uc3m | Universidad Carlos III de Madrid

Master Thesis

Optimization of Financial Asset Portfolios through Deep Reinforcement Learning

Author: Ledicia Díaz Lago

Advisor: Francisco Javier García Polo

Department of computer science

July, 2020

Abstract

The purpose of this master thesis is to optimize a portfolio comprised by financial assets, maximizing its expected return minimizing the risk. Concretely, the idea is to learn a policy that optimally rearranges the assets of the portfolio by buying and selling them at the end of each market session, depending on their price evolution. Due to the the complexity of the market and the non linear patterns found in it, the decision process of reallocating the portfolio is treated from a Deep Reinforcement Learning approach, modeling the portfolio management problem as a Markov decision process. Then, while the Deep learning approach is in charge of learning the optimal policy, the Reinforcement Learning one considers the sequential nature of the process, and the effects of the rearranged assets at each trading period. In order to evaluate the risk and profitability of the trained system the back-test strategy, which splits the data into a training set (so as to train the model) and a test set (so as to assess the model on data that has not been seen by the model), is applied. Finally, this master thesis results conclude with the correct adaptability of the Deep RL agent when treating financial market environments.

Resumen

El propósito de este trabajo de fin de máster es optimizar una cartera compuesta por activos financieros, maximizando su rendimiento esperado y minimizando el riesgo. Concretamente, la idea es aprender una estrategia que reorganice de manera óptima los activos de la cartera comprándolos y vendiéndolos al final de cada sesión de mercado, en función de la evolución de sus precios. Debido a la complejidad del mercado y los patrones no lineales que se encuentran en él, el proceso de decisión que permite reorganizar la cartera se trata desde un enfoque de Aprendizaje por refuerzo profundo, modelando el problema de gestión de carteras como un proceso de decisión de Markov. Por lo tanto, mientras que el enfoque de aprendizaje profundo se encarga de aprender la estrategia óptima, el aprendizaje de refuerzo considera la naturaleza secuencial del proceso y los efectos que tiene la reorganización de los activos en la cartera para cada período de negociación. Con el fin de evaluar el riesgo y la rentabilidad del sistema, los datos se dividen en un conjunto de entrenamiento (para entrenar el modelo) y un conjunto de prueba (para probar su funcionamiento en datos que el modelo no ha visto). Finalmente, los resultados de este trabajo de fin de máster concluyen con la adaptabilidad correcta del agente de Deep RL al tratar entornos de mercado financiero.

I am especially grateful to Mateo, for being my proofreader, and to Juan, for listening and discussing with me all the things that did not make sense.

Contents

List of Figures

1 Introduction

Financial investing is the process of purchasing assets at a given time t in order to obtain a profitable return. Although the first financial market was the Wall Street market, which opened in 1792, it was not until the late 1960′s that the portfolio investment strategy was studied and formalized. Concretely, a portfolio is nothing but group of any kind of financial assets, such as stocks, bonds, commodities, currencies, and funds, that an investor purchases to produce income and meet financial goals [1].

This master thesis focus its attention on the portfolio management problem, which can be considered as a decision making process of rearranging capital into a number of assets aiming to maximize the return and minimize the risk. Going deeper into matter, let's imagine a portfolio composed of m assets, each of them represented by a weight, which indicates what percentage of the investment portfolio is comprised by an specific asset [2]. The portfolio can be defined in terms of a weight vector $\vec{w}$ composed by the individual weights of all the assets at a fixed period of time. However, due to the price movements in the market, the value of the investment products change over time, being necessary to buy or sell relevant assets in order to meet profits and avoid losses in the total portfolio value. As a way of example, let's assume a portfolio comprised of 2 assets such as Telefónica and Santander. Depending on its price evolution, an agent would invest more money (buy more shares) on one company or the other. For example, if Santander's expected return is higher than the expected return for Telefónica, then the agent would invest more money in Santander's company, presenting a greater influence in the portfolio value and therefore, a greater weight for this asset. The evolution of the market may change over time, and then the agent may decide to rearrange the capital and invest more money in Telefónica, and so on. Moreover, each of the agent decisions has consequences on the future benefits that him/her will end up obtaining.

Therefore, as it will be better explained later in this document, the portfolio management problem is inherently a sequential decision problem where a sequence of states, actions and rewards take place over time. This work studies how to model the problem as a Markov decision process for its subsequent resolution using Reinforcement Learning (RL) [3], and, more specifically, through Deep RL, which combines neural networks and Reinforcement Learning to solve complex problems [4].

1.1 Motivation

There are many works trying to solve financial market problems. However, most of them try to predict price movements, focusing on price forecasting. The problem of this approach, is that the agent's performance is subject to the accuracy with which the prices are predicted, being those very difficult to forecast. Therefore, this master thesis, inspired in the work of Jiang et al. in [5] and Kanwar in [6], focuses its attention on implementing a Deep RL approach on which the agent learns an investment strategy. The reason to explore Deep RL techniques to solve the portfolio management problem is justified by the fact that RL considers the sequential nature of the problem and the effects of the reallocating actions on the expected portfolio return, and deep neural networks (DNN), trained on the historic prices of the assets, can capture the complex market features. Thus, the RL framework generates a sequence of states, actions and reinforcements, and

the goal is to maximize a long-term measure of the reinforcement.

Specifically, the deep neural network is in charge of evaluating the immediate potential of growth of each asset, and pass this result to the final softmax layer whose outcome will be the new portfolio weights for the upcoming trading period [7]. The weights define the market action of the reinforcement learning agent, which will purchase those assets with an increased target weight, and sell the ones with decreased weight, taking under consideration the transaction costs of the trading operations.

1.2 Objectives

The main goal of this master thesis is the study and the application of Deep RL techniques to solve the portfolio management problem. In order to fulfill this main goal, the following subgoals must be studied first:

1. **Study of the modelization of the portfolio management problem as a MDP**. The portfolio management problem can be seen as a sequential decision process, and, therefore, it can be modeled as a MDP. In this work, we describe the state and action spaces and the reward function, which will allow us to later solve this problem through RL.

2. **Study of different Deep RL techniques to solve the previous MDP**. Recently, Deep RL has achieved an enormous popularity in solving complex problems. In this work, we study its use to solve the portfolio management problem.

3. **Train a learning agent through the selected Deep RL technique in order to maximize the expected portfolio return**. For this purpose, it is required to gather the data from a Spanish financial market in order to feed the model. Particularly, the agent is going to pretend to be back at a specific period $t_{\text{train}} \in (t_0, t_1, \ldots, t_{\text{current}})$[1], and then use the data from $[t_0, t_1, \ldots, t_{\text{train}}]$ to train the agent.

4. **Test the performance of the trained agent**. Once the agent has been trained its performance is going to be evaluated by applying a back test strategy. In this case, the data from $(t_{\text{train}}, \ldots, t_{\text{current}}]$ is used to test its performance [8].

1.3 Structure of the Master Thesis

So, in order to meet the goals described above, Section 2 provides a theoretical background in order to better understand the proposed work. Section 3 provides a detail mathematical description of the portfolio management problem, and Section 4 its subsequent conversion to a RL problem. Finally, Section 5 shows the results obtained, and Section 6 summarizes the main conclusions.

[1] t_0 and t_{current} stand for the period at which the first trading information is provided, and the current trading period respectively

2 Theoretical Background

This section introduces the theoretical framework and some key concepts necessary to understand this work. This is achieved both by first providing related work to portfolio management optimization (Section 2.1) and second by offering the theoretical basis of RL (Section 2.2) and Deep RL (Section 2.3).

2.1 Portfolio Management Optimization

The idea of optimizing a portfolio of assets is to select the best asset distribution, which is the one that maximizes the return of the portfolio and minimizes the risk. With this purpose, different optimization methods were born, starting from traditional algorithms, such as Online Moving Average Reversion [9], or Machine Learning approaches such as regularization [10], Cased-Based Reasoning [11, 12], or Support Vector Machines (SVM) [13].

However, each of these approaches has its own drawbacks. For instance, case-based reasoning may require a prohibitive number of cases to achieve an optimal behavior, and SVM is only directly applicable for two-class tasks. Furthermore, in all these cases the portfolio management optimization problem is approached from a *static* point of view and not as a *dynamic* sequential decision problem that evolves over time. Moreover, in this problem, one should be willing to make decisions to sacrifice short-term benefits if that means greater long-term benefits. So, this problem can be seen as a sequential decision problem and, in particular, as a Markov Decision Process (MDP), which can be solved using Reinforcement Learning techniques.

Therefore, inspired in the works by Jiang et al. [5] and Kanwar [6], whose previously addressed this problem using Deep RL techniques, this master thesis also explores a complete Deep RL approach, trying to apply Jian et al. [5] model to the Spanish stock market, and modifying the some important parameters, structures and functions with the purpose of improving the agent's performance.

2.2 Reinforcement Learning

Sequential decision making problems are the processes where a decision maker chooses an action given a concrete situation within an environment. Therefore, they provide a dynamic description of the environment in which they operate, and can be modeled by defining four fundamental components:

- Agent: It is the decision maker, and it is going to be trained based on a policy in order to choose those decisions that maximize a long term reward.
- Environment: It is the 'world' with which the agent interacts. The different situations or states are the different configurations of this environment. In the portfolio management problem, the environment corresponds to the market.
- Action: It is the decision taken by the agent so as to maximize a long term reward.

- Reward: It is the reward the agent receives from each of its actions.

RL typically is formalized as a sequential decision process, and, in particular, as a MDP. The training process in RL is based on the interactions between the agent and the environment, from which the agent gets a reward and gradually improves its performance [3]. Specifically, the agent will interact with the environment by selecting actions which cause an impact on it, and obtaining a reward as a function of this impact. Since the impact causes a response in the environment, a new situation or state is presented to the agent, which will have to select another action given the new situation and the reward of the last action [14]. Figure 1 summarizes the dynamics of a RL problem.

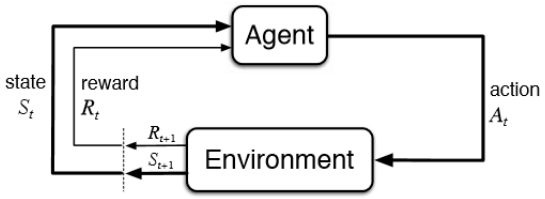


Figure 1: Reinforcement learning process [3].

Although there are several kinds of sequential decision making problems, the portfolio management problem is a Markov decision process. These type of processes are the most common ones when modeling a problem through RL.

2.2.1 Markov Decision Processes

A MDP is a sequential decision making problem which accomplishes the *Markov property*, where the values of the next state, s_{t+1}, and reward, $\mathfrak{r}_{t+1}$, depend only on the current state, s_t and action, a_t[2] [3]. This kind of processes can be described by the tuple $< S, A, \mathcal{T}, \mathfrak{R} >$ [15], where:

- S**:** The state space, i.e., the states $s_t \in S$ in which an agent can be located at each time step t.
- A**:** The action space, i.e., the actions $a_t \in A$ that the agent can select at a given time step t.
- $\mathcal{T} : S \times A \to \mathcal{P}(S) \in [0, 1]$**:** The transition function which describes the probability of occurrence of a state s_{t+1} given the state s_t and the action a_t taken in that state s_t.
- $\mathfrak{R} : S \times A \to \mathbb{R} : \mathfrak{r}(s, a)$**:** The reward function which assigns numerical rewards to transitions.

Therefore, the agent's goal is to find the *policy* $\pi : S \times A \in [0, 1]$ where $\pi_t(s, a)$ is the probability of selecting an action a_t at a given state s_t so as to maximize a long term reward.

[2]The action taken in that state, s_t

The action that the agent performs at each time step t depends only on the state s_t in which the agent is and not on past actions already taken. Thus, it does not matter what the agent has done so far, because it is the current state s_t the only required information by the agent to make a decision on the next action a_t to be performed in that state. This is the well-known *Markov Property* mathematically described in Equation 2.2.1.

$$\mathcal{P}_t = \mathcal{P}(s_{t+1}, \mathfrak{r}_{t+1}, s_t, a_t) = \tag{2.1}$$
$$= Pr(s_{t+1}, \mathfrak{r}_{t+1}|s_t, a_t) \underset{\text{Markov Property}}{=} Pr(s_{t+1}, \mathfrak{r}_{t+1}|s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0)$$

Equation 2.2.1 means that the probability that an agent at time step $t+1$ is in a state s_{t+1} and receives a reward $\mathfrak{r}_{t+1}$ only depends on the state s_t in which it was and on the execution of the action a_t it performed in that state s_t. Since $\mathcal{P}_t$ specifies a probability distribution of s_t and $\mathfrak{r}_t$ for each choice of s_{t-1} and a_{t-1} [3], the Markov decision process is subject to the constrain in Equation 2.2.

$$\sum_{t=1}^{tf} \mathcal{P}_t = \sum_{s_{t+1}\in S} \sum_{\mathfrak{r}_{t+1}\in\mathfrak{R}} \mathcal{P}(s_{t+1}, \mathfrak{r}_{t+1}, s_t, a_t) = \sum_{s_{t+1}\in S} \sum_{\mathfrak{r}_{t+1}\in\mathfrak{R}} Pr(s_{t+1}, \mathfrak{r}_{t+1}|s_t, a_t) = 1 \tag{2.2}$$

where the sums are over all the probabilities of occurrence of each of the states and rewards ($\mathcal{P}_t$).

2.2.2 Policies and Value Functions

As mentioned previously, the final goal of the agent is to find a *policy* $\pi : S \times A \in [0, 1]$, where $\pi_t(a, s)$ is the probability of selecting an action a_t at a given state s_t, so as to maximize a long term reward measurement[3] [3]., which will be a function of the onward rewards. Concretely, the return can be defined by the following equation:

$$G_t = \sum_{k=0}^{T} \gamma^k \mathfrak{r}_{t+k+1} \tag{2.3}$$

where γ is the discount factor and T the final period. Depending on whether the process is finite or not, this parameters have different values:

- **Finite process:** $T = t_f$, therefore $\gamma = 1$ cause no discount factor is needed as the sum is over a finite set.
- **Infinite process:** $T \to \infty$, thus $\gamma < 1$, placing a greater weight on the most recent rewards, and assuring the convergence of the sum.

[3]As it was explained before, the agent's action causes an impact on the environment. Since this impact influences not just immediate rewards, but also subsequent states, and through those, future rewards, the goal of the agent based on which it selects new actions is the one that maximizes the long term reward and not just the immediate ones

In order to find the optimal policy π, it is necessary to evaluate the actions taken under this strategy. This can be done by estimating the *value functions* of the system, which are two functions, one of the states and the other of the action-state pairs, that estimate how good it is for the agent to be in a given state [3], which is a consequence of the previous action.

1. State value function ($V_\pi(s)$):

Is the expected value of the return when starting in a state s_t, and following a policy π [3]. Therefore, the optimal policy, π^*, is the one that maximizes the state value function:

$$V_\pi(s_t) = \mathbb{E}_\pi(G_t|s_t) \Rightarrow V_{\pi^*}(s_t) = \max_\pi V_\pi(s_t) \tag{2.4}$$

2. Action value function ($Q_\pi(s,a)$):

Is the expected return when starting in a state s_t, taking an action a_t, and following a policy π [3]. Hence, the optimal policy, π^*, is the one that maximizes the action value function:

$$Q_\pi(s_t,a_t) = \mathbb{E}_\pi(G_t|s_t,a_t) \Rightarrow Q_{\pi^*}(s_t,a_t) = \max_\pi Q_\pi(s_t,a_t) \tag{2.5}$$

2.2.3 Bellman's Equations

The Bellman equations represent a necessary condition to find the optimal policy, and describe the dynamics of the system since they relate the value functions at period t with the value functions at the following period $t+1$ (*recursive relationship*).

The recursive relationship between two successive states can be derived from the recursive property of the return function, which relates the returns at successive time steps by the following equation [3]:

$$\begin{aligned} G_t &= \sum_{k=0}^{T} \gamma^k \mathfrak{r}_{t+k+1} = \mathfrak{r}_{t+1} + \gamma \mathfrak{r}_{t+2} + \gamma^2 \mathfrak{r}_{t+3} + \cdots = \\ &= \mathfrak{r}_{t+1} + \gamma(\mathfrak{r}_{t+2} + \gamma \mathfrak{r}_{t+3} + \dots) = \mathfrak{r}_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.6}$$

Therefore, equation (2.6) expresses a relationship between the value of a return and the value of its successor return. Starting from this equation, similar equations can be written for the state and action value functions:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi(G_t|s_t) = \mathbb{E}_\pi(\mathfrak{R}_{t+1} + \gamma G_{t+1}|s_t) = \\ &= \sum_a \pi(a_t,s_t) \sum_{s_t,\mathfrak{r}_t} \mathcal{P}(s_t,\mathfrak{r}_t,s_{t-1},a_{t-1}) \left[\mathfrak{r}_t + \gamma \mathbb{E}(G_{t+1}|s_{t+1})\right] = \\ &= \sum_a \pi(a_t,s_t) \sum_{s_t,\mathfrak{r}_t} \mathcal{P}(s_t,\mathfrak{r}_t,s_{t-1},a_{t-1}) \left[\mathfrak{r}_t + \gamma V_\pi(s_{t+1})\right] \end{aligned} \tag{2.7}$$

where:

- $\pi(a_t, s_t)$ is the probability of choosing an action given a state s_t.
- $\pi(a_t, s_t)\mathcal{P}(s_t, \mathfrak{r}_t, s_{t-1}, a_{t-1})$ is the probability of obtaining an action given the previous state and action.
- $G_t = \mathfrak{r}_t + \gamma V_\pi(s_{t+1})$ is the expected return defined in terms of the next state value function.

Since the action value function gives the expected return for taking action, a_t in a state, s_t, and thereafter following an optimal policy, it is possible to define $Q(s_t, a_t)$ in terms of $V(s_{t+1})$, as the latter describes the expected return when starting in a state s_{t+1}.

$$\begin{aligned} Q_\pi(s_t, a_t) = \mathbb{E}_\pi(G_t|s_t, a_t) = \mathbb{E}_\pi(\mathfrak{R}_{t+1} + \gamma G_{t+1}|s_t, a_t) = \\ = \sum_{a_t} \pi(a_t, s_t) \sum_{s_t, \mathfrak{r}_t} \mathcal{P}(s_t, \mathfrak{r}_t, s_{t-1}, a_{t-1}) \left[\mathfrak{r}_t + \gamma \mathbb{E}(G_{t+1}|s_{t+1}, a_{t+1})\right] = \\ = \sum_{a_t} \pi(a_t, s_t) \sum_{s_t, \mathfrak{r}_t} \mathcal{P}(s_t, \mathfrak{r}_t, s_{t-1}, a_{t-1}) \left[\mathfrak{r}_t + \gamma Q_\pi(s_{t+1}, a_{t+1})\right] \end{aligned} \tag{2.8}$$

Then, the Bellman equation is given by maximizing the Equations (2.7) and (2.8).

2.2.4 Exploration vs Exploitation

The RL training process consists on learning which actions to take, based on the interaction between the agent and the environment. This involves two processes which are related to each other:

- **Exploitation:** Choose the best action using the current information about the system.
- **Exploration:** Choose random actions in order to explore new states and therefore new possible solutions so as to gather more information and finding the optimal strategy.

Therefore, there exist a trade of between these two concepts, and, a equilibrium must be found in order to find the optimal policy for the agent.

Although there are several algorithms designed to ensure this equilibrium, this work is going to implement the ε-greedy method, which balances exploration and exploitation by choosing between exploration and exploitation randomly (ε refers to the probability of choosing to explore).

2.2.5 Resolution Methods

Bellman equations allows a recursive definition of the state-value function $V(s)$ and the action-value function $Q(s, a)$. However, they require the full model of the environment in order to be used, i.e., they assume we know the transition and the reward functions.

However, in a real world scenarios this is not always the case. Model-free RL algorithms allow dealing with environments for which we do not know both the transition and the reward function. There are several kinds of model-free RL algorithms, all destined to find the optimal policy π. However, it is possible to make a subdivision depending on whether the algorithms use value functions to select the action (algorithms based on value functions, e.g., Q-Learning) or not (policy gradient algorithms).

Q-Learning

Q-learning algorithm [16] is one of the most widely used model-free RL algorithms for computing the action-value function. The pseudocode of Q-learning is depicted in Algorithm 1.

Algorithm 1: Q-learning (off-policy TD control) for learning π

```
Algorithm parameters: step size α ∈ [0, 1], γ ∈ [0, 1], ϵ > 0, H, K;
Initialize Q(s, a), for all s ∈ S, a ∈ A(s), arbitrarily;
Initialize k = 0, h = 0;
while k < K do
    Initialize s_0;
    while h < H do
        Choose a_h from s_h using policy derived from Q (e.g., ϵ-greedy);
        Take action a_h, observe r_h, and next state s_{h+1};
        Q(s_h, a_h) ← Q(s_h, a_h) + α[r_h + γ max_a Q(s_{h+1}, a) − Q(s_h, a_h)];
        h ← h + 1;
    end while
    k ← k + 1;
end while
```

Algorithm 1 receives as input parameters the learning rate α, the discount factor γ, the exploration parameter ϵ, and the maximum number of episodes K and steps per episode H. At each time step, the algorithm chooses an action a_h in the current state s_h following an exploration/exploitation strategy (e.g., ϵ-greedy). Then, the agent takes a_h in state s_h, perceives the reward r_h from the environment, and transits to a new state s_{h+1}. The agent uses the experiences $< s_h, a_h, s_{h+1}, r_h >$ to update the Q-table and, in that way, learning a policy π. Despite it simplicity, most RL algorithms are based on the same principles of Q-learning.

Therefore, the Q-learning algorithm is a model free algorithm (it does not require a model of the environment) which learns a policy π telling the agent what action to take [17]. It updates a state–action pair through the following equation:

$$Q^{new}(s_t, a_t) = \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \left(\underbrace{\mathfrak{r}_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right) \tag{2.9}$$

Then, the learned action-value function, Q, approximates the optimal action-value function, Q_*, independent of the policy being followed. The policy's role is to determine which state–action pairs are visited and updated [16].

Policy Gradient Algorithms

Policy gradient algorithms turn out to be very useful when modeling not just deterministic policies, but also stochastic ones. Furthermore, since it is relatively easy to handle continuous actions with this algorithm, they have been chosen as the method to solve the portfolio management problem.

In particular, policy gradient algorithms parametrize the policy π using a *policy parameter vector* $\vec{\theta} \in \mathbb{R}^d$ whose dimensions are less than the number of states. The algorithm is focused on finding the optimal policy parameter vector $\vec{\theta}$ which maximizes a scalar performance measure $J(\vec{\theta})$. Then, the optimal policy is obtained using a gradient ascent algorithm which updates the parameter vector at each iteration [18]:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha \widehat{\nabla J(\vec{\theta}_t)} \tag{2.10}$$

Since the reinforcement learning goal is to maximize the expected return following a policy π, the scalar measure that is going to be maximized is the expected value of the reward $J(\vec{\theta}) = \mathbb{E}_\pi(G_t)$ when following a parametric policy.

2.3 Deep Reinforcement Learning

This section presents the key concepts of Deep RL to endow the reader with the required background to better understand the rest of the document. First, Section 2.3.1 shows the basics of neural networks and Section 2.3.2 the main concepts of Deep Learning. Afterwards, Section 2.3.3 presents the key concepts on Deep Reinforcement Learning.

2.3.1 Basics of Neural Networks

A Neural Network is a set of algorithms designed to recognize patterns. They are composed of several basic units or neurons, whose input is a finite set of m samples. Each sample is multiplied by a weight (parameter), forming a weighted combination of samples which are summed up, then, a bias is added to the sum and it is passed through a non-linear activation function to get the output $\hat{y}$ [19].

It is important that the activation function is non-linear so as to introduce non-linearities in the network and therefore, get a better approximation of complex functions. This wouldn't be possible if linear activation functions were used because they always produce linear decisions.

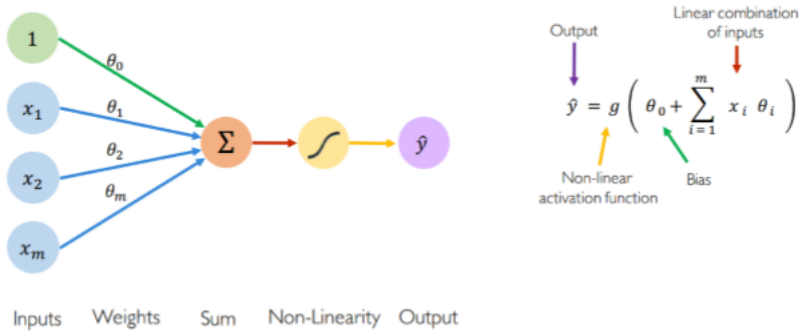


Figure 2: Basic Unit or Neuron [19].

The NN is then a combination of this basic units in charge of computing the output to predict. However, before being able to predict the expected outputs, the NN needs to be trained. During the training process, the NN sees the input samples, computes its outputs, and updates its weight parameters so as to minimize an objective function, also called loss function.

2.3.2 Basics of Deep Learning

Deep Learning is a machine learning algorithm which uses a special architecture of NN to extract features from the input data. The term 'deep' stands for the fact that the NN implements multiple layers, as increasing the number of layers allow the NN to compute more complex features from the raw input (Figure 3).

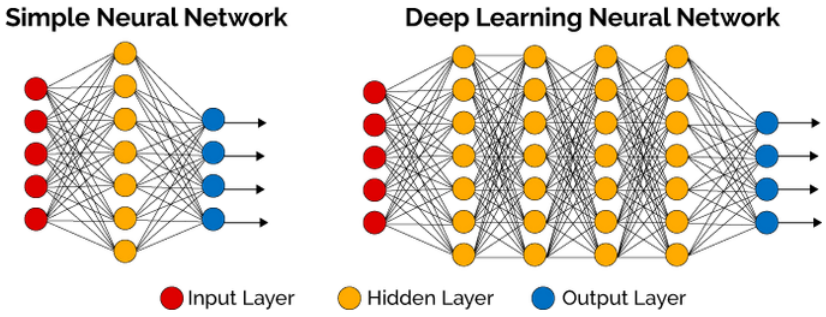


Figure 3: Neural Networks vs. Deep Neural Networks [20].

There is no a definite answer to how many layers a network have to have in order to qualify it as 'deep', but usually having two or more hidden layers counts as 'deep'. Although from a theoretical point of view it has not been demonstrated why having many layers is beneficial, it has been empirically demonstrated in various tasks and domains. For instance, Goodfellow [21] demonstrated this phenomenon in the transcription of multi-digit numbers from photographs of addresses. Figure 4 shows the results of this experiment and how the test accuracy consistently increases with increasing depth.

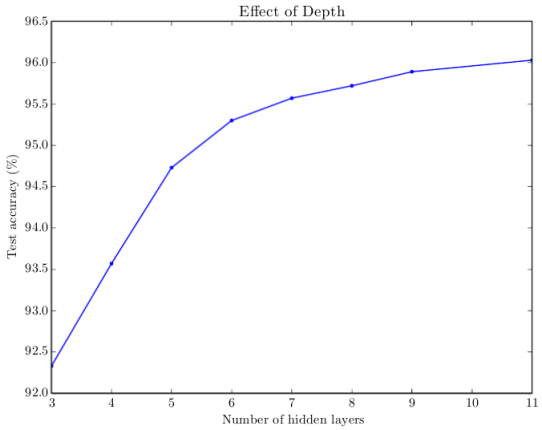


Figure 4: Empirical results showing that deeper networks perform better when used to transcribe multi-digit numbers from photographs [21].

Figure 3 shows the basic structure of a deep NN, however, there are more advanced architectures, such as Convolutional NN (CNN) and Recurrent NN (RNN), whose have shown exemplary performance on several task and domains, especially in those related to image classification. The most relevant particularities of each of these architectures are described below.

- **Convolutional NN:**

  Figure 5 shows the general shape of a Convolutional Neural Networks (CNN) [22]. It is divided in two phases: *Feature extraction* and *Classification*. The *Feature extraction* phase consists of sequential convolution and pooling operations carried out on an input image. The purpose of these operations is to extract relevant features from the input image and to reduce its dimensionality. Instead, the *Classification* phase consists of a fully connected NN which receives as input the features extracted in the *Feature extraction* phase, and produces as output the class of the object to be classified.

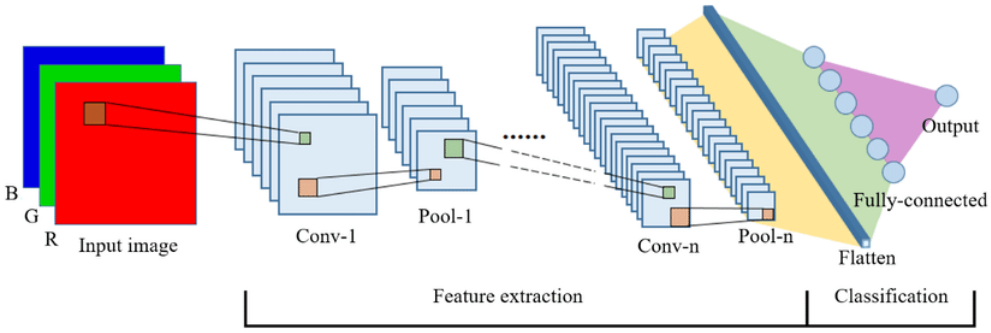


Figure 5: Convolutional Neural Network [23].

Therefore, what makes this network special are the convolution and pooling operations carried out at the beginning of the network. These operations are able to capture the temporal and spatial dependencies shown in an input matrix by applying relevant filters whose parameters are modified when training. One of its main

advantages is that a single filter is applied across different parts of an input to produce a feature map, sharing the parameters and therefore reducing the needed number of them (Figure 6):

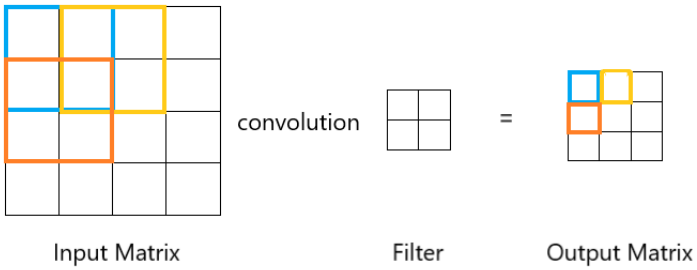


Figure 6: Convolutional operation and parameter sharing.

Although CNN are mostly used in image classification, this master thesis uses them as the algorithm to explore the potential of growth of each asset, since, using the appropriate filters, it is possible to explore the time series of each asset individually.

- **Recurrent NN:**

  Recurrent Neural Networks [24] are artificial NN with a recurrent connection on the hidden state so that they can take as input the current input example together with what they have explored previously in time. Therefore, the decisions made by a RNN in the previous time step affects the decision it will reach at time step t. A diagram of how those recurrent connections work can be found in the following image:

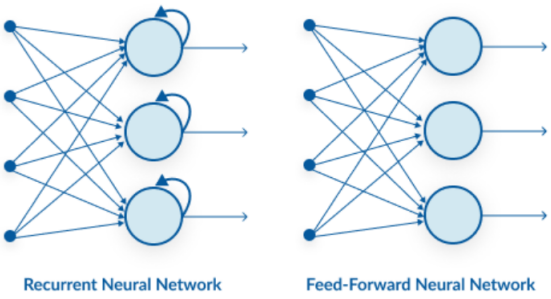


Figure 7: Recurrent NN vs NN [25].

that simplify the model more complex architectures able to capture local information such as neighbour data

2.3.3 Basics of Deep Reinforcement Learning

Due to the success of Deep Learning, Deep RL arises by combining the advantages of Deep Learning and RL to solve more complex problems than those allowed by the traditional methods discussed so far.

Although classical RL algorithms like Q-Learning are effective when solving simple problems, they are ineffective when facing problems with a greater number (possibly infinite) of states. To deal with a very large number of possible states, such as those that arise in the financial markets, deep neural networks turn out to be very useful, as they can explore all of them. To get a better understanding of why NN are better at dealing with a larger number of states, the difference between Q-Learning and Deep Q-Learning is shown in Figure 8.

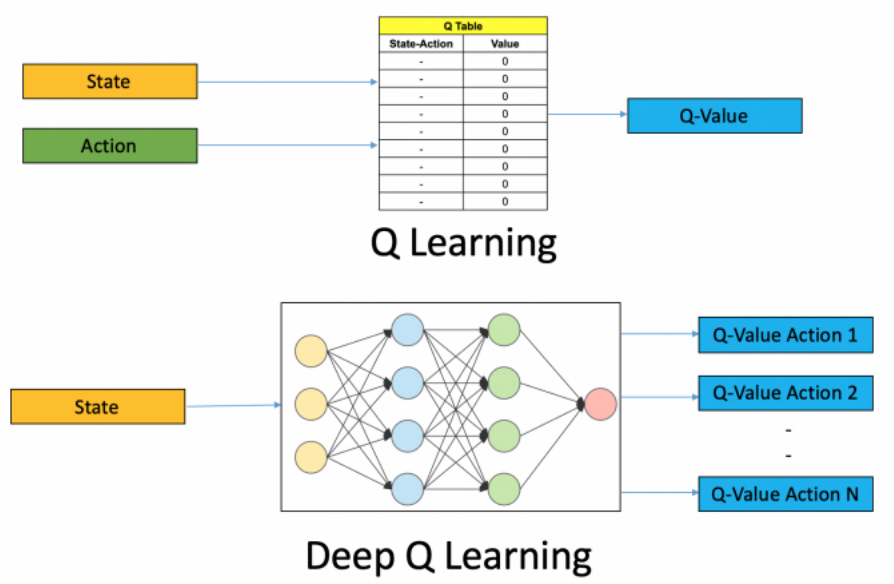


Figure 8: Q-Learning vs. Deep Q Learning [26].

While traditionally Q-Learning tries to approximate the Q-value function by a table that represents a finite number of states and actions, Deep Q Learning uses a Deep NN to approximate the Q-value function that can manage a much larger state space. It is important to be aware of the fact that, in the case of Deep Q Learning, the state is given as the input and the Q-value of all possible actions is generated as the output. Therefore, the best action will be the one with the highest network output. Additionally, the architecture of the Deep NN used as function approximation can be adapted depending on the problem to be addressed (e.g., CNN, Recurrent NN, etc).

Since the portfolio management problem needs to deal with a large number of states, this master thesis is going to implement Deep Reinforcement Learning (DRL) as the algorithm to explore the problem. In the portfolio management problem, the trading agent is represented by a DNN in charge of computing the action for the given state. Concretely, the state fed into the NN contains the prices of the assets along a defined interval of time, then, the NN computes the potential of growth of the assets, and finally the action vector is calculated by a softmax layer. The DNN is then going to be trained so as to minimize a loss function which is nothing but the minus expected value of the reward[4]:

$$\text{Loss} : \vec{L}(\theta) = -\vec{J}(\theta) = -\mathbb{E}_{\pi}(G_t) \tag{2.11}$$

[4]Minimizing the minus expected value of the reward is the same as maximizing it.

Anyway, the following sections will delve into the mathematical description of the portfolio problem (Section 3) and the mapping of this problem as a Deep RL task (Section 4).

3 Mathematical Description of a Portfolio

This section is oriented to give a brief introduction of the financial concepts needed to describe the portfolio management problem. Below, the concept of financial portfolio, its properties, and the market features that characterize the portfolio are going to be studied.

3.1 Portfolio

A **portfolio** is a group of financial assets (tangible or not tangible) which can be represented by the following equation:

$$P_t = \sum_{i=0}^{m} p_{i,t} \cdot A_{i,t} = \underbrace{\vec{p}_t \cdot \vec{A}_t}_{\text{dot product}} = \langle \vec{p}_t, \vec{A}_t \rangle \tag{3.12}$$

where:

- P_t stands for the portfolio at period t.
- $m+1$ is the total number of assets in the portfolio (m risky assets and a non risky asset which is the cash).
- $p_{i,t}$ is the price of the asset i at period t.
- $\vec{p}_t = (p_{0,t}, p_{1,t}, \ldots, p_{m,t})$ is the price vector of all the assets in the portfolio at period t.
- $A_{i,t}$ is the amount of shares of asset i in the portfolio.

Therefore, the value of the portfolio depends on how much money is invested in each asset. This information is encrypted in the portfolio weight vector $\vec{w}_t$, which indicates the importance of each asset in the portfolio value. Specifically, the weight of an asset i is given by its monetary value divided by the monetary value of the portfolio [2].

From this definition, the portfolio optimization process consists on selecting the best asset distribution which is represented by the weight vector $\vec{w}_t$. The process can be summarize in the following image:

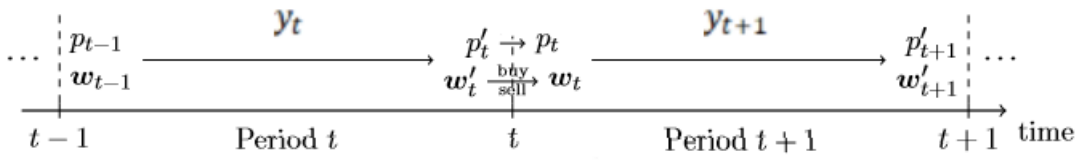


Figure 9: Portfolio management problem [27].

As it can be seen in figure 9, at the beginning of period t the portfolio vector is still $\vec{w}_{t-1}$, but because of different price movements in the market, the portfolio vector evolves

to $\vec{w}'_t$. So, the agent's goal is to rearrange relevant assets at the end of each period in order to reallocate the portfolio vector $\vec{w}'_t$ into $\vec{w}_t$ so as to maximize the expected return and minimize the financial risk. But to do so, the portfolio value at time period t_f and its expected return need to be defined.

3.2 Portfolio returns

The purpose of the portfolio management problem is to evaluate the need of updating the portfolio in order to maximize its final value and not the immediate rewards. Since the return informs about the profits and losses of an investment, it needs to be consider in order to calculate the portfolio final value P_f. Specifically, the return of an asset is defined as the change of the asset's price relative to the moment with which is being compared[28]. Starting from this definition, the simple return (ρ_t) can be computed as it follows:

$$\rho_t = \frac{p_t - p_{t-1}}{p_{t-1}} = \frac{p_t}{p_{t-1}} - 1 \tag{3.13}$$

However, due to the fact that in a portfolio the wealth accumulated by time t is reallocated in time $t+1$ [29], the compound return, which takes under consideration the reallocation of capital, is going to be used. Nevertheless, the continuous nature of the market price series has not been contemplated yet. In order to consider this, a definition of the fractional interest needs to be provided.

The compounded rate equation is obtained by considering the reallocation of capital from one period t to the next one ($t+$one year), but it also can be deduced by reallocating the funds from one period t to the following one $t+1$, where the lengths of the periods are less than one year [30]. Let n be the number of equal periods on which a year is subdivided, then, the compounded rate return can be defined as:

$$P_t = P_0(1+i)^{\Delta t} = P_0\,(1+i_n)^{n\Delta t} \tag{3.14}$$

where i is the annual compound rate of return, i_n the fractional compound rate of return, P_t the value of the portfolio at period t, P_0 the value of the portfolio at period t_0, Δt the number of years between P_t and P_0, and $n\Delta t$ the number of periods between P_t and P_0.

Now, in order to consider the fact that the market price series are continuous, the limit of the compound return when the number of periods in a year tends to infinity is calculated[5]:

$$P_t = P_0\,(1+i_n)^{n\Delta t} \Rightarrow P_t = P_0\left(1+\frac{r}{n}\right)^{n\Delta t} \tag{3.15}$$

$$P_t = \lim_{n\to\infty} P_0\left(1+\frac{r}{n}\right)^{n\Delta t} = P_0 \lim_{n\to\infty}\left(\left(1+\frac{1}{n/r}\right)^{n/r}\right)^{r\Delta t} = P_0 e^{r\Delta t}$$

[5]Since the period of one year is subdivided in n equal periods, the fact that n tends to infinite means that the sub divisions are so close together that the the intervals can be considered continuous.

where r is the continuous compound rate of return also called *logarithmic return*:

$$r_t = \log\left(\frac{P_t}{P_{t-1}}\right) \tag{3.16}$$

Since $\frac{P_t}{P_{t-1}}$ represents the change in the total portfolio value during the *t-th* session, it can be computed as a function of the change rate of the asset prices during the actual session, $\vec{y}_t$, and the portfolio vector before updating at the beginning of period t, $\vec{w}_{t-1}$ [27].

$$P_t = P_{t-1}\sum_{i=1}^{M} y_{i,t} \cdot w_{i,t-1} = P_{t-1}\vec{y}_t \cdot \vec{w}_{t-1} \tag{3.17}$$

This introduces the need of defining a price relative vector $\vec{y}_t$. Let $\vec{p}_t$ be the price vector containing the price of all the assets, then the change rate of the assets price $\vec{y}_t$ is given by.

$$\vec{p}_t = (1, p_{1,t}, \ldots, p_{M,t}) \Rightarrow y_t = \left(1, \frac{p_{1,t}^{(c)}}{p_{1,t}^{(o)}}, \ldots, \frac{p_{M,t}^{(c)}}{p_{M,t}^{(o)}}\right) \tag{3.18}$$

where the first asset is equal to $p_{0,t} = 1$ because is the quoted currency or *cash*.

Each element in the $\vec{y}_t$ vector represents the quoting of closing prices relative to the opening prices and it is called price fluctuating vector or the rate of change in the prices of the assets at period t [27].

Hence, it is possible to express the final portfolio value ($P_{t_f} = P_f$) in terms of the rate of return and the initial portfolio value ($P_{t_0} = P_0$):

$$r_t = \log\left(\frac{P_t}{P_{t-1}}\right) \underbrace{=}_{\text{equation 3.17}} \log(\vec{y}_t \cdot \vec{w}_{t-1}) \tag{3.19}$$

$$P_f = P_0 \exp\left(\sum_{t=1}^{t_f+1} r_t\right) = {}^{6}P_0 \prod_{t=1}^{t_f+1} (\vec{y}_t \cdot \vec{w}_{t-1}) \tag{3.20}$$

where P_0 is the initial investment amount, and P_f is the amount that is going to be maximize for a given time frame.

3.3 Transaction costs

Although equation 3.20 defines the portfolio value at period t_f, it does not represent the real value of the portfolio since transaction costs have not been considered yet. Every time the portfolio vector $\vec{w}'_t$ is modified by buying or selling shares, expenses due to

[6]The summation ends at period $t_f + 1$ because the reward obtained by reallocating the portfolio from $\vec{w}'_{t_f}$ into $\vec{w}_{t_f}$ is given at the beginning of period $t_f + 1$ as the portfolio vector of period t_f is reallocated at the end of this period.

transaction costs are incurred [31]. Below figure 9 is modified in order to illustrate the effect of the transaction costs:

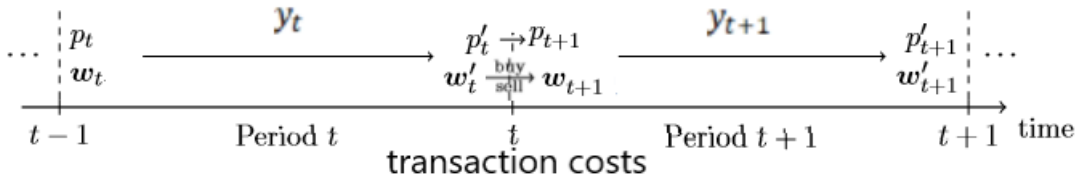


Figure 10: Effect of the transaction costs. The market movement during Period t is represented by the price-relative vector $\vec{y}_t$ [27].

At the beginning of period t, the portfolio vector is still $\vec{w}_{t-1}$, but because of different price movements in the market, the portfolio vector evolves to $\vec{w}'_t$. Since the weight of an asset i in the portfolio is determined by dividing the monetary value of the asset by the total monetary value of the portfolio [2], $\vec{w}'_t$ is given by:

$$\vec{w}'_t = \frac{\vec{y}_t \cdot \vec{w}_{t-1}}{\sum_{i=0}^{M} \vec{y}_{i,t} \cdot \vec{w}_{i,t-1}} \tag{3.21}$$

Then, the task of the portfolio manager is to reallocate the portfolio vector from w'_t to w_t by selling and buying relevant assets. Concretely, each day the trading agent observes the stock market by analyzing data and then reallocates his portfolio [29]. Since the selling and buying operations have commission fees, the reallocating action reduces the portfolio value by a factor $\mu_t \in (0, 1]$ called *transaction remainder factor*[31].

Taken under consideration the transaction costs, the real portfolio value at the beginning of period $t+1$ is given by $P_t = \mu_t P'_t$. Consequently, the logarithmic return can be expressed as a function of μ_t:

$$r_t = \log\left(\frac{P_t}{P_{t-1}}\right) = \log\left(\frac{\mu_t \cdot P'_t}{P_{t-1}}\right) = \log\left(\mu_t \vec{y}_t \cdot \vec{w}_{t-1}\right) \tag{3.22}$$

Therefore, the final portfolio value, which is also the factor to maximize, is given by:

$$P_{t_f} = P_0 \exp\left(\sum_{t=1}^{t_f+1} r_t\right) = P_0 \prod_{t=1}^{t_f+1} \mu_t \vec{y}_t \cdot \vec{w}_{t-1} \tag{3.23}$$

Nevertheless, the remainder factor μ_t has not been determined yet. To do so, it is necessary to calculate the transaction cost incurred when selecting the portfolio $\vec{w}_t$ by reallocating assets from $\vec{w}'_t$. Since the reallocating operation involves selling relevant assets and buying new ones the process is going to be split in these two actions. Let $\vec{w}'_{i,t} P'_t$ be the monetary value at the *i-th* asset before rearranging the capitals, and $\vec{w}_{i,t} P_t$ the monetary value at the same asset after rearranging, then:

1. **Amount of money obtained by selling relevant assets:**

If the amount of money at the *i-th* asset in the portfolio before rearranging funds is greater than the amount of money at the same asset after rearranging them, then the right option is to sell, because the asset's value in the portfolio has been depreciated as a consequence of the reallocating process [32]:

$$P'_t \vec{w}'_{i,t} > P_t \vec{w}_{i,t} \rightarrow \vec{w}'_{i,t} > \mu_t \vec{w}_{i,t} \tag{3.24}$$

Let c_s be the commission fees for selling, then the amount of money obtained by selling relevant assets is given by:

$$\text{Income} = (1 - c_s) P'_t \sum_{i=1}^{M} \left(w'_{i,t} - \mu_t w_{i,t}\right)^+ \tag{3.25}$$

where $(x)^+ = \text{ReLu}(x) = \begin{cases} x & \text{if x} > 0 \\ 0 & \text{otherwise} \end{cases}$ meaning that, if equation $w'_{i,t} > \mu_t w_{i,t}$ is not true, no transaction is made [32].

2. Amount of money used to buy new relevant assets:

The selling process generates an income which is used to improve the investor's portfolio as it covers the value of the shares to purchase. So, the stemming income and the original cash reserve $P'_t w'_{0,t}$ minus the actualized cash reserve due to reallocating process $\mu_t P'_t w_{0,t}$ is going to be used to purchase new assets [27].

$$(1 - c_p)\, P'_t \left[w'_{0,t} + (1 - c_s) \sum_{i=1}^{M} \left(w'_{i,t} - \mu_t w_{i,t}\right)^+ - \mu_t w_{0,t} \right] = P'_t \sum_{i=1}^{M} \left(\mu_t w_{i,t} - w'_{i,t}\right)^+ \tag{3.26}$$

Equations 3.25 and 3.26 define the dynamics of buying ans selling assets, therefore, it is possible to calculate the transaction remainder factor from this two equations:

$$\mu_t = \frac{1}{1 - c_p \cdot w_{0,t}} \left[1 - c_p w'_{0,t} \cdot (c_s + c_p - c_S c_p) \cdot \sum_{i=1}^{M} (w'_{i,t} - \mu_t \cdot w_{i,t})^+ \right] \tag{3.27}$$

However, this equation does not have an analytic solution, so either it is solved iteratively or an approximation of the transaction remainder factor is done [33]. In this framework, the transaction costs are going to be approximated by the following equation:

$$\text{costs}_t = \sum_{i=1}^{M} |\vec{w}'_{i,t} - \vec{w}_{i,t}| \cdot c \tag{3.28}$$

$$P_t = P'_t - \text{costs}_t = P'_t \cdot \mu_t \rightarrow \mu_t = 1 - \sum_{i=1}^{M} |\vec{w}'_{i,t} - \vec{w}_{i,t}| \cdot c$$

where c is the commission fee charged when buying or selling an asset.

3.4 Portfolio management hypothesis

As it was was explained in section 2.1 the portfolio is comprised of m risky assets which have to be selected from the market. These assets must accomplish two hypothesis:

1. Since the portfolio is rearranged every day trades must be carried out immediately at the last price when a order is place [34]. Therefore, the liquidity of the assets need to be very high, for which the selected assets have to be top volume in the market.

2. The market should not be influenced by the operations made by the trading agent [34]. Thus, the assets should also be top volume assets as the capital invested on them will have less influence on the market.

4 Mapping the Portfolio Management Problem onto Deep Reinforcement Learning

Once the mathematical description of the problem has been settled in Section 3, the following step is to build a deep reinforcement learning approach to train the trading agent. This approach combines deep neural networks (DNN), with a RL framework for sequential decisions, in such a way that:

- The DNN is in charge of computing the portfolio weight vector $\vec{w}_t$ at each period. As it will be better explained in subsection 4.1, the portfolio weight vector represents the actions taken by the RL agent when reallocating the portfolio. Therefore, the RL agent is represented by this NN, which follows a parametric policy π_θ.
- The RL framework is responsible for maximizing the expected return G (function of the future rewards).

4.1 Reinforcement learning framework

So as to model the portfolio management problem as a Markov decision process, the tuple $(S, A, P, r, \rho_0, \gamma)$ is going to be defined, as it represents the states of the market, the actions of the agent, and the rewards received from its actions. In this tuple S is a set of states, A is a set of actions, $P : S \times A \times S \to \mathbb{R}$ is the transition probability distribution, $\mathfrak{r} : S \to \mathbb{R}$ is the reward function, $\rho_0 : S \to \mathbb{R}$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor [29]. Below, each element of the tuple is going to be defined in terms of the portfolio mathematical description:

4.1.1 State $s \in S$

The environmental state of the market is described by the financial indexes of the assets time series in a fixed window of size n [7]. Thus, the resultant representation for the *current state* is given by the input tensor X_t as shown in Figure 11.

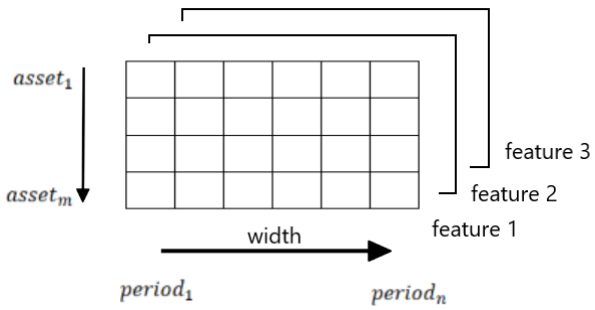


Figure 11: NN input tensor.

X_t is composed of f features for all the m assets at the n previous historic periods:

[7] n is the number of previous historic periods.

- **Selecting the number of features f:**

  A *feature* is a distinctive characteristic of a good or service that sets it apart from similar items [37]. Since the market environment is very complex and it is comprised of a large number of features, it is impossible for a trading agent to get all the information of a market state. Nevertheless, as the most relevant information is reflected in the prices of the assets [38], the closing $p_{i,t}^{(c)}$, highest $p_{i,t}^{(hi)}$, and lowest $p_{i,t}^{(lo)}$ prices of an asset i are going to be the input features of the DNN. However, this prices are going to be normalized by the opening price of each day in order to avoid different data scales and ensure a faster convergence.

- **Selecting the number of previous periods n:**

  The state of the market at each period can be represented by the selected features of the assets throughout the market's history up to the moment where the state is at [38]. However, since the available data is too large, only a subset of the n previous periods of the asset features is used as the DNN input. Because the market prices present a strong correlation at lags that are closer to the current moment and its correlation decreases as lags are further back in history, a window size of n=50 days is used [38].

- **Selecting the assets m:**

  The portfolio is composed by m non-cash assets. Although the number of chosen assets can be any, it is important that they are top volume (section 3.4).

The input tensor X_t (Figure 11) is therefore given by stacking together the following 3 relative price vectors as shown in Equation (4.29).

$$
\begin{aligned}
V_t^{(c)} &= \left[\frac{\vec{p}_{t-n+1}^{(c)}}{\vec{p}_{t-n+1}^{(o)}}, \frac{\vec{p}_{t-n+2}^{(c)}}{\vec{p}_{t-n+2}^{(o)}}, \ldots, \frac{\vec{p}_{t-1}^{(c)}}{\vec{p}_{t-1}^{(o)}}, \frac{\vec{p}_{t}^{(c)}}{\vec{p}_{t}^{(o)}}\right] \\
V_t^{(hi)} &= \left[\frac{\vec{p}_{t-n+1}^{(hi)}}{\vec{p}_{t-n+1}^{(o)}}, \frac{\vec{p}_{t-n+2}^{(hi)}}{\vec{p}_{t-n+2}^{(o)}}, \ldots, \frac{\vec{p}_{t-1}^{(hi)}}{\vec{p}_{t-1}^{(o)}}, \frac{\vec{p}_{t}^{(hi)}}{\vec{p}_{t}^{(o)}}\right] \\
V_t^{(lo)} &= \left[\frac{\vec{p}_{t-n+1}^{(lo)}}{\vec{p}_{t-n+1}^{(o)}}, \frac{\vec{p}_{t-n+2}^{(lo)}}{\vec{p}_{t-n+2}^{(o)}}, \ldots, \frac{\vec{p}_{t-1}^{(lo)}}{\vec{p}_{t-1}^{(o)}}, \frac{\vec{p}_{t}^{(lo)}}{\vec{p}_{t}^{(o)}}\right]
\end{aligned}
\tag{4.29}
$$

where each of the elements of V_t is a column vector.

Together with tensor X_t, the previous action $\vec{w}_{t-1}$ is also going to be considered as a part of the state. The reason to do so is to ensure that the Markov property [8] is accomplished, since the previous action $\vec{w}_{t-1}$ does have an influence on the new portfolio value through $\vec{w}_t'$, r_t and μ_t (Equations 3.21, 3.22 and 3.3) [40]:

$$S_t = (X_t, \vec{w}_{t-1}) \tag{4.30}$$

[8] Markov property: the effects of an action taken in a state depend only on that state and not on the prior history [39].

where, X_t represents the external state, and $\vec{w}_{t-1}$ the internal one. The portfolio value at each period P_t is not included since it has little influence in the market (hypothesis 2 of section 2.5).

4.1.2 Action $a \in A$

The agent's purpose at the end of period t is to reallocate the portfolio vector $\vec{w}'_t$ into $\vec{w}_t$ in order to maximize the return and minimize the risk. Then, the agent's action $\vec{a}_t$ is equal to the reallocated portfolio vector $\vec{w}_t$.

$$\vec{a}_t = \vec{w}_t \tag{4.31}$$

4.1.3 Reward function R

The reinforcement learning framework is based on the interactions between the agent and the environment, from which the agent gets a reward and gradually improves its performance. Specifically, the agent's goal is to maximize the long term reward along a set of trading periods so as to maximize the portfolio final value P_f. However, since the agent has no control over the length of the whole portfolio management train process, $(t_f - t_0)$, the average of the reward function is what is going to be maximize [40].

Two different reward functions are going to be explored. The first one (Logarithmic Return) is the one maximized by the agent in Jiang et al. work [5], the second one (Sharpe Ratio) is tested so as to study if the Network's performance is improved while trying to reduce volatility of the portfolio:

- **Logarithmic Return:**

  The immediate return obtained by the agent is given by 3.22, therefore, the average of the returns obtained along a train period is given by:

$$R_1(s_1, a_1, \ldots, s_{t_f}, a_{t_f}, s_{t_f+1}) = \frac{1}{t_f} \log\left(\frac{P_f}{P_0}\right) = \sum_{t=1}^{t_f+1} \log(\mu_t \vec{y}_t \cdot \vec{w}_{t-1}) = \frac{1}{t_f} \sum_{t=1}^{t_f+1} r_t \tag{4.32}$$

  where the r_t is the immediate reward for an individual episode.

- **Sharpe Ratio:**

  Although equation 3.22 contains the information about the returns obtained along the training period, it does not consider the risk of the portfolio. In order to take this under consideration, the Sharpe ratio is going to be maximized. Concretely, the Sharpe ratio is the average return earned in excess of the risk-free rate per unit of volatility (total risk) $SR = \frac{\text{mean}(r)}{\text{std}(r)}$ [41].

$$R_2(s_1, a_1, \ldots, s_{t_f}, a_{t_f}, s_{t_f+1}) = \frac{1}{t_f} \sum_{t=1}^{t_f+1} SR_t \tag{4.33}$$

4.2 Deep Reinforcement Learning Framework

Finally, the combination of the DNN framework, and the RL framework culminates in the deep reinforcement framework for portfolio management. Concretely, the agent is a DNN whose loss function is the minus reward obtained from the RL framework [9]. Then, the policy function π_θ (Section 2.3) is build by training the agent with the RL reward function.

4.2.1 Deep Neural Network Architecture

The chosen architecture for the DNN agent uses Ensemble of Identical Independent Evaluators (EIIE) [35], ensuring that the prices time series of the assets are evaluated independently (Figure 12).

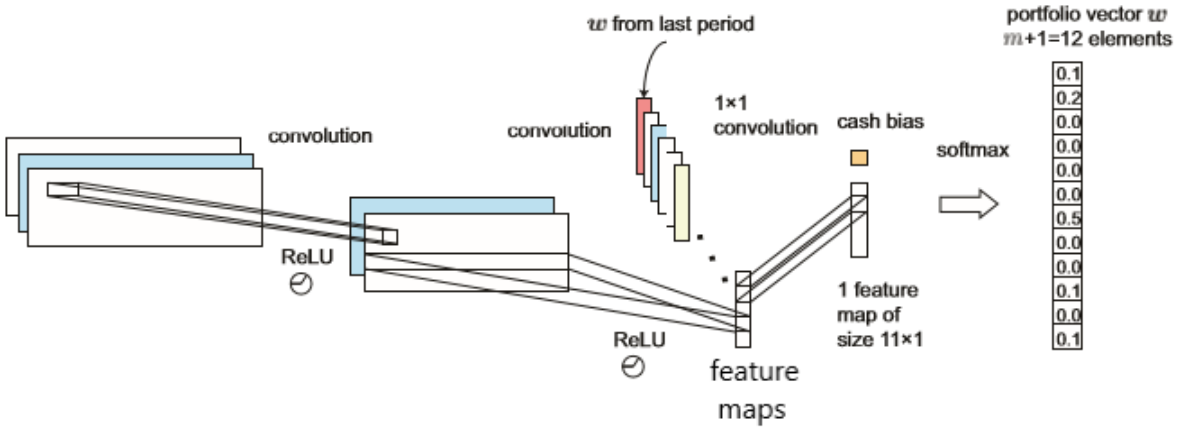


Figure 12: NN architecture [35].

Below, each layer is explained in detail:

- **First Layer:**

  Convolutional layer composed of 3 filters of size [1,3] in charge of exploring each asset individually. Its input is shown in Figure 11 (shape [assets, periods, features]). The output has shape of [assets, periods-2, 3].

- **Second Layer:**

  Convolutional layer comprised of 48 filters of size [1, periods-2]. The output has shape of [assets, 1, 48].

- **Third Layer:**

  Convolutional layer comprised of 1 filter of size [1, 1]. The previous action $\vec{w}_{t-1}$ is concatenated with the output of the layer two so as to create the input of this layer. The output has shape of [assets] and it is a vector (rank one tensor) containing the potential of growth of each asset.

[9] Minimizing the minus reward is the same as maximizing the reward.

- **Last Layer:**

  Softmax layer in charge of computing the action for the upcoming period $\vec{w}_t = \vec{a}_t$.

4.2.2 Deep Neural Network Training

So that the agent can maximize the cumulated reward the training process is going to use mini-batches. Mini-batches are tensors composed of several samples [10] which are fed into the NN. The network analyses all the samples, computes an action for each sample, and calculates the immediate rewards for each action. Then, the cumulated reward over batch is computed and the NN updates its parameters.

Mini-batches are build from *bs* samples settled in time-order. To explore the whole training set, each epoch (number of times the NN sees the training set) is comprised by a number of batches, each of which starts in a period $t_b \in$ trainig set which is picked with a geometrically distributed probability.

Besides, in order to ensure exploration, the ε-greedy algorithm described in section 2.2.4 is employed.

[10] Each sample in the batch has the shape of Figure 11

5 Experimental Results

This section's purpose is to evaluate the performance of a DRL agent in different back-test experiments. Therefore, the portfolio selection, the data preparation and the training process are explained below.

5.1 Experimental setting

The chosen portfolio is comprised of 9 assets (Telefónica, Santander, Grifols, Biosearch, Faes-Farma, Airbus, Indra, Inditex and Solaria) of the Spanish Stock Market. The stocks time series were downloaded from Yahoo Finance[11], filling up the missing prices with the close price of the previous day.

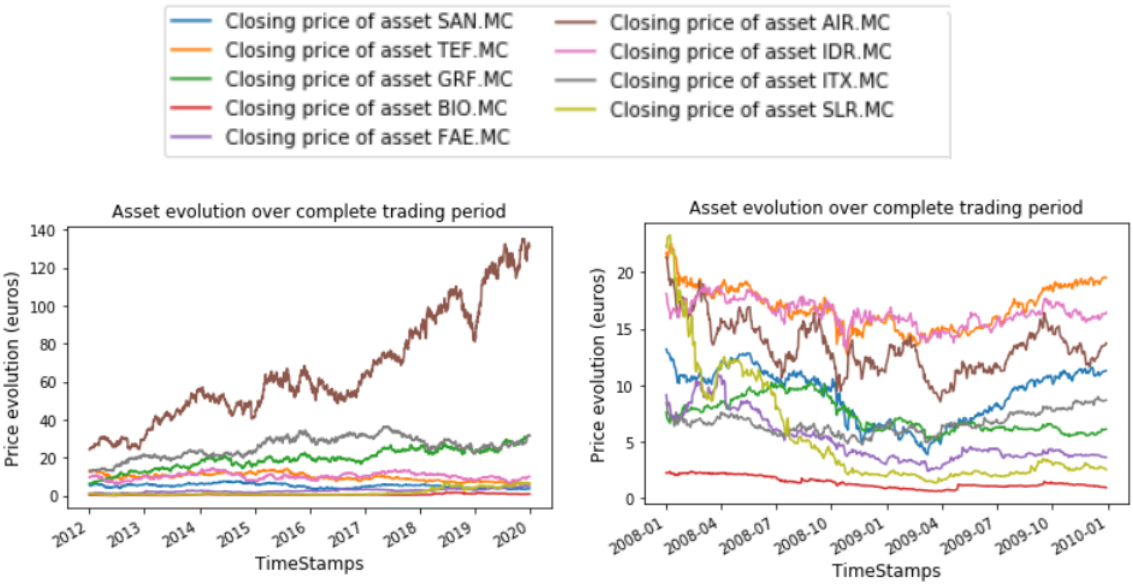
Besides, so as to avoid overfitting, the data was normalized. Concretely, two normalizations are studied. The first one divides the prices of each of the asset time series by the opening price of each day, while the second one divides them by the highest closing price for each specific asset. This step is very important since ensures the proper training of the NN, and its effects are going to be studied in section 5.2.

Several back-test periods are explored so as to evaluate the performance of the trading agents. The experiments explore two objective functions or reward functions, the logarithmic return and the Sharpe Ratio (Section 4.1.3), and the idea of introducing the cash as a non risky asset in the portfolio. Since an RL agent is trained by maximizing a reward function, in this experimental section two agents are considered: one trained to maximize the average of the logarithmic namely Agent 1, and other trained to maximize the Sharpe Ratio namely Agent 2.

Furthermore, the adaptability of the trained agent to different trading periods where the behaviour of the assets has nothing to do with the one seen by the agent in the training process is also studied. In order to do so, two main trading periods are considered. The first one goes from 2012/01/01 to 2020/01/01 and the second one goes from 2008/01/01 to 2010/01/01[12]. Figure 13 graphically represents the evolution of the prices of the assets for each of these trading periods. To make things simpler, lets call trading period 1 to the one that goes from 2012/01/01 to 2020/01/01 and trading period 2 to the one that goes from 2008/01/01 to 2010/01/01.

[11] https://finance.yahoo.com/

[12] For 2011 there are several prices missing for two of the assets so this year was not considered in the trading periods.



(a) (b)

Figure 13: (a) Evolution of the prices of the assets along trading period 1 and (b) Evolution of the prices of the assets along trading period 2.

Regarding the selected training and back-test sets, and the experiments conducted:

- For the trading period 1, Agent 1 and 2 are trained on the price data set from 2012/01/02 to 2016/10/17, leaving periods 2016/10/18-2018/05/24, 2018/05/25-2019/12/30 and 2016/10/18-2019/12/30 for back-test experiments. The goals for these experiments are: to select the best agent, and to explore the idea of introducing the cash as a non-risky asset. The results of these analysis can be found on Section 5.2.
- For the trading period 2, only the agent with the best results in trading period 1 is considered. The aims of these experiments are: to evaluate the adaptability of the agent to other trading periods, and to use data from 2008/01/01 to 2009/03/12 so as to retrain the agent in order for him to learn a more dynamic trading strategy. The outcomes of these studies can be found on Section 5.3.

Summarizing, the main goals for these experiments are on the one hand assess the objective function which provides the best results and, on the other, evaluate the adaptability of the agents in periods which differ a lot from the one used to train them.

5.2 Results for Trading Period 1

During the experiments, a very interesting fact related to the normalization of the data emerged. At the beginning, the prices of the assets time series were normalized by the opening price of each day, capturing the intraday's price variation[13]. Nonetheless, this

[13]How much the closing, highest and lowest prices changed compared to the opening price, and therefore along the session.

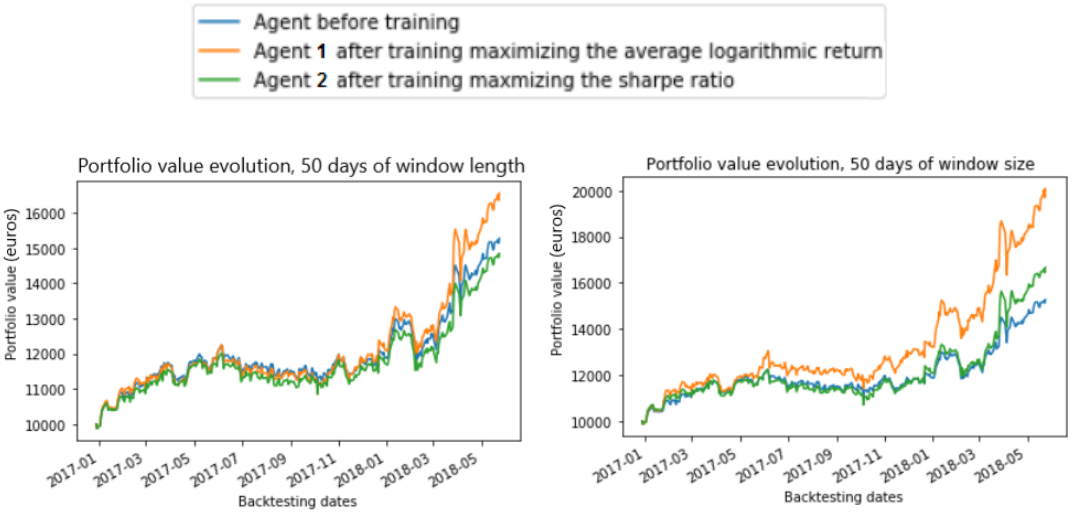
lead to a major overfitting of the data. Then, so as to ensure that the NN was able to capture the long-term price variation, all the prices of each asset were normalized by the highest closing price of the considered asset[14]. Therefore, if $\vec{p}$ is the highest closing price of each asset, tensor X_t from Equation (4.29) transforms to Equation (5.34).

$$V_t^{(c)} = \left[\frac{\vec{p}_{t-n+1}^{(c)}}{\vec{p}}, \frac{\vec{p}_{t-n+2}^{(c)}}{\vec{p}}, \ldots, \frac{\vec{p}_{t-1}^{(c)}}{\vec{p}}, \frac{\vec{p}_{t}^{(c)}}{\vec{p}}\right] \tag{5.34}$$

$$V_t^{(hi)} = \left[\frac{\vec{p}_{t-n+1}^{(hi)}}{\vec{p}}, \frac{\vec{p}_{t-n+2}^{(hi)}}{\vec{p}}, \ldots, \frac{\vec{p}_{t-1}^{(hi)}}{\vec{p}}, \frac{\vec{p}_{t}^{(hi)}}{\vec{p}}\right]$$

$$V_t^{(lo)} = \left[\frac{\vec{p}_{t-n+1}^{(lo)}}{\vec{p}}, \frac{\vec{p}_{t-n+2}^{(lo)}}{\vec{p}}, \ldots, \frac{\vec{p}_{t-1}^{(lo)}}{\vec{p}}, \frac{\vec{p}_{t}^{(lo)}}{\vec{p}}\right]$$

So as to analyze the overfit phenomenon, the corresponding agents' performance for the back-test period 2016/10/18-2018/05/24 is explored in both normalization frameworks.



(a) (b)

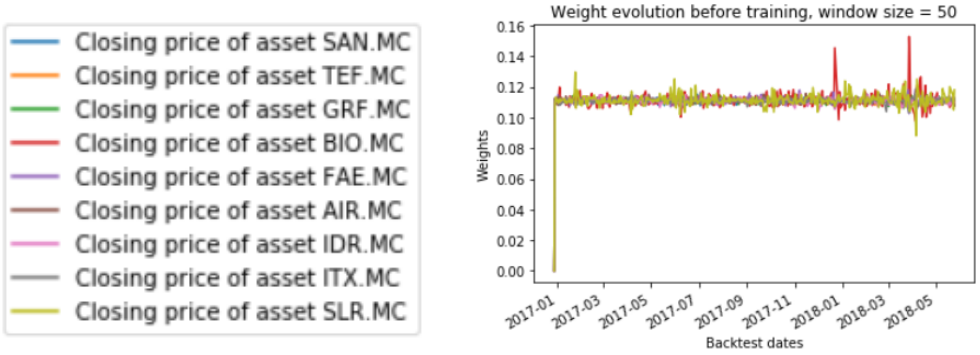
Figure 14: Agents' performances for back-test period 2016/12/28-2018/05/24 when (a) prices normalized by the opening price of each day and (b) prices normalized by the closing price of each day.

Although the back test experiment shown in Figure 14 starts in 2016/10/18, since the window length consists on 50 days, the agent needs the prices of the 50 previous days to start computing actions.

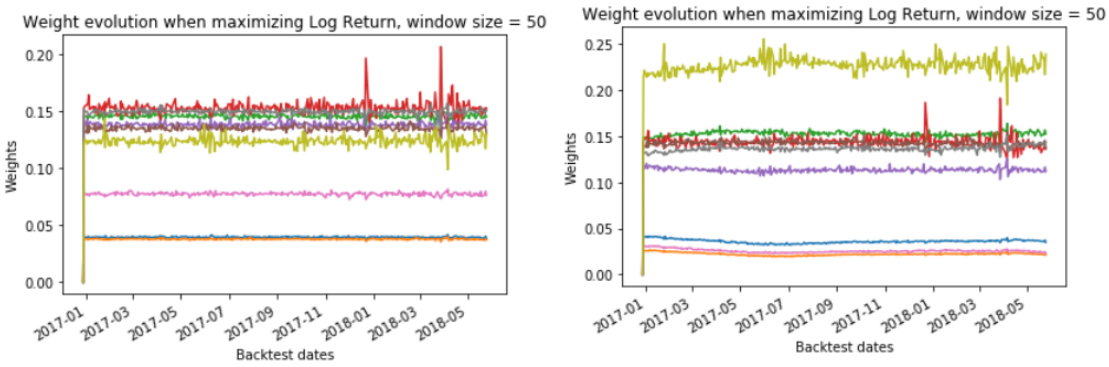
As it can be seen, the NN performance improved dramatically when normalizing the prices of each stock by the same price (the highest closing price of each asset). This can be explained easily when thinking that, if all the prices of an asset are normalized by the same price, the NN would see higher normalized prices if the session prices grow along

[14]There is one closing highest price per asset.

a long-term period, and vice versa when prices decrease. Figure 15 shows clearly that, while when the prices are normalized by the opening prices of each day the NN is not able to change the weight benchmark (straight line with intraday peak variations, Figure 15 (b)), however, if prices of an asset are normalized by the same price, the NN is able to change the benckmark and also capture the intraday's variations (the straight line curves depending on whether prices grow or decrease, Figure 15 (c)).



(a)



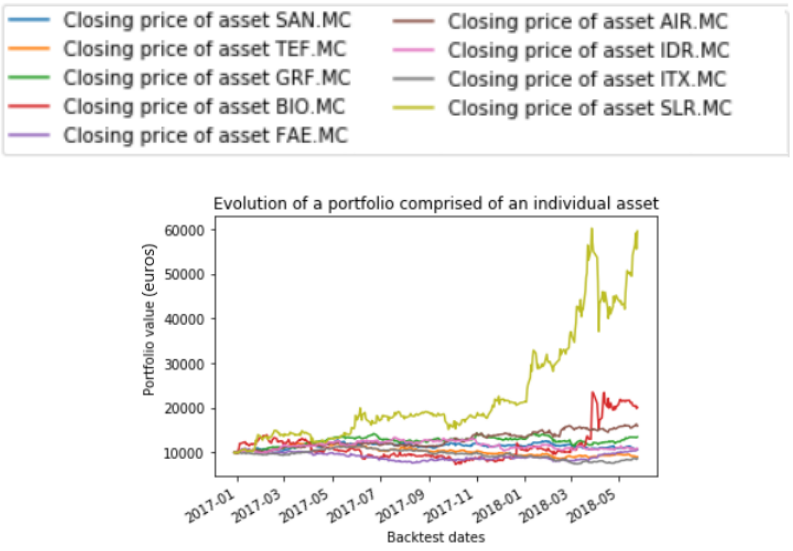
(b) (c)

Figure 15: Weight evolution for the back-test experiment in Figure 14: (a) weights before training, (b) weights of Agent 1 when prices are normalized by the opening price of each day and (c) weights of Agent 1 when prices are normalized by the highest closing price.

Besides, by analyzing the weights of the NN before training and after training with prices normalized by the opening price of each day (Figures 15 (a) and 15 (b)), it can be concluded that the training process overfits NN parameters to perform pretty good on the training set, since the only difference between both graphs are the benchmark prices computed by the NN. This also explains why Agent 2 performs worse once it has been trained than before the training process along the period from 2017/01 to 2018/05 (Figure 14 (a)). Nevertheless, there are other back-test periods where Agent 2 performs better.

Computing the portfolio evolution in case it would be comprised by a single asset for the studied back-testing period (2016/12/28-2018/05/24) shows clearly the need of a dynamic change in the portfolio weights (Figure 16). Along this back-testing period, the asset with the biggest potential of growth is SLR.MC (Solaria). This asset presents a

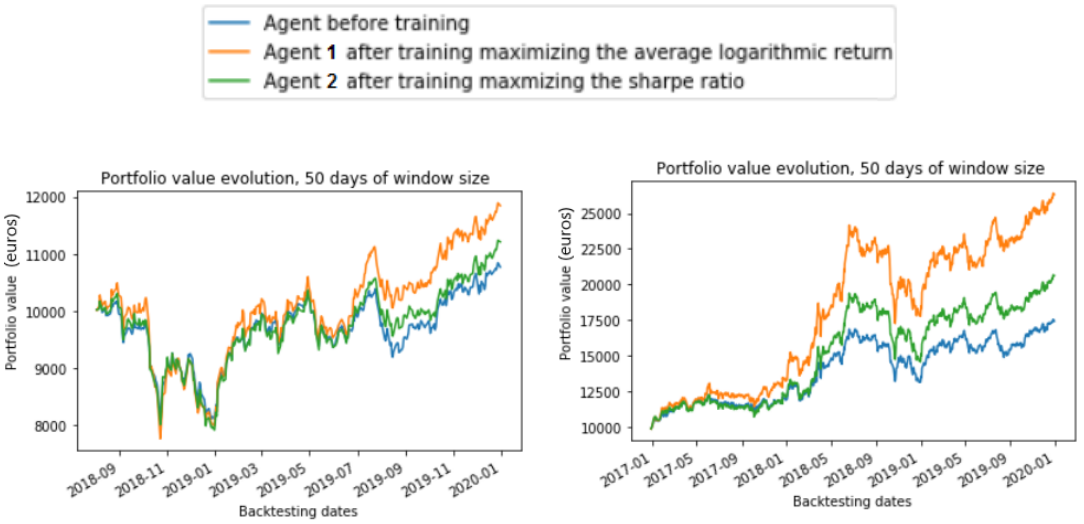
growth in its price between 2017/05 and 2017/07. The weights computed by Agent 1 in Figure 15 (c) capture this growth in the asset's price by increasing the weight on SLR.MC. The network also responds when the price of the asset is devalued, as occurs around the fourth month of 2018 (2018/03-2018/05). Again, the weights computed by Agent 1 in Figure 15 (c) capture its price devaluation by dropping its weight during around this specific period.



(a)

Figure 16: Returns of the portfolios comprised by a single asset (all the money inested in just one asset).

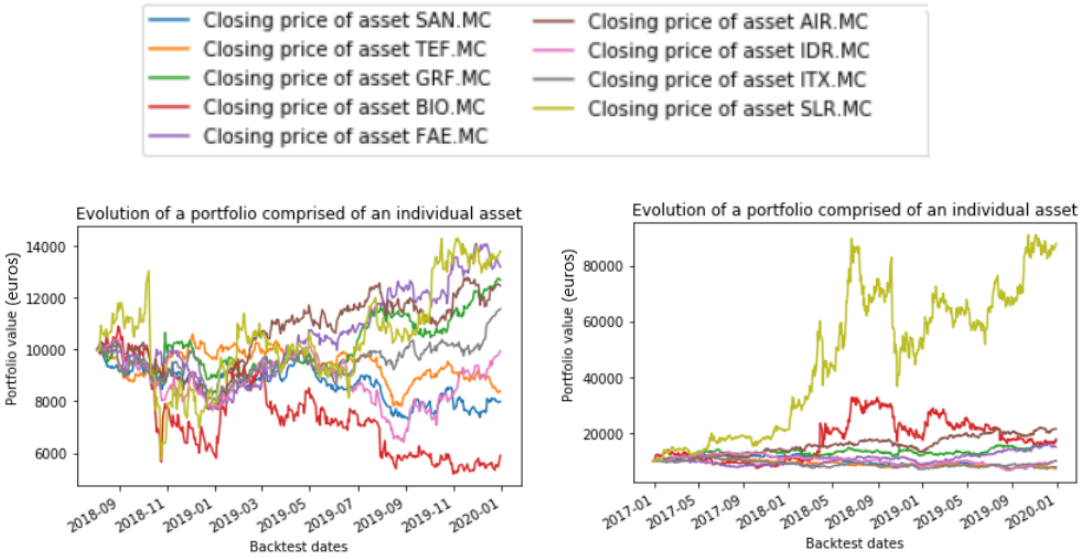
Figure 14 (b) also allows one to conclude that Agent 1 performs much better than Agent 2. Two more back-test experiments are included in Figure 17 to reinforce this conclusion.



(a) (b)

Figure 17: Back-test experiments for (a) period 2018/08/03-2019/12/30 and (b) period 2016/12/28-2019/12/30.

Indeed, Agent 1 is much better than Agent 2. Going deeper into matter, let's compute what would be the results of a portfolio consisting of a single asset (Figure 18), and the weights of the assets (Figure 19).



(a) (b)

Figure 18: Returns of the portfolios comprised by a single asset (all the money invested in just one asset) for (a) period 2018/08/03-2019/12/30 and (b) period 2016/12/28-2019/12/30.

Therefore, depending on the starting date, the portfolio evolution has a very different ending. Graphing the weights of each agent for both periods helps to understand why the performance of Agent 1 is far greater than the one for Agent 2.

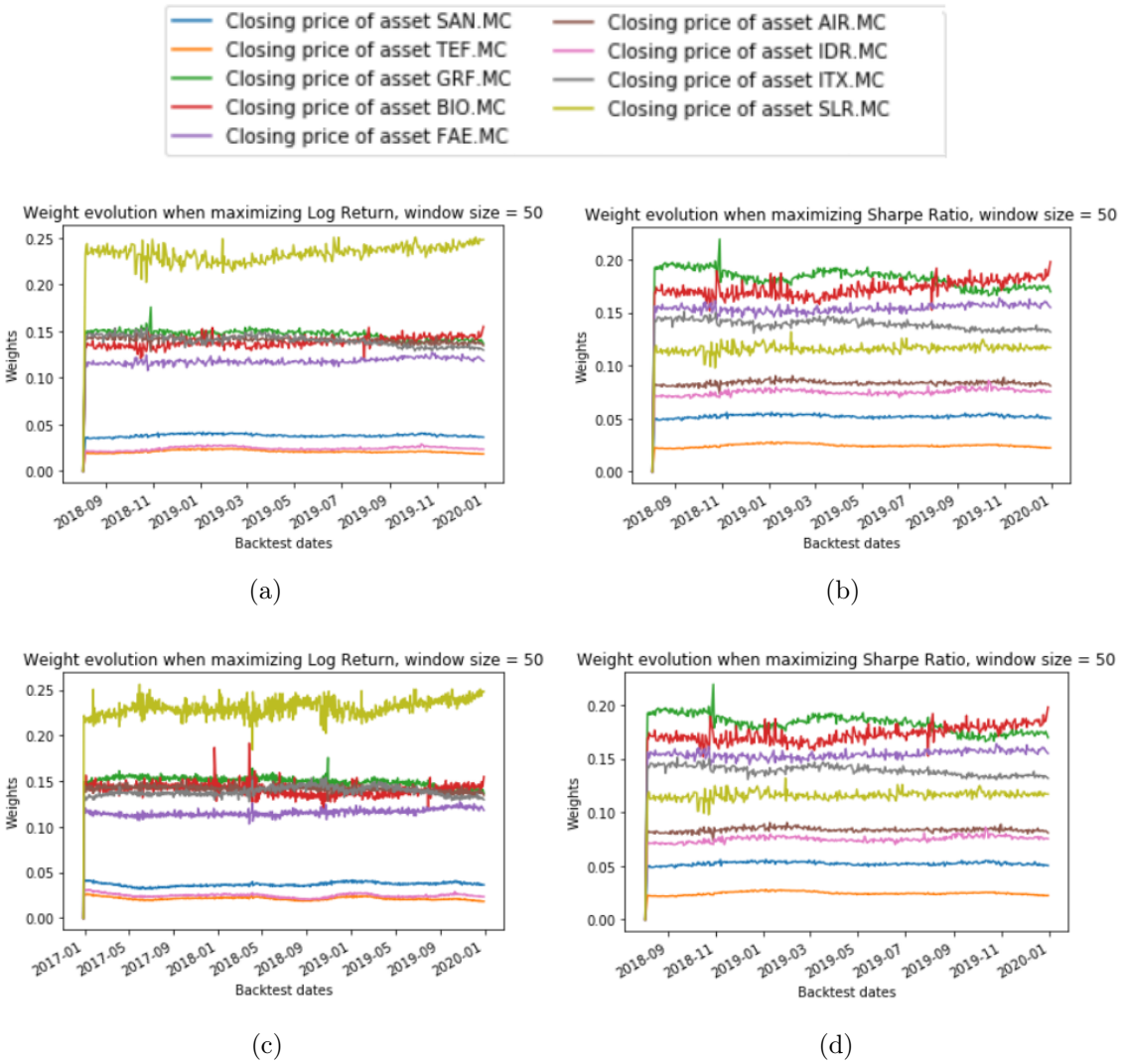


Figure 19: Weight evolution for (a) Agent 1 and period 2018/08/03-2019/12/30, (b) Agent 2 and period 2018/08/03-2019/12/30, (c) Agent 1 and period 2016/12/28-2019/12/30, and (d) Agent 2 and period 2016/12/28-2019/12/30.

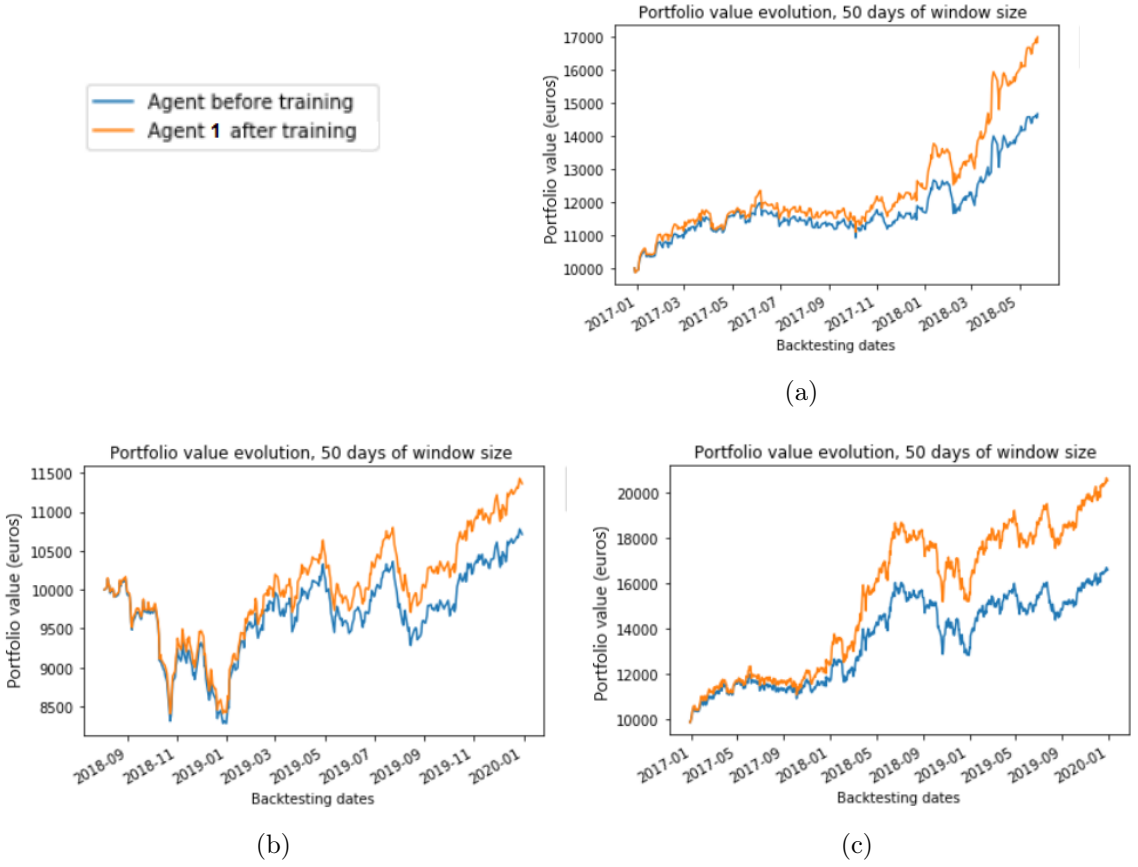
The major conclusion that can be draw from Figure 19 is that Agent 1 invests more money in assets whose potential of growth is much bigger, but also assuming a higher risk (asset SLR.MC (Solaria) is the asset with the biggest potential of growth, but also with the steepest price drops). Concretely, for period 2018/08/03-2019/12/30, Figure 19 (a) shows clearly that, when SLR.MC prices drop (2018/11, Figure 18 (a)), the weight assigned to this asset drops (around 2018/11 the curve for SLR.MC is convex in Figure 19 (a)), and the weights assigned for assets whose price does not decrease are increased a little bit (around 2018/11 the curve for TEF.MC is concave in Figure 19 (a)). However, this is not enough to avoid the devaluation of the portfolio, being necessary to increase this effect.

On the other hand, Agent 2 invests more money in secured assets, such as GRF.MC (Grifols) whose potential of growth is much smaller, but its risk is also significantly lower. This makes sense since the objective function maximized by Agent 2 is the Sharpe Ratio, which measures the excess of return per unit of risk, avoiding large falls in the portfolio

value. Specifically, analyzing the fall in the portfolio value around 2018/11 (Figure 17 (a)) it can be seen that Agent 2 tries to reduce the portfolio devaluation.

So, in order to minimize the risk of Agent 1, the idea of including the cash in the portfolio as a non risky asset is explored. This is intended for the agent to learn to assign more weight to the cash asset when prices of the other assets are falling[15]. So as to introduce the cash as a non risky asset, unlike Jiang et al. work [5] where the cash was included as a bias before the softmax layer (figure 12), the cash return[16] is introduced in the input tensor X_t for the agent to learn to assign more weight to the cash when normalized prices of the assets are smaller than the cash return. The reason for not including the cash just before the softmax layer is that, by using this approach, the agent sells all the money at the beginning and is not able to capture the variations in the assets prices.

From now on only Agent 1 is going to be considered since Agent 1 is much better than Agent 2.



(a)

(b)

(c)

Figure 20: Returns of the portfolios for Agent 1 and (a) period 2016/12/28-2018/05/24, (b) period 2018/08/03-2019/12/30, and (c) period 2016/12/28-2019/12/30.

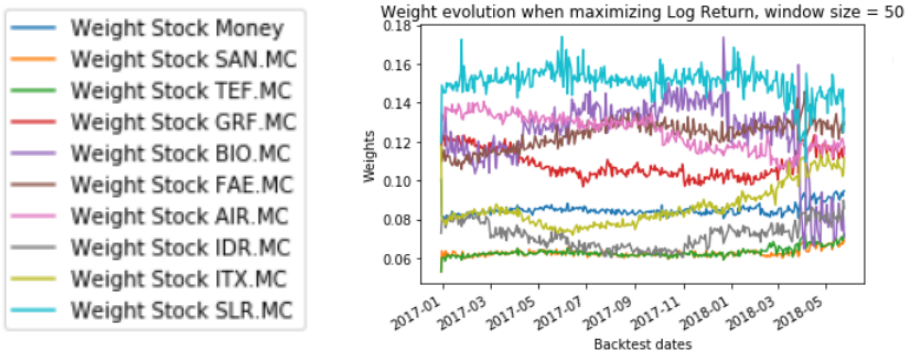
Comparing Figure 20 (b) with Figure 17 (a) it can be seen that around 2018/11 this

[15]Sell risky assets whose prices are falling and hold their cash value before they are devalued.

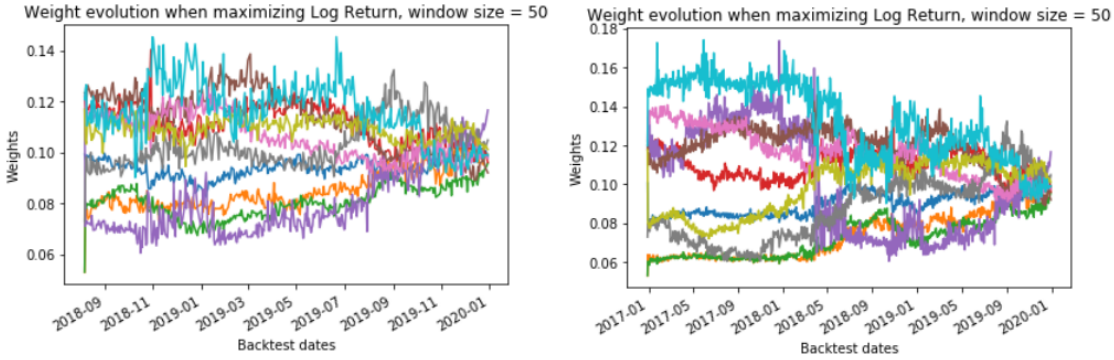
[16]Money is being assumed to be in a deposit whose return is pretty close to one (1.00008). Then, the potential of growth of the cash is equal to how much money someone would earn by leaving the money in the deposit.

new approach reduces a little bit the fall in the portfolio value. However, it does not prevent it from falling, and the final portfolio value is somewhat reduced compared to the approach where the cash was not included[17].

When studying the evolution of the weights (Figure 21) the conclusion obtained is that now, Agent 1 is able to decide higher weight variations. This seems pretty obvious when comparing Figures 19 (a) and 19 (b) with Figures 21 (b) and 19 (c).



(a)



(b) (c)

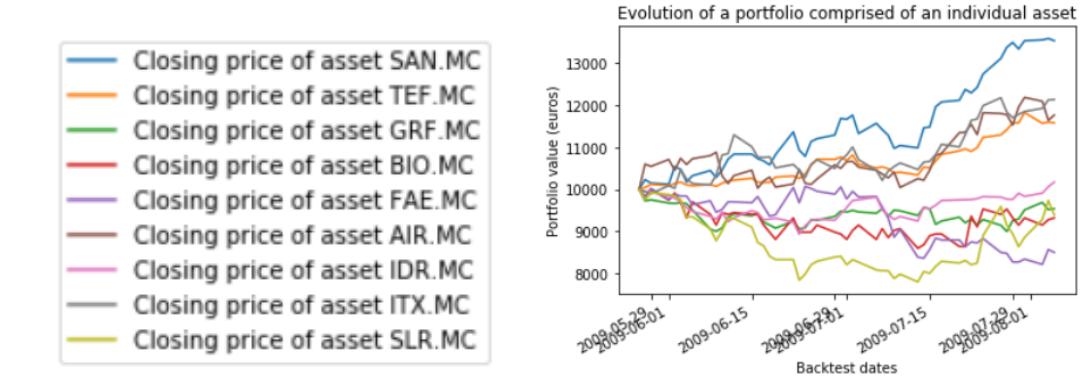
Figure 21: Weight evolution for (a) Agent 1 and period 2016/12/28-2018/05/24, (b) Agent 1 and period 2018/08/03-2019/12/30, and (c) Agent 1 and period 2016/12/28-2019/12/30.

5.3 Results for Trading Period 2

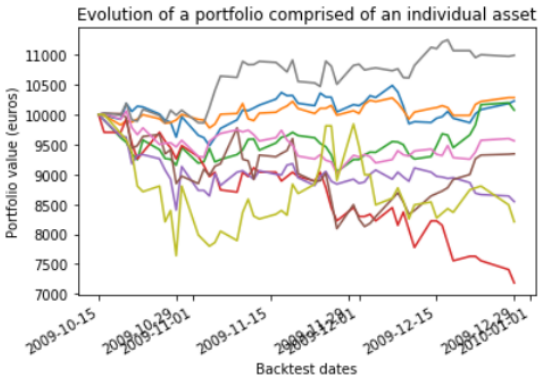
This section studies the adaptability of the trained agent to different trading periods for two models: the model that does not include the cash as an asset namely Model 1, and the one that does namely Model 2. Moreover, it is important to be aware of the fact that all the experiments in this section have been carried out with the Agent 1.

The new trading periods considering a window length of 50 days are 2009/05/27-2009/08/05, 2009/10/15-2009/12/29, and 2009/05/27-2009/12/29. The evolution of portfolios made up of a single-asset for these new periods is included below, which is useful for assessing the performance of agents.

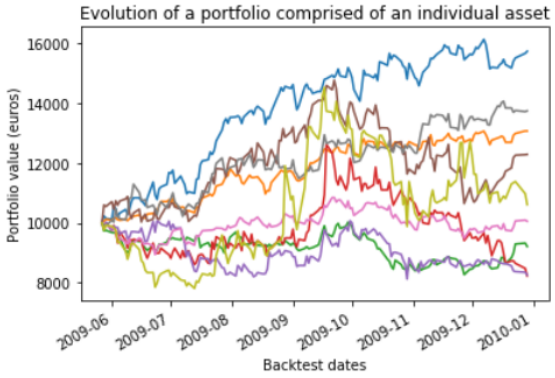
[17]When introducing the cash, the agent is assigning a weight to the cash whose return is much smaller than the return of any other asset when its price is not decreasing.



(a)



(b)

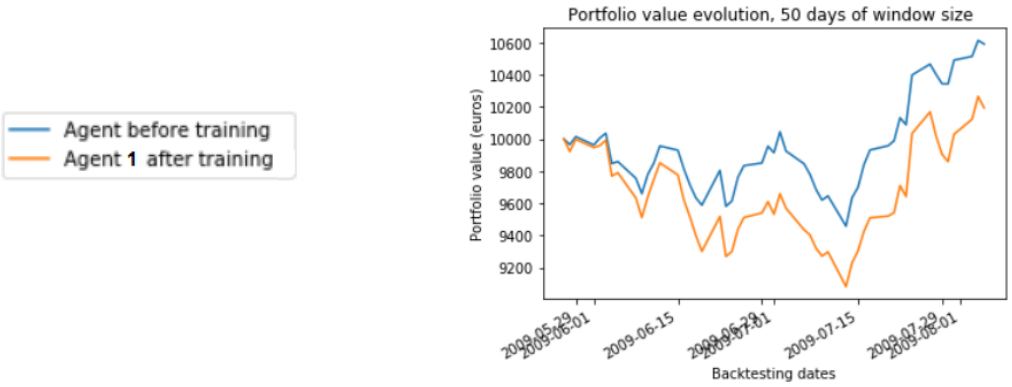


(c)

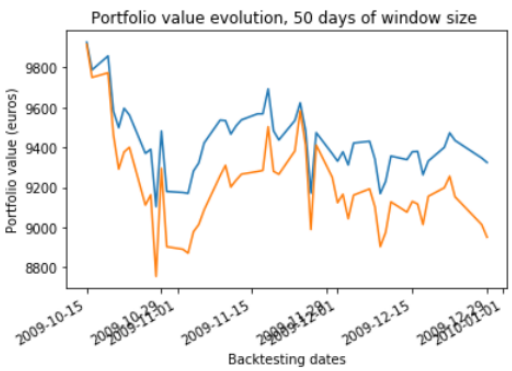
Figure 22: Returns of the portfolios comprised by a single asset (all the money invested in just one asset) for (a) period 2009/05/27-2009/08/05, (b) period 2009/10/15-2009/12/29, and (c) period 2009/05/27-2009/12/29.

5.3.1 Model 1

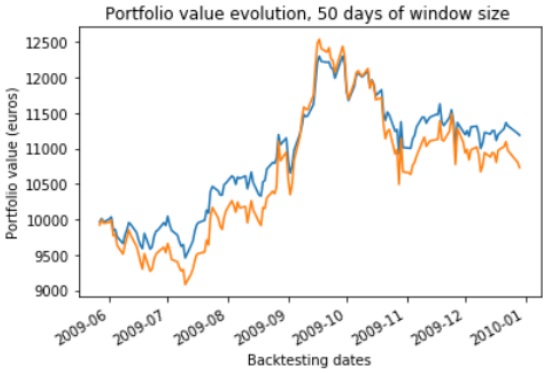
This section is focused on the model that does not include the cash as an asset. Figure 23 shows the investing decisions made by the Agent 1 which has been trained on period 2012/01/02-2016/12/30. The agent's performance is worse than the performance of an agent which has never been trained, which leads to thinking that the parameters are overfitted.



(a)



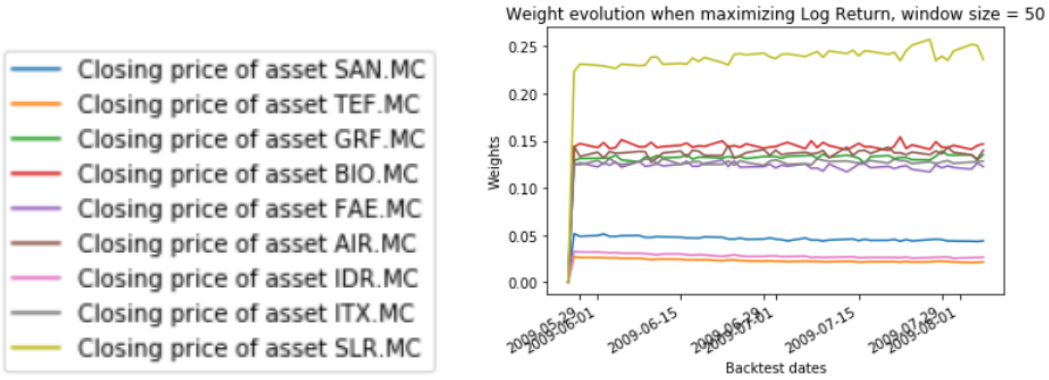
(b)



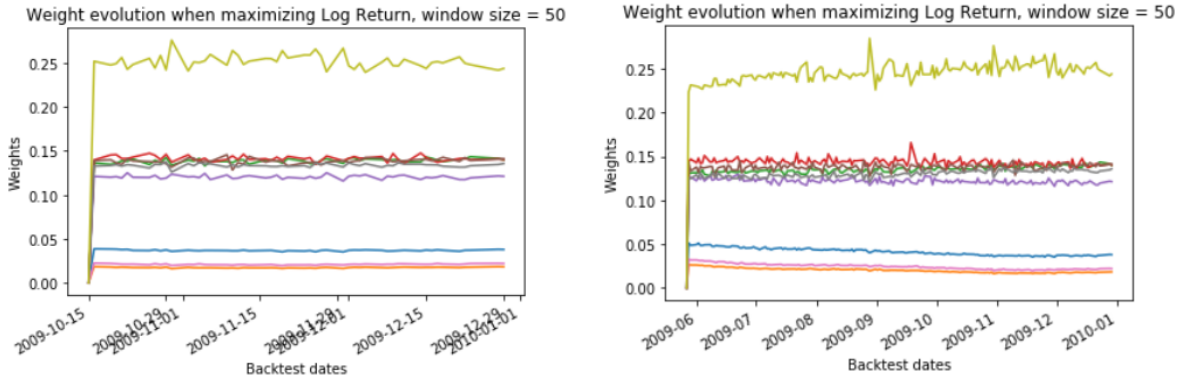
(c)

Figure 23: Returns of the portfolios for Agent 1 and (a) period 2009/05/27-2009/08/05, (b) period 2009/10/15-2009/12/29, and (c) period 2016/12/28-2019/12/30.

The possibility of overfitting is analyzed by comparing the weight evolution throughout these new periods (Figure 24) with the one belonging to trading period 1 (Figure 15 (c), Figure 19 (a), and Figure 19 (c)). For the three back-test periods of trading period 1, the asset with the biggest potential of growth was SLR.MC (Figures 16 and 18), so it was the one that had assigned a greater weight in the portfolio. However, in these three new back-test periods, asset SLR.MC has very little potential of growth (Figure 22), and therefore its weight should decrease so that the portfolio does not lose money. Nevertheless, Figure 24 shows that this is not the case, and the agent still invests more money SLR.MC asset. Thus, the conclusion is that the agent's parameters are overfitted to training period 2012/01/10-2016/10/18.



(a)

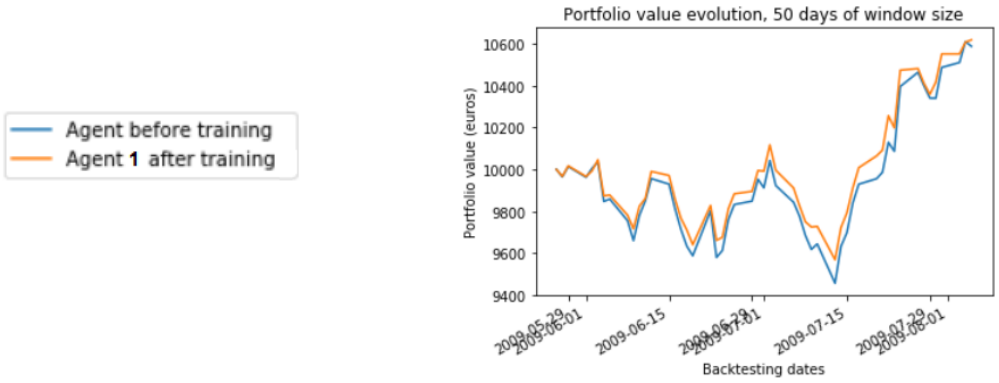


(b) (c)

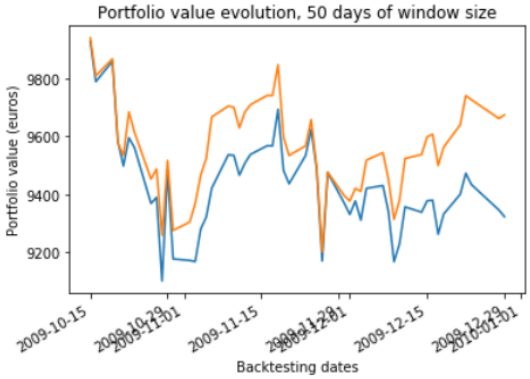
Figure 24: Weight evolution for (a) Agent 1 and period 2009/05/27-2009/08/05, (b) Agent 1 and period 2009/10/15-2009/12/29, and (c) Agent 1 and period 2009/05/27-2009/12/29.

So as to improve the performance of the agent, it is retrained on prices from period 2008/01/01-2009/03/12 using the weights computed in the last training period as the starting weights. The purpose is for the agent to explore very different market states being able to learn from them and avoid overfitting. Once the agent has been retrained all the back-test periods (including the ones from the Trading Period 1) are studied again so as to assess its advances.

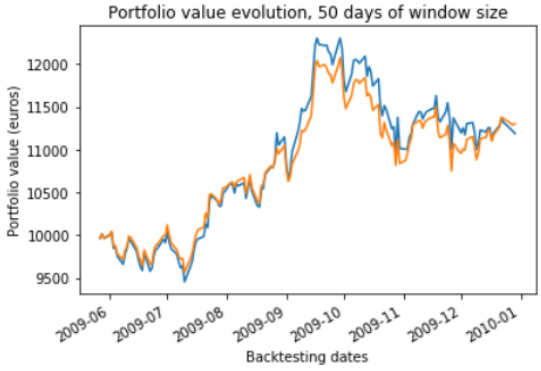
Figure 25 shows some improvements regarding to figure 23, however, for back-testing period 2009/10/15-2009/12/29 the agent still does not manage to avoid loosing money, and for periods 2009/05/27-2009/08/05, and 2016/12/28-2019/12/30 its execution is almost the same as that of the agent who has never been trained.



(a)



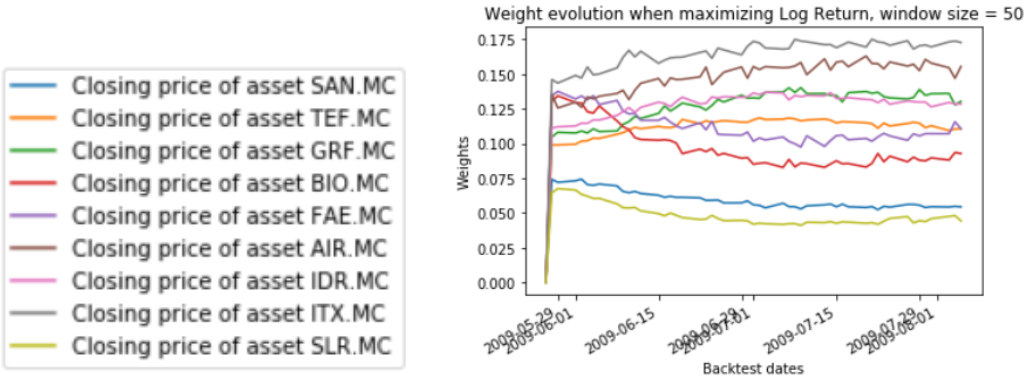
(b)



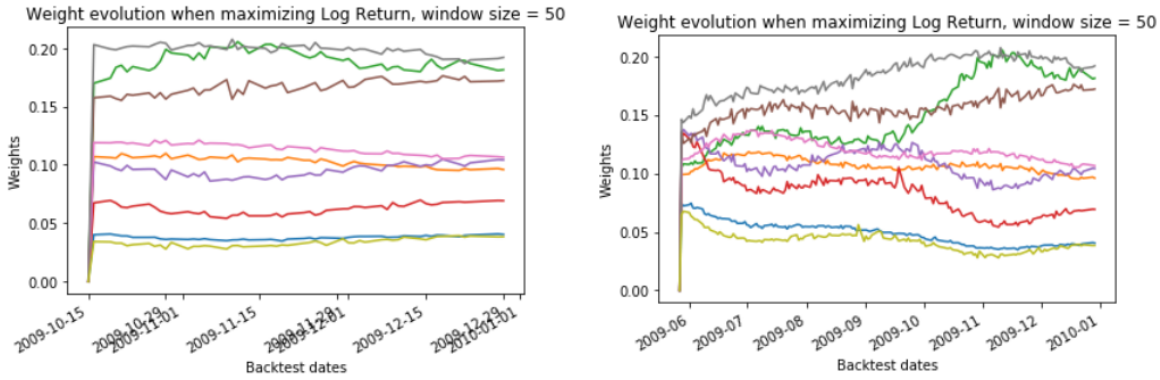
(c)

Figure 25: Returns of the portfolios for Agent 1 and (a) period 2009/05/27-2009/08/05, (b) period 2009/10/15-2009/12/29, and (c) period 2016/12/28-2019/12/30.

Studying the new computed weights (Figure 26), and comparing them with the ones from Figure 24 the conclusion that can be draw is that now, the trading agent assigned a bigger weight to some growing assets, such as ITX.MC (Inditex), but not to all of them, since the weight assigned to asset SAN.MC (Santander) is quite small, and it presents one of the biggest growth potentials for these periods (Figure 22). Therefore, the main conclusion is that again, the agent‘s parameters have been overfitted when training in the new trading period.



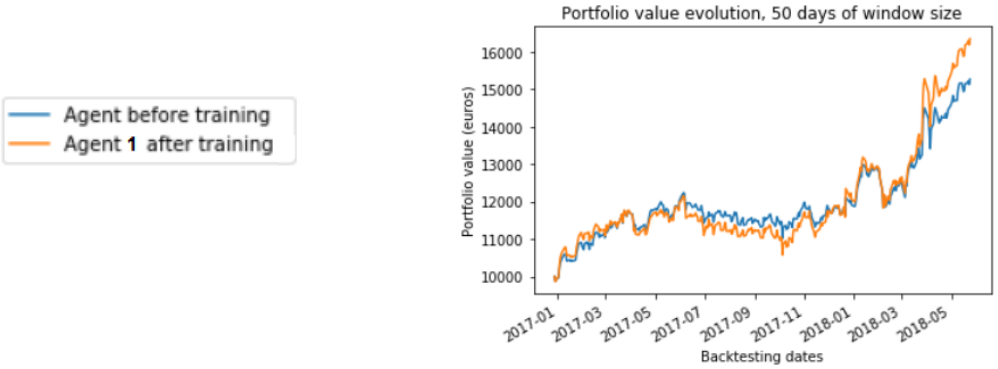
(a)



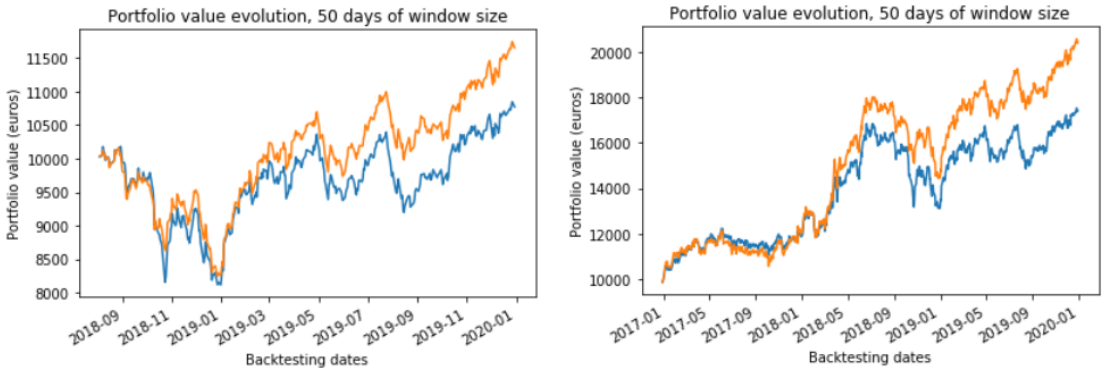
(b) (c)

Figure 26: Weight evolution for (a) Agent 1 and period 2009/05/27-2009/08/05, (b) Agent 1 and period 2009/10/15-2009/12/29, and (c) Agent 1 and period 2009/05/27-2009/12/29.

Finally, the retrained agent is tested on dates spanning from 2016/01 to 2020/01.



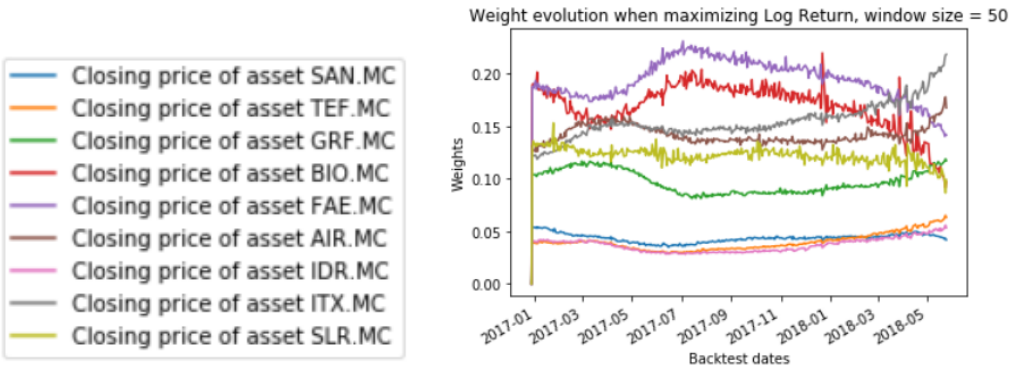
(a)



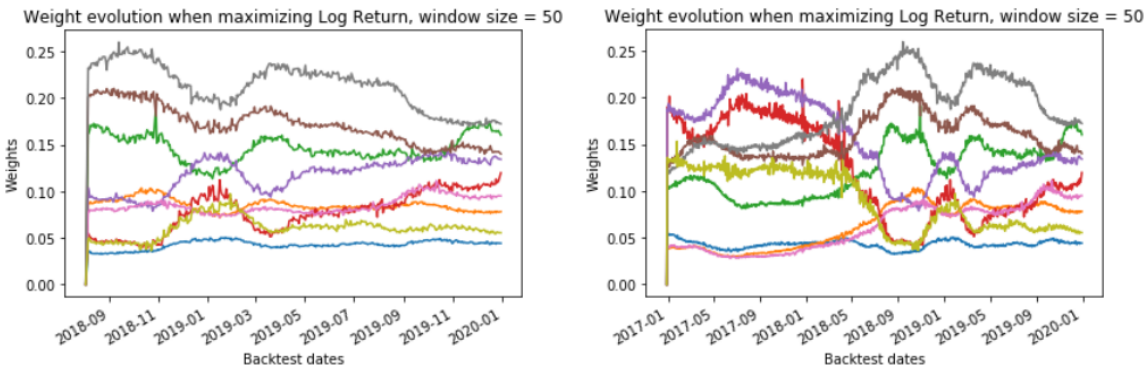
(b) (c)

Figure 27: Returns of the portfolios for Agent 1 and (a) period 2016/12/28-2018/05/24, (b) period 2018/08/03-2019/12/30, and (c) period 2016/12/28-2019/12/30.

The returns of the portfolios for the back-test periods from Figure 27 (a) and 27 (c) are worse than the ones from Figure 15 (c) and 19 (c), which correspond to the agent before being retrained. However, their returns are similar for back-testing period 2018/08/03-2019/12/30 (Figure 27 (b) and Figure 19 (a)). This is easily explained by analyzing the the evolution of the weights and the behaviour of a single-asset portfolio for this back-test periods (Figures 16 and 18). Concretely, for periods 2016/12/28-2018/05/24 and 2016/12/28-2019/12/30, the asset with the biggest potential of growth is SLR.MC, asset which during the training period is slightly increasing with some growth peaks, and therefore the agent could be overfitting. Nevertheless, for period 2018/08/03-2019/12/30, this asset loses money around 2018/11 (Figure 18 (a)) increasing its price right after. This loss it hardly captured by the agent before being retrained (Figure 19 (a)), but, once it has been retrained, the weight for this asset is quite small, increasing a bit immediately after this date (Figure 28). Then, although after the re-training process the agent's execution is not good, it is more dynamic than before, being able to adapt better to the changes in the prices of the assets. Furthermore, after the re-training, the agent is able to change the weight composition of the portfolio, as proven in Figure 26, where its weight pattern differs a lot from the one in Figure 28), ruling out the overfitting possibility.



(a)

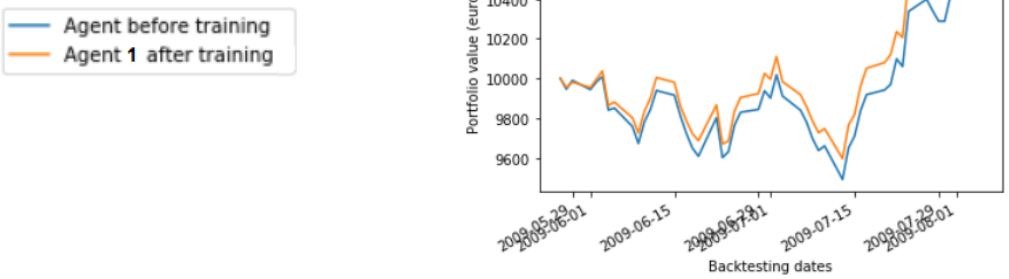


(b) (c)

Figure 28: Weight evolution for (a) Agent 1 and period 2016/12/28-2018/05/24, (b) Agent 1 and period 2018/08/03-2019/12/30, and (c) Agent 1 and period 2016/12/28-2019/12/30.

5.3.2 Model 2

As mentioned previously, this section is focused on the model that includes the cash as an asset. Figure 29 shows the investing decisions made by an agent which has been trained on period 2012/01/02-2016/12/30. Although the evolution of the portfolios for these new back-test periods differs a lot from the evolution for the first three back-test periods (Figures 16, 18 and 22), this agent manages to do better than an agent which has never been trained. Nevertheless, this does not prevent the agent from losing money in Figure 29 (b).

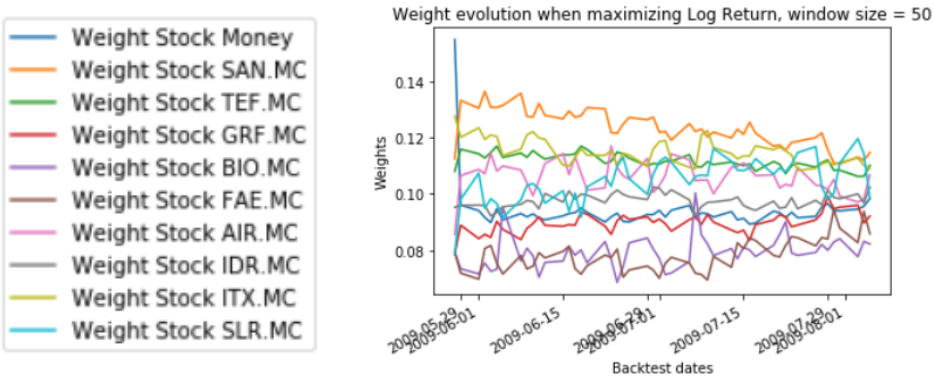


(a)

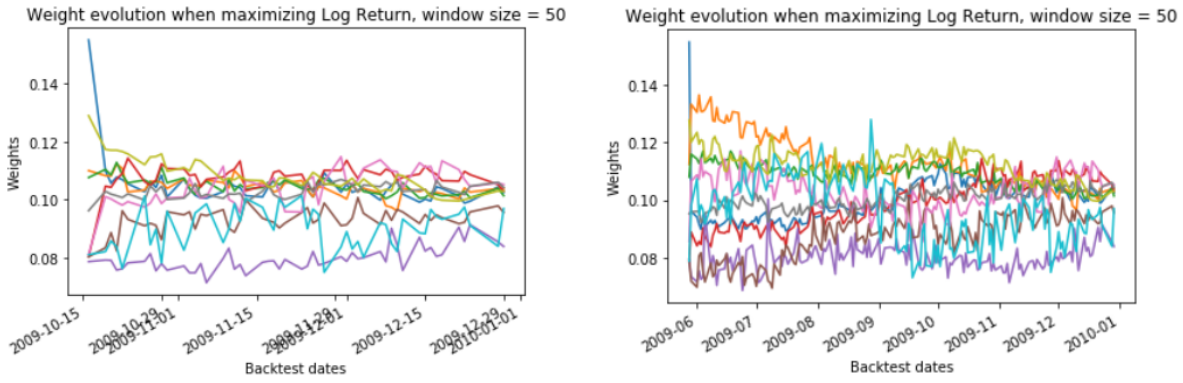
(b) (c)

Figure 29: Returns of the portfolios for Agent 1 and (a) period 2009/05/27-2009/08/05, (b) period 2009/10/15-2009/12/29, and (c) period 2016/12/28-2019/12/30.

The possibility of overfitting is ruled out by comparing the weight evolution throughout these new periods (Figure 30) with the one belonging to trading period 1 (Figure 15 (c), Figure 19 (a), and Figure 19 (c)). Specifically, in the three back-test periods of trading period 1, the asset with the biggest potential of growth was SLR.MC (Figures 16 and 18), so it was the one that had assigned a greater weight in the portfolio. However, in these three new back-test periods, asset SLR.MC has very little potential of growth (Figure 22), and therefore its weight is now exceeded by other assets whose growth potential, and therefore its weight, was for trading period 1 very small (SAN.MC). The agent can also be consider quite dynamic since, despite of SLR.MC losses throughout most of the dates of these 3 new back-test periods, it manages to capture its peaks of growth (i.e. 2009/09 in Figure 22 light green, and 2009/09 in Figure 30 light blue). Even so, the agent's performance cannot be considered as good.



(a)

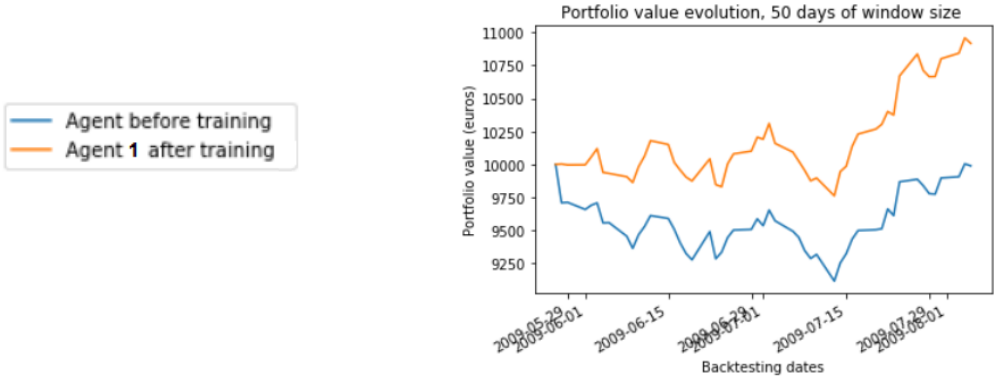


(b) (c)

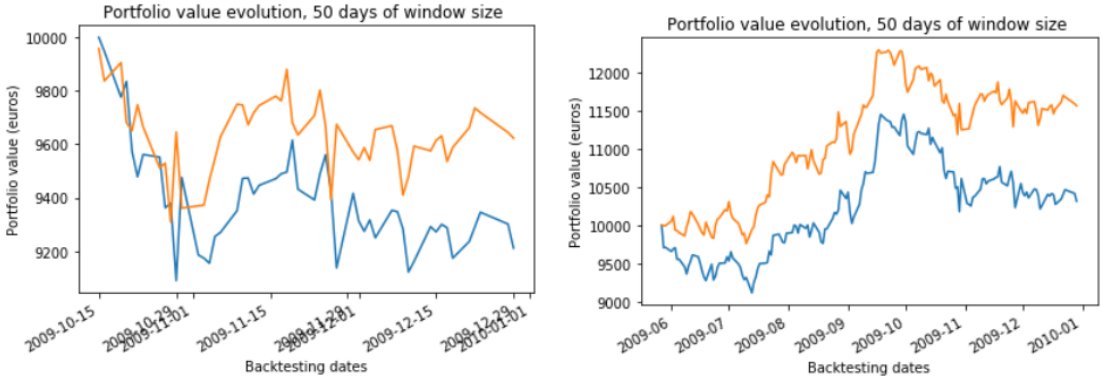
Figure 30: Weight evolution for (a) Agent 1 and period 2009/05/27-2009/08/05, (b) Agent 1 and period 2009/10/15-2009/12/29, and (c) Agent 1 and period 2009/05/27-2009/12/29.

So as to improve the execution of the agent, it is retrained on prices from period 2008/01/01-2009/03/12 using the weights computed in the last training period as the starting weights. The purpose is for the agent to explore very different market states being able to learn from them. Once the agent has been retrained all the back-test periods (including the ones from the Trading Period 1) are studied again so as to assess its advances.

Figure 31 shows some improvements regarding to Figure 29, however, for back-testing period 2009/10/15-2009/12/29 the agent still does not manage to avoid loosing money.



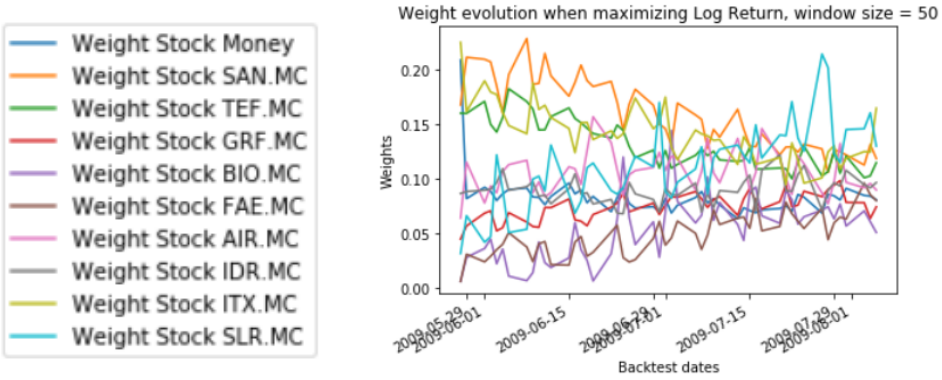
(a)



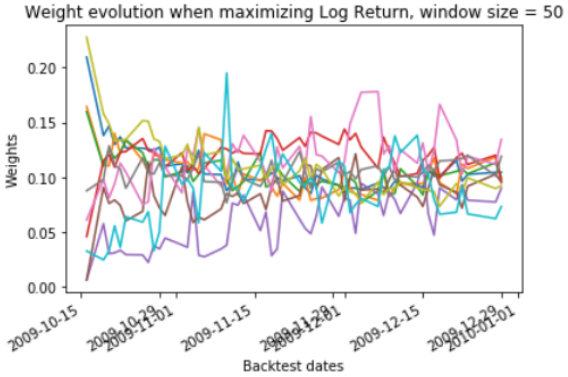
(b) (c)

Figure 31: Returns of the portfolios for Agent 1 and (a) period 2009/05/27-2009/08/05, (b) period 2009/10/15-2009/12/29, and (c) period 2016/12/28-2019/12/30.

The weights computed for the trading agent during this three periods help to understand the improvements on the final portfolio value. Comparing the graphs from Figure 32 with the ones from Figure 30 the main conclusion is that now, trading agent assigned a bigger weight to the growing assets, increasing the amount of money gained, but is not able to avoid losses because the patterns of graph 30 and graph 32 are very similar. This explains the similarities in the portfolio return patterns before and after the agent has been trained.



(a)

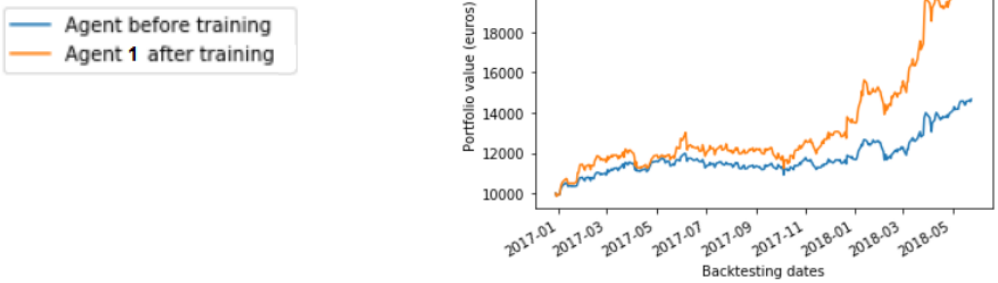


(b)

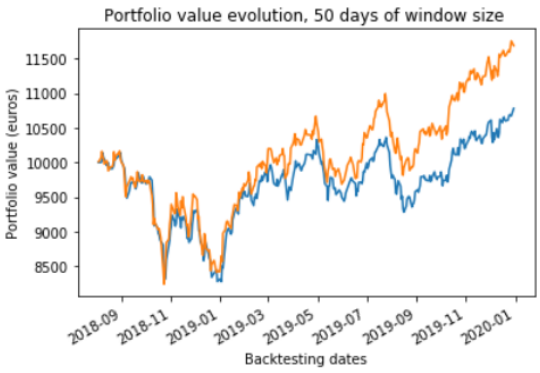
(c)

Figure 32: Weight evolution for (a) Agent 1 and period 2009/05/27-2009/08/05, (b) Agent 1 and period 2009/10/15-2009/12/29, and (c) Agent 1 and period 2009/05/27-2009/12/29.

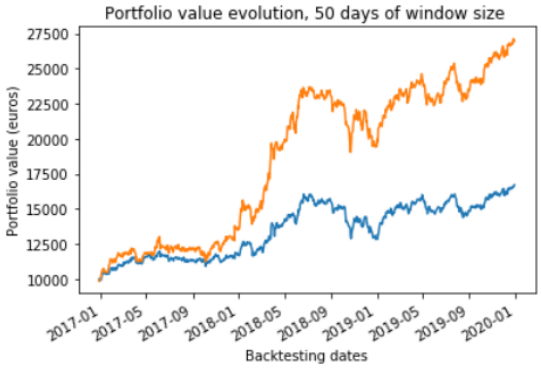
Finally, the retrained agent is tested on dates spanning from 2016/01 to 2020/01.



(a)



(b)

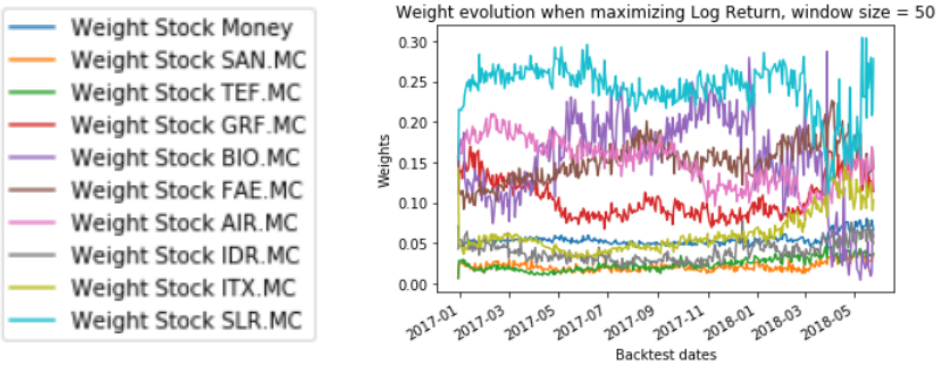


(c)

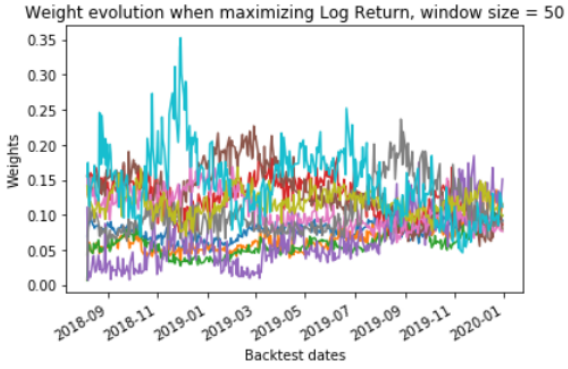
Figure 33: Returns of the portfolios for Agent 1 and (a) period 2016/12/28-2018/05/24, (b) period 2018/08/03-2019/12/30, and (c) period 2016/12/28-2019/12/30.

Clearly, the retrained agent (Figure 33) outweighs its performance from before being retrained (Figure 20). Again, to understand the growth in the portfolio value, an analysis

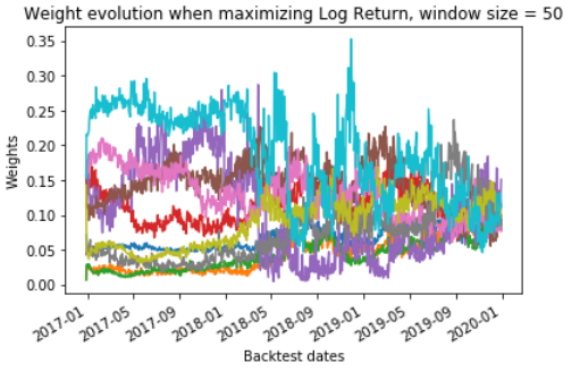
of the evolution of the weights before (Figure 21) and after (Figure 34) being retrained is necessary. By comparison, once the agent has been retrained, its trading strategy becomes more dynamic, being able to adapt to both long-term and short-term variations. Concretely, Figure 34 (c) shows that the retrained agent is able to adjust better to the ups and downs of the price of asset SLR.MC between 2018/07 and 2019/01 (Figure 18 (b)) than the same agent before being retrained (Figure 20 (c)).



(a)



(b)



(c)

Figure 34: Weight evolution for (a) Agent 1 and period 2016/12/28-2018/05/24, (b) Agent 1 and period 2018/08/03-2019/12/30, and (c) Agent 1 and period 2016/12/28-2019/12/30.

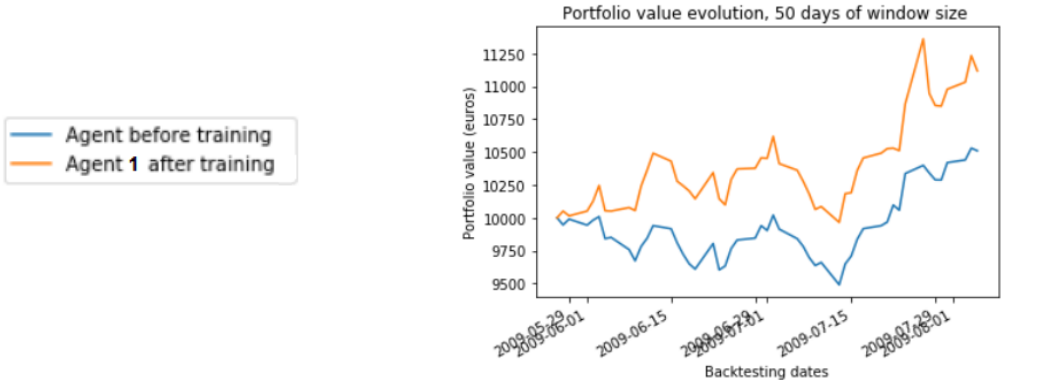
5.3.3 Model 2, retrained 50 epochs

Finally, since Model 2 presents a quite significant growth potential, it is going to be retrained in another 30 epochs of 2008/01/01-2009/03/12 data. The first training was carried out on 50 epochs of the 2012/01/01-2016/10/18 period. However, the re-training was carried out only in 20 epochs, since it is a much smaller period. Nevertheless, it does not matter that the agent sees repeated data since, the process of training and updating network parameters is carried out in batches, maximizing the objective function for each batch, and as the index in which it begins to collect the samples of the batch is random, and the portfolio value depends a lot on when the agent starts investing, the overfitting probability is low.

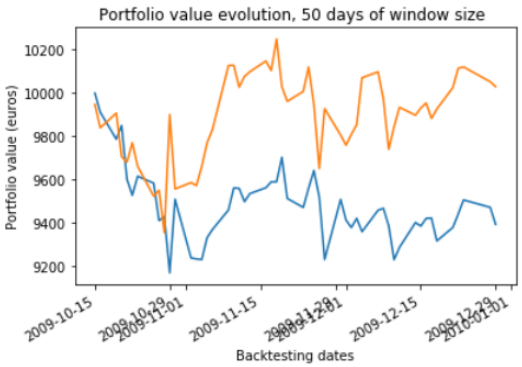
The new retrained agent will be tested on the periods 2009/05/27-2009/08/05, 2009/10/15-2009/12/29, 2009/05/27-2009/12/29, and on the periods 2016/12/28-2018/05/24, 2018/08/03-2019/12/30, and 2016/12/28-2019/12/30. The results obtained in each of these tests are presented below.

1. Testing on periods 2009/05/27-2009/08/05, 2009/10/15-2009/12/29, 2009/05/27-2009/12/29:

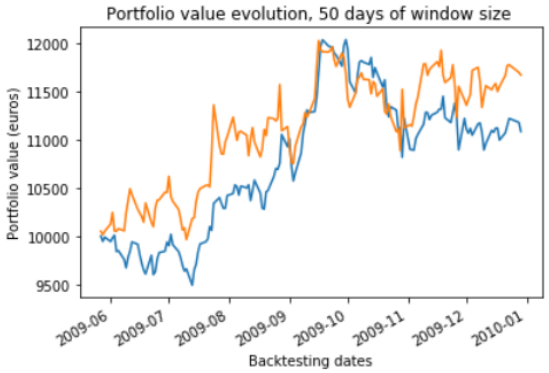
Figure 35 shows the results obtained by the retrained agent on the periods 2009/05/27-2009/08/05, 2009/10/15-2009/12/29, 2009/05/27-2009/12/29.



(a)



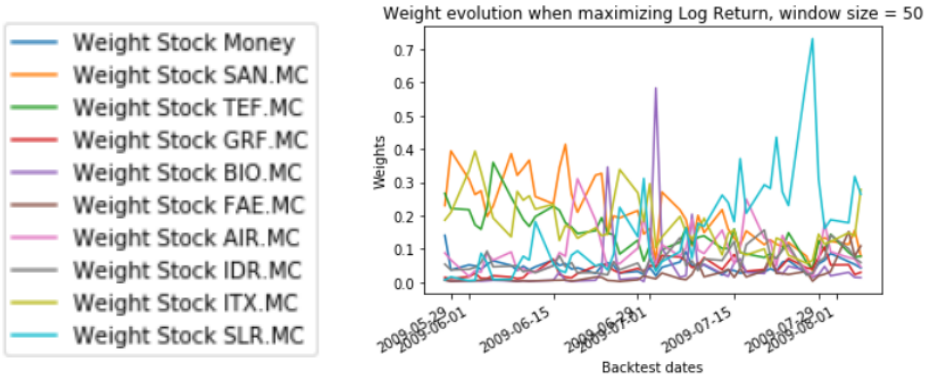
(b)



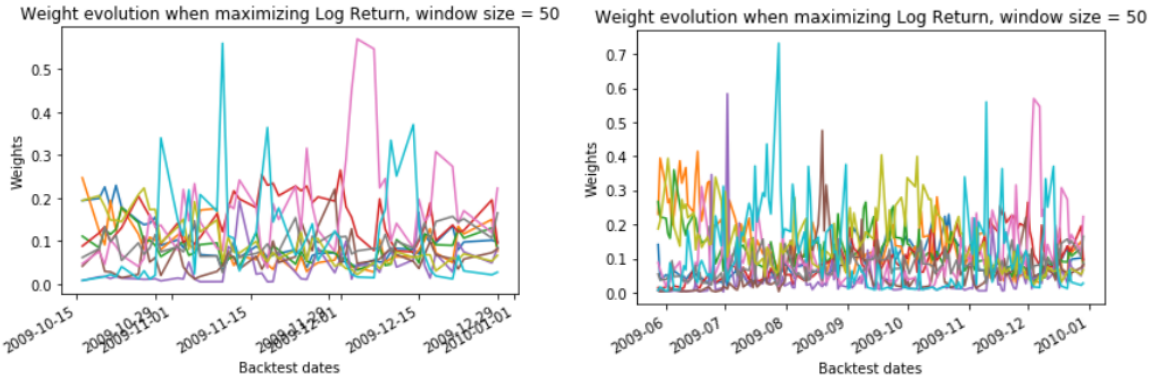
(c)

Figure 35: Returns of the portfolios for Agent 1 and (a) period 2009/05/27-2009/08/05, (b) period 2009/10/15-2009/12/29, and (c) period 2016/12/28-2019/12/30.

Figure 35 shows an improvement from Figure 29, since now, the agent manages to not lose money along period 2009/10/15-2009/12/29. Comparing the weight evolution from Figure 32 (b) with the one from Figure 36 (b) the conclusion for this improvement lies in that now, the agent gives more importance to the growth peaks of the assets.



(a)

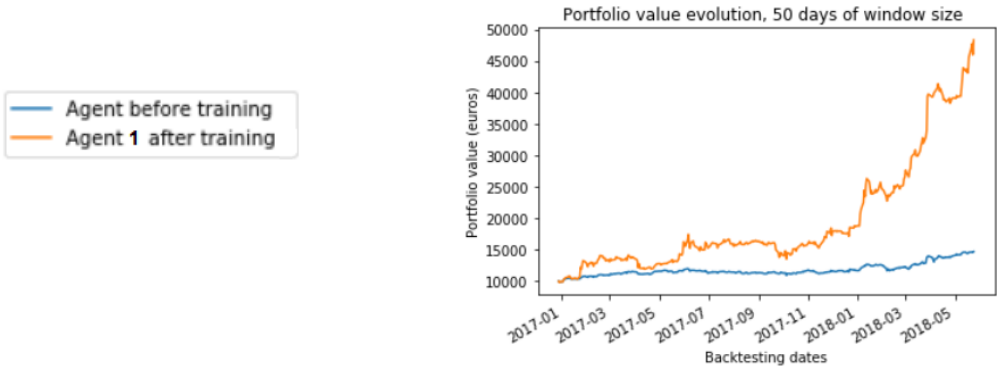


(b) (c)

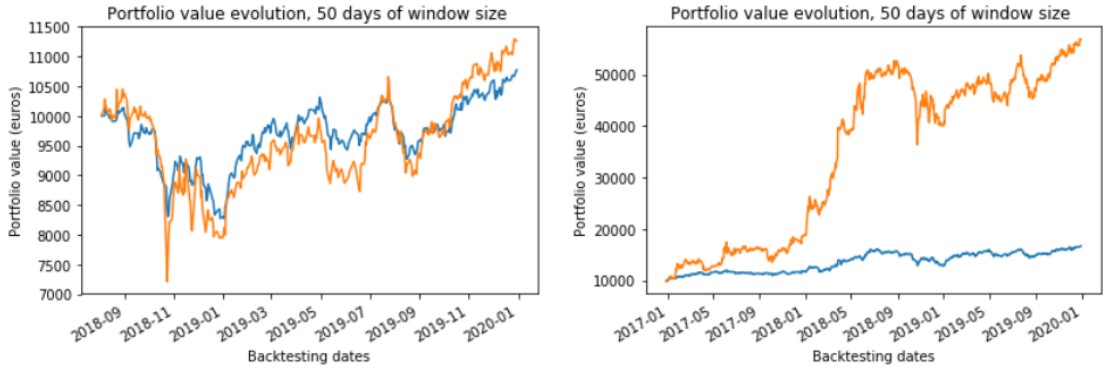
Figure 36: Weight evolution for (a) Agent 1 and period 2009/05/27-2009/08/05, (b) Agent 1 and period 2009/10/15-2009/12/29, and (c) Agent 1 and period 2009/05/27-2009/12/29.

2. Testing on periods 2016/12/28-2018/05/24, 2018/08/03-2019/12/30, and 2016/12/28-2019/12/30:

Figure 37 shows the results obtained by the retrained agent on the periods 2016/12/28-2018/05/24, 2018/08/03-2019/12/30, and 2016/12/28-2019/12/30.



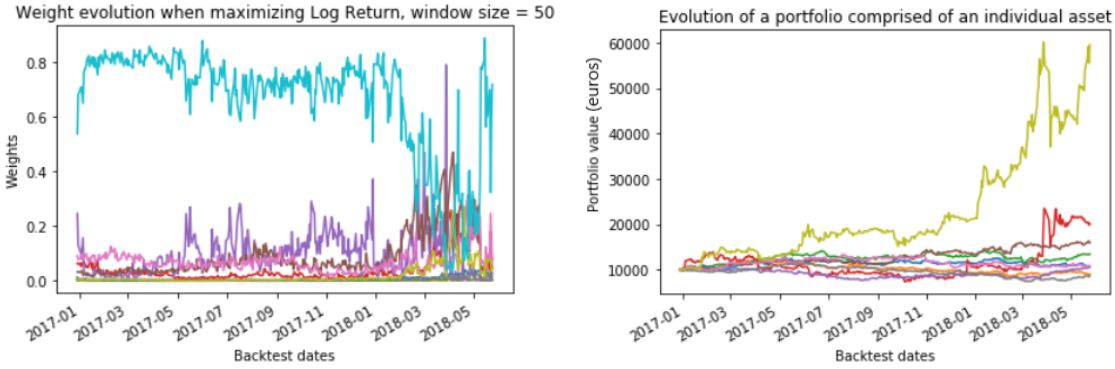
(a)



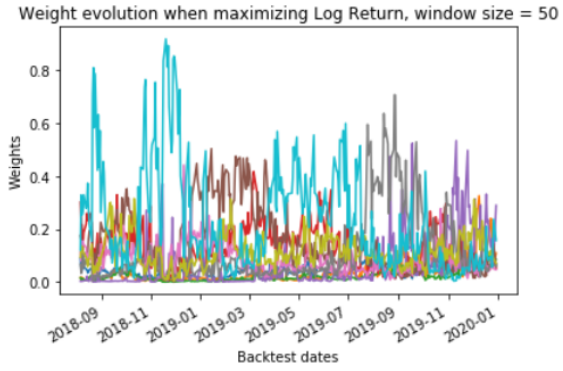
(b) (c)

Figure 37: Returns of the portfolios for Agent 1 and (a) period 2016/12/28-2018/05/24, (b) period 2018/08/03-2019/12/30, and (c) period 2016/12/28-2019/12/30.

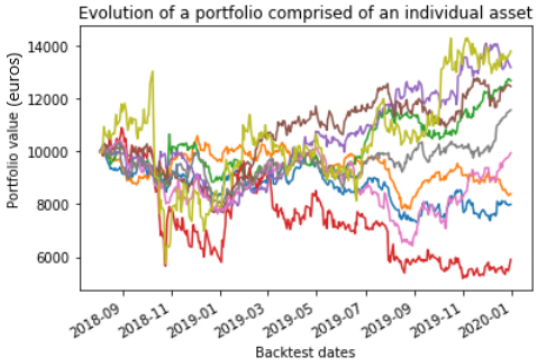
Clearly, by retraining the agent another 30 epochs, its execution improves dramatically for periods 2016/12/28-2018/05/24, and period 2016/12/28-2019/12/30, but, instead, for period 2018/08/03-2019/12/30 its performance cannot be considered as good. The reason is that now the agent is able to increase the weights of the assets whose potential of growth is much bigger, investing most of the money on them, but it is not dynamic enough. Let's compute simultaneously the weight evolution and the evolution of the single-asset portfolios. Figure 38 shows on the first column the graphs corresponding to the weight evolution and on the second column the evolution of the single-asset portfolios. Figure 39 shows the legend for each of these columns.



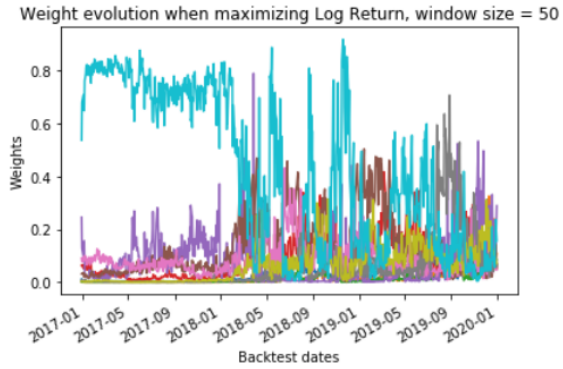
(a) (b)



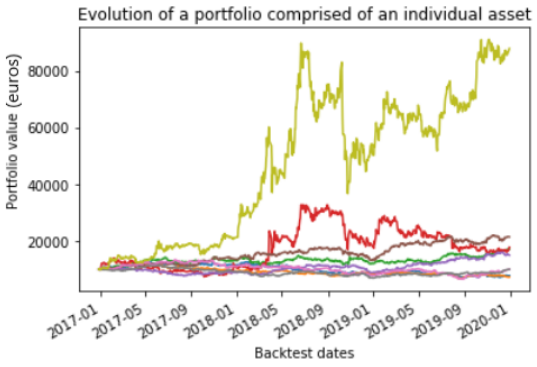
(c)



(d)



(e)



(f)

Figure 38: (a) Weight evolution of Agent 1 and period 2016/12/28-2018/05/24, (b) Evolution of single-asset portfolios along period 2016/12/28-2018/05/24, (c) Weight evolution of Agent 1 and period 2018/08/03-2019/12/30, (d) Evolution of single-asset portfolios along period 2018/08/03-2019/12/30, (e) Weight evolution of Agent 1 and period 2016/12/28-2019/12/30, and (f) Evolution of single-asset portfolios along period 2016/12/28-2019/12/30.

Figure 39: Asset legend for (a) the weight evolution, and (b) the single-asset portfolios.

The conclusion drawn from Figure 38 is that now, the agent is increasing the weights of the profitable assets following the falls and growths in the prices of the assets. This is

clear when assessing asset SLR.MC, and explains why suddenly the portfolio value raised highly. However, for period 2018/08/03-2019/12/30, the agent decision of reducing the weight of asset SLR.MC comes too late, assuming significant losses around 2018/11 in Figure 37 (b)).

6 Conclusions

This master thesis has been focused to solve the portfolio management problem through Deep RL techniques. The use of these techniques as a mechanism to create a trading agent with the purpose of rearranging capital into a number of assets allows the extraction of the following conclusions:

- The portfolio management problem is inherently a sequential decision problem aiming to maximize long-term return. In this context, the agent should be willing to make decisions to sacrifice *short-term* return if that means greater *long-term* return. Therefore, this problem can be better modeled as an MDP, and solved by RL techniques and, in particular, by Deep RL.
- The Convolutional Neural Network has been postulated as an architecture capable of solving this problem, and not only games (e.g., Atari) where an image is required as the input of the network. Nonetheless, a careful implementation needs to be carried out since they can suffer from the vanishing gradient problem which prevents the NN fro being trained. Concretely, the best approaches are the use of the ReLu activation function and the use of a random initialization of the parameters proportional to the square of the variance [43].
- From a financial point of view, the best objective function for maximizing the return of the portfolio through the RL is the logarithmic return. This makes sense since the other objective function is the Sharpe Ratio, which measures the excess of return per unit of risk, avoiding large falls in the portfolio value. However, the assets with a bigger volatility are usually the ones that provides higher profits.
- In general, the trading agents assume a high risk. However, despite the agent ends up not losing money for the studied periods when they are large enough, it does follow a very similar pattern to the one of an agent that has never been trained, so a risk policy is needed so as to avoid high losses.
- Regarding to the introduction of the cash as a non risky asset, it is found that, even though it initially produces smaller returns than without the cash, the weight evolution is more dynamic, and, as shown in Section 5.3.2, it adapts better to other trading periods.
- As shown in the experiments, the policy learned by the Deep RL agent is able to adjust the weights of the assets depending on its market evolution. The asset that has benefited the most is Solaria (SLR.MC), since it presents the biggest growth potential as most of the time is monotonously increasing.
- Section 5.3.3 proves that training the agent throughout along number of epochs improves the agent performance since the portfolio value, and therefore its returns, differ a lot depending on the date the trading agent starts investing money.

Therefore, the Deep RL framework proves to be useful for the trading strategy, obtaining quite good results for some of the back-test periods. However, there is a lot of future work related to finding a risk management strategy that is dynamic enough to improve the returns of the portfolio but also to avoid excessive losses.

References

[1] Understanding The History Of The Modern Portfolio. Retrieved from https://www.investopedia.com/articles/07/portfolio-history.asp, (2020).

[2] What Is Portfolio Weight?. Retrieved from https://www.investopedia.com/terms/p/portfolio-weight.asp, (2020).

[3] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An introduction.* The MIT press, pp. 47-49, 53-55, 58-62 (1998).

[4] A Beginner's Guide to Deep Reinforcement Learning. Retrieved from https://pathmind.com/wiki/deep-reinforcement-learning, (2020).

[5] Z. Jiang, D. Xu, J. Liang, *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem* , (2017).

[6] N. Kanwar, *Deep Reinforcement Learning-based Portfolio Management*, (2019).

[7] Z. Jiang, D. Xu, J. Liang, *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem* , pp. 2 (2017).

[8] Backtesting Definition. Retrieved from https://www.investopedia.com/terms/b /backtesting.asp, (2020).

[9] B. Li, S. CH Hoi, D. Sahoo, Zhi-Yong Liu, *Moving average reversion strategy for on-line portfolio selection.*, Artificial Intelligence, pp. 104–123, (2015).

[10] G-Y Ban, N. E. El Karoui, A E. B. Lim *Machine learning and portfolio optimization.* Management Science, 64(3), pp. 1136-1154 (2018).

[11] K.P. Sycara, D. Zeng, K.Decker, *Intelligent Agents in Portfolio Management.* In: Jennings N.R., Wooldridge M.J. (eds) Agent Technology. Springer, Berlin, Heidelberg, (1998).

[12] C. Musto, G. Semeraro, P. Lops, M. de Gemmis, G. Lekkas. *Personalized finance advisory through case-based recommender systems and diversification strategies.* Decision Support Systems, (2015).

[13] H. Ince, T. B. Trafalis, *Kernel methods for short-term portfolio management.* Expert Systems with Applications, vol. 30, Issue 3, (2006).

[14] Reinforcement Learning Demystified: Markov Decision Processes (Part 1). Retrieved from https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690, (2020).

[15] F. J. García Polo, F. Fernández Rebollo, *Aprendizaje por Refuerzo para la Toma de Decisiones Segura en Dominios con Espacios de Estados y Acciones Continuos* . Universidad Carlos III de Madrid, pp. 13 (2012).

[16] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An introduction.* The MIT press, pp. 131-132 (1998).

[17] Q-learning. Retrieved from https://en.wikipedia.org/wiki/Q-learning, (2020).

[18] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An introduction*. The MIT press, pp. 321-326 (1998).

[19] N. Kanwar, *Deep Reinforcement Learning-based Portfolio Management*, pp. 13-19 (2019).

[20] Deep Learning fácil con DeepCognition. Retrieved from https://planetachatbot.com/deep-learning-fácil-con-deepcognition-9af43b2319ba, (2020).

[21] I. Goodfellow, Y. Bengio, A.n Courville, *Deep Learning*, (2016).

[22] A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, (2020).

[23] X. Wu, Z. Peng, J. Ren, C. Cheng, W. Zhang, D. Wang, *Rub-impact fault diagnosis of rotating machinery based on 1-D convolutional neural networks*, IEEE Sensors Journal, (2019).

[24] Recurrent Neural Networks. Retrieved from https://pathmind.com/wiki/lstm, (2020).

[25] Recurrent Neural Network (RNN) – What is an RNN and why should you use it?. Retrieved from https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/, (2020).

[26] A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python. Retrieved from https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/, (2020).

[27] Z. Jiang, D. Xu, J. Liang, *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem* , pp. 5-6 (2017).

[28] What is a Return in Finance?. Retrieved from https://www.investopedia.com/terms/r/return.asp, (2020).

[29] Z. Liang, H. Chen, J. Zhu, K. Jiang, Y. Li, *Adversarial Deep Reinforcement Learning in Portfolio Management*, pp. 2 (2018).

[30] J. Mira Navarro, *Introducción a las Operaciones Financieras*, pp. 56-62 (2013).

[31] Z. Jiang, D. Xu, J. Liang, *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem*, pp. 5-7 (2017).

[32] M. Ormos, A, Urbán, *Performance analysis of log-optimal portfolio strategies with transaction costs*, Quantitative Finance, pp. 4-7 (2010).

[33] J. Moody, L. Wu, Y. Liao, M. Saffell. *Performance functions and reinforcement learning for trading systems and portfolios*. Journal of Forecasting, 17(56), pp. 441–470, (1998).

[34] Z. Jiang, D. Xu, J. Liang, *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem*, pp. 8 (2017).

[35] Z. Jiang, D. Xu, J. Liang, *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem*, pp. 13-15 (2017).

[36] N. Kanwar, *Deep Reinforcement Learning-based Portfolio Management*, pp. 39 (2019).

[37] What is feature? definition and meaning. Retrieved from http://www.businessdictionary.com/definition/feature.html, (2020).

[38] Z. Jiang, D. Xu, J. Liang, *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem*, pp. 8-10 (2017).

[39] B. Givan, R. Parr, Y. Li, *An Introduction to Markov Decision Processes*, Purdue University and Duke University, pp. 3.

[40] Z. Jiang, D. Xu, J. Liang, *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem* , pp. 10-11 (2017).

[41] What Is The Sharpe Ratio?. Retrieved from https://www.investopedia.com/terms/s/sharperatio.asp, (2020).

[42] N. Kanwar, *Deep Reinforcement Learning-based Portfolio Management*, pp. 37 (2019).

[43] Andrew Ng. [Deeplearning.ai]. Weight Initialization for Deep Networks. [Video]. Retrieved from https://www.youtube.com/watch?v=s2coXdufOzE