Reinforcement Learning

José Luis Aguilar Charfén Simone Reynoso Donzelli

March 29, 2022

1 Introduction

In the recent years,[1] Reinforcement Learning (RL) has become an important research topic. Its applicability in various areas, such as self-driving cars and robotics, videogames, finance, optimization processes, dynamic programming or healthcare systems, has risen its popularity. RL is used in these areas, mainly to access and process in a fast way large amounts of data.

Considered as a Machine Learning (ML) technique, RL enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. The learning by reinforcement combines two main tasks; the first is exploring new situations; and the second involves the use of the experience acquired during the first situation to make better decisions, decisions that improve with a given time, and have the main objective to achieve a final task.

Due to its sudden boom, many RL examples can be found on the internet. However, despite the number, the vast majority look forced, mainly due to the fact that common ML examples are wrongly fitted to these RL examples.Though both supervised and reinforcement learning use mapping between input and output, unlike supervised learning where the feedback provided to the agent is the correct set of actions for performing a task, reinforcement learning uses rewards and punishments as signals for positive and negative behavior. As compared to unsupervised learning, reinforcement learning is different in terms of goals. While the goal in unsupervised learning is to find similarities and differences between data points, in the case of reinforcement learning the goal is to find a suitable action model that would maximize the total cumulative reward of the agent. In summary, RL is better suited for applications in which sequential and complex decisions are needed, in order to obtain a long-term goal. ML models might get to a certain point; however, RL excels in environments with direct feedback.

1.1 Basic concepts

Some of the key elements found in an RL [1] model are:

1. Agent: The RL algorithm that learns from trial and error.
2. Environment: World in which the agent operates. May be physical in nature.
3. State (s_i): Current situation of the agent, which is in the state space $\mathcal{S}$ if discrete, and $\mathcal{X}$ if continuous in time.
4. Action (a_i): All the possible steps the agent can take, part of the action space $\mathcal{A}$.
5. Reward (r_i): Feedback from the environment.
6. Policy (π): The approach that the agent uses to determine the next action based on the current state. It does not need to be deterministic, it can also be stochastic in nature.

7. Value: Future reward that an agent would receive by taking an action in a particular state.

The relationship between all these elements can be schematically represented as follows:

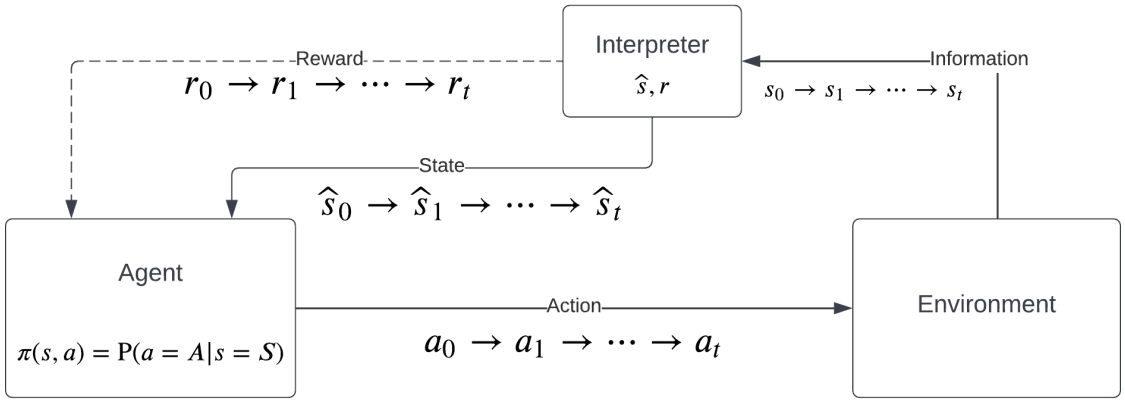

Figure 1: *Reinforcement learning basic diagram.*

Besides the basic concepts declared above, it's important to define some additional ones, which often appear in the models used in RL [1]:

- Discount factor (γ): the discount factor essentially determines how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future. If $\gamma = 0$, the agent will be completely myopic and only learn about actions that produce an immediate reward. If $\gamma = 1$, the agent will evaluate each of its actions based on the sum total of all of its future rewards.

- Learning rate (α): The learning rate is a parameter that controls how much the model changes, in response to the estimated error each time the model weights are updated. Choosing the correct learning rate is very difficult; a large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights; while a smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train.

- Exploitation: using the already exploited information to heighten the rewards

- Exploration: exploring and capturing more information about an environment, regardless to the low reward

Usually the way exploration/exploitation is prioritized is through an $\epsilon-$greedy algorithm. Basically, this means that the probability of choosing exploration over exploitation is ϵ, while the probability of exploiting known information is $1 - \epsilon$, where ϵ tends to –but is not required to– be small (> 0.1).

1.2 Context

The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. When an infant plays, waves its arms, or looks about, it has no teacher, but it does have a senses, a connection to the environment it lives in. Exercising this connection produces a wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals.

Such interactions provide valuable information to go along in life. Whether we are learning to drive a car or to hold a conversation, we attempt to understand how our actions affect our environment, and we seek to influence what happens through our behavior. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence.

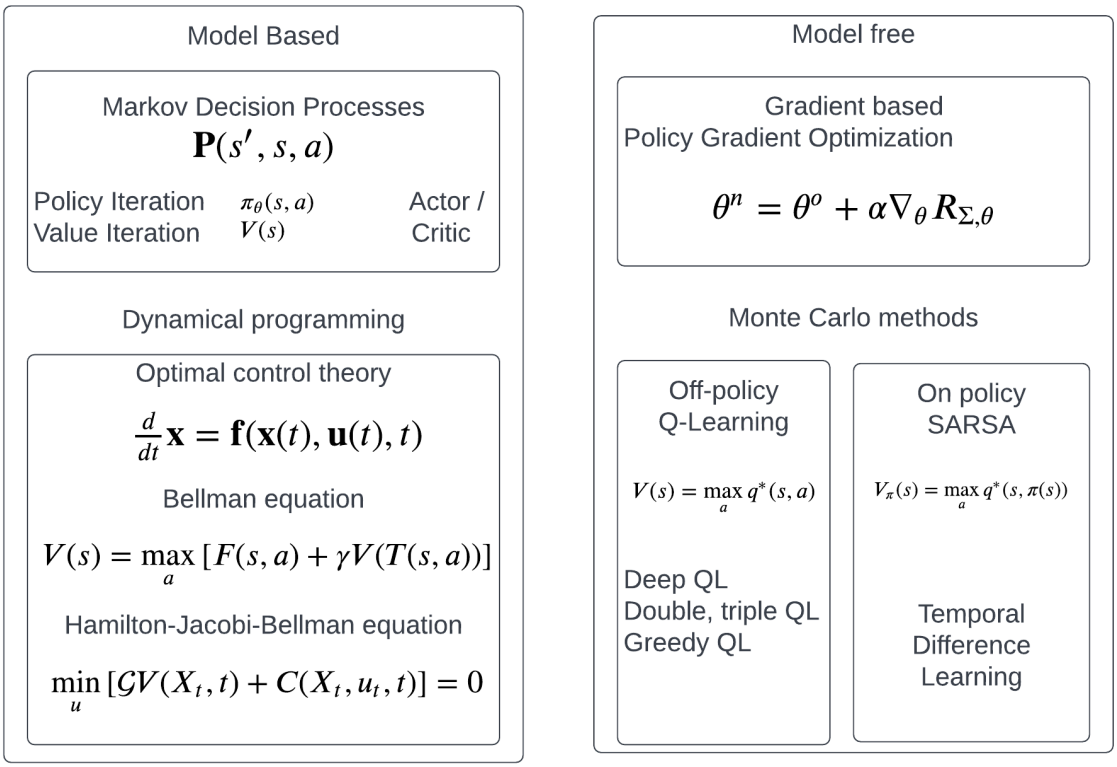

Figure 2: *Brief map of reinforcement learning, based on [2].*

One of the first reinforcement learning algorithms could be argued to be classical conditioning, first discussed by Ivan Pavlov, in which given a neutral stimulus (a bell) and a strong positive stimulus (food, for example), Pavlov could make his dog respond by salivating due to the neutral stimulus (an action, trying to get the strong positive stimulus, or reward); learning by trial and error.

On another hand, the development of dynamical programming for simplifying complex problems, and in particular its application in finding solutions to the value equations posed in optimal control helped to develop certain tools which are fundamental in RL. Even if in its early days it did not involve learning, the work made by Richard Bellman and others in developing the theory of Hamilton and Jacobi was extremely important, and today is frequently used in several algorithms. This approach uses the concepts of a system's state and a value function, and by solving the Bellman equation (which is described further in this text), then the tools of dynamical programming enabled solving general stochastic general control problems, and by finding the actions taken for this control, then policies could be implemented.

These ideas on how learning can occur, and on tools that were essential for reinforcement learning, allowed for the development of this field. On figure 2, a small map of subfields in reinforcement learning can be seen. Topics seen in this map will be briefly discussed on this text.

2 Model based learning

2.1 Markov Decision Processes (MDP)

Markov decision process (MDP) is one of the most important topics in RL, mainly because all the model based and some of the model free learning derive from its properties; to better understand MDP, some basic concepts must be presented.

In a Markov process, such as any RL process, the interaction between the environment and the state were the agent stands, causes an action[1]. The Markov Property:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t] \tag{1}$$

Describes is that the transition from one state to another, S_t to S_{t+1} , is entirely independent of the past, meaning that previous states are not stored or used to perform an action to change state. This transition has a certain probability, commonly known as state transition probability; for a transition of states following the Markov Property, the probability of this movement is described as:

$$\mathcal{P}_{s,s'} = \mathbb{P}[S_{t+1} = s'|S_t = s] \tag{2}$$

In Markov Chains all the states obey the Markov Property, and the transition probability for all possible movements $\mathcal{P}_{s,s'}$ can be written in a matrix form, where each row in the matrix represents the probability of moving form one given state to another. It's important to point out, that the sum of each row equals 1.

Before going to Markov Reward process (MRPs), it's important to define some basic concepts that will help to better understand MRPs. Rewards are the numerical values that the agent receives on performing some action at some state in the environment. The numerical value can be positive or negative based on the actions of the agent. In RL its cared to maximize the cumulative reward, instead of the reward the agent receives from the current state. The total sum of rewards is commonly referred as return (G_t). In the return function (G_t), all the rewards have the same *weight*, if different *weights* are set to be assigned, the discount factor may be included in the return function, having the following expression:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3}$$

Once defined reward and the return function, the Markov Reward Process can be derived form all these definitions. MRPs is a function which depends on the states, the transition probability, the rewards and the discount factor. Defined as the expected value of the immediate reward that one will get in a specific state, mathematically this can be expressed as:

$$R_s = \mathbb{E}[R_{t+1}|S_t] \tag{4}$$

The main focus of the MRPs is to maximize the accumulated value of all this expected rewards. Discount factors are commonly used in Markov reward and decision processes; the main reason behind this, is to avoid infinite return loops, to properly represent future uncertainties, and it results mathematically convenient for solving these type of problems. Intuitively it also makes sense to value near future rewards more than those far in the future, which is a concept frequently used in other disciplines such as economics and finance.

Before actually defining MDPs, it's very important to remark two extra concepts:

1. Policy function
2. Value function

In one hand, the policy is a simple function, that defines a probability distribution over actions $a \in A$ for each state $s \in S$. If an agent at time t follows a policy π then $\pi(a|s)$ is the probability of the agent taking action a at a particular time step t.In Reinforcement Learning the experience of the agent determines the change in policy. Mathematically, a policy is defined as follows:

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \tag{5}$$

On the other hand, the value function returns the long-term value of a state (s). Defined as follows:

$$\mathcal{V}_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{6}$$

This equation gives the expected returns starting from the state(s) and going to successor states thereafter, with the policy π. One thing to note is the returns are stochastic whereas the value of a state is not stochastic.

Finally, we get to the Markov Decision Process, which it's basically the same as MRPs, apart from the additional property of action taking by the agent. Changing equations 2 and 4 applying this property, we get the following equations, which describe MDPs:

$$\mathcal{P}^a_{s,s'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] \tag{7}$$

$$R^a_s = \mathbb{E}[R_{t+1}|S_t, A_t = a] \tag{8}$$

These type of transformation, adding the dependence of the action in the functions, can be applied to all the functions described earlier. Then, in summary, a Markov Decision Process can be described as a 4-tuple in the following manner:

$$MDP := (\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{s,s'}, R^a_s) \tag{9}$$

Where $\mathcal{S}$ is the state space, or set of all possible states, and $\mathcal{A}$ is the action space, or set of all possible actions in each state. A special cases arises with the state-value function, which gives the so called q-learning function, RL method which will be described in the following sections of the paper.

Two derived algorithms used in Markov Decision Processes are value iteration and policy iteration, whose names imply how we arrive to the optimal policy. First considering the value iteration:

$$V(s) = \max_a \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) \left(R(s', s, a) + \gamma V(s')\right) \tag{10}$$

Describes the optimal value that can be achieved given the state and action taken. It is described as a Bellman equation, which is further discussed in the next section of the text. Then, to extract the optimal policy, we only need to obtain the argument in the following fashion:

$$\pi(s, a) = \arg\max_a \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) \left(R(s', s, a) + \gamma V(s')\right) \tag{11}$$

On the other hand, policy iteration is a two step iteration algorithm, where intuitively we first lock in a policy π, and iteratively update the value function. Instead of adding over all possible actions a, we simply apply what we think our best policy is and update the values, described in the following fashion:

$$V_\pi(s) = \max_a \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, \pi(s)) \left(R(s', s, \pi(s)) + \gamma V_\pi(s')\right) \tag{12}$$

$$\pi(s) = \arg\max_a \mathbb{E}\left[R(s', s, a) + \gamma V_\pi(s')\right] \tag{13}$$

Usually, $V_\pi(s)$ doesn't start as an accurate approximation of the value function, but if it results good enough to improve the policy, then after improving the policy the iteration starts over, updating value function and afterwards the policy.

2.2 Optimal Control – the Hamilton-Jacobi-Bellman (HJB) equation

Before introducing the Hamilton-Jacobi-Bellman (HJB) equation, we'll first see the discrete counterpart, the Bellman equation. The Bellman equation is a necessary condition for optimality associated with mathematical optimization in dynamic programming. This equation breaks the optimization problem into a sequence of simpler subproblems.

Let the state s at a given time step n be s_n, and that the initial state $s_0$at time 0 is given. Let $a_n \in \mathcal{A}(s_n)$ be the action taken at time n, and the set of possible actions depends of the state of the system. Let the evolution of the system change in the following manner: $s_{n+1} = T(s_n, a_n)$, evolving when the action a_n is taken, and that the payoff from taking action a_n in state s_n is $F(s_n, a_n)$. Finally, let's assume that payoffs further in time are less valuable in the present, represented by a discount factor $\gamma \in (0, 1)$. Then, the optimal value problem can be described in the following fashion:

$$V(s_0) = \max_{\{a_n\}_0^\infty} \sum_{n=0}^{\infty} \gamma^n F(s_n, a_n) \tag{14}$$

Where $V(s_0)$ is the value function, subject to the assumed constraints. This formulation is equivalent to the return function, stated in equation 3 The equivalent formulation, when describing a stochastic policy is the following:

$$V^\pi(s_0) = \max_{\{a_n\}_0^\infty} \mathbb{E}\left[\sum_{n=0}^{\infty} \gamma^n F(s_n, a_n)\right] \tag{15}$$

The Bellman principle of optimality states that for an optimal policy, whatever the initial state and action are, the following decisions taken must also be optimal with regard to the next state and action. Then, going further a single step in time, the problem can be described in the following fashion:

$$V(s_0) = \max_{a_0} \left\{ F(s_0, a_0) + \beta \left[\max_{\{a_n\}_1^\infty} \sum_{n=1}^{\infty} \gamma^n F(s_n, a_n) \right] \right\} \tag{16}$$

Subject to the constraints $a_0 \in \mathcal{A}(s_0), x_1 = T(s_0, a_0)$. What's inside the brackets on the right side of the equation is merely the value of time 1, and describes the same decision problem. Therefore, this equation can be written in a recursive manner, in the following way:

$$V(s_0) = \max_{a_0} \{F(s_0, a_0) + \gamma V(s_1)\} \tag{17}$$

Subject to the same constraints. Dropping the subscripts and plugging in the value of the next state then gives the following equation:

$$V(s) = \max_{a} \{F(s, a) + \gamma V(T(s, a))\} \tag{18}$$

Then, solving the Bellman equation means finding the value function V. Then, if the optimal value function is obtained, we can also find $a(s)$ –the argument of the maximum–, the optimal action to be taken as a function of the state, which can also be called the *policy function* $\pi(s)$. Having known this optimality principle, then we can introduce the Hamilton-Jacobi-Bellman equation.

The HJB equation gives a necessary and sufficient condition for optimality of a control with respect to a loss function[3]. Considering the following representation of a dynamical system:

$$\frac{d}{dt}\mathbf{x} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{19}$$

Where $\mathbf{x}(t)$ represents the state, t represents time, and $\mathbf{u}(t)$ represents the control action, then the optimal control problem over an interval of time $[0, T]$ can be described in the following way:

$$V_T(x(0), 0) = \min_{\mathbf{u}} \left\{ \int_0^T C[\mathbf{x}(t), \mathbf{u}(t)]dt + D[\mathbf{x}(T)]] \right\} \tag{20}$$

Where $V_T(\mathbf{x}(t), t)$ is the value function, $C[\mathbf{x}(t), \mathbf{u}(t)]$ is the cost rate function given the state and control action, and the initial state is assumed to be given. The state of the system is subject to equation 19. Then, by applying the Bellman optimality principle, and going forward in time a step Δt:

$$V_T(\mathbf{x}(t), t) = \min_u \left\{ V_T(x(t + \Delta t), t + \Delta t) \int_t^{T+\Delta t} C[\mathbf{x}(s), \mathbf{u}(s)]ds \right\} \tag{21}$$

Subject to the terminal condition $V_T(x(T), T) = D[x(T)]$. Taking the Taylor expansion to the first order of the value function at a given state x and time t:

$$V_T(\mathbf{x}(t), t) = V_T(\mathrm{x}, \mathrm{t}) + \frac{\partial}{\partial t} V_T(\mathbf{x}(t), t)\Big|_{\mathrm{x,t}} + \frac{\partial}{\partial \mathbf{x}} V_T(\mathbf{x}(t), t)\Big|_{\mathrm{x,t}} \mathbf{x}'(t) + o(\Delta t) \tag{22}$$

Then, taking the limit as $\Delta t \to 0$ of equation 21, we obtain:

$$V_T(\mathbf{x}(t), t) = \min_u \left\{ V_T(\mathrm{x}, \mathrm{t}) + \frac{\partial}{\partial t} V_T(\mathbf{x}(t), t)\Big|_{\mathrm{x,t}} + \frac{\partial}{\partial \mathbf{x}} V_T(\mathbf{x}(t), t)\Big|_{\mathrm{x,t}} \mathbf{x}'(t) + C[\mathbf{x}(t), \mathbf{u}(t)] \right\} \tag{23}$$

$$= V_T(\mathrm{x}, \mathrm{t}) + \frac{\partial}{\partial t} V_T(\mathbf{x}(t), t)\Big|_{\mathrm{x,t}} + \min_u \left\{ \frac{\partial}{\partial \mathbf{x}} V_T(\mathbf{x}(t), t)\Big|_{\mathrm{x,t}} \mathbf{x}'(t) + C[\mathbf{x}(t), \mathbf{u}(t)] \right\} \tag{24}$$

$$0 = \frac{\partial}{\partial t} V_T(\mathbf{x}(t), t) + \min_u \left\{ \frac{\partial}{\partial \mathbf{x}} V_T(\mathbf{x}(t), t) \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) + C[\mathbf{x}(t), \mathbf{u}(t)] \right\} \tag{25}$$

Which is the HJB equation, valid for continuous time. The relationship between this optimal control problem and reinforcement learning was not noticed at first, but its importance results evident when trying to model values for given actions or policies ($\mathbf{u}(t)$ results analogous to the policies taken by an agent). Even if this equation is usually solved backwards in time, its theoretical value is large for model based reinforcement learning, due to the fact that the Bellman function can be approximated in general, frequently (but not restricted) with artificial neural networks.

There is also a stochastic equivalent, which can be derived using the same Bellman optimality principle. It's described by the following equation:

$$\min_{u_t} \{ \mathcal{G} V(X_t, t) + C(X_t, u_t, t) \} = 0 \tag{26}$$

Subject to the terminal condition $V(X_T, T) = D(X_T)$, where $\mathcal{G}$ is the stochastic differentiation operator, X_t is the stochastic process to optimize, and u_t is the steering, and the value and cost functions are described equivalently.

3 Model-free learning

Model-free learning does not need an a-priori model of how the environment in which the agent acts works; instead, it attempts to learn without this model, directly learning through value functions.

3.1 Gradient free

Gradient free methods do not parametrize nor attempt to optimize parameters in any way the value functions used to obtain the optimal policies. They attempt to solve Bellman equations, with small differences between each class of algorithms.[4] The main group of gradient-free methods involve temporal-difference (TD) learning (either in one step or several steps; all algorithms can be extended from their described form in this text), but Monte Carlo methods are also used (not described in this text, but could be understood as an $\infty-$step TD learning).

3.1.1 Off policy: Q-Learning

Q-Learning is based on finding a value or "quality" function $Q(s,a)$ such that it follows the updating rule[4]:

$$Q_{t+1}(s_t,A_t) \leftarrow Q_t(s_t,A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,A_t)\right] \tag{27}$$

$$Q_{t+1}(s_t,A_t) \leftarrow (1-\alpha)Q_t(s_t,A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q_t(s_{t+1},a)\right] \tag{28}$$

In this case, the learned action-value function, Q, directly approximates q^*, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enables convergence proofs. This is a minimal requirement in the sense that any method guaranteed to find optimal behavior in the general case must require it. Under this assumption and a variant of the usual stochastic approximation conditions on the sequence of step-size parameters, Q has been shown to converge with probability 1 to q^*.[5], [6] It can be seen that this is just a Bellman equation. The relationship between the traditional formulation, given at equation 18 and the optimal action-value function is the following:

$$V(s) = \max_a q^*(s,a), \quad \forall s \in \mathcal{S} \tag{29}$$

And the manner in which the optimal policy is obtained is the following:

$$\pi^*(s) = \arg\max_a q^*(s,a), \quad \forall s \in \mathcal{S} \tag{30}$$

Theorem 1. *Given some rewards $\|R_n\| \leq M$, learning rates $0 \leq \alpha_n < 1$, such that:*

$$\sum_{i=1}^{\infty} \alpha_{n^i(x,a)} = \infty, \sum_{i=1}^{\infty} \left[\alpha_{n^i(x,a)}\right]^2 < \infty, \forall x, a,$$

Then $Q_n(x,a) \to q^*(x,a)$ when $n \to \infty$ with probability 1.

The demonstration can be found in [5], [6].

3.1.2 On policy

On-policy methods, even if as their off-policy methods they also work in temporal-difference learning, differ slightly from their counterparts in focus. The first step is to learn an action-value function rather than a state-value function. Instead of finding the $q-$function, we estimate q_π for the current policy π for states s and actions a, and at the same time change π toward greediness with respect to q_π. Described as an equation, similar to Q-Learning, the updating rule is the following[4]:

$$Q_{t+1}(s_t,A_t) \leftarrow Q_t(s_t,A_t) + \alpha\left[R_{t+1} + \gamma Q_t(s_{t+1},A_{t+1}) - Q_t(s_t,A_t)\right] \tag{31}$$

The previous approach is known as $SARSA$, or state-action-reward-state-action, due to the quintuple of events $(s_t, A_t, R_{t+1}, s_{t+1}, A_{t+1})$. It is a Bellman equation, just as in the case

of Q-Learning. The main difference between approaches is the same as when studying value iteration and policy iteration in Markov Decision Processes: whether when optimizing, the value function is updated directly (value iteration, or Q-Learning), or if a policy is implemented an then the action-value function is updated, and the policy afterwards (policy iteration, or on-policy methods).

3.2 Gradient based

3.2.1 Policy Gradient Optimization

Policy gradient optimization is based on the fact that if the policy, value or quality (Q) function can be parametrized with a set of parameters θ, then techniques used for gradient optimization may be used. Some examples include Newton's or quasi-Newton methods, or gradient descent in order to get an estimate of the policy function. This technique can be described as the following equation:

$$\theta_n = \theta_o + \alpha \nabla_\theta R_{\Sigma,\theta} \tag{32}$$

References

[1] P. W. P. D, *Reinforcement Learning: Industrial Applications of Intelligent Agents*, 1st edition. S.l.: O'Reilly Media, Dec. 1, 2020, 408 pp., ISBN: 978-1-09-811483-1.

[2] Steve Brunton, director, *Reinforcement Learning Series: Overview of Methods*, Jan. 3, 2022. [Online]. Available: `https://www.youtube.com/watch?v=i7q8bISGwMQ` (visited on 03/28/2022).

[3] J. Yong and X. Y. Zhou, *Stochastic Controls: Hamiltonian Systems and HJB Equations*. Springer Science & Business Media, Jun. 22, 1999, 472 pp., ISBN: 978-0-387-98723-1.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., red. by F. Bach, ser. Adaptive Computation and Machine Learning Series. Cambridge, MA, USA: A Bradford Book, Nov. 13, 2018, 552 pp., ISBN: 978-0-262-03924-6.

[5] F. S. Melo, "Convergence of Q-learning: A simple proof," p. 4,

[6] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1, 1992, ISSN: 1573-0565. DOI: `10.1007/BF00992698`. [Online]. Available: `https://doi.org/10.1007/BF00992698` (visited on 03/26/2022).