Master Thesis

Reinforcement Learning in Stock Market

Author:
Pablo Carrera Flórez de Quiñones

Supervisors:
Valero Laparra Pérez-Muelas
Jorge Muñoz Marí

Contents

List of Figures

List of Tables

1 Introduction

INCLUIR:

- [1] : models vs. algorithms
- [2] : rising of RL
- [3] : TD-gammon
- [4] : AlphaGo
- [5] : AlphaStar
- [6] : Montezuma Revenge

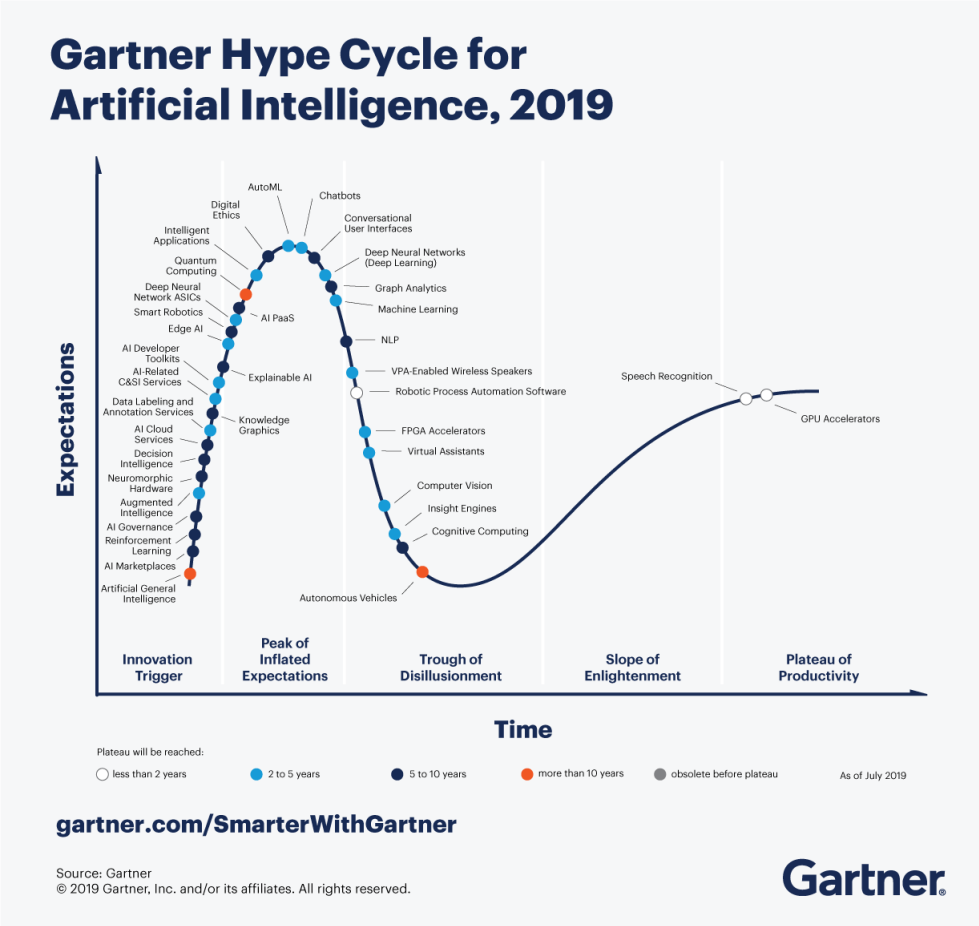

Figure 1: *Gartner Hype Cycle for Artificial Intelligence for 2019. We can see that the expectations on Reinforcement Learning are rising. Source: [2]*

2 Financial Markets

In this section we are providing an introduction to financial markets. First, we are going to define them, to explain their properties and actors and to divide them by the type trade they

involve. Then we are going deeper into one of the most popular of the financial markets, the stock market and the characteristics of the assets involved in it. Finally, we are going to make a operational introduction to the activity known as trading and the different approaches to make profits with it.

2.1 Fundamentals

A financial market, as defined in [7] or [8], is a market in which people exchange, or trade, financial assets. Although the existence of these markets is not a necessary condition for the creation and exchange of a financial asset, these operations can be realized outside of them, nowadays in most economies financial assets are created and subsequently traded in some type of financial market.

The function of financial assets is to transfer funds from those who have surplus funds to those who need funds to invest in tangible assets, in such a way that they also redistribute the unavoidable risks associated with tangible assets among those seeking and providing the funds. In this sense, financial markets have three properties:

- First, the interactions of buyers and sellers in a financial market determine the price of the traded asset. That is, the return on a financial asset is not given or fixed by prior in the market.
- Second, financial markets provide a mechanism for an investor to sell a financial asset. That is, they allow investors to get rid of the obligation of keeping an asset by transferring it to another investor.
- Third, financial markets reduce the cost of transacting. The search costs, related to looking for a potential buyer or seller, and information costs, related to the assessment of the financial merits of an asset, are almost totally avoided.

so from these properties surge many ways to classify financial markets:

- Nature of claim: we can distinguish between debt markets, where funds are borrowed and lent, and equity markets, where ownership of securities are issued and subscribed.
- Maturity of claim: we can distinguish between money markets, that allow firms to borrow funds on a short term basis, and capital markets, that allow firms to gain long-term funding to support expansion.
- Seasoning of claim: we can distinguish between primary markets, which deal with newly issued claims, and secondary markets, which deal with financial claims previously issued.
- Seasoning of claim: we can distinguish between cash markets, that trade assets directly, and derivatives instruments markets, that trade financial instruments based on an underlying asset.

2.2 Stock Market

Stocks, also known as equity securities or equities, represent ownership shares in a corporation [8]. Each share of a stock entitles its owner to one vote on any matters of corporate governance that are put to a vote at the corporation's annual meeting and to a share in the financial benefits

of ownership when those earning are distributed in the form of dividends. There are two types of stocks, the so called common stocks and the preferred stocks. The key difference between these two forms lies in the degree to which the participate in the distribution of earnings and the priority going to each in the distribution of earnings, typically preferred stockholders are entitled to a fixed dividend before common stockholders may receive dividends. We will center our discussion in common stocks since they are by far more the most common type of equity. The two most important characteristics of common stock as an investment are

- **Residual claim**: this means that stockholders are the last in line of all those who have a claim on the assets and income of the corporation. In a liquidation of the firm's assets the share-holders have a claim to what is left after all other claimants such as the tax authorities, employees, suppliers, bondholders, and other creditors have been paid. For a firm not in liquidation, shareholders have claim to the part of operating income left over after interest and taxes have been paid.
- **Limited liability**: this means that the most shareholders can lose in the event of failure of the corporation is their original investment. Unlike owners of unincorporated businesses, whose creditors can lay claim to the personal assets of the owner, corporate shareholders may at worst have worthless stock. They are not personally liable for the firm's obligations.

Stocks are created by the necessity of companies to raise new capital to achieve growth. Through an initial public offering, or IPO, the shares of the company are sold to institutional and individual investors. This process, also known as floating, transforms a privately held company into a public company. An IPO is underwritten by one or more investment banks, who also arrange for the shares to be listed on one or more stock exchanges. When a company lists its securities on a public exchange, the money paid by the investing public for the newly-issued shares goes directly to the company (primary offering) as well as to any early private investors who opt to sell all or a portion of their holdings (secondary offerings). After the IPO, when shares are traded freely in the open market, money passes between public investors, making the stock market then a secondary market, but note that the company will not be affected by these movements and is never required to repay the capital to its public investors. Those investors must endure the unpredictable nature of the open market to price and trade their shares.

Figure 2: *Biggest IPOs of history in terms of total monetary volume. We can see that the economic consequences of this type of operations are noticeable. Source: Statista.*

For a common stock its total value, also known as the market capitalization of the stock, is the sum of the price of all the shares, that is, the price per share multiplied by the number of shares outstanding. For a common stock investor, the return realized by holding one of these shares come from two sources:

- **Dividend payments**: dividends are distributions made by a corporations to its owners, typically in the form of cash or additional shares of the stock. The payment of dividends is not compulsory, usually young companies do not pay them, but as time goes on and they mature they start doing it.
- **Changes in the price of the stock**: while holding a share, if the price at a future date is higher than the purchase price there is a capital gain, and if the price at a future date is lower than the purchase price there is a capital loss. When a stockholder is calculating the return from holding a stock from the purchase date to a given point in time, it is actually calculating the gain or loss that he will get in case of selling the stock at this time.

The desire of stockholders to trade their shares has led to the establishment of stock exchanges, organizations which provide marketplaces for trading shares and other derived financial products. Those exchanges can be physical locations, such as the New York Stock Exchange (NYSE), or digital platforms, such as the NASDAQ. The biggest stock exchange in Spain is the Bolsa de Madrid, followed by the Borsa de Barcelona. These markets, according to the properties of the equities they trade are classified as secondary capital markets. Each stock exchange imposes its own listing requirements upon companies that want to be listed on that exchange. Such conditions may include minimum number of shares outstanding, minimum market capitalization, and minimum annual income.

Symbols	Open	Low	Close	High	Volume
ANA	94.400	93.450	93.800	94.400	36596.000
ACX	9.950	9.920	10.045	10.045	240375.000
ACS	35.070	35.020	35.650	35.800	396662.000
AENA	171.050	170.500	170.500	172.750	73393.000
AMS	72.700	72.200	72.800	73.300	148692.000
MTS	15.600	15.510	15.620	15.650	220308.000
SAB	1.038	1.034	1.040	1.046	5437795.000
SAN	3.720	3.685	3.730	3.743	19491991.000
BKIA	1.890	1.885	1.903	1.903	2016010.000
BKT	6.550	6.520	6.532	6.572	564270.000
BBVA	4.995	4.953	4.983	5.004	5624533.000
CABK	2.785	2.763	2.798	2.798	8844742.000
CLNX	38.480	37.840	38.370	38.870	221575.000
CIE	21.180	21.000	21.080	21.380	67281.000
ENG	23.040	22.690	22.740	23.110	504112.000
ENC	3.668	3.612	3.670	3.718	731818.000
ELE	24.200	23.730	23.790	24.200	676458.000
FER	26.550	26.500	26.970	26.970	1125539.000
GRF	31.600	31.290	31.430	31.640	198479.000
IAG	7.300	7.182	7.220	7.362	877608.000
IBE	9.216	9.174	9.180	9.250	6724758.000
ITX	31.520	31.160	31.450	31.710	726006.000
IDR	10.100	10.080	10.180	10.250	150119.000
COL	11.200	11.190	11.360	11.450	198963.000
MAP	2.382	2.360	2.360	2.402	1876403.000
TL5	5.520	5.504	5.660	5.660	292443.000
MEL	7.890	7.850	7.860	7.975	105154.000
MRL	12.650	12.560	12.790	12.790	199986.000
NTGY	22.610	22.400	22.400	22.710	742966.000
REE	18.115	17.860	17.925	18.115	411070.000
REP	14.100	13.930	13.930	14.100	2221149.000
SGRE	15.500	15.500	15.635	15.695	441154.000
TRE	23.400	23.300	23.800	23.800	107676.000
TEF	6.293	6.200	6.227	6.300	7967845.000
VIS	47.900	47.100	47.100	47.900	19412.000

Table 1: *Daily values for prices and volumes for the stocks composing the IBEX 35 as of December 31, 2019. IBEX 35 is the benchmark stock market index of the Bolsa de Madrid, Spain's principal stock exchange. It is a market capitalization weighted index comprising the 35 most liquid Spanish stocks traded in the Madrid Stock Exchange General Index and is reviewed twice annually. Source: own*

Nowadays stock trading is mostly electronic. This has been possible due to the liberalization of the markets and the technological advances for monitoring the markets and executing orders, which also has enforced the competitiveness between markets, lowering the costs and making trading available to the general public and the institutions.

2.3 Trading

As we said before, the globalization of stock market has made it available to both, the particular investor and the institutional investor. When an investor wants to buy or sell a share of common stock, the price and conditions under which the order is to be executed must be

communicated to a broker, then the broker arranges the trade and charges a commission to the investor. These orders can be of many types:

- Market order: is a buy or sell order that are to be executed immediately at current market price. A buy order is made at bid price (best offered price) and a sell order is made at ask price (best asked price). These is a gap between both prices known as bid-ask spread.
- Conditional order: prices can change between the time a order is made and the time a order is executed, so investors also may place orders specifying prices at which the are willing to buy or sell a share.

All these orders are included in the order book, which constitutes a record of the interest of buyers and sellers in a particular financial instrument, in this case stock shares. Then, a matching engine uses the book to determine which orders can be fully or partially executed. Consequently, the price of a stock fluctuates fundamentally due to the theory of supply and demand. The usual tool for studying these fluctuations are candlestick diagrams, such the one in Figure 3, which represent the open, close, low and high values for the price of a stock in a given time lapse.

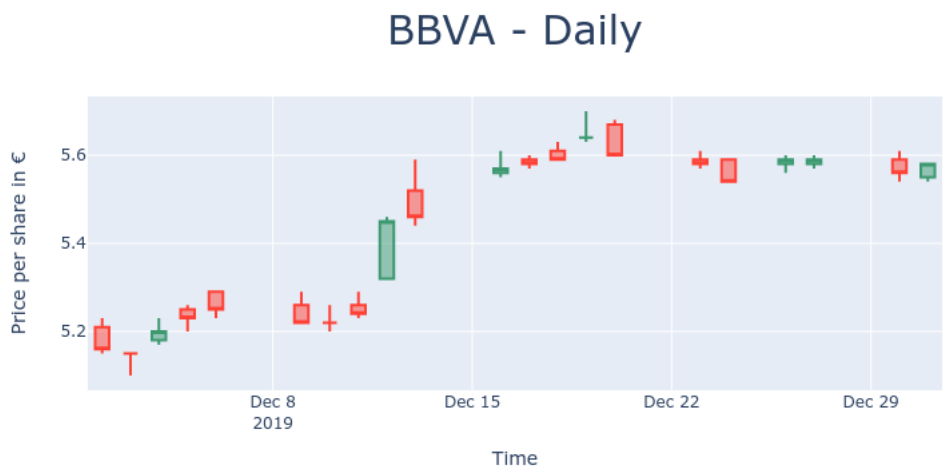

Figure 3: *Example of candlestick diagram for a stock. Data shown correspond to BBVA stock from December 2019. Each box, known as candlestick, contains lots of information. If the box is red it means that the closing price is lower than the opening price, consequently the lower edge of the box denotes the closing price of the stock this day, and the upper box denotes the opening price. If the box is green it means that the closing price is higher than the opening price, consequently the lower edge of the box denotes the opening price of the stock this day, and the upper box denotes the closing price. The lowest whisper, called lower shadow, always denotes the lowest price of the stock this day, and the upper whisper, called upper shadow, always the highest price. Source: own.*

In addition to supply and demand, there are many factors that influence the demand for a particular stock and the fields of fundamental analysis and technical analysis attempt to understand market conditions that lead to price changes in order to make profits. These two fields represent two different approaches to the market:

- **Fundamental analysis**: it consists on the analysis of the financial states of a company in order to determine its value. In this context stands out the idea of value investing [9], that supports that markets may incorrectly price a stock in the short run but the correct price

will eventually be reached at some point. Under this assumption, investors can make profits by buying/selling the wrongly priced stocks and then waiting for the market to reprice the security to sell/buy it and gain profit from the difference.

- **Technical analysis**: it consists on the use use of past market data, such as price and volume, to develop forecasts about the direction of prices [10]. It relies heavily in the use of candlestick diagrams, as the one shown in Figure 3, and techniques of time-series analays, but it also exploits more advanced techniques from other fields, such as statistics or signal analysis.

These two approaches were a hot topic for many years, but now there is some consensus in the fact that stock market's prices are essentially unpredictable. This is partially based on the efficient-market hypothesis [11], which states that all the available information of an asset is contained in its prices, which makes impossible to "beat the market" consistently since these prices will only react to new, unavailable, information. However, some of the methods deviated in these approaches can still be used to take advantage from other investors in the market and make profits. This idea led to the growth of a new approach, algorithmic trading.

Algorithmic trading consists in the application of the ideas provided by fundamental analysis and technical analysis to automate trading using computer programs. These algorithms can handle much more information at the same time, and much faster, than a human trader.

REVISAR Y ACABAR

Chan - Quantitative Trading: How to Build Your Own Algorithmic Trading Business

Chan - Algorithmic Trading: Winning Strategies and Their Rationale

AÑADIR ALGO DE HIGH FREQUENCY TRADING

2.4 Machine Learning

AÑADIR ALGO, CITANDO [1]

- **Stock price forecasting**: the prediction of stock prices is the most classic problem. Despite efficient market hypothesis states that it is not possible to predict stock prices and that stocks behave in random walk manner, technical analysts believe that most information about the stocks are reflected in recent prices and so if trends in the movements are observed then prices can be easily predicted. In addition, stock market's movements are affected by many macro-economical factors such as political events, firms' policies, general economic conditions, commodity price index, bank rates, bank exchange rates, investors' expectations, institutional investors' choices, movements of other stock markets, psychology of investors, ... The literature in this topic is extensive [12], [13], [14], but the publications of machine learning techniques used to forecast stock market movements can be categorised according to the machine learning technique used (ARIMA models, Suport Vector Machines, Neural Networks, Genetic Algorithms, ...), the forecasting timeframe (minutely, hourly, daily, monthly, ...), the input variables used (lagged index data, exchange rates, candlestick values, unemployment rates, ...), and the evaluation techniques employed. It is found that there is some consensus between researchers stressing the importance of stock index forecasting, since it gives an overall picture of economy, and the positive results of the use of Neural Networks because of their ability to learn very nonlinear relations.

- **Portfolio optimization**: another classical problem is the portfolio optimization, which describes the process of selecting the best portfolio of stocks out of the set of all portfolios being considered, according to some objective such as the maximization of expected return or the minimization of financial risk [15]. Some noticeable approaches to this problem can be the development of eigenporfolio theory, which compute orthogonal varying portfolios using Principal Component Analysis [16], or hierarchical risk parity models, which exploits hierarchical clustering algorithms to find clusters of the assets to re-allocate risk over them recursively [17]. These approaches generate portfolios with such a high quantity of assets that make unfeasible its treatment by human traders, but can get acceptable results when managed automatically.

AÑADIR ALGO QUE MOTIVE EL USO DEL RL, CITANDO [1] OTRA VEZ QUIZÁ

REVIEWS DE RL FOR TRADING

[18] [19]

3 Reinforcement Learning

In this section we are going to make a operational introduction to the field known as Reinforcement Learning [20]. First we are going to introduce Markov Decision Processes, which are the classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations. Then we are going to present the classical solutions to these problems in terms of Dynamic Programming, Monte Carlos methods and its combination in the form of Temporal-Difference methods. Finally, we are going to introduce the use of approximate solutions to these methods, which allow us to use state-of-the-art algorithms, such as Neural Networks, to solve efficiently real-world problems.

3.1 Markov Decision Processes

Now we are going to introduce Markov Decision Processes, or MDPs. MDPs are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal [21]. The learner and decision maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent seeks to maximize over time through its choice of actions.

3.1.1 Agent-environment interface

More formally, the agent and environment interact with each other in a sequence of discrete time-steps, $t = 0, 1, 2, 3, ...$ At each time step t, the agent receives some representation of the environment's state, $S_t \in \mathcal{S}$, and on that basis selects an action, $A_t \in \mathcal{A}(S_t)$. One time step later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R}$, and finds itself in a new state S_{t+1}. Then, the MDP and agent give rise to a sequence, or trajectory, that begins like this $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$

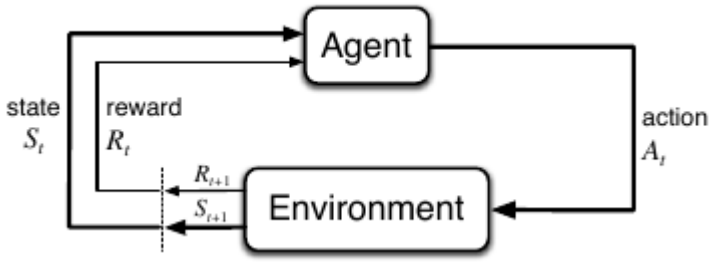

Figure 4: *Schema of the agent-environment interaction in a MDP. Source: [20]*

In a finite MDP, the sets of states $\mathcal{S}$, actions $\mathcal{A}$, and rewards $\mathcal{R}$ all have a finite number of elements. In this case, the random variables R_t and S_t have well defined discrete probability distributions dependent only on the preceding state and action:

$$p(s', r|s, a) \doteq \mathrm{P}(S_t = s', R_t = t|S_{t-1} = s, A_{t-1} = a) \tag{1}$$

so the function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \longrightarrow [0, 1]$, that defines the dynamics of the MDP, is an ordinary deterministic function of four arguments. And since it involves a conditional probability distribution, it satisfies that

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \tag{2}$$

which implies that the probability of each possible value for S_t and R_t depends only on the immediately preceding state and action, S_{t-1} and A_{t-1}, and not at all on earlier states and actions. This is best viewed a restriction not on the decision process, but on the state, which must include information about all aspects of the past agent-environment interaction that make a difference for the future. If it does, then the state is said to have the Markov property.

3.1.2 Rewards and returns

In reinforcement learning, the goal of the agent is formalized in terms of a special signal, called the reward, passing from the environment to the agent. At each time step t, the reward is a real number $R_t \in \mathbb{R}$. Informally, the agent's goal is to maximize the total amount of reward it receives, that is, maximizing not immediate reward, but cumulative reward in the long run. If we want it to solve a problem for us, we must provide rewards to it in such a way that in maximizing them the agent will also achieve our goals. It is thus critical that the rewards we set up truly indicate what we want to accomplish.

More formally, we seek to maximize the expected return, where the return is defined as some specific function of the reward sequence $G_t \doteq f(R_{t+1}, R_{t+2}, ..., R_T)$, where T is a final time step. In the simplest case the return can be defined as the sum of the rewards,

$$G_t \doteq R_{t+1} + ... + R_T = \sum_{k=t+1}^{T} R_k \tag{3}$$

so this approach makes sense in applications in which there is a natural notion of final time step T, that is, when the agent-environment interaction breaks naturally into subsequences, which are usually called episodes. Each episode ends in a special state called the terminal state S_T, followed by a reset to a standard starting state. Then the next episode begin independently of how the previous one ended. Thus, the episodes can all be considered to end in the same terminal state, with different rewards for the different outcomes. Tasks with episodes of this kind are called episodic tasks.

However, in many cases the agent-environment interaction does not break naturally into identifiable episodes, but goes on continually without limit. Then the above return formulation is problematic for continuing tasks because the final time step would be $T = \infty$ and then also the return. We can solve this problem by introducing the concept of discounting, with this the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses A_t to maximize the expected discounted return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=t+1}^{\infty} \gamma^{k-(t+1)} R_k \tag{4}$$

where $0 \leq \gamma \leq 1$ is called the discount rate. If $\gamma < 1$, the infinite sum has a finite value as long as the reward sequence is bounded, and in the extreme case of $\gamma = 0$ the agent will be concerned only with maximizing immediate rewards. As γ approaches 1, the return objective takes future rewards into account more strongly, that is, the agent becomes more farsighted. In this sense is important to note the following recursive rule

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{5}$$

It is useful to establish one notation that enables us to talk precisely about both cases simultaneously. This can be achieved by considering episode termination to be the entering of a special absorbing state that transitions only to itself and that generates only rewards of zero. Then we get the same return whether we sum over the first T rewards or over the full infinite sequence.Thus, we can define the return, in general, using the convention of omitting episode numbers when they are not needed, as

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-(t+1)} R_k \tag{6}$$

where we include the possibility of $T = \infty$ and $\gamma = 1$, but not both at the same time.

3.1.3 Policies and value functions

Reinforcement learning algorithms usually involve estimating how good it is for the agent to be in a given state, which is defined in terms of the expected return. The rewards the agent can

expect to receive in the future depend on what actions it will take, so value functions are defined with respect to particular ways of acting, called policies. More formally, a policy is a mapping $\pi : \mathcal{S} \times \mathcal{A} \longrightarrow [0, 1]$ from states and actions to probabilities of selecting each possible action, if the agent is following policy π at time t, then $\pi_t(a|s)$ is the probability that $A_t = a$ is chosen if we are in $S_t = s$. Like p, π is an ordinary function which defines a probability distribution over a $A(s)$ for $s \in S$. Reinforcement learning methods specify how the agent's policy is changed as a result of its experience.

Then, the value of a state, that is, of being in that state, is associated to a given policy. This is formalized by the definition of the value function v_π for policy π. The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t | S_t = s \right] \tag{7}$$

for $s \in \mathcal{S}$. Note that the value of the terminal state, if any, is always zero.

Similarly, we define the value of taking action a in state s under a policy π, denoted $q_\pi(s, a)$, as the expected return of taking the action a in the state s following policy π

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{8}$$

and we call q_π the action-value function for policy π. By definition the following relation is satisfied

$$\begin{aligned}
&v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t | S_t = s \right] && (9)\\
&\implies v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] && (10)\\
&\implies v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) && (11)
\end{aligned}$$

A fundamental property of value functions is that they satisfy recursive relationships similar to that which we have already established for the return

$$\begin{aligned}
&v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t | S_t = s \right] && (12)\\
&\implies v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s \right] && (13)\\
&\implies v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | S_t = s' \right] \right] && (14)\\
&\implies v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right] && (15)
\end{aligned}$$

which is known as the Bellman equation [22] and represent the relation between the value of an state and the subsequent states. Then, the value function $v_\pi(s)$ can be seen as the unique solution to the Bellman equation [20]. For the action-value function

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] \tag{16}$$
$$\implies q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1}+\gamma G_{t+1}|S_t=s,A_t=a\right] \tag{17}$$
$$\implies q_\pi(s,a) = \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r|s,a)\left[r+\gamma v_\pi(s')\right] \tag{18}$$
$$\implies q_\pi(s,a) = \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r|s,a)\left[r+\sum_{a'\in\mathcal{A}}\pi(a'|s')q_\pi(s',a')\right] \tag{19}$$

we obtain a similar result, which is denoted as the Bellman equation for the action-value function. Again, the action-value function $q_\pi(s,a)$ is the unique solution to the Bellman equation [20].

3.1.4 Optimality

Solving a reinforcement learning task means, roughly, finding a policy that achieves the most reward over the long run. For finite MDPs value functions define a partial ordering over policies, a policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π for all states. In other words, $\pi > \pi'$ if and only if $v_\pi(s) > v_{\pi'}(s)$ for all $s \in \mathcal{S}$. There is always at least one policy that is better than or equal to all other policies, the optimal policy. Although there may be more than one, we denote all the optimal policies by π_*. They share the same state-value function, called the optimal state-value function,

$$v_*(s) \doteq \max_\pi v_\pi(s) \tag{20}$$

for all $s \in \mathcal{S}$. Optimal policies also share the same optimal action-value function

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \tag{21}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

Because $v_*(s)$ is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values, but because it is the optimal one its consistency condition can be written in a special form without reference to any specific policy. Intuitively, the Bellman optimality equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state,

$$v_*(s) = \max_a q_*(s,a) \tag{22}$$
$$\implies v_*(s) = \max_a \mathbb{E}\left[G_t|S_t=s,A_t=a\right] \tag{23}$$
$$\implies v_*(s) = \max_a \mathbb{E}\left[R_{t+1}+\gamma G_{t+1}|S_t=s,A_t=a\right] \tag{24}$$
$$\implies v_*(s) = \max_a \mathbb{E}\left[R_{t+1}+\gamma v_*(S_{t+1})|S_t=s,A_t=a\right] \tag{25}$$
$$\implies v_*(s) = \max_a \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r|s,a)\left[r+\gamma v_*(s')\right] \tag{26}$$

or in terms of the action-value function

$$q_*(s,a) = \mathbb{E}\left[G_t | S_t = s, A_t = a\right] \tag{27}$$
$$\implies q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a\right] \tag{28}$$
$$\implies q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] \tag{29}$$
$$\implies q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a\right] \tag{30}$$
$$\implies q_*(s,a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) \left[r + \gamma \max_{a'} q_*(s', a')\right] \tag{31}$$

so for finite MDPs the Bellman optimality equation for v_*, or q_*, also has a unique solution [20]. If the dynamics p of the environment are known, then in principle one can solve this system of equations using any one of a variety of methods for solving systems of nonlinear equations.

Once one has v_*, it is relatively easy to determine an optimal policy. For each state s, there will be one or more actions at which the maximum is obtained in the Bellman optimality equation. Any policy that assigns nonzero probability only to these actions is an optimal policy. We can also think of this as a one-step search, if we have the optimal value function, v_* , then the actions that appear best after a one-step search will be optimal actions. So, by means of v_* , the optimal expected long-term return is turned into a quantity that is locally and immediately available for each state. Hence, a one-step-ahead search yields the long-term optimal actions.

Having q_* makes choosing optimal actions even easier, the agent does not even have to do a one-step-ahead search, for any state s, it can simply find any action that maximizes $q_*(s,a)$. The action-value function effectively caches the results of all one-step-ahead searches. It provides the optimal expected long-term return as a value that is locally and immediately available for each state–action pair.

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solving the reinforcement learning problem. However, this solution is rarely directly useful, it leads to an exhaustive search, looking ahead at all possibilities, computing their probabilities of occurrence and their desirabilities in terms of expected rewards. This solution relies on at least three assumptions that are rarely true in practice: we accurately know the dynamics of the environment, we have enough computational resources to complete the computation of the solution and the system satisfies the Markov property. But for the kinds of tasks in which we are interested, one is generally not able to implement this solution exactly because various combinations of these assumptions are violated. Then many reinforcement learning methods can be understood as approximately solving the Bellman optimality equation, using actual experienced transitions in place of knowledge of the expected transitions.

3.2 Dynamic Programming

The term dynamic programming (DP) refers to a collection of algorithms that can bused to compute optimal policies given a perfect model of the environment as a MDP. Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense, but they are still important theoretically to set the basis of the learning process.

First we consider how to compute the state-value function v_π for an arbitrary policy π, this is called policy evaluation. If the environment's dynamics are completely known, then the Bellman

equation is a system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknowns (the $v_\pi(s)$, $s \in \mathcal{S}$). So in principle, its solution is a straightforward computation, but, for our purposes, iterative solution methods are most suitable. Consider then a sequence of approximate value functions $v_0, v_1, v_2, ...$, each mapping $\mathcal{S}^+$ to $\mathbb{R}$. The initial approximation v_0 is chosen arbitrarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation as an update rule

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left[r + \gamma v_k(s') \right] \tag{32}$$

for all $s \in \mathcal{S}$. Each iteration updates the value of every state once to produce the new approximate value function. Clearly, $v_k = v_\pi$ is a fixed point for this update rule because the Bellman equation for v_π assures us of equality in this case. So, it is usually applied until the difference between two consecutive iterations of the value function estimate go beyond a given threshold θ.

Then, we can use the value function for a policy to find better policies, which is called policy improvement. Suppose we have determined the value function v_π for an arbitrary deterministic policy π. For some state s we would like to know whether or not we should change the policy to deterministically choose an action $a \neq \pi(s)$. Consider selecting a in s and thereafter following the existing policy, π. The value of this way of behaving is $q_\pi(s, a)$ and then, the key criterion is to check whether this is greater than or less than $v_\pi(s)$. If it is greater, that is, if it is better to select a once in s and thereafter follow π than it would be to follow π all the time, then one would expect it to be better still to select a every time s is encountered, and that the new policy would in fact be a better one overall. A natural extension is to consider changes at all states and to all possible actions, selecting at each state the action that appears best according to $q_\pi(s, a)$. In other words, to consider the new greedy policy π' given by

$$\pi'(s) = \text{argmax}_a \left[q_\pi(s, a) \right] \tag{33}$$

$$\implies \pi'(s) = \text{argmax}_a \left[\sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} p(s', r|s, a) \left[r + v_\pi(s') \right] \right] \tag{34}$$

where argmax_a denotes the value of a at which the expression that follows is maximized (with ties broken arbitrarily). Suppose the new greedy policy π' is as good as, but not better than, the old policy π, then $v_\pi = v_{\pi'}$, and then for all $s \in \mathcal{S}$:

$$v_{\pi'}(s) = \max_a \left[\mathbb{E} \left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a \right] \right] \tag{35}$$

$$\implies v_{\pi'}(s)' = \max_a \left[\sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} p(s', r|s, a) \left[r + v_\pi(s) \right] \right] \tag{36}$$

but this is the same as the Bellman optimality equation, and therefore, $v_{\pi'}$ must be v_* , and both π and π' must be optimal policies. Policy improvement thus must give us a strictly better policy except when the original policy is already optimal.

Finally, once a policy π has been evaluated to get v_π, and it is used to yield a better policy π', we can then compute $v_{\pi'}$ and improve it again to yield an even better π''. We can thus obtain a sequence of monotonically improving policies and value functions

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} ... \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}$$

where $\xrightarrow{E}$ denotes a policy evaluation and $\xrightarrow{I}$ denotes a policy improvement. Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). Because a finite MDP has only a finite number of policies, this process must converge to an optimal policy and optimal value function in a finite number of iterations. This way of finding an optimal policy is called policy iteration.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Figure 5: *Pseudocode for policy iteration algorithm. Source: [20].*

However, although we have complete knowledge of the environment, it would not be easy to apply these DP methods to compute the value function. DP methods require the distribution of next events, all of the probabilities must be computed before DP can be applied, and such computations are often complex and error-prone.

3.3 Monte Carlo methods

One solution to the problem of building a model of the environment can be found in Monte Carlo (MC) methods. MC methods are a broad class of algorithms that rely on repeated random sampling to obtain numerical results, the underlying concept is to use randomness to solve problems that might be deterministic in principle. In this sense MC methods, for Reinforcement Learning, require only experience-sample sequences of states, actions, and rewards from actual or simulated interaction with an environment in order to average over them. So, despite a model is required, the model need only generate sample transitions, not the complete probability distributions of all possible transitions that is required for DP.

First, as in DP, suppose we wish to estimate $v_\pi(s)$, the value of a state s under policy π, given a set of episodes obtained by following π and passing through s. Each occurrence of state s in an episode is called a visit to s. Then, the first-visit Monte Carlo methods estimate $v_\pi(s)$ as the

average of the returns following first visits to s in each episode, and the every-visit Monte Carlo methods average the returns following all visits to s in each episode. However, since a model of environment is not available, it is particularly useful to estimate action values rather than state values. So we are doing this computation for $q_\pi(s, a)$ taking into account visits to state-action pairs instead of states alone.

Then, after this policy evaluation, policy improvement is done by making the policy π greedy with respect to the current action value function q_π. For any action-value function q, the corresponding greedy policy is the one that, for each state s deterministically chooses an action with maximal action-value. Then policy improvement then can be done by constructing each π_{k+1} as the greedy policy with respect to q_k.This leads to a series of evaluations and improvements, analogous to the ones in the policy iteration algorithm in DP.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

Figure 6: *Pseudocode for first-visit Monte Carlo algorithm. Source: [20].*

3.4 Temporal-Difference Methods

Temporal-Difference (TD) learning is a combination of Monte Carlo (MC) ideas and Dynamic Programming (DP) ideas. On the one hand, like MC methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. And, on the other hand, like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome.

The problem with MC methods is that the updates of $V(S_t)$, that serve as a estimation of $v(s)$, are done using the expected return G_t as a target, that is,

$$V(S_t) \longleftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

so we have to wait until the end of the episode to compute the value of the returns G_t that allow us to update the values of $V(S_t)$. The idea beyond TD methods is not to wait until the ed of the episode but to make updates in each step, so we can use the recurrence property of the returns to perform a more useful update such as

$$V(S_t) \longleftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

which constitute the simplest TD method, the TD(0). This can be complicated including more steps, constituting the so called $T(\lambda)$ methods [20]. With this we get the best of both approaches, we get the online and fully-incremental computational power of DP methods, and the lack of necessity of a model of the environment from MC methods.

3.4.1 Exploration vs. exploitation

The policy evaluation problem for action values is to estimate $q_\pi(s, a)$, the expected return when starting in state s, taking action a, and thereafter following policy π. A state action pair (s, a) is said to be visited in an episode if the state s is visited and the action a is taken in it. The only complication is that many state-action pairs may never be visited. For policy evaluation to work for action-value pairs, we must assure continual exploration.

This problem is formulated more generally in the context of the so called exploration vs. exploitation trade-off, which shows the necessity to balance the exploitation of behaviours that the agent know that provide high return, with the exploration of new behaviors that the agent does not know its returns yet. The common approach to assuring that all state-action pairs are encountered, is to consider policies that are stochastic with a nonzero probability of selecting all action in each state. That is, these policies most of the time will select the best possible action, performing exploitation, but sometimes they will select a random action, performing exploration. There are two approaches to ensuring this:

- **On-policy methods**: which attempt to evaluate or improve the policy what is used to make decisions. In on-policy methods the policy is generally soft, meaning that $\pi(a|s) > 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$, but gradually shifted closer and closer to a deterministic optimal policy. A example of these policies are ε-greedy policies, meaning that most of the time they choose an action that has maximal estimated action values, but with probability ε they instead select an action at random. That is, all nongreedy actions are given the minimal probability of selection $\varepsilon/|\mathcal{A}|$, and the remaining bulk of probability $1 - \varepsilon + \varepsilon/|\mathcal{A}|$ is given to the greedy action. Using the natural notion of greedy policy for ε-soft policies, one is assured of improvement in every step. But with the drawback that we can only achieve the best policy among the ε-soft policies, but not the best policy overall.
- **Off-policy methods**: which evaluate or improve a policy different from that used to generate the data. The on-policy approach is actually a compromise, it learns action values not for the optimal policy, but for a near-optimal policy that still explores. A more straightforward approach is to use two policies, one that is learned about and that becomes the optimal policy, the target policy, and one that is used to generate behavior, the behavior policy. In this case we say that learning is from data "off" the target policy, and the overall process is termed off-policy learning. A typical case is when the target policy is made as the deterministic greedy policy with respect to the current estimate of the action-value function given by the behavior policy.

3.4.2 On-policy methods

On the one hand, the most straightforward application of the TD idea in the form of a on-policy method is the SARSA algorithm [23]. Is is defined by the update rule

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

which is done after every transition form a nonterminal state S_t. If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is set to zero. This rule uses every element of the quintuple $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$ that make up a transition from one state-action pair to another, which gives rise to the name SARSA for the algorithm. We continually estimate q_π for the policy π and at the same time change π towards greediness with respect to q_π. The detail of this algorithm is shown in Figure 7.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figure 7: *Pseudocode for Sarsa algorithm. Source: [20].*

3.4.3 Off-policy methods

On the other hand, we can build a off-policy version of this algorithm doing a little change to the update rule. This algorithm is usually known as Q-learning [24] and is defined by the update rule

$$Q(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

where the function Q directly approximates q_* the optimal action-values function, independent of the policy being followed, so it constitutes an off-policy algorithm. However, the policy still has still an effect since it determines which state-action pairs are visited and updated, but all that is required for correct convergence is that all pairs continue to be updated. The detail of this algorithm is shown in Figure 8.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 8: *Pseudocode for Q-learning algorithm. Source: [20].*

However, the problem of this algorithm is that, since it involves a maximization in the construction of their target policies, it can lead to a significant positive bias in the estimation of Q-vales, hurting perfomance and potentially leading to suboptimal policies. One way to view the problem is that it is due to using the same samples both to determine the maximizing action and to estimate its value. Te idea of doule Q-learning [25] solve this problem by dividing the samples in two sets and using them to learn two independent estimates, call them $Q_1(a)$ and $Q_2(s)$, each an estimate of the true value $q(a)$ for all $a \in \mathcal{A}$. We could then use one estimate, say Q_1, to determine the maximizing action $A^* = \text{argmax}_a Q_1(a)$, and other, say Q_2, to provide the estimate of its value $Q_2(A^*) = Q_2(\text{argmax}_a Q_1(a))$. This estimate will then be unbiased in the sense that $\mathbb{E}[Q_2(A^*)] = q(A^*)$. We can also repeat the process with the role of the two estimates reversed to yield a second unbiased estimate $Q_1(A^*) = Q_1(\text{argmax}_a Q_2(a))$, so the two approximate value functions are treated completely symmetrically eliminating the bias. The detail of this algorithm is shown in Figure 9.

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \text{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \text{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Figure 9: *Pseudocode for Double Q-learning algorithm. Source: [20].*

3.5 State-of-the-art

3.5.1 Approximate Solutions and Experience Replay

All the methods that we have presented are based in a tabular set-up of the data, so in principle they are able to being applied to problems with arbitrarily large state spaces. However,

in many of the tasks to which Reinforcement Learning can be useful the state space is combinatorial and enormous, or even continuous and non-easily discretizable. In such cases we cannot expect to find an optimal policy, or an optimal value function, even in the limit of infinite time and data. In addition to this, the other he problem with large state spaces is that almost every state encountered will never have been seen before, so to make sensible decisions in such states it is necessary to generalize from previous encounters with different states that are in some sense similar to the current one.

The goal instead is to find a good approximate solution using limited computational resources.

Fortunately, generalization from examples has already been extensively studied, and we do not need to invent totally new methods for use in reinforcement learning. To some extent we need only combine reinforcement learning methods with existing generalization methods. The kind of generalization we require is often called function approximation because it takes examples from a desired function (e.g., a value function) and attempts to generalize from them to construct an approximation of the entire function. Function approximation is an instance of supervised learning, the primary topic studied in machine learning, artificial neural networks, pattern recognition, and statistical curve fitting. In theory, any of the methods studied in these fields can be used in the role of function approximator within reinforcement learning algorithms, although in practice some fit more easily into this role than others.

This function is usually chosen to be a Deep Neural Network due to its hability to learn very complex and non-linear funcitons [26].

This is very related to the concept of Experience Replay [27]. In tabular Reinforcement Learning, the update is performed online, in a sample-by-sample manner, that is, every time a new transition is made, the value function is updated. The problem with this is that typically we will need a huge amount of episodes to achieve an optimal, or near optimal policy, in this new set-up. The reason for this is that, if weights are adjusted for one certain state-action pair, then unpredictable changes also occur at other places in the state-action space.

3.5.2 Deep Q-learning

[28] [29] [30] [31]

A Q-network can be trained be minimising a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}\left[(y_i - Q(s,a;\theta_i))^2\right] \tag{37}$$

where $y_i = \mathbb{E}_{s'\in\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i+1}|s,a)\right]$ is the target for iteration i and $\rho(s,a)$ is a probability distribution over sequences s and actions a that we refer as the behaviour distribution.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 10: *Pseudocode for Deep Q-learning algorithm. Source: [29].*

3.5.3 Others

[32] [33]

4 Experimental Design

[26] [34]

IMÁGENES LSTM : https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[35] [36]

4.1 Environment

4.2 Agents

4.3 Algorithms

5 Results

6 Conclusions

References

[1] Leo Breiman. "Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)". In: *Statist. Sci.* 16 (2001), pp. 199–231. DOI: 10.1214/ss/1009213726.

[2] Kenneth Brant, Jim Hare, and Svetlana Sicular. *Hype Cycle for Artificial Intelligence, 2019.* Technical Report G00369840. Gartner Research, 2019. URL: https://www.gartner.com/en/documents/3953603/hype-cycle-for-artificial-intelligence-2019.

[3] Gerald Tesauro. "Temporal Difference Learning and TD-Gammon". In: 38.3 (1995), pp. 58–68. DOI: 10.1145/203330.203343.

[4] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550 (2017), pp. 354–359. DOI: 10.1038/nature24270.

[5] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575 (2019), pp. 350–354. DOI: 10.1038/s41586-019-1724-z.

[6] Adrien Ecoffet et al. *Go-Explore: a New Approach for Hard-Exploration Problems.* 2019. arXiv: 1901.10995 [cs.LG].

[7] Frank J. Fabozzi, Franco P. Modigliani, and Frank J. Jones. *Foundations of Financial Markets and Institutions.* 4th. Pearson, 2013. ISBN: 9781292021775.

[8] Zvi Bodie, Alex Kane, and Alan Marcus. *Investments.* 11th. McGraw-Hill, 2018. ISBN: 9781259277177.

[9] Benjamin Graham and Jason Zweig. *The Intelligent Investor (Revised 1973 edition).* Harper Business, 2003. ISBN: 9780060752613.

[10] John J. Murphy. *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications.* New York Institute of Finance, 1999. ISBN: 9780735200661.

[11] Eugene F. Fama. "Efficient Capital Markets: A Review of Theory and Empirical Work". In: *The Journal of Finance* 25 (1970), pp. 383–417. DOI: 10.2307/2325486.

[12] Bjoern Krollner, Bruce Vanstone, and Gavin Finnie. "Financial time series forecasting with machine learning techniques: A survey". In: *Proceedings of the 18th European Symposium on Artificial Neural Networks (ESANN 2010)* (2010), pp. 25–30.

[13] Dev Shah, Haruna Isah, and Farhana Zulkernine. "Stock Market Analysis: A Review and Taxonomy of Prediction Techniques". In: *International Journal of Financial Studies* 7 (2019), pp. 383–417. DOI: 10.3390/ijfs7020026.

[14] Dongdong Lv et al. "An Empirical Study of Machine Learning Algorithms for Stock Daily Trading Strategy". In: *Mathematical Problems in Engineering* 2019 (2019).

[15] Harry Max Markowitz. "Portfolio Selection". In: *The Journal of Finance* 7 (1952), pp. 77–91. DOI: 10.2307/2975974.

[16] Marco Avellaneda and Jeong-Hyun Lee. "Statistical arbitrage in the US equities market". In: *Quantitative Finance* 10 (2010), pp. 761–782. DOI: 10.1080/14697680903124632.

[17] Marcos Lopez de Prado. "Building Diversified Portfolios that Outperform Out-of-Sample". In: *Journal of Portfolio Management* 10 (2016), pp. 761–782. DOI: 110.3905/jpm.2016.42.4.059.

[18] Thomas G. Fischer. *Reinforcement Learning in Financial Markets - a survey.* FAU Discussion Papers in Economics 12/2018. 2018. URL: http://hdl.handle.net/10419/183139.

[19] Terry Lingze Meng and Matloob Khushi. "Reinforcement Learning in Financial Markets". In: *Data* 4 (2019). DOI: 10.3390/data4030110.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* 2nd. Adaptive Computation and Machine Learning. MIT Press, 2018. ISBN: 9780262039246.

[21] Richard E. Bellman. "A Markovian Decision Process". In: *Indiana University Mathematics Journal* 6 (1957), pp. 679–684.

[22] Richard E. Bellman. *Dynamic Programming.* Adaptive Computation and Machine Learning. Princeton University Press, 1957. ISBN: 9780691146683.

[23] Gavin A. Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Technical Report CUED/F-INFENG/TR 166. Engineering Department, Cambridge University, 1994.

[24] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8 (1992), pp. 279–292. DOI: 10.1007/BF00992698.

[25] Hado van Hasselt. "Double Q-learning". In: *Advances in Neural Information Processing Systems 23*. 2010, pp. 2613–2621.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[27] Long-Ji Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Machine Learning* 8 (1992), pp. 293–321. DOI: 10.1007/BF00992699.

[28] Martin Riedmiller. "Neural Fitted q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method". In: *Proceedings of the 16th European Conference on Machine Learning*. ECML'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 317–328. ISBN: 3540292438. DOI: 10.1007/11564096_32.

[29] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[30] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[31] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: 1509.06461 [cs.LG].

[32] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning*. 2015. arXiv: 1511.06581 [cs.LG].

[33] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].

[34] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[35] Greg Brockman et al. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG]. URL: https://github.com/openai/gym.

[36] Martí Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://github.com/tensorflow/tensorflow.