Deep Deterministic Policy Gradient based portfolio management system

Firdaous Khemlichi
SIGER Laboratory
Sidi Mohamed Ben Abdellah University
Fez, Morocco
firdaous.khemlichi@gmail.com

Hiba Chougrad
SIGER Laboratory
Sidi Mohamed Ben Abdellah University
Fez, Morocco
chougrad.hiba@gmail.com

Youness Idrissi Khamlichi
SIGER Laboratory
Sidi Mohamed Ben Abdellah University
Fez, Morocco
youness.khamlichi@usmba.ac.ma

Abdessamad El Boushaki
SIGER Laboratory
Sidi Mohamed Ben Abdellah University
Fez, Morocco
abdessamad.elboushaki@gmail.com

Safae Elhaj Ben Ali
SIGER Laboratory
Sidi Mohamed Ben Abdellah University
Fez, Morocco
safae.elhajbenali@usmba.ac.ma

Abstract— The Deep Reinforcement Learning Process (DRL) has made considerable progress since its inception towards the production of various independent systems. Nowadays, deep learning gives Reinforcement Learning (RL) the ability to conform to several dysfunctions that were previously difficult, if not impossible to resolve. In this paper, we explore the power of Deep Reinforcement Learning in order to solve the problem of portfolio management which is a process that allows the selection, prioritization and management of funds by following a defined policy. Our objective is to obtain an optimal strategy to maximize the expected return while minimizing the costs. To do this, we use the Deep Deterministic Policy Gradient which is an off-policy algorithm adapted for environments with continuous action spaces. The obtained results demonstrate that the model achieves a higher rate of return than the strategy of "Uniform Buy and Hold" stocks and the strategy of "Buy Best Stock in last month".

Keywords—Portfolio Management, Reinforcement learning, Deep Learning, Deep Deterministic Policy Gradient

I. INTRODUCTION

The financial market is an environment known for its complex composition. It is characterized by uncertain and changing information, dynamic opportunities, multiple goals and strategic considerations, which make it a system for which it is difficult to design an optimal and profitable strategy that maximizes returns while avoiding risks [1]. Reinforcement Learning can help build such decision systems. It is based on agents who tend to obtain an optimal policy through trial and error by interacting directly with the environment [2]. In this paper, we adopt this type of learning to find the best possible behavior for Portfolio management.

Portfolio management is the decision making process of continuously reallocating an amount of funds into a number of different financial investment products, aiming to maximize the return while restraining the risk [3][4]. To analyze the allocation, risk and return of a given portfolio, different benchmarks can be used. In this work, we will exploit two benchmarks which are: "Uniform Buy and Hold" and "Buy Best Stock".

"Uniform Buy And Hold" [5], also called market strategy refers to a passive investment strategy in which an investor buys carefully selected stocks and holds them for a long time, regardless of market fluctuations. This strategy is based, among other things, on the assumption that the economy will continue to grow over the long term and that earnings and stock prices will continue to rise. This allows the long-term investor to build a high-performance and balanced portfolio while devoting very little management time to it.

As for the "Buy Best Stock" [5], this is a strategy where the agent knows the growth rate of each share between the first day and the last day of the examination. The agent simply invests all of its assets in one stock, the one with the highest growth rate, and holds it for the entire period. In this case, the return for each time step is equal to the growth rate of a better stock.

Reinforcement learning has a good capacity for decision making, however, it has a major drawback in perception [6]. Contrarily, Deep Learning (DL) is strong in perception, but it has a weak capacity for making decisions [7]. Indeed, for this case study, traditional Machine Learning (ML) methods perform financial transactions by recognizing patterns from raw data, and in this context, it can be used to understand the rules of purchase, sale and execution [8]. Deep Learning, on the other hand, is even more effective insofar as there is a repetition of the reception of new information from the raw data. Therefore, Reinforcement Learning and Deep Learning in a way complete each other and their combination can then be used to solve the problem of perceptual decision making in complex systems [9]. Among the most well-known Reinforcement learning algorithms is DDPG [10]. It is an Actor-Critic algorithm designed to be used in environments with continuous action spaces. It is a model free off-policy and so as the name suggests, it is deterministic which means that

we have a network of policies which, under some observations of the environment, only results in the one that is considered to be the most appropriate action.

In this paper, we investigate the effectiveness of DDPG to create a profitable strategy. We explore the potential of deep Reinforcement Learning in optimizing a Portfolio Management strategy in order to maximize the return on investment. Experiments show that the proposed algorithm achieves a higher rate of return compared to baseline strategies like “Buy And hold” or “Buy Best Stock”.

This document is organized as follows: Section 2 generally reviews some related work on RL and DRL. In section 3 we briefly introduce the stock market system, in section 4 we detail the main concepts related to portfolio optimization, we formulate the problem in section 5, in section 6, we present the MDP formulation, then we detail in section 7 the concept of deep reinforcement learning and the actor critic methods, we describe also the algorithm of deep deterministic policy gradient and we discuss the implementation of the model. In section 8, we present the results obtained and we conclude with some final remarks.

II. RELATED WORK

Reinforcement learning has been applied successfully in many fields such as: resource management [11], helicopter flight[12] and chemistry [13] among others. It has demonstrated effective performance on a variety of problems such as robotic control [14][15], the inverted pendulum [16], job scheduling, telecommunications, and backgammon [17].

However, although RL has enjoyed these successes over the past few decades, old approaches and previous theories [18] lacked evolution and were strictly limited to problems of rather old dimensions. In addition, we have recently witnessed the rise of deep learning [19][20][21], which relies on fairly powerful approximation properties of functions and learning representations of deep neural networks, and which has given us new tools to solve these dysfunctions. The notion of deep learning has made considerable progress in RL, notably through the use of deep learning algorithms within RL which defines the domain of Deep Reinforcement Learning (DRL) [22].

Among recent works in the field of DRL, there have been two exploits that have proven to be remarkable. The first that spawned the DRL revolution, which represented the development of an algorithm used to learn how to launch games like Atari 2600 [23] at a high level, directly from image pixels, thus providing means for solving the instability of function approximation techniques in Reinforcement Learning. A second considerable success consisted in the development of a hybrid DRL system, AlphaGo [24], which brought its victory over a human world champion in Go. Indeed, DRL algorithms have been previously employed and used in a field of problems, such as robotics [25], health [26][27], transportation [28] and video games [29][30].

Deep Reinforcement learning has also been applied to the field of financial markets in many different contexts. In [31], the authors proposed a new strategy for portfolio management. The method is based on the Deep Q-Network algorithm. First, they discretized the action space in order to adapt the DQN to stock market production. Then, in order to improve the recognition capacity of the algorithm, the authors opted for the combination of the dueling Q-net with the neural network. The results obtained after the tests show that the proposed method surpasses the ten other traditional strategies and that it is the least risky investment method. Another work [32] based on Q-learning proposed a new portfolio management framework which is composed of a set of local agents and a global agent. The local agent is based on deep Q-learning, double deep Q-learning and dueling double deep Q-learning and the global agent describes the global reward function. After testing the framework on a crypto portfolio consisting of four cryptocurrencies, it turned out that the proposed framework is a promising approach for dynamic portfolio optimization. Another strategy has been proposed in the work of [33] consisting in training an agent so that he can identify an optimal trading action using deep Q-learning. In order to super form trading, the proposed strategy is based on three functionalities which are as follows: The first is a mapping function whose objective is the transformation of an action found but unrealizable into an achievable action closest to the ideal action initially proposed. In order to derive a multi-asset trading strategy, the authors [33] proposed as a second functionality to establish agent and Q network models. And for the third functionality, they designed an agent to simulate all the actions that can be done in each state and this to have a technique that aims to derive a well-suited multi-asset trading strategy. After performing the tests, the proposed approach was able to obtain better results than the reference strategies. As for the authors of [34], they proposed a new model-based RL architecture using an infused prediction module (IPM) and a behavior cloning module (BCM), extending the standard actor-critic algorithms. In addition, the authors have designed a back-testing engine which also allows transactions to be executed by interacting with the RL agent in real time. After performing the tests, the results obtained demonstrate that the proposed model is robust, profitable and risk sensitive, compared to basic trading strategies and model-free RL agents used in previous works. Another work [35] has addressed the main research problems linked to the optimization of financial portfolio management policies. The authors proposed to use deep RNNs (GRUs) and a policy gradient in order to find the optimal policy function. They presented in their paper a study of each type of RL approaches and their integration with the DL method while solving the policy optimization problem.

III. STOCK MARKET

The stock market is the main component of the capital market and one of the most important ways for public corporations to raise financing. The stock market's volatility is strongly linked to the economic market's success or recession.

The mode of financing for public enterprises is capital. Public firms can generate idle funds by issuing stocks and bonds on the stock market to remedy the problem of a temporary capital deficit.

Shares are one of the most easily accessible financial products for private investors. Personal assets will appreciate to some extent if they are held in appropriate stocks and managed properly. Stock trading is also the primary business of securities firms and investment banks.

Stock trading and price prediction are all about predicting the future worth of a company's stock, which has been around since the beginning of the stock market. A good projection of stock prices in the future can bring in a lot of money. Many macro and micro elements, such as interrelated political and economic indicators, the supply-demand connection for

stocks, corporate operational circumstances, and so on, affect the stock market. As for portfolio management, it helps in making the decision on what to trade. This involves security analysis, risk, modeling, correlations, liquidity, etc. More details about this process in the next section.

IV. PORTFOLIO MANAGEMENT

a) The process of Portfolio Management

A portfolio is a collection of securities or a combination of assets put together with the owner's specific goals in mind. Commodities, currencies, futures or corporate debt, and so on are all examples of securities, and a portfolio is established when various assets are chosen and then integrated. The portfolio must be tailored to the investor's investing objectives in order to be most effective, which implies that the assets picked and assembled must coincide with the investor's goals.

There are three basic steps to managing a portfolio:

Investment policy: This step identifies the goals in terms of profitability, risk tolerance, degree of volatility, time horizon, as well as constraints such as tax considerations, the need for cash and income, and any other factors that may influence investment decisions. This stage influences the securities, actions, and assets that will be included in the portfolio, as well as the management style; everything else is influenced by this stage.

Execution: Portfolio managers assess the risk and return of various asset classes to determine fund allocation in this stage. The manager uses a top-down analysis to consider current economic conditions including macroeconomic forecasts such as interest rates, GDP growth and inflation. This makes it possible to identify the asset classes that correspond to the investor's portfolio. Then, there's bottom-up analysis, which examines securities inside certain asset classes.

Feedback: Over time, investors' preferences change, the risk and return of asset classes also changes, and as market prices of securities change, portfolio composition also changes. The manager must assess these characteristics and rebalance the portfolio. This process includes buying and/or selling securities to readjust the weighting to the desired percentages. Additionally, the portfolio manager should compare the performance of the portfolio to the benchmark and make any necessary changes.

b) Types of Portfolio management

Portfolio management strategies are diverse; however, they generally fall into four categories:

Active Portfolio management. In this case, investors and portfolio managers assume that they can outperform the market by taking advantage in particular of market anomalies and will therefore aim to outperform a benchmark index, they must be able to buy securities or assets that are expected to do better than the market and to sell assets that are supposed to underperform the market. This implies that active managers must bear additional analysis costs and transaction costs that make active management expensive. Quantitative market analysis, extensive diversification, and a thorough understanding of the economic cycle are all required for this strategy.

Passive Portfolio Management Investors who believe in the efficiency of the financial markets, i.e. they believe that the prices reflect at all times all the information available on the securities will opt for passive management, the objective of which is to replicate a reference index or a Benchmark by trying to limit transactions and management costs. Their management cost is among the lowest, because they do not use any strategy and their only function is to replicate an index.

Discretionary Portfolio Management is aimed at investors who have significant assets and wish to delegate day-to-day investment decisions based on a clearly defined investment policy. Discretionary management is a service whereby an individual gives a professional investment manager permission and the authority or discretion to act on their behalf proactively in light of changing investment markets, changes in geopolitical risks and changes in day-to-day investment performance.

Non-Discretionary Portfolio Management A non-discretionary manager is nothing more than a financial adviser, his power of attorney is limited, he advises the investor on the best course of action. Although the advantages and disadvantages are clearly stated, the investor is free to choose his own way. It is only until the manager has received authorization to act on behalf of the investor that he acts.

c) Investment risk profiles

There are three primary types of investment risk profiles: conservative, balanced, and aggressive.

Conservative risk profile is one in which the investor has a low risk tolerance and capacity, and is unable to take risks due to a large number of financial applications and a low income. As a result, he should allocate 0-10 percent of his assets to equity and a maximum of 90-100 percent of his assets to debt and cash.

Balanced risk profile. This type of investor has a medium level of risk-taking capacity and tolerance; they can take on risk but not excessively owing to financial obligations and small income, and they can invest a maximum of 40% to 60% in stock and 40% to 60% in debt or cash.

Aggressive risk profile. This type of investor has a high level of risk tolerance and capacity. They may take large risks since their financial commitments are met, and they earn a lot of money and have a lot of money. These individuals can put 90 to 100 percent of their money into equity and only 0 to 10% in debt or cash.

d) Basic Techniques for managing stock markets problems

The basic techniques for managing stock markets problems are divided into two categories: The fundamental analysis and the technical analysis.

Fundamental analysis, according to fundamental analysis, economic forces influence prices, so we look into the economics of a company to see if it is cheap, costly, or the proper value. With fundamental analysis, we use existing data to estimate the value of a stock by taking the time to evaluate all of the available information, which can be in the form of quantitative or qualitative information and can impact the business economically. This information is used to construct ratios and metrics that measure the health, performance, and growth rates of a company. Future growth rates are forecasted using industry data and economic factors such as interest rates and retail expenditure. After evaluating numerous models and ratios, a fair value is determined.

Technical analysis Technical analysis is a field that uses charts to analyze the movement of a financial market. The chart structure reflects all the information available to buyers and sellers. Unlike the fundamental analysis, the technical analysis is simply concerned with the price's evolution, not with the external factors that influence it. The curve on the chart is the trace left by the balance of power between the buyers and sellers. The curve rises when buying interests are dominant, and vice versa. Technical analysis allows to identify the moments when a market evolves in trend. The technical analyst's task is to qualify the emergence of a trend as quickly as feasible until it reaches its exhaustion phase. In the past, there have been major price levels that have resulted in arbitrages and the collective psychology of investors is now known during market rises (greed) and declines (fear). Trends or geometric structures on the chart depict this collective psychology. These chart patterns have no predictive value but they do reflect market sentiment. They allow for the formulation of theories on its evolution.

V. PROBLEM STATEMENT

We wish through our work to manage the portfolio and this by distributing our investment in a number of shares according to the market. Our environment is defined as follows:

Let N be the number of shares we are going to invest in. Our close / open relative price vector is defined as follows:

$$y_t = \left[1, \frac{v_{1,t,close}}{v_{1,t,open}}, \frac{v_{2,t,close}}{v_{2,t,open}}, \ldots, \frac{v_{N,t,close}}{v_{N,t,open}}\right] \quad (1)$$

where $\frac{v_{i,t,close}}{v_{i,t,open}}$ represents the relative price of stock i at time stamp t. With y [0] is the relative price of cash, which is always equal to 1. Our portfolio weight vector is defined as follows :

$$w_t = [w_{0,t}, w_{1,t}, \ldots, w_{N,t}] \quad (2)$$

We note that $w_{i,t}$ is the fraction of investment on stock i at time stamp t and $\sum_{i=0}^{N} w_{i,t} = 1$.

With $w_{0,t}$, the fraction of cash we have and therefore the profit after the timestamp T is :

$$p_T = \prod_{t=1}^{T} y_t . w_{t-1} \quad (3)$$

Note that w_0=[1,0,⋯,0]. Let μ be the negotiation cost factor, the cost of negotiating each timestamp is:

$$\mu_t = \mu \sum \left| \frac{y_t \odot w_{t-1}}{y_t . w_{t-1}} - w_t \right| \quad (4)$$

Then we have:

$$P_T = \prod_{t=1}^{T} (1 - \mu_t) y_t . w_{t-1} \quad (5)$$

VI. MARKOV DECISION PROCESS FORMULATION

We model our problem as a Partially Observable Markovian Decision Process (MDP) which generalizes MDP for planning under partial observability and when there exist multiple sources of uncertainty [36]. Besides the system's stochastic dynamics, there are also observation noises [37].

a) State and Action

Our state s_t is defined as o_t, with o_t representing the observation of the timestamp t. Since the effect of historical data decreases over time, we will be content with the price of the historical and current news of the day in a fixed length window W. From where,

$$o_t = \left[\overrightarrow{v_{1,t}}, \overrightarrow{v_{2,t}}, \ldots, \overrightarrow{v_{N,t,}}\right] \quad (6)$$

with,

$$\overrightarrow{v_{1,t}} = \begin{bmatrix} v_{i,t} - w \\ v_{i,t} - w + 1 \\ : \\ . \\ v_{i,t-1} \end{bmatrix} \quad (7)$$

The portfolio weight vector W_t represents the action a_t. Our objective is to form a network of policies $\pi_\theta(a_t, o_t)$.

b) State Transition

We have no control over the market, so what we can get is price and timeliness i.e. state of observation.

So, Instead of o_{t-1}, o_t is given by the dataset, this is because we collect the price history of various stocks.

c) Reward

We have the logarithm of the equation (3) instead of having the reward 0 at each timestamp and p_t at the end:

$$\log p_t = \log \prod_{t=1}^{T} \mu_t y_t . w_{t-1} = \sum_{t=1}^{T} \log(\mu_t y_t . w_{t-1}) \quad (8)$$

So, in order to avoid the sparse reward problem [38] where an environment rarely produces a useful reward signal, which seriously calls into question the way ordinary DRL tries to learn [38], we have $\log(\mu_t y_t . w_{t-1})$ which rewards each timestamp.

VII. METHODOLOGY

A. Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning method in which agents tend to learn an optimal policy through trial and error [7]. By interacting with the environment, an RL agent is supposed to be diligent in succeeding in sequential tasks when making decisions. The Reinforcement Learning agents can interact with their

environments by observing the repercussions of their actions. They should be able to learn, to rectify, and to modify their own actions in response to the rewards received [7].

An independent agent in the RL configuration, observes and supervises a state s_t from its environment at time step t. This agent engages in interaction with the environment by taking an action in the state s_t. When taking an action, the agent and its environment transfer to a new state s_{t+1}, depending on the action chosen and the current state. The rewards provided by the environment determine the best sequence of actions. The environment also provides an r_{t+1} scalar reward to the agent in return each time it transitions to a new state. The agent's objective consists in assimilating a control strategy r which maximizes the expected return (i.e. the cumulative, discounted reward). We consider a state, a policy – a mapping of observations into actions –; a better policy is one which maximizes the expected profitability in the entourage. To this end, Reinforcement Learning strives to provide a solution to the same problem as optimal control. On the other hand, what we take up as challenge in RL is that the agent is able to learn about the repercussions of actions in the environment by errors and by trials, since, a model of the state transition dynamics is not available to the agent. The information that the agent uses to update his knowledge is produced by each interaction with the produced environment. (see Figure 1).

Fig. 1. Architecture of Reinforcement Learning

B. *Deep Learning*

Machine learning algorithms adjust the parameters of their calculations based on the examples given to them. This makes it possible to adapt their operations to the data provided. Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction [39]. It is based on Artificial Neural Networks, a technique inspired by the theory of brain development and can be learned in the sense of three learnings: unsupervised learning, supervised learning and semi-supervised learning.

The neural network structure consisting of Input, Output and Hidden layers.

In addition, depending on the task of interest, there are specific architectures involved when training the model using these networks, we cite the Artificial Neural Networks (ANN), Deep Neural Networks (DNN), Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM).

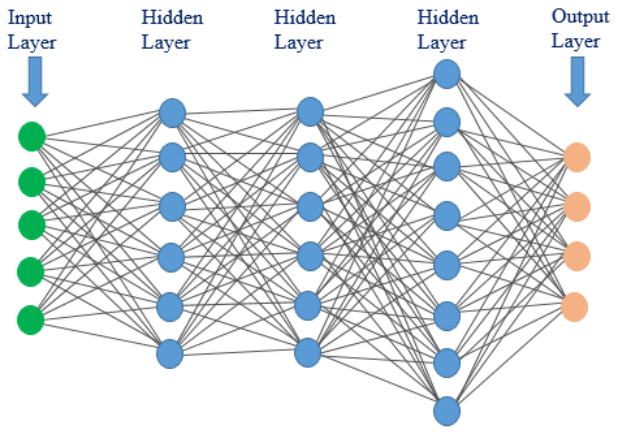


Fig. 2. The structure of a Deep Neural Network

1) Artificial Neural Network (ANN) : An artificial neural network (ANN) [40] is designed on the basis of the working mechanism of the human nervous system [40]. A neural network learns to perform a task by examining labeled training examples. It is made up of a number of data processing agents; that are called neurons and which receive signals from each other and pass them on to each other. A neuron takes information as input and develops outputs according to its internal activation function [42]. The parameters controlling the learning of ANNs are the number of hidden layers, the number of neurons per hidden layer. These two choices directly condition the number of parameters (of weight) to be estimated and therefore the complexity of the model. There is also, the maximum number of iterations, the maximum tolerated error and a possible term of ridge regularization (decay). We have also, the learning rate as well as a possible strategy for its evolution and the size of the sets or batches of observations considered at each iteration. For the computation, several activation functions can be used, such as: Sigmoid, Softmax, Rectified Linear unit…

Artificial Neural Networks (ANNs) based on multiple layers hidden between the input and output layers [43][44] are called Deep Neural Network (DNNs). An example of DNN architecture is shown in figure 2.

2) Recurrent Neural Networks (RNN) : Recurrent Neural Networks [45] , capture information about sequences or time series data. They are particular networks, which make it possible to take into account the notion of sequence of observations.

The new state of the recurrent neural network at time T is a function of its old state at time t minus 1 and of the input at time which is X_t. This function is the basic idea of RNNs. X_t : is the entry at time step t , S_t : state at time step t , F_w : is the recursive function.

$$S_t = F_w(S_{t-1}, X_t) \qquad (9)$$

Simple RNNs work with formula (9) and the recursive function used is a tanh. We multiply the input state X with weights W_x and the previous state by W_s then we go through an activation tanh to get the new state.

$$S_t = tanh(W_s S_{t-1} + W_x X_t) \qquad (10)$$

W_x and W_s are the weights. Now, to get the output vector we multiply the new state that is S_t with W_y.

$$Y_t = W_y S_t \quad (11)$$

3) Long Short Term Memory (LSTM) : Recurrent neural networks [45] are widely used in applications such as automated language processing. However, they show deficiency in learning when the sequences to be processed become too long [46]. As the gradient can become too small at the end of the chain. Long Short Term Memory [47] was introduced to solve this problem thanks to their gate system. It is a variant of RNN and its blocks are known as memory cells with adaptive multiplicative gates. The first gate of the LSTM is responsible of deciding which information to eliminate from the state of the cell which is carried out by a sigmoid layer and it is called the "forgot gate layer". For the second gate called the "entry gate", it decides what information can be stored from the current step and decides how much current information to transmit. As for the output gate, it decides whether the contents of the cell should influence the output of the neuron.

C. Deep Reinforcement Learning

Tabular representations represent the most accessible tools for storing learned estimates like values, models, or policies, of which each pair of case actions is attached to a discrete estimate [48]. When estimates are represented discretely, each additional feature tracked in the state leads to an exponential growth in the number of state-action pair values that must be stored [48]. In the literature, this dysfunction is generally referred to as the "dimensionality curse", a term that was originally coined by Bellman [49].

Usually in simple environments, this is rarely a concern, however it can lead to insoluble problems in real life algorithms, because of computational and / or memory constraints which remain essential as parameters. It is possible to obtain useful policies over a large state action space, however, it can take an unacceptable amount of time. Several fields of everyday reality include spaces of continuous state and / or action; the latter are presumed to remain discretized in various cases.

On the other hand, interesting steps of discretization are supposed to limit certain performances which one can concretize in a domain, whereas meticulous discretization steps can result in a considerable state reaction interval where it is impractical to obtain a sufficient sample size dependent on each state-action pair [48].

Alternatively, the function approximation can be employed for the purpose of generalization by states and / or actions, whereby the function approximation is employed to retrieve and store certain estimates [48]. It turns out that the approximation of functions remains a fairly active area of research in RL, providing a way to orient continuous state and / or action spaces, to minimize the explosion of state space, action, and then generalize the previous experience to state-action pairs that were previously invisible [48].

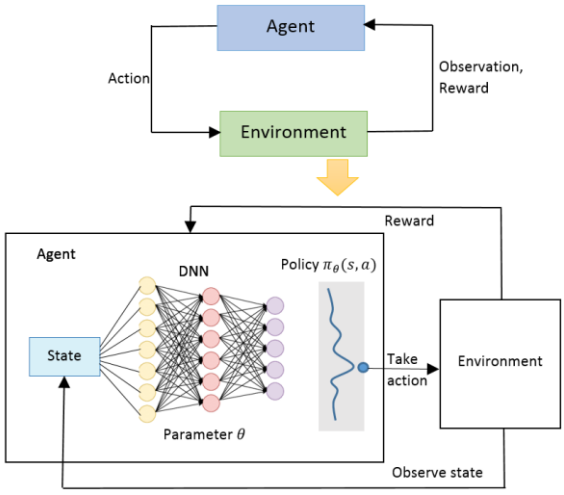


Fig. 3. Architecture of Reinforcement Learning and Deep Reinforcement Learning

Neural networks are being used on a recurring basis to implement function approximation. Several works have employed deep neural networks as a means of approximating functions; this emerging model is known as Deep Reinforcement Learning (DRL) [22]. (See Fig. 3).

Among the most widely used reinforcement learning techniques, we find: Actor-critic methods. In the next two sections, we will introduce the concept of these methods as well as the Deep Deterministic Policy Gradient (DDPG) algorithm which is based on them.

D. Actor-critic Methods

Actor-critic methods are methods that combine value-based and policy-based into a single algorithm [50]. As their name suggests, they use two neural networks: An actor (policy-based) and a critic (value-based).

The objective of the actor $\pi_\theta : s \longrightarrow A$ is to control the behavior and to get directly close to the policy of the agent, it is indeed a mathematical function. The policy is just a probability distribution over the set of actions where we take a state as input and produce a selection probability of each action.

On the other hand, the critic $Q_\phi^\pi : S \times \mathrm{A} \longrightarrow R$ aims to measure the quality of the action taken, in other words, it is used to approximate the value function. The critic acts like any other critic, telling the actor how good or bad each action is, depending on the value of the resulting state.

The two networks are working together to find out how to best act in the environment. The actor selects the action, the critic assesses the condition, and then the result is compared to the environmental rewards. Over time, the critic becomes more precise in estimating state values, allowing the actor to select the action that leads to those states from a practical point of view.

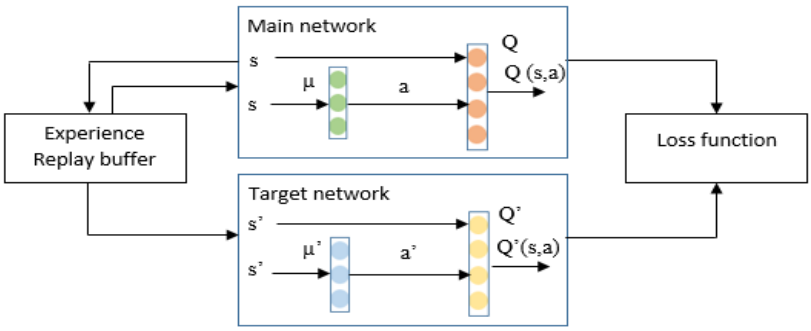
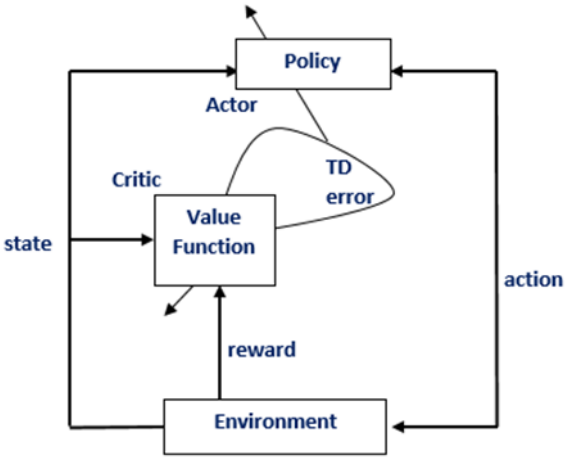


Fig.4. Architecture of Deep Deterministic Policy Gradient

Actor-critic methods belong to the class of algorithms called Temporal Difference Learning [51]. The basic idea of TD methods is that the learning is based on the difference between temporally successive predictions [51], so we will update the weights of our DNN at each time step. This is a way of saying that we are going to estimate the difference in values of the successive states that is to say of the states separated by a time step. As with any DL problem, we calculate the cost function; in particular, we have 2 cost functions, one to update our critic and the other to update our actor. Figure 4 represents the architecture of the Actor-critic methods.

Fig.5. Architecture of Actor-Critic algorithm



Actor updates are done always in the same way:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \, log \pi_\theta(a_t|s_t) G_t \qquad (12)$$

where G_t is the evaluation of long-term returned by the critic for s_t. Critic updates are done to evaluate the current policy

$$w \leftarrow w + \alpha \delta \nabla_\theta V_W(a_t|s_t) \qquad (13)$$

where δ is the estimated error in evaluating the s state.

E. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [10] is a model-free off-policy algorithm for learning continuous actions. In particular, we use a replay memory where, instead of just learning from the most recent state transition the agent has experienced, it will keep track of the sum total of its experiences, then randomly sample that memory to each time step to get a batch of memories to update the weight of its deep neural network. The other innovation is the use of target networks [10]. The architecture of the DDPG algorithm is shown in figure 5.

DDPG uses the Actor-Critic structure which means that it has two networks. For the network of actors, it takes observation and gives action. For the critical network, it takes the observation and the corresponding action and produces the value Q. For the deterministic part, this comes from the fact that we have a network of policies which, under a certain observation of the environment, only gives the one which is considered to be the most appropriate action.

Now, this poses a problem called the exploration-exploitation dilemma [7] and it is present in all Reinforcement Learning problems, it is a fundamental concept in the field. The basic idea is that our agent tries to build a model of the world. When the agent takes an action out of the optimum it is called exploration and when it takes the optimal action, it is called exploitation because it is simply exploiting the best known action and the solution here, is to take the output of our current network and apply some extra noise to it.

So, we can break down the algorithm into the following: Experience replay, Actor and critic network updates, target network updates and exploration.

As used in many RL algorithms, DDPG uses a replay buffer to sample the experience in order to update the parameters of the neural network. During each trajectory deployment, we save all experience tuples (state, action, reward, next_state) and store them in a cache of finite size "replay buffer". Then, when updating the parameters of the neural network, the experiences inside this recovery memory are randomly selected, which avoids introducing a bias during learning.

Date	Time	AAPL_open	AAPL_High	AAPL_Low	AAPL_Close	AAPL_Vol
2018-01-02	173100	170.18	170.19	169.88	169.94	17654
2018-01-02	173200	169.90	169.93	169.67	169.69	8200
2018-01-02	173300	169.67	169.88	169.58	169.58	9380
2018-01-02	173400	169.46	169.46	169.27	169.33	5348
2018-01-02	173500	169.45	169.60	169.44	169.59	9700

TABLE I. EXAMPLE OF RAW DATA FOR THE AAPL STOCK

1) Actor (Policy) & Critic (Value) Network Updates

The network of values is updated in the same way as in Q-learning. The updated Q-value is obtained by the Bellman equation:

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'}) \mid \theta^{Q'}\right) \quad (14)$$

However, in DDPG, the calculation of Q values for the next state is done with the target value network and the target strategy network. Next, we minimize the mean square loss between the updated Q-value and the original Q-value :

$$Loss = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \quad (15)$$

For the political function, our objective is to maximize the expected return :

$$J(\theta) = E[Q(s,a) \mid_{s=s_t, a_t=\mu(s_t)}] \quad (16)$$

To calculate the policy loss, we take the derivative of the objective function with respect to the font parameter. Knowing that the function of actor (political) is differentiable, we must therefore apply the chain rule.

$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_a Q(s,a)\, \nabla_{\theta^\mu} \mu(s|\theta^\mu) \quad (17)$$

But as we update the policy in a non-political way with lots of experience, we take the average of the sum of the gradients calculated from the mini-batch:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q) \mid_{s=s_i,\ a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i} \quad (18)$$

2) Target network update

We make a copy of the parameters of the target network and slowly follow them with those of the learned networks via "software updates", as illustrated below:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \quad (19)$$

where $\tau \ll 1$

3) Exploration

In reinforcement learning for discrete action spaces, exploration is done via the probabilistic selection of a random action such as epsilon-greedy [52] or Boltzmann exploration. For continuous action spaces, exploration is done by adding noise to the action itself. Authors in [53] add noise to the output of the action :

$$\mu'(s_t) = \mu\left(s_t \mid \theta_t^\mu\right) + N \quad (20)$$

The pseudo-code is described in the section above.

Algorithm. Deep Deterministic Policy Gradient

Randomly initialize critic network Q(s, a|θ^Q) and actor μ(s,a|θ^μ) with weights θ^Q and θ^μ

Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer R

For episode = 1, M do

Initialize a random process N for action exploration

Receive initial observation state s_1

For t = 1, T do

Select action $a_t = \mu(s_t \mid \theta^\mu) + N_t$ according to the current policy and exploration noise

Execute action a_t and observe reward r_t and observe new state s_{t+1}

Store transition (s_t, a_t, r_t, s_{t+1}) in R

Set $s_{t+1} = s_t$

Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

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

Update critic by minimizing the loss:
$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q) \mid_{s=s_i,\ a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

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

End for

End for

VIII. EXPERIMENTS AND RESULTS

1) Dataset:

In this work, we assess the performance of the Deep Deterministic Policy Gradients on Portfolio Management. We have formed the model with 15 Stock price data (AAPL, BAC,

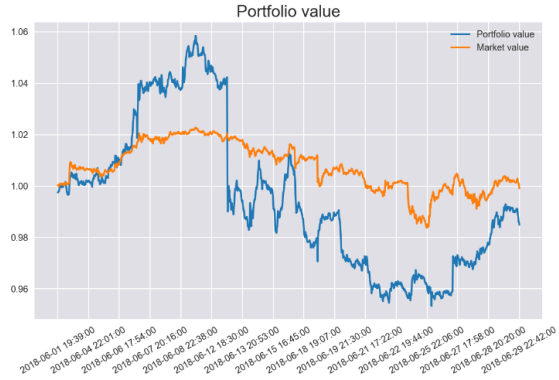
BA, GE, GOOG, JNJ, KO, MA, MRK, MSFT, NVDA, PFE, SBUX, T, XOM) from 2018/01/01 to 2018/10/29, recorded in minutes with open, close, high, low, column features. Table 1 gives an example of raw data from the AAPL Stock Price.

2) Parameter settings and results :

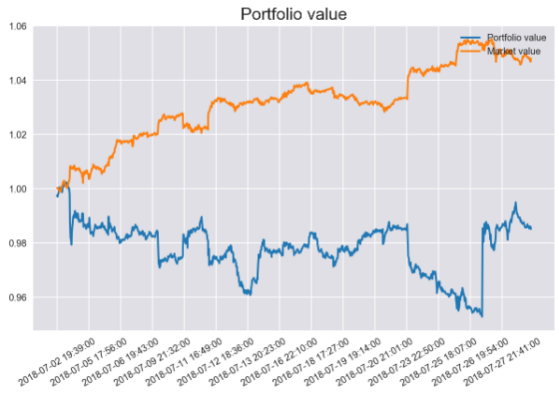
We use the algorithm designed for environments with continuous action spaces which is the DDPG, in order to learn the policy network $\pi_\theta(a_t, s_t)$. The configurations are as follows :

- The Critic Network: Refers to the value function.
- The Actor Network: which designates the network of policies.
- Exploration noise: Ornstein-Uhlenbeck with zero mean, 0.3 sigma and 0.15 theta.
- We train the models with different settings in 1500 episodes.

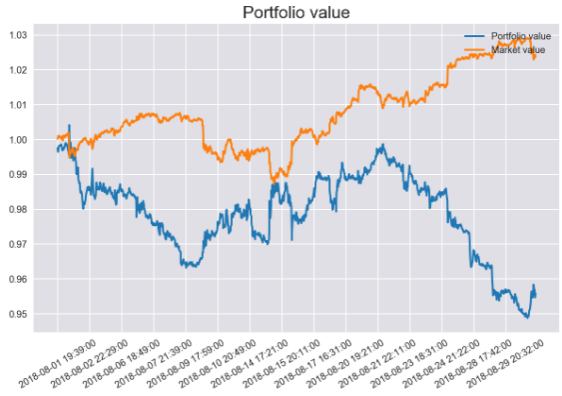
a) : DDPG results for the month of May



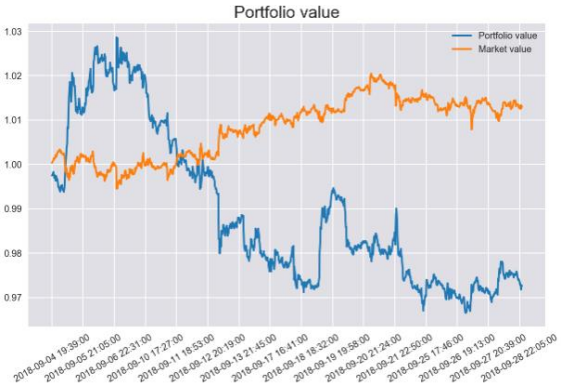
(b) : DDPG results for the month of June



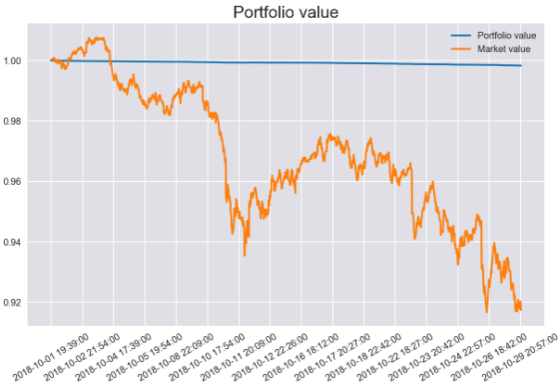
(c) : DDPG results for the month of July



(d) : DDPG results for the month of August



(e) : DDPG results for the month of September



(f) : DDPG results for the month of October

Fig. 6. DDPG result : Portfolio value vs market value for the six months on the 15 Stocks : May, June, July, August, September and October

As for the basic parameters : the stock contains the cash position, the long position of 15 stocks and the short position of 15 stocks, the stock price data is observed every minute, but the action is done only every 7 minutes. In addition to the original (s, a, r, s'), other state-action pairs of "inferred steps" were also collected and stored in the replay buffer at each step.

We have normalized the historical price instead of using the open, high, low and close gross price so that whatever the actual price of the historical prices is, they are going to be in the same scale. The historical price is normalized as: (closeopen - 1) × scale, and we heuristically set the scale factor to 100.

For building our learning model we adopted a recurrent Long short-term memory network (LSTM) to solve the portfolio management problem which gave promising results.

The market value is obtained by equally distributing the investment to all the stocks. Figure 6 represents the value of the portfolio (Orange) versus the market value (Blue) during the last four months of testing. The figure shows that the model only buys and sells with a very small portion of the portfolio, it hasn't changed positions very often during the month.

We used the data from the previous month to build the RL model in a time series rolling diagram and we tested it over the following month. As for the results, the model achieved a rate of return of 14% for the selected period compared to a of return of 5.6% using a “Uniform Buy and Hold” strategy of the 15 stocks and maintaining a rate of return of -16.8% using the “Buy Best Stock” strategy in the last month.

IX. CONCLUSION

In this article, we have explored the Deep Deterministic Policy Gradient method and investigated its efficiency when applied to stock trading as a controller. Our model was able to achieve a higher rate of return compared to baseline strategies which are “Buy And hold” or “Best Stock”.

Although stock trading with agents, through Reinforcement Learning, has shown some efficiency, it is not yet reliable as it sometimes makes mistakes and loses money.

The results obtained can be improved in several ways. For instance, we would like to test Reinforcement Learning algorithms on live trading platforms rather than historical data.

We could also select the most predictive characteristics by adjusting the model with various hyper-parameters. Finally, we could compare different Reinforcement Learning algorithms under similar conditions and data sources.

REFERENCES

[1] COOPER, Robert G., EDGETT, Scott J., et KLEINSCHMIDT, Elko J. Portfolio Management. Pegasus, New York, 2001.

[2] SUTTON, Richard S. Introduction: The challenge of reinforcement learning. In : Reinforcement Learning. Springer, Boston, MA, 1992. p. 1-3.

[3] Robert A Haugen. Modern investment theory. Prentice Hall, 1986.

[4] Harry M Markowitz. Portfolio selection: efficient diversification of investments, volume 16. Yale university press, 1968.

[5] B. Li and S. C. H. Hoi, “Online portfolio selection: A survey,” ACM Computing Surveys, vol.46, pp.35, 2014.

[6] LIN, Long-Ji. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning, 1992, vol. 8, no 3-4, p. 293-321.

[7] SUTTON, Richard S., BARTO, Andrew G., et al. Introduction to reinforcement learning. Cambridge : MIT press, 1998.

[8] MENG, Terry Lingze et KHUSHI, Matloob. Reinforcement Learning in Financial Markets. Data, 2019, vol. 4, no 3, p. 110.

[9] TAN, Runjia, ZHOU, Jun, DU, Haibo, et al. An modeling processing method for video games based on deep reinforcement learning. In : 2019 IEEE 8th joint international information technology and artificial intelligence conference (ITAIC). IEEE, 2019. p. 939-942.

[10] LILLICRAP, Timothy P., HUNT, Jonathan J., PRITZEL, Alexander, et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[11] MAO, Hongzi, ALIZADEH, Mohammad, MENACHE, Ishai, et al. Resource management with deep reinforcement learning. In : Proceedings of the 15th ACM Workshop on Hot Topics in Networks. 2016. p. 50-56.

[12] ABBEEL, Pieter, COATES, Adam, QUIGLEY, Morgan, et al. An application of reinforcement learning to aerobatic helicopter flight. In : Advances in neural information processing systems. 2007. p. 1-8.

[13] ZHOU, Zhenpeng, LI, Xiaocheng, et ZARE, Richard N. Optimizing chemical reactions with deep reinforcement learning. ACS central science, 2017, vol. 3, no 12, p. 1337-1344.

[14] BAGNELL, J. Andrew et SCHNEIDER, Jeff G. Autonomous helicopter control using reinforcement learning policy search methods. In : Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164). IEEE, 2001. p. 1615-1620.

[15] KIM, H. Jin, JORDAN, Michael I., SASTRY, Shankar, et al. Autonomous helicopter flight via reinforcement learning. In : Advances in neural information processing systems. 2004. p. 799-806.

[16] MICHIE, Donald et CHAMBERS, Roger A. BOXES: An experiment in adaptive control. Machine intelligence, 1968, vol. 2, no 2, p. 137-152.

[17] TESAURO, Gerald. Practical issues in temporal difference learning. In : Advances in neural information processing systems. 1992. p. 259-266.

[18] Liu H, Cocea M. Traditional Machine Learning[J]. 2018.

[19] Russakovsky, O.et al. Imagenet large scale visual recognition challenge. Int. J. Compute. Vis. 115, 211–252 (2015).

[20] WU, Yonghui, SCHUSTER, Mike, CHEN, Zhifeng, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144, 2016.

[21] OORD, Aaron van den, DIELEMAN, Sander, ZEN, Heiga, et al. Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499, 2016.

[22] Y. Li, “Deep reinforcement learning: An overview,” arXiv preprint arXiv:1701.07274, 2017.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[24] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., “Mastering the game of go with deep neural networks and tree search,” nature, vol. 529, no. 7587, pp. 484–489, 2016.

[25] Gu, S.; Holly, E.; Lillicrap, T.; Levine, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In Proceedings of the IEEE international conference on robotics and automation (ICRA), Singapore, 29 May–3 June 2017; pp. 3389–3396.

[26] KHALID, Muhammad, AWAIS, Muhammad, SINGH, Nishant, et al. Autonomous Transportation in Emergency Healthcare Services: Framework, Challenges, and Future Work. IEEE Internet of Things Magazine, 2021, vol. 4, no 1, p. 28-33.

[27] BAUCUM, Matthew, KHOJANDI, Anahita, et VASUDEVAN, Rama. Improving Deep Reinforcement Learning with Transitional Variational Autoencoders: A Healthcare Application. IEEE Journal of Biomedical and Health Informatics, 2020.

[28] MALDONADO-RAMIREZ, Alan, RIOS-CABRERA, Reyes, et LOPEZ-JUAREZ, Ismael. A visual path-following learning approach

for industrial robots using DRL. Robotics and Computer-Integrated Manufacturing, 2021, vol. 71, p. 102130.

[29] ZHA, ZhongYi, TANG, XueSong, et WANG, Bo. An Advanced Actor-Critic Algorithm for Training Video Game AI. In : International Conference on Neural Computing for Advanced Applications. Springer, Singapore, 2020. p. 368-380.

[30] YE, Deheng, CHEN, Guibin, ZHANG, Wen, et al. Towards playing full moba games with deep reinforcement learning. arXiv preprint arXiv:2011.12692, 2020.

[31] GAO, Ziming, GAO, Yuan, HU, Yi, et al. Application of deep q-network in portfolio management. In : 2020 5th IEEE International Conference on Big Data Analytics (ICBDA). IEEE, 2020. p. 268-275.

[32] LUCARELLI, Giorgio et BORROTTI, Matteo. A deep Q-learning portfolio management framework for the cryptocurrency market. Neural Computing and Applications, 2020, vol. 32, no 23, p. 17229-17244.

[33] PARK, Hyungjun, SIM, Min Kyu, et CHOI, Dong Gu. An intelligent financial portfolio trading strategy using deep Q-learning. Expert Systems with Applications, 2020, vol. 158, p. 113573.

[34] YU, Pengqian, LEE, Joon Sern, KULYATIN, Ilya, et al. Model-based deep reinforcement learning for financial portfolio optimization. In : RWSDM Workshop, ICML. 2019. p. 2019.

[35] HU, Yuh-Jong et LIN, Shang-Jen. Deep reinforcement learning for optimizing finance portfolio management. In : 2019 Amity International Conference on Artificial Intelligence (AICAI). IEEE, 2019. p. 14-20.

[36] MONAHAN, George E. State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms. Management science, 1982, vol. 28, no 1, p. 1-16.

[37] WANG, De-Xin et CAO, Xi-Ren. Event-based optimization for POMDPs and its application in portfolio management. IFAC Proceedings Volumes, 2011, vol. 44, no 1, p. 3228-3233.

[38] ZAI, Alexander et BROWN, Brandon. Deep reinforcement learning in action. Manning Publications, 2020.

[39] LECUN, Yann, BENGIO, Yoshua, et HINTON, Geoffrey. Deep learning. nature, 2015, vol. 521, no 7553, p. 436-444.

[40] HOPFIELD, John J. Artificial neural networks. IEEE Circuits and Devices Magazine, 1988, vol. 4, no 5, p. 3-10.

[41] Hassoun, M. H. (1995). Fundamentals of artificial neural networks. MIT press.

[42] Daniel, G. (2013). Principles of artificial neural networks (Vol. 7). World Scientific.

[43] Bengio, Y. (2009). Learning deep architectures for AI. Foundations and trends R in Machine Learning, 2(1), 1-127.

[44] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85-117.

[45] MEDSKER, Larry R. et JAIN, L. C. Recurrent neural networks. Design and Applications, 2001, vol. 5.

[46] YU, Yong, SI, Xiaosheng, HU, Changhua, et al. A review of recurrent neural networks: LSTM cells and network architectures. Neural computation, 2019, vol. 31, no 7, p. 1235-1270.

[47] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[48] KIRAN, B. Ravi, SOBH, Ibrahim, TALPAERT, Victor, et al. Deep reinforcement learning for autonomous driving: A survey. IEEE Transactions on Intelligent Transportation Systems, 2021.

[49] R. Bellman, Dynamic Programming. Princeton, NJ, USA: Princeton University Press, 1957.

[50] KONDA, Vijay R. et TSITSIKLIS, John N. Actor-critic algorithms. In : Advances in neural information processing systems. 2000. p. 1008-1014.

[51] SUTTON, Richard S. Learning to predict by the methods of temporal differences. Machine learning, 1988, vol. 3, no 1, p. 9-44.

[52] Watkins, C.: Learning from Delayed Rewards. PhD thesis, University of Cambridge, Cambridge, England (1989).

[53] UHLENBECK, George E. et ORNSTEIN, Leonard S. On the theory of the Brownian motion. Physical review, 1930, vol. 36, no 5, p. 823.