16.55250	16.65500	16.51250	14.34485	6347600	NKE	3	0.56340	137.99542	129.58540	56.79537	56.42995	13.08096	134.17088	130.67151	[[4.56627754e-04 5.35930060
0	2009-12-31	17.44782	17.53321	17.25806	11.23810	27063136	PFE	3	0.58923	137.94973	129.59061	56.75335	12.16863	5.00298	134.15357	131.02162	[[4.56627754e-04 5.35930060
0	2009-12-31	61.52000	61.53000	60.56000	42.65570	5942200	PG	3	0.54799	137.73332	129.62396	55.64399	26.60677	9.20348	134.06146	131.35095	[[4.56627754e-04 5.35930060
0	2009-12-31	44.47451	44.53115	43.62492	33.18238	4200680	RTX	3	0.47715	137.24640	129.78000	54.98833	-17.39353	2.43657	133.99165	131.66713	[[4.56627754e-04 5.35930060
0	2009-12-31	50.19000	50.55000	49.84000	37.75843	2320000	TRV	3	0.24983	136.99763	129.63188	51.73878	-79.51919	6.18810	133.84879	131.93120	[[4.56627754e-04 5.35930060
0	2009-12-31	30.95000	31.15000	30.45000	25.70593	5354200	UNH	3	0.28440	136.73925	129.72921	55.28010	13.94886	5.06308	133.83559	132.19321	[[4.56627754e-04 5.35930060

Figure 5.1: The first training environment state, obtained after the initial preprocessing. The last column contains the covariance matrix.

5.1.2 Training and Backtest

After setting the environments, one model was trained for each of the four algorithms previously described. The implementations of the algorithms were the ones of the StableBaseline3 library, incorporated in the classes of DLR agent and StockPorfolioEnv proper of the Finrl library.

It is important to point out that the presence of “ent_coef” among the hyperparameters of each model, apart from DDPG, indicates that the orther three utilize the Entropy Regularized framework previously introduced in section 3.2.6, as it will result useful to know it in the hyperparameters tuning section. For this first round of training though, the hyperparameters of the models were left as default.

After the training step, the models were tested by making predictions on the test environment and the results were fed into the timeseries.perf_stats function of the Pyfolio library, a library providing tools for financial strategies evaluation, in order to extrapolate useful information about performances. The output of the perf_stats function for each model is summarized in (fig 5.2) while a description of the performance indicators computed by it is provided in (fig 5.3).

	Market_Stats	A2C	DDPG	PPO	SAC
0	Annual return	0.14445	0.11594	0.15561	0.11248
1	Cumulative returns	0.29511	0.23399	0.31944	0.22667
2	Annual volatility	0.27647	0.28576	0.28027	0.28383
3	Sharpe ratio	0.62676	0.52715	0.65682	0.51797
4	Calmar ratio	0.40680	0.31509	0.44029	0.31256
5	Stability	0.13095	0.00690	0.08008	0.00158
6	Max drawdown	-0.35508	-0.36797	-0.35344	-0.35988
7	Omega ratio	1.14697	1.12199	1.15392	1.11726
8	Sortino ratio	0.88279	0.74425	0.92658	0.72933
9	Skew	-0.26930	-0.19847	-0.33530	-0.28011
10	Kurtosis	13.27558	12.36902	12.66583	11.66863
11	Tail ratio	0.84510	0.84584	0.88308	0.88913
12	Daily value at risk	-0.03414	-0.03540	-0.03458	-0.03518

Figure 5.2: A summary of the performance indicators provided by the Pyfolio perf_stats function

Then, a comparison between the models was performed, in order to select the one which achieved the better results. The choice, despite the many indicators, was carried out mainly according to the cumulative return, the annual volatility, and the Sharpe ratio, as all the other indicators were either slight modifications of the Sharpe, as the Omega and Sortino ratio, other risk measures, or measures more useful to understand the general shape of the return distribution, as the Kurtosis and the skew. These indicators were, therefore, considered less informatives, since the main objective of the analysis was to understand which of the models was achieving the highest overall return.

From the extracted measures, it clearly emerged that the PPO algorithm was outperforming all the others, achieving not only 2.5% more points of cumulative return off of the second, A2C, but also having an higher Sharpe ratio despite higher annual volatility.

Indicator	Definition
Annual Return	Average Annual return.
Cumulative Returns	Total cumulative returns, expressed as a % of the initial value of the portfolio.
Annual Volatility	Average standard deviation of returns.
Sharpe Ratio	Amount of excess return in % over the risk-free rates per unit of risk (in standard deviation).
Calmar Ratio	Annual rate of return/ Max Drawdown.
Stability	Indicator of discrepancies between the forcasted distribution of returns and the realized one.
Max Drawdown	Largest drop in % between a peak (return in the highest point before a decrease) and a valley (return in the lowest point before an increase).
Omega Ratio	Risk-Return performance metric taking into account all four moments of the return distribution (Sharpe ratio only considers the first two)
Sortino Ratio	Modified Sharpe in which downside deviation, a measure of negative volatility, replace standard dev.
Skew	The third moment of the returns distribution.
Kurtosis	The fourth moment of the returns distribution.
Tail Ratio	Ratio between the 95th and 5th percentile of the distribution of daily returns.
Daily Value at Risk	How much you would lose, in %, at max for taking the position of this portfolio one more day (computed on final portfolio).

Figure 5.3: A legend of all performance indicators evaluated by the Pyfolio perf_stats

Furthermore, to back up the choice, PPO achieved also the lowest max drawdown and the highest Sortino and Calmar ratios.

Once chosen PPO as the best model, to evaluate its performances in a more contextual way, a backtest step was performed. This meant to compare the returns of the trading strategy under analysis with the ones achieved by just buying the same portfolio at the start of the considered period and selling it at the end. The baseline for comparison was, therefore, the cumulative returns achieved by the actual Dow Jones 30 index over the trade set period: 1/1/19-31/12/20. The results

obtained are summarized in the following figures, the plots being constructed with the function create_full_tear_sheet of the Pyfolio library.

The results were good, with the best model outclassing the Dow Jones 30 index by almost 5% on cumulative returns, as the index grew by "only" 27%, and achieving an overall equal standard deviation: 28%

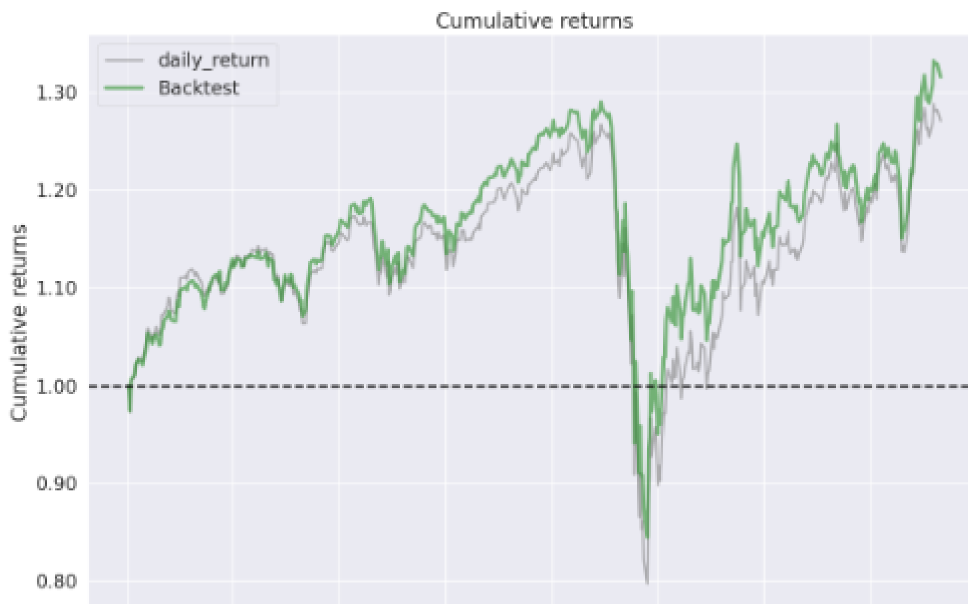


Figure 5.4: Cumulative returns chart o the PPO model (green) vs Dow Jones 30 Index (grey)

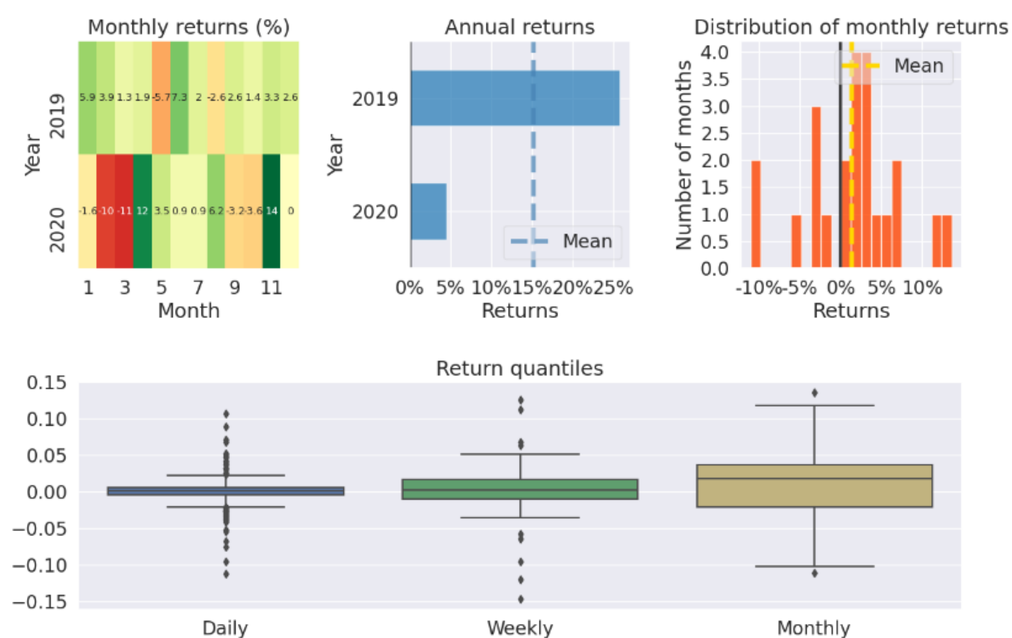


Figure 5.5: Charts describing varius aspects of the PPO returns distribution

5.1.3 Hyperparameter Tuning

The next step after the model initial evaluation was the hyperparameter tuning. The rationale behind this step is to tweak the parameters on which the model depends in order to find the combination of them that yields the best result. In the case of PPO, four parameters were given: the number of steps in between each policy update, n_steps, the entropy trade-off coefficient α introduced in 3.2.6, ent_coef, the learning rate for the gradient ascent algorithm, learning_rate, and, finally, the size of the minibatch of trajectories sampled from D at each update step, bach_size.

To carry out the procedure, a grid search algorithm was set up. The idea was to specify several combinations of parameters, train all the models resulting from those combinations, and evaluate which one was the best. The "grid" to search in was constructed by proposing each time 2/3 variation of a parameter. N_steps, for example, was always either 1024, 2048, or 3072 while ent_coef could only be 0.005 or 0.001, obtaining a total of 24 potential candidates.

The evaluation of the models, again carried out using mainly cumulative return and Sharpe ratio, selected as the best the one having 1024 as n_steps, 0.005 as ent_coef, 0.0001 as learning_rate, and 256 as batch size.

The backtest step was also carried out as previously described. The results obtained are summarized in the figures below. The selected model outclassed the Dow Jones 30 buy and hold strategy by a significant 13% points of cumulative returns, while also achieving lower annual volatility. In addition, it outperformed the previously best model by almost 10% points of cumulative returns, striking a very good improvement and a great result overall.

Annual return	0.19558
Cumulative returns	0.40830
Annual volatility	0.27742
Sharpe ratio	0.78308
Calmar ratio	0.57696
Stability	0.36837
Max drawdown	-0.33899
Omega ratio	1.18402
Sortino ratio	1.11701
Skew	-0.19152
Kurtosis	12.61488
Tail ratio	0.85647
Daily value at risk	-0.03409

Figure 5.6: Grid Search PPO model perfomance indicators

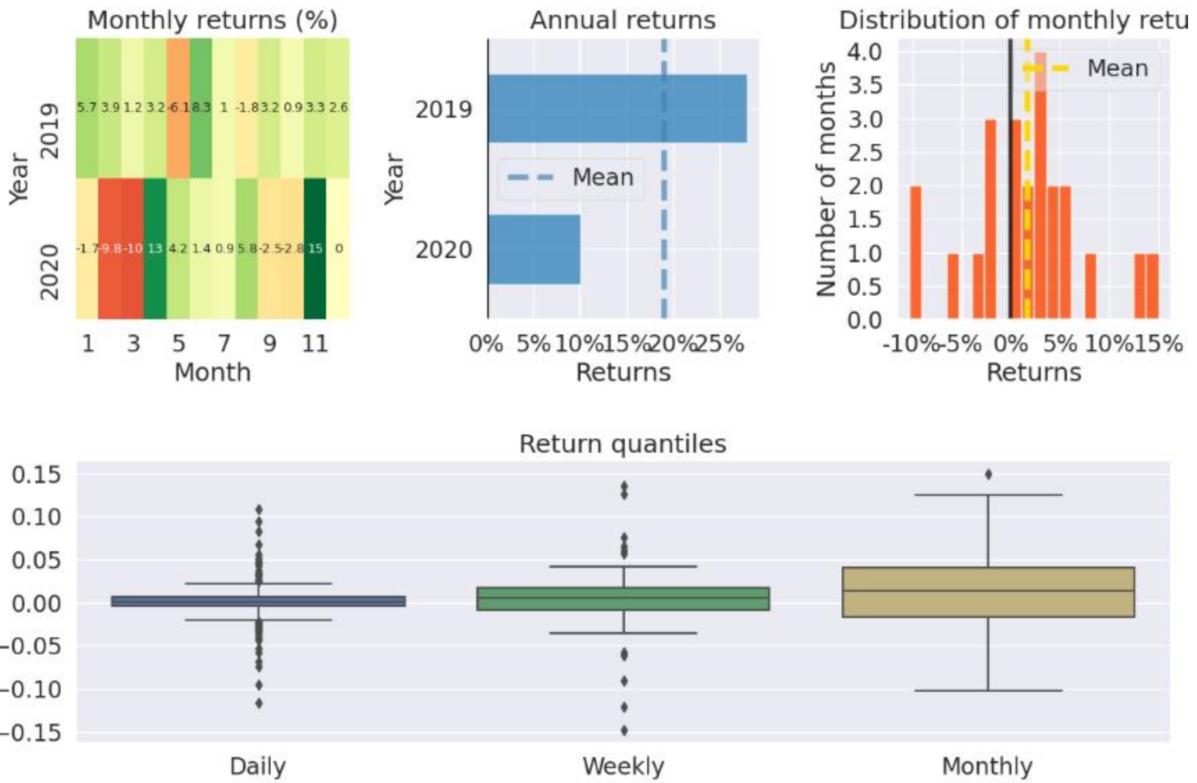


Figure 5.7: Returns distribution of PPO Grid Searh model

Figure 5.8: Cumulative returns chart of PPOGrid Search Mode (green) l vs Dow Jones 30 Index (grey)

5.2

The Bond Market

The asset chosen to build the second portfolio was bond. Bonds are, usually, the safest macro category of assets one can invest in. With a lower risk rate though, comes also a lower possibility of gains and it is exactly for this reason that bonds constituted a valid alternative field to train the models on. The idea was to compare the new results with the ones obtained in a higher risk higher rewards environment to understand, on one hand, if the inner characteristics of a different financial market could enhance or disrupt the performances of the models on a common basis and, on the other, if the change of the environment could make emerge different methods as best performing ones.

Before diving into the results it is necessary to precise that the portfolio was not actually composed of bonds. Data pertaining to a portfolio of bonds are difficult to obtain on a consistent basis, given the differences in maturities proper of this category of assets. One solution could have been to only include bonds with long-term maturities, to have a long enough period over which to train and test the models, but at the cost of diversification.

The adopted portfolio was instead composed of bond indexes, all pertaining to American firms, to restrict the analysis to the USA market, as the stock one was, and containing bonds of all the possible maturities, short medium or long, in order to represent the general structure of the market.

Indexes solve the temporal problem as they do not have a maturity themselves, they just represent an aggregate of bonds with similar characteristics. One index could for example be composed of all the short-term bonds of high-tech firms traded on Nasdaq, or by all the investiment grade bonds (a measure of risk associated with corporate bonds) of a given maturity. Indexes do not need to have a maturity since they represent the total value of the slice of the bond market they consider and, as time passes, some bonds mature and others get issued, creating a circle that rebalances itself.

The complete list of securities by which the second portfolio was composed can be found in Appendix 1.

5.2.1 Environment Set Up, Training and Backtest

As for the Stock market, the first step to train the models was to set up the environment. This was done in an almost identical way to the previous section, with the only difference of the time periods covered by the train and test set. Some of the selected indexes, in fact, were traded for a lower amount of time than others, the most recent ones going back to just 2014. The start of the training set was, therefore, the day in which the youngest index was born, 16/12/2014, and from there a 66%-33% split was carried out to obtain the two partitions. The train, as a result, covered from 16/12/2014 to 4/12/2018, while the test went from there to 31/03/2021.

Again, four models were trained, one for each algorithm, and were then tested in the test environment to evaluate their performances. The results obtained were summarized in a table (fig 5.9), in order to carry out a comparison between them.

	Bond_Stats	A2C	DDPG	PPO	SAC
0	Annual return	0.07649	0.07091	0.06907	0.07535
1	Cumulative returns	0.18556	0.17142	0.16678	0.18268
2	Annual volatility	0.07073	0.07357	0.07056	0.08064
3	Sharpe ratio	1.07748	0.96804	0.98196	0.94133
4	Calmar ratio	0.51908	0.44811	0.44684	0.44644
5	Stability	0.89728	0.86468	0.85945	0.85943
6	Max drawdown	-0.14735	-0.15824	-0.15457	-0.16879
7	Omega ratio	1.35388	1.30739	1.31362	1.29624
8	Sortino ratio	1.53100	1.34933	1.34751	1.31552
9	Skew	-0.10749	-0.41586	-0.92236	-0.42320
10	Kurtosis	43.24003	42.32406	41.52911	41.03405
11	Tail ratio	1.01746	1.00936	1.03673	1.03975
12	Daily value at risk	-0.00861	-0.00899	-0.00861	-0.00986

Figure 5.9: A summary of the performance indicators evaluated for the Bond models

The numbers were very interesting, since the new environment and setup made effectively emerge different algorithms as the most efficient ones. From the computed metrics, A2C and SAC were clearly the more suited for the situation, with A2C gaining the edge due to slightly higher returns, 0.3% more, and lower volatility, almost 1% less. PPO, instead, was the worse performing one in this case

Then, the usual backtest step was performed, but this time, as no predefined index containing all the assets in the portfolio already existed, a general American corporate bonds index, the iShares Core U.S. Aggregate Bond ETF, was selected as the baseline.

The selected best model, A2C, outperformed the buy and hold strategy by a significant amount. The cumulative returns of the baseline, in fact, barely reached 14% over the considered time span, while the value of the A2C portfolio grew over 18.5% points over the same period. The annual volatility of the baseline was, instead, lower, but the higher Sharpe ratio of the algorithm portfolio, almost 3% more, well expressed how much that excess volatility was worth it.

Figure 5.10: Cumulative return chart of A2C model (green) vs the iShares Core U.S. Aggregate Bond ETF (grey)

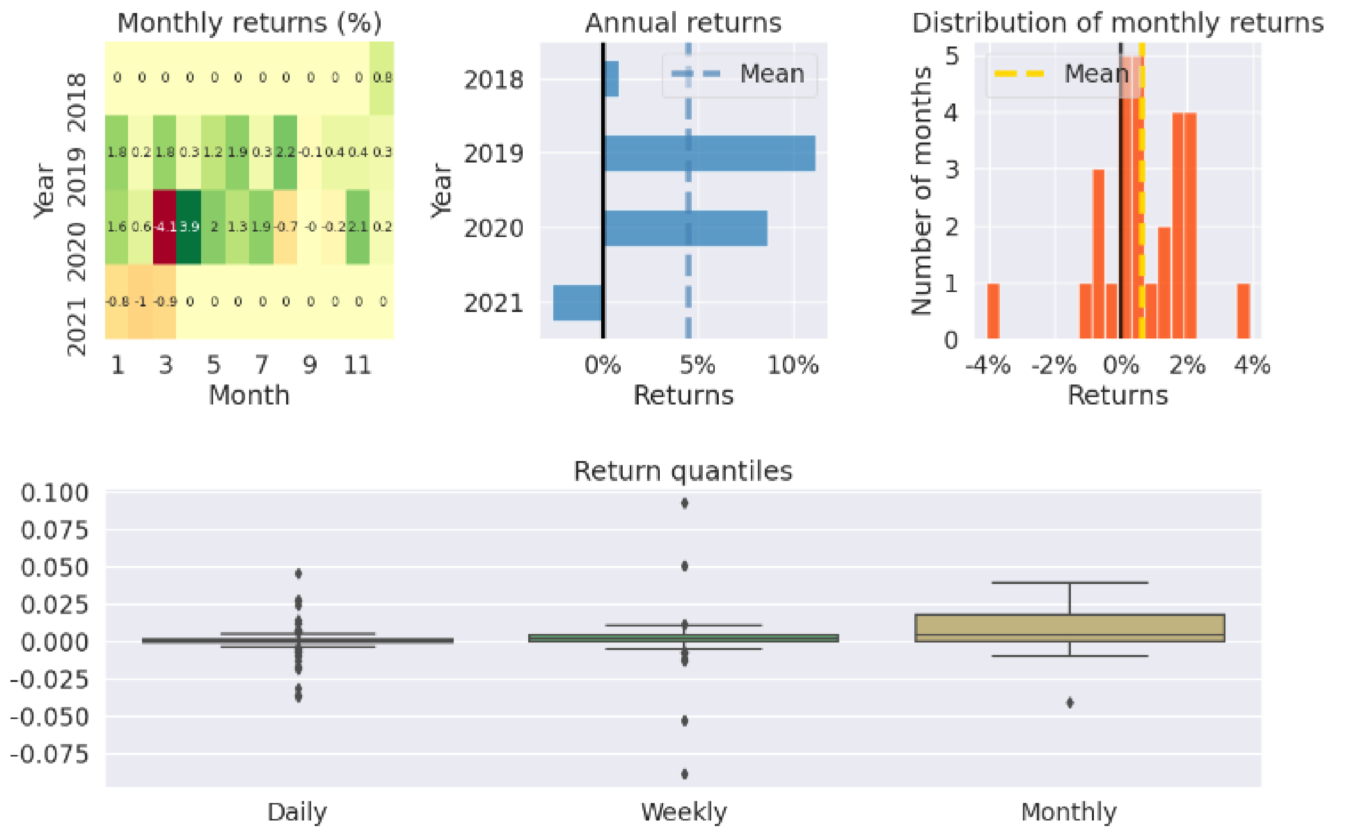


Figure 5.11: A2C model returns distribution plots

5.2.2 Hyperparameter Tuning

Starting from these good results, the hope was to increase the performances by performing an hyperparameter tuning step.

The parameters at A2C disposal were the same as the PPO algorithm, with the only difference being the lack of the bach_size one. The grid of candidate models was therefore composed by specifying four alternatives for n_steps and three for the remaining parameters, ent_coef and learning_rate, obtaining a total of 36 candidates.

Unfortunately, the Grid Search did not produce any meaningful result this time, with the best-trained model barely achieving the same results as the original, with a cumulative return of 17.9%

5.3

The Cryptocurrency Market

Cryptocurrencies were chosen as the asset composing the last portfolio.

Cryptocurrencies are one of the more interesting financial securities of recent times. Their connection with the blockchain technology makes them incredibly volatile, as the whole ecosystem they are part of is relatively new, as well as very diversified.

Cryptocurrencies are, in fact, just a functional mechanism of a bigger machine and can be designed to fulfil multiple roles. Bitcoin, for example, is programmed to behave like a traditional currency and to be utilized mostly for transactions while Ethereum, instead, is more of a functional currency, created to support the development and the execution of smart contracts on the Ethereum platform.

In addition, the general excitement generated in recent years around the blockchain has led to the creation of an enormus variety of cryptocurrencies, all satisfying the needs of a specific niche of users. Monero, for example, tries to attract buyers more concerned with privacy, granting complete fungibility between its coins thanks to the total anonymity of its transactions, while Tether tries to capture the interest of more risk-adverse investors, by linking its value to the one of fiat currencies.

Another inner characteristic of the cryptocurrency market is the degree of uncertainty that surrounds it. On one hand, if all the technical problems related to these assets will be solved, the blockchain could really be one of the game-changing technologies of this century, and cryptocurrencies with it. On the other, there is no real guarantee that this is going to happen, and with no solid basis to support them, cryptocurrencies prices are constantly, and dramatically, changing depending on the investors' believes.

The cryptocurrency market seemed therefore the missing piece to the puzzle as it was providing at the same time a much more volatile environment with respect to the previous ones, and the possibility of inspecting the capabilities of RL in a field that could be a really important player in the future of finance.

The portfolio itself was then constructed to incorporate the cryptocurrencies with the higher market capitalization as of today, to concurrently perform an informed selection, leaving aside the less infuential ones, and get the biggest slice of the market to have a good representation of it. The composition of the CMC All Crypto Index was then chosen as the one to be used, since the index not only reflected all the desired characteristics but provided, at the same time, an already constructed portfolio to use as a baseline. More information about the CMC ALL Crypto index is provided in Appendix 1

5.3.1 Environment Set Up, Training and Backtest

The first step of the training process was setting the environment. Also this time, as for the bond portfolio, the disposable time period needed to be adjusted considering the last asset that started to be traded. This translated into a dataset covering from 10/06/2018 to 31/03/2021. To obtain train and test, a 66%-33% split was adopted.

Then the four algorithms were again trained in the train environment and tested on the test one. The results are summarized in the figure below.

	Crypto_Stats	A2C	DDPG	PPO	SAC
0	Annual return	3.34984	3.29002	2.78583	2.89688
1	Cumulative returns	4.46116	4.37453	3.65199	3.80991
2	Annual volatility	0.69350	0.68091	0.68516	0.66891
3	Sharpe ratio	2.47186	2.48532	2.29084	2.37428
4	Calmar ratio	12.47023	12.47122	10.99046	11.15154
5	Stability	0.84290	0.84230	0.83759	0.86495
6	Max drawdown	-0.26863	-0.26381	-0.25348	-0.25977
7	Omega ratio	1.54265	1.53577	1.49454	1.50592
8	Sortino ratio	3.84536	3.83146	3.50988	3.61519
9	Skew	-0.09300	-0.20701	-0.19872	-0.30568
10	Kurtosis	2.54685	1.96287	2.13988	1.88825
11	Tail ratio	1.22886	1.20095	1.22558	1.24096
12	Daily value at risk	-0.08057	-0.07907	-0.08009	-0.07797

Figure 5.12: A summary of the performance indicators evaluated for the Cryptocurrency Models

Looking at the different performance metrics, it is immediately clear that the cryptocurrency market moves on a completely different scale with respect to the other two previously explored. Across all four algorithms, the one obtaining the lowest cumulative return (PPO) still achieved a 365% increase on the initial value of the portfolio. To compensate for that a much higher annual volatility, 66% at best, is present across the returns distributions, but the incredibly high Sharpe ratios, minimum 2.29, indicate that investors are adequately rewarded for their risk.

The best performing algorithms were, this time, DDPG and A2C, with the latter being again selected as the best one thanks to 10% higher cumulative returns.

The successive step was backtest. This time, unfortunately, the best model trained did not succeed in outperforming the baseline. The CMC ALL Crypto index gained, over the test period, an incredible +500%, with the A2C algorithm achieving a +446% at best (fig 5.13). The annual volatility of the index was also substantially lower, at 55%, than the one of the model portfolio, at 69%, and, consequently, the former had also a much higher Sharpe ratio, 2.9, with respect to the latter's, 2.47.

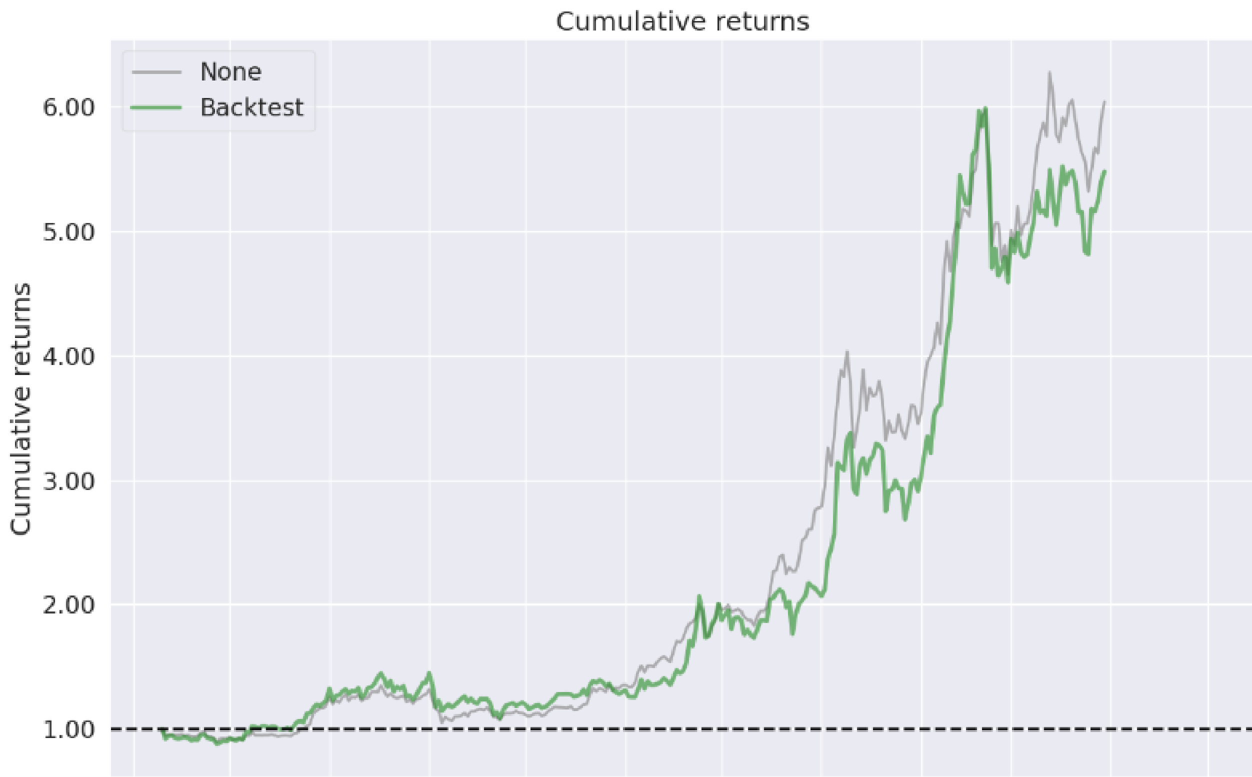


Figure 5.13: Cumulative returns chart of A2C model (green) vs the CMC ALL Crypto index (grey)

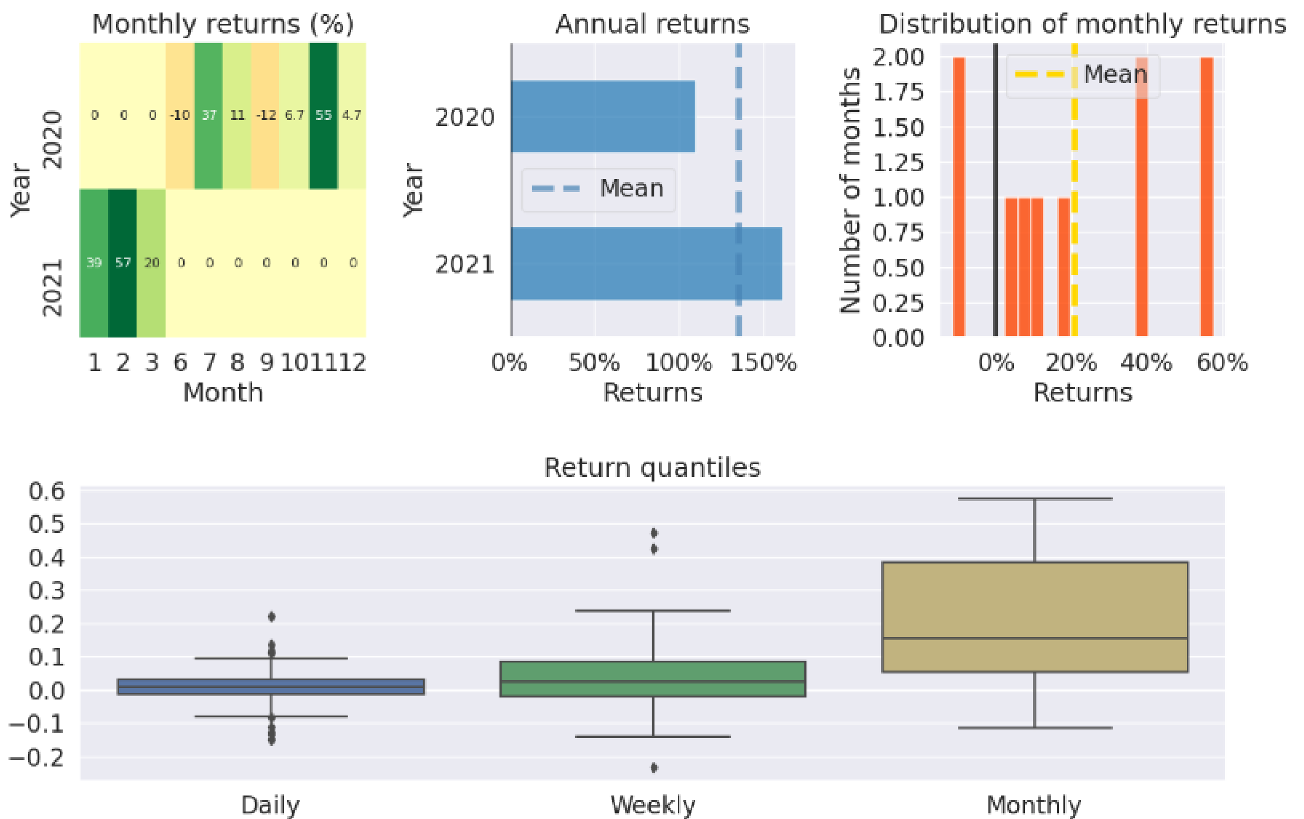


Figure 5.14: A2C model returns distribution plots

5.3.2 Hyperparameter Tuning

After the initial selection, an hyperparameter tuning step was performed to try to improve the results previously obtained. Again, as the selected algorithm was A2C, the tree parameter utilized to obtain the grid were n_steps, ent_coef, and learning_rate. Again four variations were given for the first one and three for the other two, obtaining 36 possible candidates.

The grid search algorithm selected as the best model the ones having the same ent_coefficient and n_steps as the first one, respectively 0.005 and 5, but a lower learning rate, 0.0001 instead of 0.0002. Unfortunately though, despite an increase in performances, the obtained model did not reach the cumulative returns obtained by the baseline, achieving an increase in the value of the portfolio of only 474%.

Figure 5.15: Cumulative returns chart of A2C Grid Search Model (green) vs CMC All Crypto Index (grey)

5.4

Results Discussion

The main objective of this chapter was to get an understanding of the potential of Reinforcement Learning methods in the field of finance and, specifically, in solving the problem of Efficient Portfolio Allocation. In addition, the hope was to highlight, adopting portfolios composed of different kinds of securities, discrepancies in performance across various algorithms to understand which of them, if any, was more suited for a particular market.

The results obtained from the analysis were diversified.

For what concerns the Stock market, only two of the four algorithms outperformed the Dow Jones 30 Index, but thanks to subsequent hyperparameter tuning, the best performing method, PPO, achieved cumulative returns higher than the baseline by over 13%. So, in this case, both objectives were reached: on one hand, the results proven the usefulness of RL applied to the subject, on the other a best model was identified in PPO.

For the Bond market the situation was quite similar. This time, all four of the trained models outperformed the baseline by at least 2% in returns, the worse being PPO, and the best one, A2C, reached a peak of 18.5%, 4.5% higher than the buy and hold strategy. Unfortunately though, the hyperparameter tuning step did not produce any additional increase in models' performances. The reasons for this are probably to be found in the low risk-low opportunity set up proper of the bond market itself: as not much variability is involved in the value of this kind of assets, after a certain point, the edges to be gained become very small. Generally speaking though, also for this market, both objectives were reached: RL algorithms performed better than the baseline, and a better model was selected in the form of A2C.

Finally, for the Cryptocurrencies market, the story was a little bit different. Not only none of the trained models outclassed the performance of the baseline, but neither one of them was able to even reach its returns. The reason for this is probably twofold. On one hand, the cryptocurrency market was, by far, the one for which fewer data were available for the composition of the environments. The total time span covered was of barely three years, with respect to six, for the bond market, and more than ten for the stock one. On the other hand, the test set was covering a period of extreme increase of the cryptocurrency market, regardless of what the algorithm was doing. This, despite in some ways seeming to be helpful for the models, could actually have hurt their performances due to the transaction fees. Rebalancing each day the composition of the portfolio when the assets contained in it are all going up in value anyway could, in the long run, turn out to be a detrimental strategy, as the edge gained by a better proportion of securities could be not enough to outset the costs paid to obtain it.

The result for the cryptocurrency market, despite not being as good as the others, seemed nevertheless promising, with the best of the trained models almost reaching the baseline returns, 474% vs 500%, and the first step of selection finding in A2C and DDPG the best algorithms for the specific environment by a substantial amount. Further iterations on this market, with a larger dataset, perhaps, or in a different period of time, could maybe produce appreciable results and are, therefore, worth pursuing.

Furthermore, different environments made actually emerge differences in performances across the four considered algorithms but, in general, policy optimization models outperformed mixed ones, with A2C being selected two times as the best model and PPO taking the remaining one. The reason for this is probably related to the complexity of the problem approached. With respect to other applications, in fact, Efficient Portfolio Allocation does not imply very high amount of difficulty, as controlling a robot in a simulated environment, for example, could. For this reason, simpler and more stable methods, as policy optimization ones, gain the edge over more complex and therefore unreliable models, since that higher complexity is indeed not necessary to efficiently solve the problem.

To sum up, the application of Reinforcement Learning models to the financial market was capable of producing significant improvements to the average buy and hold strategy for both bonds and stock portfolios, while further work would be needed to prove the same for cryptocurrencies. The analysis also highlighted the general characteristics that a model should have for efficiently solving the EPA problem while providing at the same time a starting point for the optimization, at least for the stock market, with the parameter configurations obtained in the hyperparameter tuning step.

SECTION 6: WEB APP DEVELOPMENT

6.1

Introduction and Motivations

The beauty of the financial applications of Reinforcement Learning is that also the solutions to relatively simple problems can have a real impact on the everyday investor. This is exactly the case for the Efficient Portfolio Allocation problem.

As the Modern Portfolio Theory, an algorithm capable to identify the cumulative returns maximizing portfolio at a given time could be useful, but just as a benchmark, when crafting a successful trading strategy. Reinforcement Learning settings instead, go a little more far, by being capable of constantly readjusting the portfolio composition on a day-to-day basis. This represents a real revolution as, with such premises, the trading process could be completely automatized by having an RL agent deciding how to manage the assets of each investor. Furthermore, since no deep knowledge of the subject would be needed to set up the process, the available public for such an innovation would be very large.

Following the results of the analysis carried on in Section 5 and motivated by the reasons just exposed, the natural conclusion for this work is then the development of a web app, designed to introduce the ideal everyday investor to the powerful possibilities offered by RL algorithms in finance.

It is important to notice that the app is thought to represent just a Proof Of Concept, and illustrate the point without getting too complicated, but also that, with little efforts, a small project like this could be expanded to become a fully functioning SAS (Software As Service) to be offered as a financial tool for online portfolio management.

The app is therefore limited to the Stock market, as it was the one on which the algorithms performed better in the analysis, and on a portfolio of maximum 15 stocks, to be chosen among the constituents of the Dow Jones 30 Index. The selection feature is, though, still present, as the user will be capable to choose among any combination of assets in the defined range, as well as the repeatability one, since each time the app is run a new selection of stocks will become possible.

The app was developed utilizing the Streamlit (20) library, a Python package designed to support the development of interactive web apps.

6.2

User Interface and Inner Functioning

The initial interface of the app, depicted in (fig 6.1), presents a little explanation of the porpuses of the software and describes what are ist functionalities. The user will be able, from this point, to select a portfolio of at most 15 different securities among all the ones listed in the Dow Jones 30 Index. The user will also be asked for a, preferably absolute, path, in which the app will later save the trained model for further use. The specification of this variable is completely optional but, if not given, the app will not save the model at the end of the process.

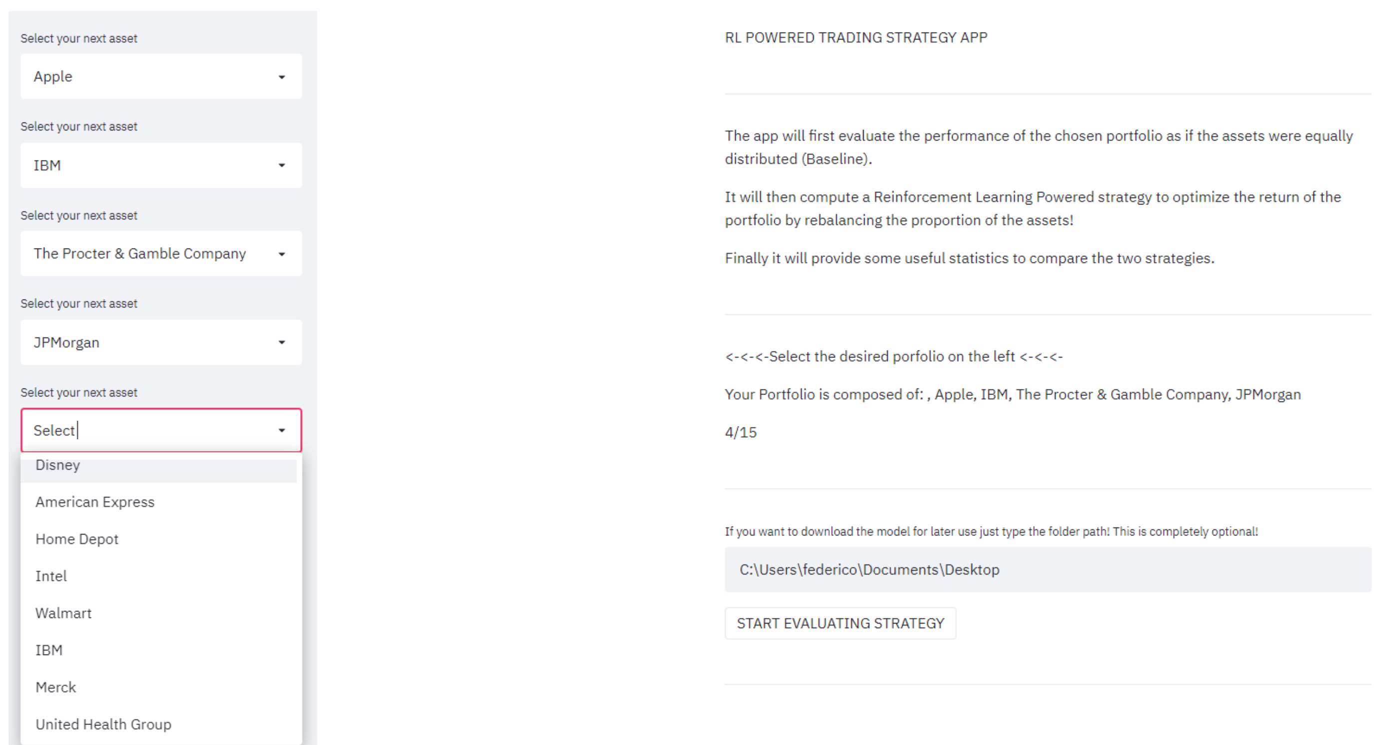
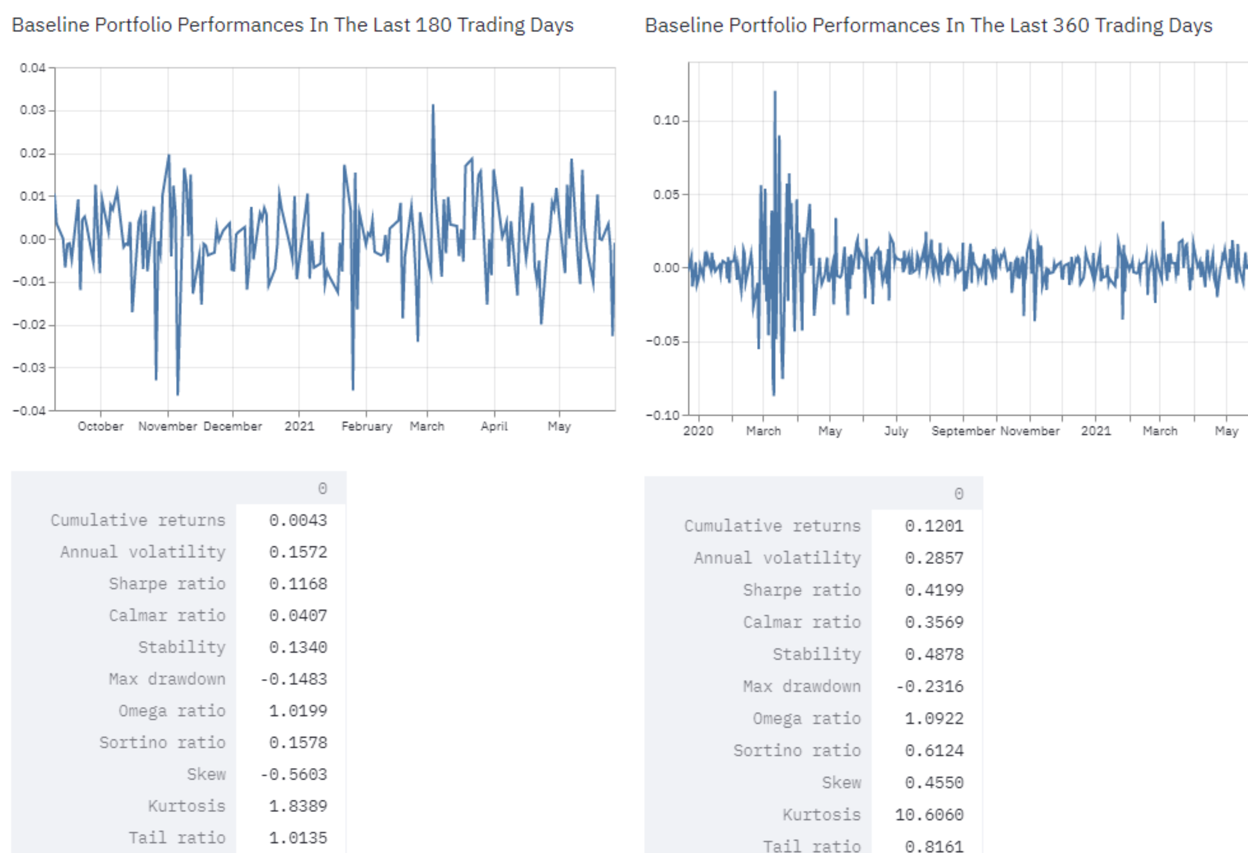


Figure 6.1: Starting User Interface of the Web App

The software will then convert the list of firms' names specified by the user into the relative tickers and exploit the Yahoo Finance APIs to download the financial information available on the site over a preset time span: 2009/01/01-2021/06/01. It will then go on to evaluate the performance of the relative baseline to have a comparison for the Agent returns. In the absence of a predefined index and in the need of a flexible way of evaluating the RL strategies, the baseline for each portfolio will be evaluated as a composition of the same portfolio having, though, an equal proportion of each asset. In this way, for each possible user choice, the baseline will be easily defined and its statistics easily computable, ensuring solidity to the software execution.

The performance of the baseline portfolio will be computed and showed as in (fig 6.2). The choice for two different timeframes to evaluate performance on, is thought to give the user a more complete feeling of the goodness of a strategy both in the long and in the medium run. This choice will be later carried on also for the Agent-curated portfolio.



	0
Cumulative returns	0.0043
Annual volatility	0.1572
Sharpe ratio	0.1168
Calmar ratio	0.0407
Stability	0.1340
Max drawdown	-0.1483
Omega ratio	1.0199
Sortino ratio	0.1578
Skew	-0.5603
Kurtosis	1.8389
Tail ratio	1.0135

	0
Cumulative returns	0.1201
Annual volatility	0.2857
Sharpe ratio	0.4199
Calmar ratio	0.3569
Stability	0.4878
Max drawdown	-0.2316
Omega ratio	1.0922
Sortino ratio	0.6124
Skew	0.4550
Kurtosis	10.6060
Tail ratio	0.8161

Figure 6.2: Baseline Portfolio Stats and Cumulative returns charts over 180 and 360 trading days.

The app will then start to set up the environment and train the agent. The choice of the algorithm implied, PPO, is the result of the analysis of the previous section, as well as the selection of the hyperparameters utilized: the same of the best performing model trianed by the grid search algorithm for the stock market.

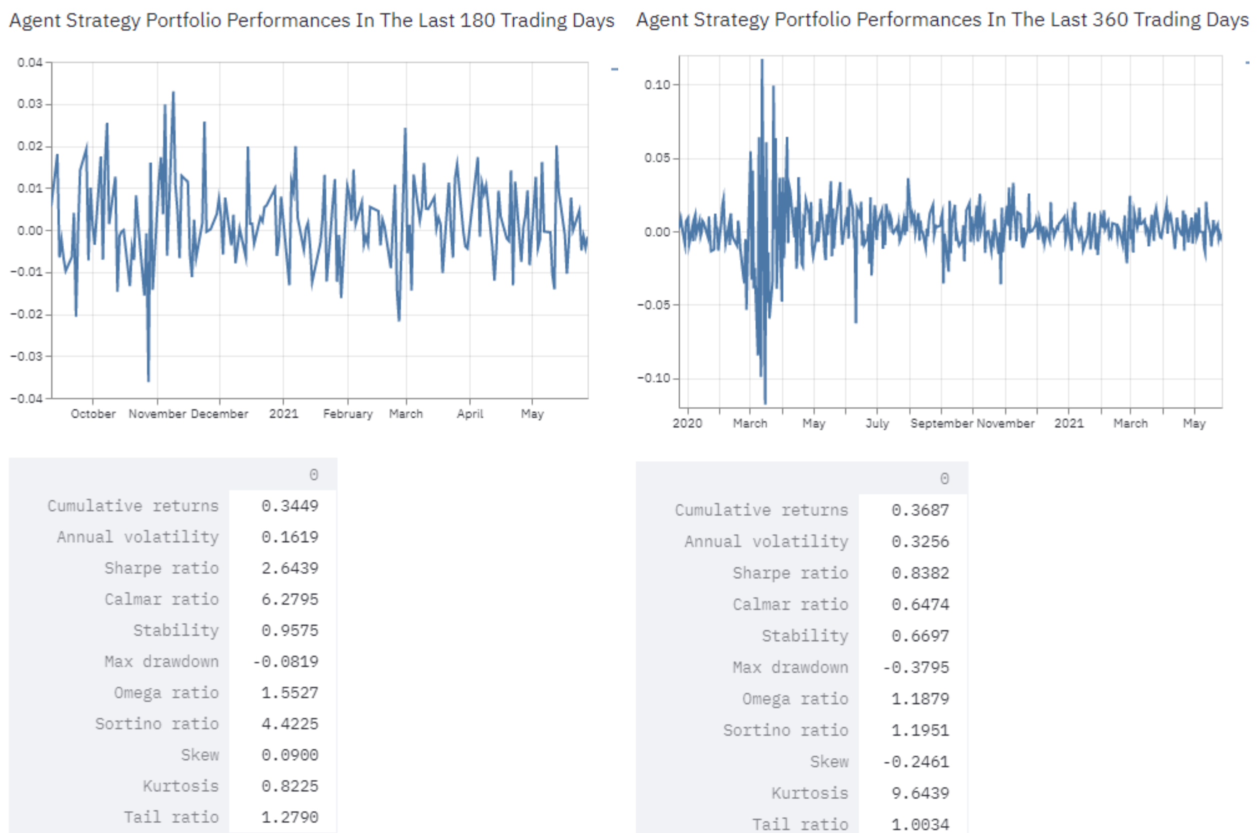
To train the Agent, the dataset acquired at the start of the execution is splitted into two, from the start to the first of January 2019 and then on, and just the first part is considered.

Once the agent is trained, its strategy is tested on the remainig data, and the performance evaluated is exposed in the same way as for the Baseline (fig 6.3).

Then a comparison plot is shown for both the 180 days range and for the 360 days one (fig6.4) and, finally, a summary of information relative to the strategy comparison is provided. The summary (fig 6.5) is composed of the edge gained or lost by the Agent strategy, evaluated as the difference in cumulative returns between the two portfolios, and by the average and standard deviation of the

proportion of each asset in the Agent portfolio. These last metrics are particularly interesting, as they provide insight respectively on which of the assets the Agent identified as the better-performing ones over a given period, and on how much the proportion for that asset was stable over time. A high number for the average paired with a low standard deviation, for example, will indicate a particulalry good asset to maintain in the portfolio.

Once the execution is completed the app saves the model on the path specified by the user, if any, and it is ready to be utilized again.



	0
Cumulative returns	0.3449
Annual volatility	0.1619
Sharpe ratio	2.6439
Calmar ratio	6.2795
Stability	0.9575
Max drawdown	-0.0819
Omega ratio	1.5527
Sortino ratio	4.4225
Skew	0.0900
Kurtosis	0.8225
Tail ratio	1.2790

	0
Cumulative returns	0.3687
Annual volatility	0.3256
Sharpe ratio	0.8382
Calmar ratio	0.6474
Stability	0.6697
Max drawdown	-0.3795
Omega ratio	1.1879
Sortino ratio	1.1951
Skew	-0.2461
Kurtosis	9.6439
Tail ratio	1.0034

Figura 6.3: Agent Portfolio Stats and Cumulative returns charts over 180 and 360 trading days.

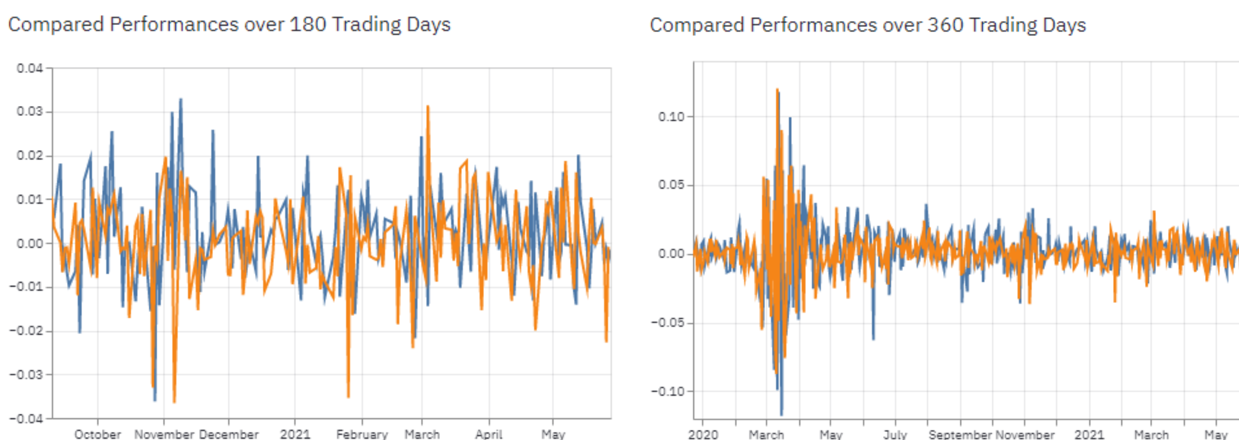


Figure 6.4: Comparison Plots (Cumulative returns) between Baseline (blue) and Agent (orange) over both 360 and 180 trading days.

Agent Strategy Porfolio achieved returns of 34.5% higher than the Baseline on a 180 trading days period

Agent Strategy Porfolio achieved returns of 24.5% higher than the Baseline on a 360 trading days period

---

These are the Average and Standard Deviation of the proportions of the Agent Strategy portfolio!

For 180 Trading Days

	Apple	IBM	JPMorgan	The Procter & Gamble Company
Mean	0.2411	0.2477	0.2467	0.2644
Standard Deviation	0.0835	0.0880	0.0931	0.0912

For 360 Trading Days

	Apple	IBM	JPMorgan	The Procter & Gamble Company
Mean	0.2468	0.2501	0.2455	0.2576
Standard Deviation	0.0853	0.0867	0.0892	0.0887

Figure 6.5: Information Summary

6.3

Further Developments

As stated in the introduction, this app represents just a Proof Of Concept, but with very little effort and a bit more resources it could be expanded to become a fully functioning SAS.

Improvements could take mainly two directions. First of all, the expansion to other markets. As it is now, the app makes the user select the stocks from a limited pool, but that could be easily expanded to comprehend more securities. Completely different markets, as the one of bonds, could be included too, adopting, for each market, the relative best-performing algorithm.

Secondly, the automatization aspect of the RL trading strategy, approximated in the app by the option of saving the model, could be developed until becoming a real service, by having the Agent fully manage the portfolio. This would imply servers, on which to run continuously the app, databases, on which to store the models for each portfolio of each user, and the APIs of online trading platforms, to make the Agent able to rebalance each day the portfolio composition.

Overall the potential is pretty huge and it is probable that services like this will become more and more popular in the following years.

SECTION 7: GENERAL CONCLUSIONS

In conclusion, this work's main objective was to uncover the potential of Reinforcement Learning algorithms applied to financial markets.

To do that first a revision of possible RL application was carried out, followed by a theoretical explanation of the methods implied. Then, the Efficient Portfolio Allocation problem history was revised and the problem itself was precisely formulated. Finally, at the core of this work, an analysis of performances among four different Reinforcement Learning algorithms, producing diversified but promising results, was carried out across three different markets and, to implement those results, a web app was developed.

The work, overall, reached its objective, by proving not only the capabilities of Reinforcement Learning on a conceptual basis with the analysis, but also its possible practical applications with the web app development.

Clearly, the disposal of greater datasets, higher computing power, and generally more resources could have led to improved models performances, but even with these settings, the analysis was capable of producing appreciable results. Therefore, this work could constitute the starting point for further research on the subject and it is looking forward to doing so.

Bibliography

1- http://finrl.org/ --- Finrl Library Complete Documentation

2- Mnih et al., 2013V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, et al.”Playing Atari with deep reinforcement learning. Technical report”.Deepmind Technologies (2013)arXiv:1312.5602 [cs.LG]

3-Tuomas Haarnoja, , Aurick Zhou, Sehoon Ha, Jie Tan, George Tucker, and Sergey Levine. "Learning toWalk via Deep Reinforcement Learning.".CoRR abs/1812.11103 (2018).

4- J. Morimoto, G. Cheng, C. G. Atkeson and G. Zeglin, "A simple reinforcement learning algorithm for biped walking," IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004, 2004, pp. 3030-3035 Vol.3, doi: 10.1109/ROBOT.2004.1307522.

5- Andrychowicz, O. M. et al. (2020) ‘Learning dexterous in-hand manipulation’, The International Journal of Robotics Research, 39(1), pp. 3–20. doi: 10.1177/0278364919887447.

6-Sallab, A., Abdou, M., Perot, E., and Yogamani, S. 2017. Deep Reinforcement Learning framework for Autonomous Driving. Electronic Imaging, 2017(19), p.70–76.

7-Wu, D., Chen, X., Yang, X., Wang, H., Tan, Q., Zhang, X., Xu, J., and Gai, K. 2018. Budget Constrained Bidding by Model-free Reinforcement Learning in Display Advertising. Proceedings of the 27th ACM International Conference on Information and Knowledge Management.

8- https://outreach.didichuxing.com/tutorial/kdd2018/ --- KDD Tutorial London (2018)

9- Arel, Itamar & Liu, C. & Urbanik, T. & Kohls, Airton. (2010). Reinforcement learning-based multi-agent system for network traffic signal control. Intelligent Transport Systems, IET. 4. 128 - 135. 10.1049/iet-its.2009.0070.

10- http://finrl.org/tutorial/finrl_multiple_stock.html --- Finrl multiple stock trading docs

11-Yuxi Li. (2019). Reinforcement Learning Applications

12-Michael Kearns, and Yuriy Nevmyvaka. (2013).Machine Learning for Market Microstructure and High Frequency Trading.

13- “Modern Perspectives on Reinforcement Learning in Finance”The Journal of Machine Learning in Finance, Vol. 1, No. 1, 2020. Kolm, Petter N. and Ritter, Gordon (September 6, 2019),

14- https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html --- Spinning OpenAi documentation on model-based RL algorithms

15- https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html --- Spinning OpenAI documentation on policy gradient

16- https://spinningup.openai.com/en/latest/algorithms/sac.html --- Spinning OpenAI documentation on SAC

17 - Jangmin,O.,Lee,J.,Lee,J.W.,& Zhang,B.-T.(2006). “Adaptive stock trading with dynamic asset allocation using reinforcement learning” . Information Sciences,176(15),2121–2147.

18- Jiang,Z.,Xu,D.,& Liang,J.(2017). ”A deep reinforcement learning framework for the financial portfolio management problem”. arXiv:1706.10059.

19- “Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder” —DeepBreath, Farzan Soleymani,Eric Paquet, NationalResearch Council.

20- https://streamlit.io/ --- Streamlit Complete Documentation

Appendix 1: Portfolios Composition

Bond Portfolio Tickers:

"VCIT","LQD","VCSH","IGSB","IGIB","SPSB","FLOT","SPIB","USIG","VCLT","ICSH","GSY","IGLB","SLQD"

Assets for the bond portfolio were selected from: https://etfdb.com/etfdb-category/corporate-bonds/

The Baseline for the bond portfolio is: https://it.finance.yahoo.com/quote/AGG?p=AGG&.tsrc=fin-srch

—

Crypto Portfolio Tickers:

"XLM-USD","ADA-USD","TRX-USD","XMR-USD","DASH-USD","NEO-USD","BTC-USD","ETH-USD","BCH-USD","LTC-USD","EOS-USD"

CMC ALL Crypto index can be foud at: https://www.cmcmarkets.com/en/cryptocurrencies/crypto-index