International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys.13, No.1 (May-23)

http://dx.doi.org/10.12785/ijcds/1301118

Reinforcement Learning: A review

Hanae MOUSSAOUI[1], Nabil EL AKKAD[1] and Mohamed BENSLIMANE[2]

[1]*LISA, Engineering, Systems, and Applications Laboratory, ENSA of Fez, Sidi Mohamed Ben Abdellah University, Fez, Morocco*
[2]*LTI Laboratory, EST of Fez Sidi Mohamed Ben Abdellah University, Fez, Morocco*

Received 28 Sep. 2022, Revised 1 May. 2023, Accepted 23 May. 2023, Published 30 May. 2023

Abstract: Reinforcement learning is considered a sort of machine learning that acquires knowledge of solving problems using the trial-and-error technique. The process starts with the main actor that is the agent interacting with a given environment and attempting to achieve a multi-step goal within this environment. The environment is characterized by a state that the agent detects and examines. On the other hand, due to the agent's several actions, the environment's state changes according to these modifications. Eventually, and at this stage, the agent gets reward signals as it proceeds nearer to its goal. The agent uses these rewards signals to determine which actions were successful and which actions were not. The state action is then repeated and the reward is looped until the agent learns how to operate effectively within the environment using the trial-and-error concept. The agent's main objective is to learn how to always choose the right action given any state of the environment that leads it closer to its goal. In this paper, we gathered all the methods used in the literature. Multi-armed bandits, the Markov decision process, Monte Carlo methods, dynamic programming as well as temporal-difference learning are some of the corresponding methods used to solve reinforcement learning issues. The current paper is organized and structured as follows: we'll start with an introduction followed by a reinforcement learning section where we discussed all the methods and techniques used in the literature. Furthermore, the third section will be about deep reinforcement learning, here we gathered deep reinforcement learning techniques. In the fourth section, we will summarize the reinforcement and deep reinforcement learning algorithms in detail. Furthermore, we will finalize the article with a discussion and a conclusion.

Keywords: Reinforcement learning, Deep reinforcement learning, multi-armed bandits, Markov decision process, Monte Carlo methods, dynamic programming, Deep model predictive control, Actor-critic network, temporal-difference learning, Deep Q-learning, Deep dueling q network, Advantage actor-critic network.

1. INTRODUCTION

Recently, reinforcement learning [1] has exploded through the birth of many challenging projects. Robotic arm manipulation, 1v1 Dota, and vintage Atari games are one of the most important implementations using reinforcement and deep reinforcement learning. Moreover, the winnings of supervised deep learning [2] have continued to cumulate, giving the example of the ImageNet classification challenge in 2012. In addition, researchers from many areas have been involved in deep neural networks to solve an important range of new projects such as comprehending intelligent attitudes and actions within a convoluted dynamic environment. Reinforcement learning is considered as a subfield of machine learning [3] and is considered one of the most favorable and helpful orientations to attain a high level of intelligence in robotics behavior. Nearly every researcher uses supervised learning [4] in machine learning projects and applications ([5], [6]), where an input is given to the neural network model while knowing exactly what output the model should produce, subsequently, gradients will be calculated by utilizing the backpropagation method to train the network to output the results. Unfortunately, by using supervised learning elements should be collected in the created dataset. This step of the process is not always easy for doing, regarding the amount number of elements that should be gathered in the dataset. Furthermore, the neural network will be trained to straightforwardly emulate the human player records and actions, knowing that an agent can never perform well at playing compared to a human. Thereafter, reinforcement learning ([7], [8], [9]) remains the idealistic choice to handle the issue of an agent being a better player than a human gamer, and also to learn how to play it by itself with no human interaction. Reinforcement learning and the normal framework of supervised learning are similar, we constantly have an input frame that is run through certain neural network models, and afterward, the network generates an output action. In reinforcement learning, and owing to the absence of a dataset it'll be impossible to recognize the target label, unlike supervised learning. Likewise, a Policy Network is defined as a network that modifies input frames to output actions, and among the uncomplicated ways or techniques to train the Policy Network is by employing the Policy Gradient. Moreover, the output of the network will consist of two numbers, the

E-mail address: hanae.moussaoui1@usmba.ac.ma, nabil.elakkad@usmba.ac.ma, mohamed.benslimane@usmba.ac.ma

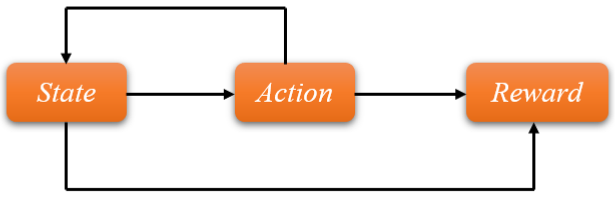

Figure 1. Policy optimization process

probability of each decision, and what to do while training. In reinforcement learning we want to authorize the agent to acquire knowledge completely by itself [9], the only given feedback, in this case, is the scoreboard. The agent in this process receives a reward of (+1) whenever it achieves to mark or score a goal, or else, in the case when the opponent was the first to score the goal, the agent is then rewarded with a penalty of (-1). The noteworthy aim of the agent is to optimize its policy to increase the amount received rewards. The policy network will be trained by first of all gathering a bunch of experiences by running a whole set of game frames through the network. Then, pick out random actions and feed them back into the system as shown in Figure 1.

The agent is randomly going to select a whole succession of actions that lead to scoring a goal. Multiple sequences of actions will be selected randomly by the agent to score goals, subsequently, it will receive a reward. A key thing to retain here is that for each episode, and in any case, we can calculate the gradients to make the agent's action, either we need a positive or a negative reward. Ordinarily, the normal gradients are chosen to increment the likelihood of the taken actions in the future. The same gradient will also be used but this time multiplied by (-1) whenever we obtain an unfavorable reward. In this case, and by using this minus sign, every action that is taken in a bad episode will be less likely. Consequently, during the policy network training all the actions that lead to an unfavorable or negative reward will be filtered out, thereafter, the positive ones will be more likely. However, the problem with policy gradients is that if in an episode the agent was lucky by taking good actions, it did badly in the last one. In this case, the gradient policy is going to presume that since that episode is missed, every taken action must be bad and reduce the probability of taking those actions in the future [10]. In reinforcement learning, this is called the "Credit Assignment problem". This issue is entirely related to the fact that we have what it's called a "Sparse Reward setting". Rather than gaining a reward for every single action, we only acquire a reward posterior to an entire episode, and the agent needs to find out in what part of its action sequence the reward has been created. In reinforcement learning, algorithms ([11], [12]) must have a training time before they can learn some useful behavior. In some extreme cases, the sparse reward setting fails; a famous example is the Montezuma's Revenge game ([13], [14]), where the main motivation of the agent is to navigate a bunch of stairs, leap over the skull, snatch a key and navigate to the way out to get to the following stage. The problem here is that by taking random actions, the agent is never going to see a singular reward; and that is due to the succession of actions that it needs to take to get that reward that is so complicated. That means that the policy gradient is never going to see a single positive reward. The traditional approach to solve this issue of sparse rewards has been the employment of 'Reward Shaping' which is the process of manually designing the reward function which needs to guide the policy to some required attitude. For the Montezuma's game, the agent could be rewarded every single time it manages to avoid the skull or reach the key. Posteriorly, these additional rewards will guide the policy to some desired behavior. Nevertheless, there are some significant downsides to reward shaping. Firstly, reward shaping is a custom process that needs to be redone for every novel environment, and the second problem is that reward shaping suffers from what is called alignment.

2. Reinforcement Learning:

A. History

In the early 1980s, reinforcement learning history [15] has two main orientations. The first thread is about learning by experience and error that began in the animal's learning psychology. The second orientation is about the optimal control issue by involving the value functions as well as the dynamic programming. Mostly, this orientation did not implicate learning. Even though these two threads have been autonomous. A unique property of reinforcement learning is that the training information is our evaluations and error vectors, this is learning to increase the amount of reward and decrease the amount of penalty. This has been called 'learning with a critic'. The critic evaluates the system's behavior and doesn't need to know what the target output should be. A unique property of reinforcement learning [16] is that the critic does not even need to see the output of the learning system, it can base its evaluation on some distal consequence of that learning system behavior that's produced through some unmodeled system. The roots of this return to the psychologist Edward Thorndike [17] who talked about learning by trial and error. His study was about putting a cat in a box, the cat was motivated to get out and stumble on the method of releasing itself, then he put it back in the box, and the cat would let itself out much more quickly. After multiple times it would release itself immediately. This is called learning by trial and error and coined this Law-of-Effect (Thorndike, 1911). Reinforcement learning [18] is about coaching search results or coaching the results of a search, which means that systems build a memory that caches the results of these searches so the system remembers what worked best for each situation and it can start from there the next time. The surprise is that trial and error learning had not been intensively studied from an artificial intelligence point of view, this is puzzling because such a common-sense idea would have been studied but not intensively. One of the issues is that trial-and-error learning is not error-correction learning (trial-and-error error correction), instead of trial-and-error it should be trial-and-evaluation. There were exceptions, Alan Turing in 1948 talked about the penalty pain and pleasure pain system that was never implemented. There are others notably Minsky [19] (1954, SNARCs), Minsky's thesis was reinforcement learning in artificial intelligence disciplines where he conferred about computational patterns of reinforcement learning, furthermore, he characterized his work with the SNARCs which is an abbreviation of the Stochastic Neural-Analog Reinforcement Calculators. On the other hand, Farley and Clark [20] (1954) came up with the idea of an alternative neural network of a programmed machine learning to learn using trial and error. Shannon (1952, Theseus), built his machine learning example which was a robotic maze-solving mouse recognized by the name of Theseus. Arthur Samuel (1959) proposed and applied a literacy system that incorporated temporal-difference concepts [21], as part of his famed checkers-playing project. Widrow and Hoff in 1960 developed LMS which is the least mean square that is a learning algorithm. LMS is used in different applications of adaptative signal processing. Michie (1961, Menace) described a learning system that depends on trial-and-error to acquire a piece of knowledge on how to play tic-tac-toe named MENACE which is an abbreviation of Matchbox Educable Noughts and crosses Engine [22]. Michie and Chambers (1968, Glee) came out with another reinforcement learning system [23] for tic-tac-toe named GLEE which is a shortened form of Game Learning Expectimaxing Engine, and the BOXES invention which is a reinforcement learning controller). Widrow, Gupta together with Maitra (1973) proposed a modification of the LMS that gives the ability to learn using the success and failure signals, rather than learning using the training examples. In 1975 Holland (CAS) redacted the book titled ground-breaking on inheritable algorithms, "Adaptation in Neural and Artificial System". In addition to that, he developed Holland's schema theorem. Harry Klopf [23] (1972, 1975, 1982) focused his research on those fundamental features of adaptive behavior. Various neuroscience techniques that have been ameliorated at this moment are well demonstrated according to temporal difference learning for Hawkins and Kandel in 1984; as well as Byrne, Gingrich, and Baxter in 1990; additionally, we find also Gelperin, Hopfield, and Tank in 1985; then Tesauro in 1986; Friston et al. in 1994, even though in most cases there was absolutely no historical relation. Schultz, Dayan, and Montague (1997) came out with a novel synopsis of some connections between neuroscience concepts and temporal-difference learning.

B. *Reinforcement learning models*

1) *Model-based in reinforcement learning*

The first biggest dichotomy is between the first type which is model-based and the second one which is model-free reinforcement learning. Initially, the model-based RL ([24], [25]) is the simplest form of reinforcement learning, in this version, we know everything about the environment and there will be no need for exploration ([26], [27]). In particular, what we know is the complete likelihood distribution of the following state, with an already specified (s_t, a_t) we know the likelihood of reaching any possible state s_{t+1} besides the reward R_{t+1}. In this situation, the value function is given as follows:

$$V^*(s_t) = \max_{a_t}(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)V^*(s_{t+1})) \quad (1)$$

This equation can be solved and then calculate the values of the value function V where $E[r_{t+1}]$ represents the expected reward. The optimal policy starting from state st is going to be as shown in equation 2 the moment that we have the value function:

$$\Pi^*(s_t) = \arg\max_{a_t}(E[r_{t+1}|s_t, a_t] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)V^*(s_{t+1})) \quad (2)$$

The important thing here is to figure out the value function for each one of the states using an algorithm with the steps:

Step 1: Initialize the value functions to arbitrary values
Step 2: Repeat this part of the algorithm until V(s) converge
For all $s \in S$
For all $a \in A$

$$Q(s, a) \leftarrow E[r|s, a] + \gamma \sum_{s' \in S} P(s'|s, a)V(s') \quad (3)$$

$$V(s) \leftarrow \max_a Q(s, a) \quad (4)$$

Where E[r|s,a] refers to the predictable reward in the given state 's' while choosing the action 'a' The second we obtain the value function; it will be able to compute the optimal policy in any state. The following algorithm allows us to estimate the optimal policy directly without going necessarily via the estimate to the value function. This algorithm is called the policy iteration, and it follows the steps below:

Step 1: Arbitrarily initialize a policy Π
Step 2: Repeat this part of the algorithm until convergence $\Pi \leftarrow \Pi'$
Calculate the values using Π by determining the linear equations:

$$V^{\Pi}(s) = E[r|s, \Pi(s)] + \gamma \sum_{s' \in S} P(s'|s, \Pi(s))V^{\Pi}(s') \quad (5)$$

Upgrade the policy in each state:

$$\Pi'(s) = arg \max_a (E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a) V^{\Pi}(s')) \quad (6)$$

2) Model-free in reinforcement learning

In reinforcement learning, model-free [28] is the opposite of model-based. It's an algorithm that does not require the use of the transition likelihood distribution, along with the reward function that are associated with the Markov decision process, which means that we don't know the P (s', r | s, a). Model-free methods [29] are trial-and-error trainees that try to optimize the value function along with the policy without trying to assess the transition and the reward function. Furthermore, and in this case, everything will be based on trying to estimate some average values over time without keeping track of what has been done before. The average reward at the state 's' in time 't' is:
$E_t(s) = (r_1 + r_2 + ... + r_t)/t$
$E_{t+1}(s) = (r_1 + r_2 + ... + r_t + r_{t+1})/(t + 1)$
Subsequently:
$E_{t+1}(s) = t/(t + 1)E_t(s) + r_{t+1}/(t + 1)$
$E_{t+1}(s) = (t + 1)/(t + 1)E_t(s) - E_t(s)/(t + 1) + r_{t+1}/(t + 1)$

$$E_{t+1}(s) = E_t(s) + 1/(t + 1)[r_{t+1} - E_t(s)] \quad (7)$$

Can be similar to:

$$E_{t+1}(s) = E_t(s) + \delta[r_{t+1} - E_t(s)] \quad (8)$$

This allows us to perform an incremental estimation of the average. If δ is small it will converge to the real average very slowly, otherwise, if δ is large it will oscillate around the average. The parameter δ is called the learning rate.

3) The main two models' differences

In model-based, the environment is represented just as Markov Decision Process using a subsequent element. Given a collection of states where a single state is designated by s, and a series of actions that are disposable in every state, where a single action is indicated by a. a transition probability function from the actual state which is s to the upcoming state denoted as s' within the action a T (s, a, s'). The reward function represents the instant reward obtained while transitioning starting with the state s achieving s' using the action a. The model-based reinforcement learning algorithms assume that we already know the model of our environment, and that model is Markov Decision Process where we know all the states and the actions that are available in the states, as well as the transition probability function including the immediate reward function. With this knowledge, there are two mutual approaches to coming up with optimal policy involving the recursive relation of the Bellman equation, the first one is called value iteration, and the second one is known as the policy iteration. Moreover, in the value iteration method, the ideal policy is gotten by picking out the action that increases the ideal state value function. Moreover, the perfect state value function will be obtained by employing an iterative algorithm, which is the reason behind the name value iteration. Furthermore, the policy function is not used during iteration, rather, the ideal state value function is updated by choosing the action that increases the estimate for the optimal state value function. The second algorithm for the model-based reinforcement learning to get the perfect policy is recognized as policy iteration. In the policy iteration method, the ideal policy is gained by discovering iteratively a preferable estimates of optimal policy function. In this approach, there're two major steps, the first one is policy evaluation as well as policy improvement. For policy evaluation case, the state value function is evaluated for the present policy, while in policy amelioration the policy is improved using the assessment of the state value function. Moreover, the key limitation of the model-based in reinforcement learning technique is that it works only when we know the model of the environment which means the Markov Decision Process corresponds to that environment when we already know all the states, actions as well as the transition probability function in addition to the reward function. However, that is not possible for all reinforcement learning games. In model-free reinforcement learning case, the simulation model of the environment is given, and by reacting with it, the necessary information about this environment is gathered, which means that we can learn about the game only if we play the game. Model-free reinforcement learning techniques don't need information about the Markov Decision Process. The most common technique for model-free reinforcement learning is the Monte Carlo method. Furthermore, the main idea here is to play the game repeatedly for a large number of times and in each play of the game, we keep track of the states that we're in, the actions that we take in those states, and the rewards that we receive for every taken action in each state. By using this information, we can assess the action value function that is the Q-value, as well as the state value function. The Monte Carlo method for reinforcement learning works by following two main steps. The first one is policy evaluation where the policy is evaluated, and the second one is policy improvement where we improve our policy. Unfortunately, Monte Carlo has many limitations, the first one is that is practicable for games that have only a few states and actions, which means in cases where the game doesn't last for a long time. Another shortcoming is that the procedure may pass a lot of time assessing suboptimal policies, which signifies that for a specified state 's' only one of the actions will correspond to optimal policy and all the other actions will correspond to suboptimal policy. In addition to that, the Monte Carlo method works only for episodic issues. Table I below shows the two different models of reinforcement learning in detail:

TABLE I. Reinforcement learning models

RL model	Policy	Strength	Weakness	Applications
Model-free	-Use sampling to set up a policy as well as value function. -Model dynamics are disregarded. -The model can foretell the optimal policy without using reward function besides the transition function. -In the environment, the agent can't reach a model.	-Has reduced assumptions that labor enormous collection of tasks -Show performance at learning intricate policies. -In some tasks, the policy may be propagated better	-Conduct mediocre decisions -In the case of a complex model, it might be exposed to overfit	-Real-world applications -Self-driving cars -Atari -AlphaStar -Robotics -Open AI
Model-based	-Employ the model to get the optimal policy -It's not required to develop a policy explicitly -Employ the reward function besides the transition function to assess the optimal policy -The agent is capable to access an environment model.	-Self-trained -Sample effectual -The mastered dynamic model is portable	-Requires retraining in order to optimize the controller for a certain task -The policy is not immediately optimized -Further hypothesis and not suitable for all tasks -A model might be more intricate than a policy	-Chess games -Applications in robotics (Motion control, surgical, kicking, walking, balancing, tracking, . . .) -Autonomous vehicles -Sanford Helicopter -Poker -AlphaGo -MuZero

C. *The multi-armed bandits*

Reinforcement learning is evaluating the actions rather than instructing, what the correct actions are with the correct set of labels. If we imagine the situation of having four different buttons to press, and by pushing each of these buttons we get a reward. The main purpose here is to detect a way to smartly explore and exploit which buttons to push. The k-armed bandit [30] is similar to the button or slot machines with different slot levers. The purpose here is to increase the reward through a given number of times steps. Moreover, the agent sends an action to push one of the four previous buttons, and the environment sends back a reward. In the situation of the k-armed bandit, it's a contrast to other reinforcement learning problems, where the state is going to be the same environment every time, and each time the agent has four buttons to push, and at all times it's going to be the same regardless of which buttons it pushes. This is an interesting characteristic of the k-armed bandit ([31], [32]) problem. The distribution can be stationary, which means that pushing one of the four buttons would have the same expected reward for the first 50-time steps. Or it could be non-stationary, for example, the reward distribution changes over. One good example of a non-stationary reward distribution [33] could be if there's a play against an opponent in an adversarial like tic-tac-toe; if the player keeps on making the same moves, the opponent might catch on to this and then change its strategy accordingly. Therefore, the reward distribution for making that move is non-stationary, it changes over time.

The idea of the k-armed bandits is to maximize the reward by having the estimation below of each button's reward:

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i I_{A_{i=a}}}{\sum_{i=1}^{t-1} I_{A_{i=a}}} \tag{9}$$

By a time, step noted as 't', the assessed value of action 'a' is given as $Q_t(a)$. where '$I_{predicate}$' indicates the arbitrary variable. '$I_{predicate}$' receives 1 in case the predicate is true, in other ways it gets a 0. The Greedy action selection rule would be to take the maximum button value that returned the most reward given by the equation of the number of rewards achieved over time.

$A_t = argmaxQ_t(a)$

We've just defined the Greedy action option and selected the button with the maximum value function estimated by our agent. ε-Greedy ([32], [33]) selection is going to have a probability ε of not selecting the maximum utility button, but rather sampling another pressed button with the uniform distribution or a uniform probability. And, just as the number of steps rises with the ε-Greedy policy, each action is going to be sampled an indefinite number of times. Q_t which is the value function concerning the action will converge to q*(a)which means the optimal value function for an action. There're different situations in which the Greedy algorithm [32] is advantageous to the ε-Greedy, in cases where the reward variance is 0 the Greedy selection only needs to take the action one time

to know the true reward given by that action, however, ε-Greedy algorithms [34] do much better when there are noisier rewards like having more variance in the distribution away from the mean of the reward on that action. It also performs much better with non-stationary rewards, as the reward distribution changes over time the ε-Greedy algorithm [35] will venture out. The simple bandit algorithm is the idea of either taking the maximum action with a likelihood of (1-ε) or taking a haphazard out action with a likelihood of ε and then updating the sample average for that action.
Initialization, for a=1 to k we get:
Q(a)receives 0
N(a)receives 0
Loop continually:
R receives bandit(A)
N(A)receives N(A)+1

$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)] \tag{10}$$

Adjusting step size is important in the case of non-stationary rewards. For example, by pushing a button, the reward given by each button is changing over time with the underlying distribution. Here we're having an δ parameter that weighting how much the expectation is updated for each pressed button.

$$Q_{n+1} = Q_n + \delta[R_n - Q_n] \tag{11}$$

This idea of unevenly weighting the most recently obtained rewards compared to the overall average rewards is known as the "Exponential Recently-Weighted Average".
$Q_{n+1} = Q_n + \delta[R_n + Q_n]$
$= \delta R_n + (1-\delta)Q_n$
$= \delta R_n + (1-\delta)[\delta R_{n-1} + (1-\delta)Q_{n-1}]$
$= \delta R_n + (1-\delta)\delta R_{n-1} + (1-\delta)^2 Q_{n-2}$
$= \delta R_n + (1-\delta)\delta R_{n-1} + (1-\delta)^2 Q_{n-2} + ... + (1-\delta)^{n-1}\delta R_1 + (1-\delta)^n Q_1$

$$Q_{n+1} = (1-\delta)^n Q_1 + \sum_{i=1}^{n} \delta(1-\delta)^{n-i} R_i \tag{12}$$

From step to step and by varying the parameter of the step size:

$$Q_{n+1} = Q_n + \delta[R_n - Q_n] \tag{13}$$

Another interesting hyperparameter is the initialization of action values in this algorithm, which helps to provide prior knowledge of reward expectations. This optimistic initialization promotes exploration and it's efficient for stationary problems, doesn't matter with non-stationary rewards. Strategic initialization is often a waste of effort in practice because in this case, we're dealing with non-stationary reward distributions, so things concerning the initial behavior of the agent don't matter that much for most real problems. Upper-Confidence-Bound (UCB) is another technique for balancing exploration and exploitation of ε-Greedy [36], it's about selecting randomly another action uniformly after deciding not to take the greedy action. How could we do a better technique of this and weigh actions based on whether they are almost greedy or if they haven't been tested. There's a lot of uncertainty about the action, this is done with the UCB equation below:

$$A_t = arg\max_a[Q_t(a) + c\sqrt{\frac{Int}{N_t(a)}}] \tag{14}$$

Where $Q_t(a)$ is the estimation of the action.
$c\sqrt{\frac{Int}{N_t(a)}}$ is the UCB term, and 'c' is the parameter that weights the balance between (lnt) and $N_t(a)$. lnt represents the ordinary logarithm of t. $N_t(a)$ presents for how many times action 'a' has been chosen before the time t. However, UCB is more complicated than the ε-Greedy to expand over bandits to more common reinforcement learning problems mainly due to non-stationary problem reward distributions and large state spaces. Another interesting way of balancing exploration and exploitation is the Gradient Bandit Algorithm. The main idea here is that it's preferred to take one action compared to the other actions, and what we have is a Soft-max distribution over our actions. Whenever the preference is large, the action will be taken more often.

$$Pr\{A_t = a\} = \frac{e^{H_t}(a)}{\sum_{b=1}^{k} e^{H_t(b)}} = \Pi_t(a) \tag{15}$$

Where:

$\Pi_t(a)$ at a time 't' presents the likelihood of picking up an action 'a'.
$H_t(a)$ presents the preference for each action 'a'.
Afterward, updating using gradient ascent:

$$H_{t+1}(A) = H_t(A_t) + \delta(R_t - \overline{R_t})(1 - \Pi_t(A_t)) \tag{16}$$

And:

$$H_{t+1}(a) = H_t(a) + \delta(R_t - \overline{R_t})\Pi_t(a) \tag{17}$$

Where:
$\underline{a} \neq A_t$
$\overline{R_t}$ is the average for all rewards.
So far, we've been talking about the agent that attempts to find a unique best action when it's about a stationary task or attempts to keep track of the best action when it varies through a time when it's a non-stationary task. however, there are several situations, and the purpose is to acquire knowledge of a policy or a mapping to the finest actions from situations. Here we're talking about the associative search task that includes trial and error to explore the finest actions and the combination of these actions together with the best status. In the literature, this is called the contextual

bandits.

D. Markov Decision Process (MDP)

Most problems that are solved using reinforcement learning are based on formalization. This concept of formalization was given by the Markov Decision Procedure or Process. In this case, the agent ([37], [38]) that reacts to the environment is the decision-maker and it interacts with the given environment. The agent receives a description of the environment at any time step and depending on this description the agent picks out an action to pick up. Afterward, the environment moves into some novel state, moreover, and as an outcome of its foregoing action, the agent will gain a reward. To summarize, the elements of an MDP include the environment and the agent besides all the possible environment states in addition to all the actions that the agent takes, and all the rewards that the agent acquires from taking actions in the environment. The procedure of picking out a particular action from a specific state moving to a new one and obtaining a reward occurs in a sequential way repeatedly, which produces what it's called a trajectory. The trajectory presents the concatenation of state actions and rewards all over the procedure ([38], [39]). Moreover, in reinforcement learning, the principal objective of the agent is to increase the entire number of recompenses that it obtains from picking up actions in certain states of the environment, which means that during the entire process, the agent concentrates on maximizing the instant rewards as well as the accumulated rewards earned overtime. In the Markov Decision Process, a group of states 'S' as well as a collection of actions 'A' and a series of rewards 'R' has a restricted number of elements. The agent will have for every time step t=0, 1, 2, ... a clear description of the environment state $S_t \in S$, according to this, the agent picks up an action $A_t \in A$, consequently, we will have the state-action pairs noted as (A_t, A_t). Thereafter, time is increased and switched from t to t+1 then the environment is changed to a novel state $S_{t+1} \in S$, at this stage of the process, the agent acquires a numeral reward $R_{t+1} \in R$ from the taken action A_t from the state S_t. Generally, we can kind of think of this procedure of gaining a reward as being a random function:

$$f(S_t, A_t) = R_{t+1} \tag{18}$$

The trajectory is showing the consecutive procedure of picking out an action from a state, thereafter moving to a novel state and gaining a reward can be schematized as:

$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$
Figure 2 below illustrates this entire idea.

Step 1: The environment begins is in the state S_t
Step 2: The agent spots the actual state and picks up an action A_t
Step 3: The environment proceeds until the state St+1 and grants to the agent a recompense R_{t+1}.
Step 4: The agent starts again for the upcoming time step t+1.

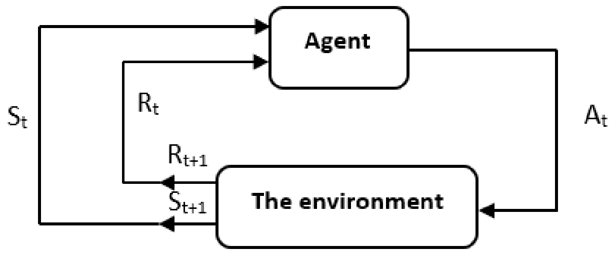

Figure 2. The process of the agent-environment

Since the set of states 'S' and the set of rewards 'R' are limited, the casual variables R_t and S_t that represent the reward in the state at time T, have competently described likelihood distributions. This means that all the probable values that can be allocated to R_t and St have some related likelihood. These allocations rely on the preceding state as well as the action that took place in the precedent time step t-1. Moreover, it requires a process to combine and formalize these accumulating rewards, for this, the predictable return of the rewards is used at a specified time step. Presently, we can consider the return as the total of the upcoming recompenses as shown below:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ...R_T \tag{19}$$

Here T represents the latest time step.
The idea of the predictable return is very significant because it's the agent's main purpose to increase the foreseeable return which drives the agent to make the decisions that it makes now. The agent-environment interplay splits up into subsequences noted as episodes [40], like the game pong where every novel round or cycle of the game can be counted as an episode, furthermore, the ending time step of each episode happens when a player gains a point. Every episode is finished in the final state at time T that is succeeded by resetting the environment to several starting state standards, or a random pattern from an allocation of probable beginning states. The following episode starts independently relying on how the preceding episode took the end. There exist several types of tasks whereas the agent-environment interconnections don't split up naturally into episodes, but instead, proceed without restriction. These types of assignments are noted as continuing tasks [41], consequently, the definition of the return at each time T problematic becomes problematic for the reason that the final time step T would be equivalent to infiniteness, therefore the return may be infinite, due to this it's required to make the way of working with the return clear. On the other hand, the expected discounted return of rewards will be maximized, specifically, the agent will be picking out an action for every time step to increase the expected return. However, the main purpose of this discounted return [42] is to push the agent to be more conscious about the instant reward than the future rewards for the reason that it will be heavily discounted. The immediate rewards will have

more influence on the agent's decision for choosing an action while it considers the rewards to gain in the future. The discounted return is described as follows:
$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{20}$$

Where γ represents the discount rate that's a number between 0 and 1.
Furthermore, the question to be asked here, is how can an agent be capable to select a specific action given a particular state. This is where policies take place. In addition to that, it's required to know how the chosen action in a given state might be benefic over another one in terms of the agent's return. To be conscious of this, the agent will have an advantage in choosing the right action in a specific state to increase its expected return, at this stage, the value function becomes functional. The symbol Π represents the policy function that delineates a particular status to probabilities of choosing every conceivable action from that state to indicate policy [43]. While speaking about policies officially, we presume that an agent follows up a policy Π at time t, at that time $\Pi(a|s)$ represents the likelihood that the action at time t is A_t = 'a' if the state at a precised time t $S_t = s$. That signifies that at time T using the policy the likelihood of choosing an action 'a' within a given state 's' is $\Pi(a|s)$, knowing that for every state $s \in S$, Π is considered as a probability distribution over $s \in A(s)$. Value functions [44] represent functions of states or of what it's called state-action pairs that evaluate how efficient it's for an agent to be in a specific state, or how functional for an agent to execute a specific action in a particular state. This concept is stated in terms of the predicted return. Since value functions are determined concerning the predicted return, this means that value functions are described concerning particular techniques of acting since the policy that it's followed by the agent is influencing directly the ways the agent takes an action, thereafter, we conclude that value functions are therefore determined for policies. Therefore, value functions are defined as being the functions of states or state action pairs, and there are two sorts of value functions, the first one is a state-value function, while the second one is the state-value function. In the case of the state-value function that is using the policy, Π is designated as $_\Pi$, and it informs us on how efficient any particular state is for an agent using policy Π, moreover, we'll be given the value of a state. The foreseeable return is described as the value of state denoted 's' using a policy Π at a time 't'. Subsequently, $_\Pi(s)$ is defined using the expression bellow:
$_\Pi(s) = E[G_t|S_t = s]$

$$_\Pi(s) = E[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{21}$$

Likewise, q represents the action-value function following the policy Π, informing us on how efficient for an agent to pick up any specific action from any specified state using a policy Π. subsequently, it gives us the value of an action under Π. Formally, the value of the action 'a' at a specific state 's' at a time 't' is defined as: $q_\Pi(s,a) = E[G_t|S_t = s, A_t = a]$

$$q_\Pi(s,a) = E[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{22}$$

The notation q refers to the action-value function which can be mentioned as a Q-function. Moreover, Q-value represents the function's return in the case of any given state-action pair, knowing that Q indicates the quality of picking up a particular action in a specified state.

E. Dynamic programming

Dynamic programming ([45], [46], [47]) is a technique in computer science frequently used for things like genetic sequence alignment or spell checking. The computation will be bootstrapped by using the estimate: $_{k+1}(s) = E_\Pi[R_{t+1} + \gamma_k(S_{t+1})|S_t = s]$

$$_{k+1}(s) =_{k+1} (s) = \sum_a \Pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma_k(s')] \tag{23}$$

Dynamic programming refers to the idea of not using a separate array that holds the previous value function estimates at time step k. What should be done rather is to update the state values in a single sweep. This might cause a little bit of noise in the updates; because updating the states might have a higher magnitude of updating state. Another interesting extension to dynamic programming that makes it applicable to reinforcement learning problems like chess and backgammon is the idea of synchronous dynamic programming. For example, in the backgammon game ([48], [49]) there are 1020 states, and to do a complete state space sweep and update the value pushed estimates of all 1020 states, would take 1000 years at a speed of 1 million states per second. Asynchronous dynamic programming refers to the idea of updating the value estimates of a subgroup of states rather than the entire set of states at every policy iteration. Asynchronous dynamic programming can also be distributed across machines.

F. Monte Carlo methods

The Monte Carlo method ([50], [51]) is a way to learn without prior awareness of the state to the next state transition given our actions. Monte Carlo methods are employed to create samples of episodes and then update the value estimates of the state-action pairs according to the actual returns that are received by experiencing the world by doing this model-free approach of trial-and-error learning. Monte Carlo generally [51] describes randomized algorithms, here it's used to describe randomly sampling episodes in the environment. Figure 3 below shows the Monte Carlo prediction or evaluation of the state's given the actual policy: To use the Monte Carlo algorithm [52], we follow the following steps:
Step 1: first take in the policy Π, then we'll have the value

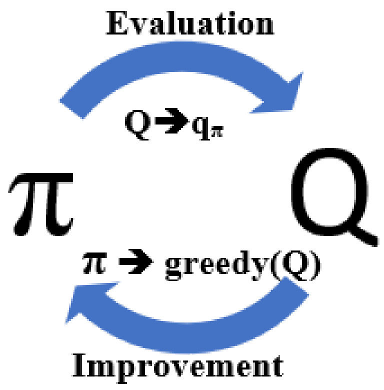

Figure 3. Monte Carlo prediction

function which is either initialized randomly or inherited from the previous iteration of the generalized policy iteration loop shown in Figure 3.
Step 2: initialize a Returns (s) array that is an empty list, which is going to aggregate all the returns every time we traverse through a given state in the state set. Returns $(s) \leftarrow$ an emptied list, for every $s \in S$
Step 3: Loop generating episodes following the policy. We'll have these sequences of actions and then different states that we traverse given the policy that's determining the state-to-action mapping.
$\Pi = S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T$
Step 4: Aggregate the achieved rewards append to the Returns array.
Step 5: Set the new values of every state as the of the returns in the Returns array. The average is going to be the sum of the Returns divided by how many times each of the states is encountered.
Transitioning from model-based learning to model-free learning, we'll be interested in Q functions compared to the value functions, because we don't know how the state-action pairs necessarily translate into the s' or the following state. The value of these state-action pairs should be explicitly evaluated. Putting this together, we'll have the generalized policy iteration framework with Monte Carlo control. According to Figure 3, the policy is used to assess the values of the state-action pairs or Q functions using this policy, then, grouping the policy and making it greedy concerning the state-action pair table. The blackjack card game is a good motivating example for using Monte Carlo control to find the preferable state-action value estimates as well as the optimal policy. In blackjack, we have got the dealer's demonstrating card, our hand, and then whether or not we have a usable ace. The decisions from these states or whether we want to hit and receive a new card, or we want to stick and hold the fort down with our current configuration, hoping that the dealer doesn't have a greater sum than we do, or can achieve it by taking the hit action. And because the estimates of every state-action pair are coming from experience rather than by using a table, it'll be a need to have a way of balancing exploitation and exploration. One way of doing this is through exploring starts, which is initializing the environment with each state-action pair, such that the entire state-action pairs are experienced throughout the succession of episodes. The Monte Carlo with exploring starts works fairly for blackjack. We can imagine this algorithm scaling very well to most of the interesting problems in reinforcement learning. Another way of adding exploration to Monte Carlo control differently from exploring starts is to use -soft policies, this is similar to the k-armed bandits where we either take the optimal action or the probability epsilon and randomly sample another action from the conceivable actions given the current state. Probably the most interesting way to achieve exploration in Monte Carlo learning algorithms is with off-policy learning. The way this is done is by keeping a target policy that refers to the policy that will try to behave optimally, and then we'll have a behavior policy 'b' which is like the exploration policy. Here, episode samples are used from the behavior policy to explore the environment and then use this to bring the target policy up to date. At this stage, Π and 'b' must be aligned with importance sampling to weigh how realistic these trajectories are for the target policy as they're experienced by the behavior policy. This significance sampling ratio is used to weigh the returns achieved by the behavior policy when used to bring the target policy up to date. This ratio is calculated by taking the product that's relating the probabilities of choosing a particular action given a certain state from the target policy [53].

$$Pr(A_t, S_{t+1}, A_{t+1}, ..., S_T | S_t, A_{t:T-1} \sim \Pi)$$

$$= \Pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\Pi(A_{t+1}|S_{t+1})...p(S_T|S_{T-1}, A_{T-1})$$

$$= \prod_{k=1}^{T-1} \frac{\Pi(A_k|S_k)}{b(A_k|S_k)} \tag{24}$$

When computing the importance sampling ratio for off-policy learning, it's computed over the trajectory of every action. There're two ways of averaging out how we're going to overall update the value estimate of the state. It's neither weighted by the length of the sequence T, nor it can be weighted by the same summation of the importance sampling ratios.

$$V(s) = \frac{\sum_{t\in\tau(s)} \rho_{t:T(t)-1} G_t}{|\tau(s)|} \tag{25}$$

$$V(s) = \frac{\sum_{t\in\tau(s)} \rho_{t:T(t)-1} G_t}{\sum_{t\in\tau(s)} \rho_{t:T(t)-1}} \tag{26}$$

G. *Temporal-difference learning*

The main idea of temporal difference ([54], [55]) is to combine learning from sampled experience with the bellman equations. The bellman equations are shown where we can assess the value of the actual state depending on the values of the states that are reached after taking certain actions and transitioning into those states. The crucial idea of temporal-difference learning is to improve the way to do model-free reinforcement learning or learning from experience. For the Monte Carlo update, the value estimate is brought of the state up to date by sampling an episode and then receiving the return Gt. Furthermore, take the error of the return minus the value under that state or the return achieved from that state as shown in the equation below:

$$V(S_t) \leftarrow V(S_t) + \delta[G_t - V(S_t)] \tag{27}$$

In temporal-difference learning, the same value estimate of the return is expected from the state but with the use of bootstrapping or the technique of introducing the bellman equations.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{28}$$

$= R_{t+1} +_{t+1}$
$G_t \cong R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \delta[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{29}$$

Using this idea, we introduce the full TD (0) [56] algorithm that is used to assess the value functions given a particular policy. Furthermore, this algorithm requires a step size $\delta \in (0, 1]$, and an initial assess of the value function V(s).
Step 1: TD takes the policy Π to evaluate as input
Step 2: Every episode begins in some initial state ‘s’. At this stage, looping through taking actions given by the policy Π for a given state. Furthermore, observing the reward in the following state S’, along with bringing the value of the state up to date based on the achieved return R at that step plus the discount factor γ times the value estimate of the subsequent state V(S’) reached, then subtracting this by the original prediction of what it was thought it would achieve from the given state V(S).

$$V(S) \leftarrow V(S) + \delta[R + \gamma V(S') - V(S)] \tag{30}$$

$S \leftarrow S'$
Step 3: Reiterate step 2 up till the episode’s ending.
For this reason, the backup diagram of the TD (0) or the one step ahead temporal-difference learning algorithm looks like what you can see in Figure 4. It starts in state S then it picks out an action A, and it finishes in state S’. Throughout temporal-difference learning, we’ll have the temporal-difference error which is where we have the reward achieved at the time step, and then we add it to the discount factor times the value approximate of the subsequent state, then we subtract this by the original value estimate of that state.

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{31}$$

One more important characteristic of free reinforcement learning algorithms is that we can save the experience by using patch updating. The main idea here is to repeatedly process all the available experiences with the novel value function to make a new gain until they converge. However, the updates are done only after producing every training

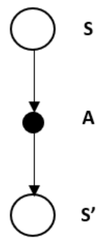

Figure 4. TD (0)

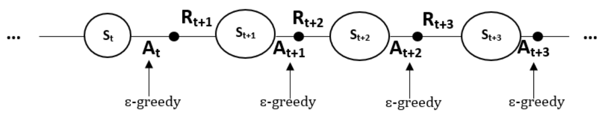

Figure 5. The trajectory of an experience

data batch. Monte Carlo converges differently than the temporal-difference learning algorithm in these batch updating. Moreover, the exploration and exploitation need to be balanced with temporal-difference learning using different algorithms like SARSA, Q-learning, Expected Sarsa [57], and Double Q-learning. Practically in these cases, we’re interested in evaluating the Q-function q (s, a) rather than the value function of states V(s) and that’s because it’s a model-free reinforcement learning where it’s not known how states transition into the next states, and this is because there’s no way to access the environmental dynamics. For this reason, it’ll be interesting to evaluate the pairs of state-action and use the value estimates of the state-action pairs to improve the policy. SARSA is a shortened form of the State Action Reward State Action that represents the On-Policy temporal-difference learning method, therefore, to balance exploration-exploitation it uses ε-greedy policy. The probability of ε will pick up the action that achieved the highest estimate with the likelihood of (1- ε). Then selecting another action from the actions that are available at that state. Figure 5 shows how the trajectories of experience have this structure of action reward: In temporal-difference learning update, a time step with the policy is taken or the Q function mapping the state to action, and then to randomly sample action from the next state achieved, and bootstrapping. This happens by first starting with an initial state, then applying one action according to the -greedy policy. After this, we’ll be in the next state. By observing the reward and accordingly, feeding the values $([R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)])$ and bring up to date the Q value of the preceding state as shown in the equation below:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \delta[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (32)$$

The full algorithm for the SARSA On-policy temporal-difference control for estimating the value functions of state-action pairs is as shown below:
Step 1: Initialize the state-action pairs with random values in the Q-table.
Step 2: In every state S that is encountered, the first thing to do is to execute the action A that comes from ε-greedy policy and get the reward R. Then processing to state S'.
Step 3: Presently, as being in state S' and planning to get to the next action A' using the ε-greedy policy at step S'.
Step 4: Referencing to the Q-table to get the value of Q (S', A').
Step 5: Update $Q(S, A) \leftarrow Q(S, A) + \delta[R + \gamma Q(S', A') - Q(S, A)]$
$S \leftarrow S'; A \leftarrow A'$
Which means considering the next state as a new initial state
Step 6: Reiterate from step 2 unto S is terminal.
Q-learning is very similar to SARSA except for one difference in Step 5 in the SARSAS algorithm [58], where instead of sampling another action from our policy, we pick out the maximum action together with the highest Q-function for that state action.

$$Q(S, A) \leftarrow Q(S, A) + \delta[R + \gamma \max_a Q(S', a) - Q(S, A)] \quad (33)$$

$S \leftarrow S'$ SARSA outperforms Q-learning because it's learning concerning its ε-greedy policy, and that's because it's an on-policy learning algorithm. It knows that its policy is going to make these random decisions. Although the Q-learning algorithm isn't learning concerning its ε-greedy policy, rather it's learning for the optimal action it would take. Another extension to SARSA is Expected SARSA which turns SARSA from On-policy to Off-policy by rather than taking the maximum action in Q-learning, expected SARSA is going to weight the Q-value of each next state-action pair by the likelihood of taking that action given that the next, that is determined by the policy.
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \delta[R_{t+1} + \gamma \mathbb{E}_\Pi[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S_t, A_t)]$

$$\leftarrow Q(S_t, A_t) + \delta[R_{t+1} + \gamma \sum_a \Pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (34)$$

The term $(a|_S t + 1)$ represents the probability distribution of actions given the current state, and $Q(S_{t+1})$ refers to the Q-value of each of the state-action pairs. A sum-up is then employed over all the actions, and that's what it's used for the bootstrap. In Q-learning, the temporal-difference method updates or the bootstrap value estimate of the following state has a maximization or a positive bias because it's taking the action that has the current highest value estimate, and this is solved with double Q-learning. Furthermore, the double Q-learning algorithm is going to maintain two separate tables $Q_1$and Q_2 of the state-action pairs, and then with a probability, using the Q_2 estimate of the Q_1 taking the action. Fundamentally use the Q_1 table to take the maximum action while doing the bootstrap step, and then evaluate that state-action pair with the Q_2 table or vice versa.

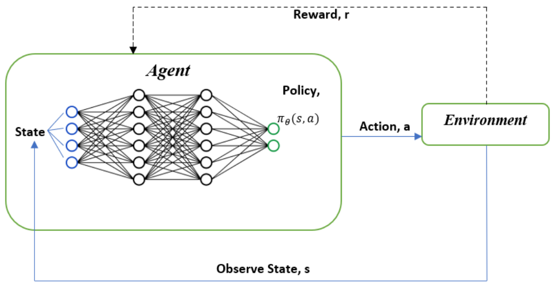

Figure 6. Deep policy network

$$Q_1(S, A) \leftarrow Q_1(S, A) + \delta(R + \gamma Q_2(S', arg \max_a Q_1(S', a)) - Q_1(S, A)) \quad (35)$$

Or

$$Q_2(S, A) \leftarrow Q_2(S, A) + \delta(R + \gamma Q_1(S', arg \max_a Q_2(S', a)) - Q_2(S, A)) \quad (36)$$

3. Deep Reinforcement Learning

Deep reinforcement learning ([59], [60]) is the most important field in artificial intelligence. It's a combination between the power and the capability of deep neural networks to represent and make sense of the world, considering the capacity to act on that understanding. As shown in Figure 6, the deep section of deep reinforcement learning is neural networks, using the frameworks and reinforcement learning where the neural network is doing the description of the world based on which the actions are made. The policy is replaced with deep neural network, and the Π is parametrized by a θ that describes the neural network. Furthermore, it maps the present state to the best probabilistic action to take in the environment.$\Pi_\theta(S, A)$
The principal idea here is to update this policy to maximize future rewards. According to the discount gamma parameter, γ^t rewards shortly are worth more than rewards in the distant future. Usually, these rewards are going to be relatively sparse and infrequent since we're in a semi-supervised learning framework, where these rewards are only occasional. Consequently, it's difficult to find out what actions gave rise to those rewards. As said at the top, the policy function Π can be parametrized as a neural network with a weight θ. Moreover, the input of the neural network represents the state and the output which is the probability of which action to take. The principal idea is to optimize weights by computing gradients based on the cumulative expected future reward using the policy gradient optimization formula: Given the cumulative future reward $R_{\Sigma,\theta}$ given a set of parameters θ for policy.

$$R_{\Sigma,\theta} = \sum_{s \in S} \mu_\theta(s, a) Q(s, a) \quad (37)$$

Where: $\mu_\theta(s)$ is the expected future distribution or the probability of being in a state 's', besides Q(s, a) which refers to the quality function of being in a state 's'. Taking the gradient of the reward concerning theta:

$$\nabla_\theta R_{\sum,\theta} = \sum_{s\in S} \mu_\theta(s) \sum_{a\in A} Q(s,a)\, \nabla_\theta\, \Pi_\theta(s,a) \qquad (38)$$

$= \sum_{s\in S} \mu_\theta(s) \sum_{a\in A} \Pi_\theta(s,a) Q(s,a) \frac{\nabla_\theta \Pi_\theta(s,a)}{\Pi_\theta(s,a)}$
$= \sum_{s\in S} \mu_\theta(s) \sum_{a\in A} \Pi_\theta(s,a) Q(s,a)\, \nabla_\theta \log(\Pi_\theta(s,a))$
$= \mathbb{E}(Q(s,a)\, \nabla_\theta \log(\Pi_\theta(s,a)))$

$$Policy \Pi_\theta(s,a) \Rightarrow \theta^{new} = \theta^{old} + \delta\, \nabla_\theta\, R_{\sum,\theta} \qquad (39)$$

A. Deep Q-Learning

One of the greatest demonstrations of deep reinforcement learning in the last five years is deep q-learning [61], where a quality function with a neural network is essentially learned. The basic formula for q-learning that represents an off-policy temporal-difference learning algorithm is as written below:

$$Q^{new}(s_t,a_t) = Q^{old}(s_t,a_t) + \delta(r_t + \gamma \max_a Q(s_{t+1},a) - Q^{old}(s_t,a_t)) \qquad (40)$$

his quality function can be parametrized by some neural network weights:

$$Q(s,a) \cong Q(s,a,\theta) \qquad (41)$$

The neural network cost function that is used when building a deep q-learner is:

$$\mathcal{L} = \mathbb{E}[(r_t + \gamma \max_a Q(s_{t+1},a_{t+1},\theta) - Q(s_t,a_t,\theta)^2)] \qquad (42)$$

The neural network is going to use its algorithm stochastic gradient descent and backpropagation to optimize the parameters θ to give the best possible q-function that minimizes the temporal-difference error. Deep q-learning which is also called human-level control, has been implemented in the deep mind Atari video game where essentially a deep q-learner with a convolutional level layer can learn the right quality function for what actions to take given a state that it finds itself.

B. Deep Dueling Q Network

Deep Dueling Q Network which is abbreviated as (DDQN) [62], essentially takes the quality function and splits it into two networks. A value network is a function of the actual state, and an advantage network tells us the advantage over the value of being in a given state for picking up an action.

$$Q(s,a,\theta) = V(s,\theta_1) + A(a,s,\theta_2) \qquad (43)$$

C. Actor-Critic Network

The main idea of an actor-critic reinforcement learner [63] is to take the best of policy-based along with value-based learning. Furthermore, the actor is trying to learn a good policy while the critic is critiquing that policy based on its estimate of the value.
Actor; policy-based: $\Pi(s,a) \cong \Pi(s,a,\theta)$ Critic; value-based:

$$V(s_t) = \mathbb{E}(r_t + \gamma V(s_{t+1})) \qquad (44)$$

By bringing up to date the parameters of the policy using the temporal-difference signal from the value learner, the value function will be updated as well. The critic is essentially giving the error signal that is employed to update the policy.

$$\theta_{t+1} = \theta_t + \delta(r_t + \gamma V(s_{t+1}) - V(s_t)) \qquad (45)$$

D. Advantage Actor-Critic Network

This theory uses the deep-dueling q-network where the actor represents a deep policy network, while the critic is a deep dueling q-network that tells us the quality of existing in a state 's' and picking up an action 'a'. The policy gradient iteration in this case is updating the deep policy network, which is faster than traditional q-learning or the traditional value iteration.

$$\theta_{t+1} = \theta_t + \delta\, \nabla_\theta\, (\log \Pi(s_t,a_t,\theta) Q(s_t,a_t,\theta_2)) \qquad (46)$$

4. Reinforcement learning algorithms summary

In actor-critic methods, [64] two deep neural networks are used. One of them is used to approximate the policy of the agent directly, knowing that the policy is just a likelihood distribution through the set of actions, where the state is taken as an input as well as an output will be a likelihood of choosing each action. The other network called the critic assesses the value function. Moreover, the critic acts like any other critic telling the actor how good each action is based on whether or not the resulting state is valuable. The two networks labor together to figure out how best to proceed in the environment. The actor [65] selects the action while the critic evaluates the states and then the result is compared to the rewards from the environment. Over time the critic will be more precise at estimating the values of states, which gives the actor the ability to select the actions that guide to those states. The actor-critic methods appertain to the temporal difference learning class of algorithms. Actor critic methods are vastly employed in fields like robotics, where applying continuous voltages to motor enjoins to actuate movement. They're necessary for dealing with continuous action spaces. In the upcoming section, we explain in detail all of the reinforcement learning algorithms and their uses case.

A. Reinforcement learning algorithms

A2C: is an abbreviation of Advantage Actor-Criticand, A2C amalgamates the policy-based as well as the value-based. The value function in this case is learned by following one policy[65]. A2C is similar to A3C except for the asynchronous section but more efficient. some of the many real-world applications are: Cooperative autonomous vehicles; Stock selection; Portfolio management; Robotics; Mobile-edge computing; Recommender systems; Tracking user mobility; Open Information Extraction (OIE); Intelligent traffic signal control; ViZDoom games; Inverted pendulum system; Visual navigation tasks; Dialogue management systems; Self-tuning PID controllers; Starcraft 2 games; Roll control for the underwater vehicle.

A3C: is an abbreviation of Asynchronous Advantage Actor-Critic, it was liberated in 2016 by Google's DeepMind group. A3C defeated DQN (Robust, faster and simpler, reached superior scores)[66], in this case, multiple agents training in parallel each one with its environment. A3C is used in the Atari domain and Autonomous drone delivery. Other real-world applications that employ A3C are: Cognitive network security, Starcraft 2 games; Hybrid flow shop scheduling; Flappy bird game; Real-time energy management; Elevator group control; Content caching policy for 5g network; Electricity-Gas-Heat Integrated Energy system; First-person shooter game; High-speed train operation system; Autonomous voltage control; Pathfinding; Mobile robot navigation

DDPG: is an abbreviation of Deep Deterministic Policy Gradient, and it combines Q-learning as well as policy gradient. DDPG uses the structure of actor-critic which means it has two networks. Moreover, the actor network presumes the observation and gives the action[67], while the critic network considers the observation and the corresponding action and outputs the Q-value. In this case, DDPG is an off-policy. Among the real-world applications that use DDPG are: energy harvesting wireless communications; Urban traffic light control; Multi-agent cooperation and competition; Learn pouring for robots; Unmanned aerial vehicles (UAVs); Navigation of mobile robots; Improvement of permanent magnet synchronous motor; Bipedal walking robot; Target tracking strategy; Robot grasping; Electricity market equilibrium; Controlling bicycle without human interaction; Radio resource scheduling for 5G; Energy management for the hybrid electric bus; Automatic Landing Control of Fixed-Wing Aircraft; Automated lane change behavior; Autonomous driving at intersections; Dosing and Surveillance in the ICU; Path planning for a humanoid arm.

TD3: which means Twin Delayed Deep Deterministic Policy Gradients. In this case, a TD3 agent [68] is an actor-critic agent exploring the appropriate policy that gives a maximized long-term reward. It's used in the case of the overestimation bias problem that emerges from deep neural network use. TD3 is applied in the following real-world applications: Decision-making algorithms; Motion planning of a robot; Oscillation damping control; Motion planning for a robot; Control of a quadrotor; Control and simulation of the 6-DOF Biped Robot; Trajectory planning for a parafoil; UAV path planning; Traffic signal timing control; UAV target tracking; Energy Harvesting wireless communication.

SAC: SAC is an abbreviation of Soft Actor-Critic, and it combines stochastic policies (TRPO, PPO) and the replay buffer (DDPG, TD3)[69]. SAC is more stable and more efficient. furthermore, it's used in many real-world applications like Path planning for multi-arm manipulators; Energy management for hybrid electric vehicles; Harnessing energy flexibility; Co-calibration -Navigation of mobile robots; Enhance indoor temperature control in buildings; Integrated energy systems; Robot skills adaptation.

C51: C51 is the acronym for the Categorical version of the DQN algorithm. Its main difference from the DQN [70] is that in the output layer instead of producing a vector C51 produces several distributions in the form of a matrix of Softmax.C51 maximizes the expected return and maintains a full distribution with a more complicated update. Moreover, it was demonstrated by Bellmar et al in 2017 on the Atari benchmark. C51 may be included in the realization of many applications, such as Product recommender for online advertising; Toolpath design; Cost-sensitive classifiers to ids.

DQN: DQN or the Deep Q-Network, occurs by amalgamating Q-learning along with the deep neural network. Although, a deep neural network that estimates the Q-function is designated as a deep Q-network [71]. By using Deep Q-learning we estimate the appropriate action that an agent might take at a specific state. DQN is used in the following real-world application: The beer games; Fault diagnosis methods for rotating machinery; Energy management for the hybrid electric bus; Human-level control; Optimization of textile manufacturing; Decision-making strategy for autonomous vehicles; Microgrid energy management; Automated stock trading; Autonomous UAV navigation; Mobile robot in path planning; Mapless navigation; Text generation; Obstacle avoidance for robots.

DDQN: represents the abbreviation of Double Deep Q-Network. DDQN improves the performance of DQN. Due to the issues found in DQN [72] about the low policy and the unstable training, double deep q-network and by using the deep q-learning implicates two isolated Q-values estimators avoid maximization bias, Double DQN achieves better results when it's compared to DQN. DDQN's main use cases are Pairs trading; Decision-making strategy for autonomous vehicles; Autonomous UAV navigation; Energy management for the hybrid electric vehicle; Scheduling multiple workflows in the cloud; Mobile robot navigation; Probabilistic Boolean control; Mobile robot collision avoidance; Proactive content publishing and recommendation systems; Sepsis treatment; Anti-jamming system for cognitive radio; Cryptocurrency trading.

PPO: is the acronym of Proximal Policy Optimization. PPO [73] is designed to obtain maximum efficiency from the captured training data. It includes several upgrades of the actor-critic algorithm, they both seek to maintain smooth gradual gradient updates so we get continuous improvement and avoid unrecoverable crashes. The major difference in PPO is how the actor loss is calculated. It's the minimum of two surrogate functions. PPO is used in: Automated lane change strategy; Metro service schedule; Maintenance of a wind farm with multiple crews; The joint replenishment problem; Target localization for a multi-agent; Mixed-autonomy traffic control; Robot running skills learning; Radiation source research; Learning muscle control for a multi-joint arm; Quadrotor control; Humanoid robot running motion; Mobile puzzle games.

TRPO: TRPO means Trust Region Policy Optimization [74], which is an on-policy algorithm that executes constrained policy updates by involving the KL divergence. TRPO is used in: Active object recognition; Customized pearl matter propagation; Control of fixed wings UAVs; Power allocation for the internet of vehicles (IoV); Collision avoidance.
ACKTR: Actor-Critic together with Kronecker-factored Trust Region. ACKTR gives better results and performances than A2C and TRPO ([75], [76]). ACKTR uses Kronecker-factored approximation to have better actor and critic. It employs trust-region optimization. Furthermore, ACKTR is widely employed in the following real-world applications: Home energy management; Learning battles in ViZDoom; Trading financial assets; Computer vision-based interface tracking; Intelligent traffic signal control; Pedestrian behavior for pedestrian vehicles; Continuous movement for robots; Locomotion behavior for robots.
SARSA: SARSA is the abbreviation of State action Reward State Action [77]. SARSA is an on-policy temporal difference learning algorithm. It's slightly different from Q-learning, however, it converges faster than Q-learning. SARSA is used for: Robot learning; Mobile edge computing; Software-defined networking; Computational mechanism of humans; Video conferencing and source rate control; Homomorphic encryption; Autonomous foraging; Electricity transaction; Task scheduling; Resource provisioning; Load balancing; Traffic signal control; Autonomous navigation and obstacle avoidance.
I2A: means Imagination-Augmented Agents. I2A merges model-free and model-based and learns to expound predictions using a learned environment model [78]. It can be applied in the following use cases: Robotic applications; Goal recognition; Digital agricultural production.
MCTS: is the abbreviation of Monte Carlo Tree Search. MCTS [79] is defined as a Heuristic search algorithm, furthermore, it's a collection of classic tree search applications and the machine learning rules of reinforcement learning. MCTS may be used for these applications: Wind farm layout; Industrial scheduling; Safe autonomous driving; Green synthetic pathways; Computer fighting games; Minecraft games; Real-time Atari games; Walking over graphs; Black box optimization; AlphaGo game; Online scheduling; Open information extraction; Motion planning; Beam orientation in radiotherapy; Backgammon game.

Besides that, previous works addressing approximately the same issue have been published in the literature. Take an example of the paper [80] where the authors gave a survey on reinforcement learning but scientifically limited, they have included no previous works nor a comparison of the existing algorithms, unlike our proposed paper where we tried to give to readers every important information related to reinforcement learning. Furthermore, in [81] authors have presented a very slight reinforcement learning summary, it doesn't encompass all the methods and algorithms. Besides that, it has no mathematical aspect, which means there are no equations or algorithms. For the paper [82], the authors gave a sprightly review of reinforcement learning techniques and their state of the art. Furthermore, reinforcement and deep reinforcement learning might be included in medical imaging as well ([83], [84]) for object and lesion detection, surgical image segmentation, and the classification of different medical images. while image segmentation ([85], [86], [87], [88], [89], [90]) is considered a challenging task, first is the fact for obtaining pixel-wise is very costly, secondly, is that the real world segmentation data is highly imbalanced which biases the performance towards the most represented categories. Consequently, it's required to minimize human labeling effort and maximize the segmentation performance simultaneously. Thinking of employing deep reinforcement learning to learn an optimum policy [91] to select small and informative image regions to be labeled from a pool of unlabeled data [92], is a good idea but still quite challenging. Moreover, reinforcement and deep reinforcement learning can be employed to solve many issues, robotics is one major use case where several papers have been published either for manipulation control [93], in healthcare [94], as well as its implementation in agriculture [95]. On the other hand, and with the exponential growth of the amount of created video games, reinforcement learning has been included in many works [96]. One other real-world application cited above that employs reinforcement learning is the urban traffic light control ([97], [98], [99]). Besides autonomous car driving ([100], [101]) including decision making [102].

5. Discussion

One of the important challenges or tradeoffs that exist in reinforcement learning is exploration versus exploitation which does not apply to any other paradigm we have seen, like supervised learning as well as unsupervised learning. Furthermore, to find the action that leads to the higher reward in a particular situation, the agent must explore its neighboring environment, because there is no other way to understand the effectiveness of doing a certain action in a particular situation. It's a necessity to try out different actions to evaluate which one will lead to a better reward in the future. However, we can't do exploration and exploitation without falling at the tradeoff task to find out which action the agent should take to maximize its reward. Furthermore, in stochastic settings where doing certain actions in the same situation will give different rewards, in this case doing more exploration is an obligation to come up with a better estimate or the long-term perspective of that particular action. Similarly, without exploration, we will never be able to know which action will lead to a higher reward. Hence, in reinforcement learning, this tradeoff is called the exploration-exploitation dilemma, and balancing them is the solution to an effective reinforcement learning algorithm. One more challenging point is that reinforcement learning focuses on the goal in an uncertain environment. Another type of challenge that we face while designing a reinforcement learning model is the reward design, which occurs while trying to decide how to give a reward for every action, which is a difficult task at the initial stage, it happens when we ignore how the game or the real-world application work. Moreover, the absence of a model. In this case, it'll be a need to gather experience over a period that will be used to design a reward as well as come up with a final optimal path in that particular case. The next challenge is the partial observability of states challenge, which happens when the environment is entirely not visible. Additionally, time-consuming operations are considered as a big challenge when opting for reinforcement learning. This goes back to the number of states and actions that are possible for every state while designing robots, games, and other real-world applications, which is going to take a lot of time. This leads to a very complex application in the end.

6. Conclusion

Reinforcement learning is a domain of machine learning that puts its focus on how an agent might take an action in the environment to maximize the recompenses. The main interest is to extend some of the algorithms cited into real life, which is a challenging task. The hardness comes from running a policy until termination because, in the real world, termination means crashing and destroying things by training and simulation, and modern simulators do not accurately depict the real world. Furthermore, they don't transfer to the real world when we deploy them, this means that in simulation the simulators work very well unlike when we deploy that policy in the real world. One of the most interesting applications in the real world is autonomous cars, where the car represents the agent and the environment is a 3d track. The possible actions here are to move left or right or to do nothing. Another interesting scope is robotics manipulation. Among the main advantages of reinforcement learning is that it's highly adaptive to the surrounding environment. Robots operating highly dynamic and ever-changing environments that's impossible to predict what's going to happen next, because of this, reinforcement learning provides a huge advantage to be accounting for these different scenarios and making sure that the robot's function is robust enough and able to handle all of the possible scenarios. Besides that, many other broad applications in real life are using reinforcement learning, like in healthcare, natural language processing, news recommendation, trading and finance, and many others.

References

[1] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.

[2] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto, "Reinforcement learning with prototypical representations," in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 920–11 931.

[3] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, no. 3, pp. 685–695, 2021.

[4] P. P. Shinde and S. Shah, "A review of machine learning and deep learning applications," in *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE, 2018, pp. 1–6.

[5] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, "A survey of generalisation in deep reinforcement learning," *arXiv preprint arXiv:2111.09794*, 2021.

[6] A. Oliver, A. Odena, C. A. Raffel, E. D. Cubuk, and I. Goodfellow, "Realistic evaluation of deep semi-supervised learning algorithms," *Advances in neural information processing systems*, vol. 31, 2018.

[7] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A gentle introduction to reinforcement learning and its application in different fields," *IEEE Access*, vol. 8, pp. 209 320–209 344, 2020.

[8] J. Shin, T. A. Badgwell, K.-H. Liu, and J. H. Lee, "Reinforcement learning–overview of recent progress and implications for process control," *Computers & Chemical Engineering*, vol. 127, pp. 282–294, 2019.

[9] Y. Qian, J. Wu, R. Wang, F. Zhu, and W. Zhang, "Survey on reinforcement learning applications in communication networks," *Journal of Communications and Information Networks*, vol. 4, no. 2, pp. 30–39, 2019.

[10] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei *et al.*, "Openspiel: A framework for reinforcement learning in games," *arXiv preprint arXiv:1908.09453*, 2019.

[11] J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. P. van Hasselt, S. Singh, and D. Silver, "Discovering reinforcement learning algorithms," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1060–1070, 2020.

[12] M. M. Afsar, T. Crump, and B. Far, "Reinforcement learning based recommender systems: A survey," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–38, 2022.

[13] T. Salimans and R. Chen, "Learning montezuma's revenge from a single demonstration," *arXiv preprint arXiv:1812.03381*, 2018.

[14] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences*, vol. 11, no. 11, p. 4948, 2021.

[15] D. Mehta, "State-of-the-art reinforcement learning algorithms," *International Journal of Engineering Research and Technology*, vol. 8, pp. 717–722, 2020.

[16] H. Fei, X. Li, D. Li, and P. Li, "End-to-end deep reinforcement learning based coreference resolution," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 660–665.

[17] E. R. Watters, "Factors in employee motivation: Expectancy and equity theories," *Journal of Colorado Policing*, vol. 970, p. 4, 2021.

[18] B. Recht, "A tour of reinforcement learning: The view from

continuous control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 253–279, 2019.

[19] K. Sharma, B. Singh, E. Herman, R. Regine, S. S. Rajest, and V. P. Mishra, "Maximum information measure policies in reinforcement learning with deep energy-based model," in *2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*. IEEE, 2021, pp. 19–24.

[20] V. Saggio, B. E. Asenbeck, A. Hamann, T. Strömberg, P. Schiansky, V. Dunjko, N. Friis, N. C. Harris, M. Hochberg, D. Englund *et al.*, "Experimental quantum speed-up in reinforcement learning agents," *Nature*, vol. 591, no. 7849, pp. 229–233, 2021.

[21] H. Helskyaho, J. Yu, and K. Yu, "Introduction to machine learning," in *Machine Learning for Oracle Database Professionals*. Springer, 2021, pp. 1–22.

[22] C. Li and M. Qiu, *Reinforcement Learning for Cyber-Physical Systems: with Cybersecurity Case Studies*. Chapman and Hall/CRC, 2019.

[23] A. G. Barto, "Reinforcement learning: Connections, surprises, and challenge," *AI Magazine*, vol. 40, no. 1, pp. 3–15, 2019.

[24] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," *arXiv preprint arXiv:2006.16712*, 2020.

[25] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine *et al.*, "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[26] C. Sun, J. Orbik, C. M. Devin, B. H. Yang, A. Gupta, G. Berseth, and S. Levine, "Fully autonomous real-world reinforcement learning with applications to mobile manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 308–319.

[27] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[28] P. Swazinna, S. Udluft, and T. Runkler, "Overcoming model bias for robust offline deep reinforcement learning," *Engineering Applications of Artificial Intelligence*, vol. 104, p. 104366, 2021.

[29] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value estimation for efficient model-free reinforcement learning," *arXiv preprint arXiv:1803.00101*, 2018.

[30] A. Balakrishnan, D. Bouneffouf, N. Mattei, and F. Rossi, "Using multi-armed bandits to learn ethical priorities for online ai systems," *IBM Journal of Research and Development*, vol. 63, no. 4/5, pp. 1–1, 2019.

[31] T. Lykouris, M. Simchowitz, A. Slivkins, and W. Sun, "Corruption-robust exploration in episodic reinforcement learning," in *Conference on Learning Theory*. PMLR, 2021, pp. 3242–3245.

[32] D. Bouneffouf, I. Rish, and C. Aggarwal, "Survey on applications of multi-armed and contextual bandits," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.

[33] R. N. Boute, J. Gijsbrechts, W. Van Jaarsveld, and N. Vanvuchelen, "Deep reinforcement learning for inventory control: A roadmap," *European Journal of Operational Research*, vol. 298, no. 2, pp. 401–412, 2022.

[34] Z. Wang and T. Hong, "Reinforcement learning for building controls: The opportunities and challenges," *Applied Energy*, vol. 269, p. 115036, 2020.

[35] Y. Liu, Y. Chen, and T. Jiang, "Dynamic selective maintenance optimization for multi-state systems over a finite horizon: A deep reinforcement learning approach," *European Journal of Operational Research*, vol. 283, no. 1, pp. 166–181, 2020.

[36] G. Farina and T. Sandholm, "Model-free online learning in unknown sequential decision making problems and games," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 6, 2021, pp. 5381–5390.

[37] T. Ding, Z. Zeng, J. Bai, B. Qin, Y. Yang, and M. Shahidehpour, "Optimal electric vehicle charging strategy with markov decision process and reinforcement learning technique," *IEEE Transactions on Industry Applications*, vol. 56, no. 5, pp. 5811–5823, 2020.

[38] B. Zhang, G. Zhang, W. Sun, and K. Yang, "Task offloading with power control for mobile edge computing using reinforcement learning-based markov decision process," *Mobile Information Systems*, vol. 2020, 2020.

[39] X. Xiang and S. Foo, "Recent advances in deep reinforcement learning applications for solving partially observable markov decision processes (pomdp) problems: Part 1—fundamentals and applications in games, robotics and natural language processing," *Machine Learning and Knowledge Extraction*, vol. 3, no. 3, pp. 554–581, 2021.

[40] S. Padakandla, P. KJ, and S. Bhatnagar, "Reinforcement learning algorithm for non-stationary environments," *Applied Intelligence*, vol. 50, no. 11, pp. 3590–3606, 2020.

[41] Y. Lin, J. McPhee, and N. L. Azad, "Comparison of deep reinforcement learning and model predictive control for adaptive cruise control," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 221–231, 2020.

[42] D. Zhou, J. He, and Q. Gu, "Provably efficient reinforcement learning for discounted mdps with feature mapping," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 793–12 802.

[43] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[44] R. Wang, R. R. Salakhutdinov, and L. Yang, "Reinforcement learning with general value function approximation: Provably efficient approach via bounded eluder dimension," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6123–6135, 2020.

[45] N. Yousefi, S. Tsianikas, and D. W. Coit, "Reinforcement learning for dynamic condition-based maintenance of a system with individually repairable components," *Quality Engineering*, vol. 32, no. 3, pp. 388–408, 2020.

[46] X. Gao, J. Si, Y. Wen, M. Li, and H. Huang, "Reinforcement learning control of robotic knee with human-in-the-loop by flexible policy iteration," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[47] Y. Wang, C. Tang, S. Wang, L. Cheng, R. Wang, M. Tan, and Z. Hou, "Target tracking control of a biomimetic underwater vehicle through deep reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[48] D. Daoun, F. Ibnat, Z. Alom, Z. Aung, and M. A. Azim, "Reinforcement learning: a friendly introduction," in *The International Conference on Deep Learning, Big Data and Blockchain*. Springer, 2022, pp. 134–146.

[49] P.-A. Andersen, M. Goodwin, and O.-C. Granmo, "Deep rts: a game environment for deep reinforcement learning in real-time strategy games," in *2018 IEEE conference on computational intelligence and games (CIG)*. IEEE, 2018, pp. 1–8.

[50] H. Yoo, B. Kim, J. W. Kim, and J. H. Lee, "Reinforcement learning based optimal control of batch processes using monte-carlo deep deterministic policy gradient with phase segmentation," *Computers & Chemical Engineering*, vol. 144, p. 107133, 2021.

[51] S. Alaniz, "Deep reinforcement learning with model learning and monte carlo tree search in minecraft," *arXiv preprint arXiv:1803.08456*, 2018.

[52] J. Subramanian and A. Mahajan, "Renewal monte carlo: Renewal theory-based reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 65, no. 8, pp. 3663–3670, 2019.

[53] I. P. Pinto and L. R. Coutinho, "Hierarchical reinforcement learning with monte carlo tree search in computer fighting game," *IEEE transactions on games*, vol. 11, no. 3, pp. 290–295, 2018.

[54] J. Bhandari, D. Russo, and R. Singal, "A finite time analysis of temporal difference learning with linear function approximation," in *Conference on learning theory*. PMLR, 2018, pp. 1691–1692.

[55] A. M. Devraj, I. Kontoyiannis, and S. P. Meyn, "Differential temporal difference learning," *IEEE Transactions on Automatic Control*, vol. 66, no. 10, pp. 4652–4667, 2020.

[56] Q. Lin and Q. Ling, "Decentralized td (0) with gradient tracking," *IEEE Signal Processing Letters*, vol. 28, pp. 723–727, 2021.

[57] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa," *IEEE Access*, vol. 8, pp. 54 074–54 084, 2020.

[58] Z.-x. Xu, L. Cao, X.-l. Chen, C.-x. Li, Y.-l. Zhang, and J. Lai, "Deep reinforcement learning with sarsa and q-learning: a hybrid approach," *IEICE TRANSACTIONS on Information and Systems*, vol. 101, no. 9, pp. 2315–2322, 2018.

[59] Y. Bao, Y. Zhu, and F. Qian, "A deep reinforcement learning approach to improve the learning performance in process control," *Industrial & Engineering Chemistry Research*, vol. 60, no. 15, pp. 5504–5515, 2021.

[60] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103500, 2021.

[61] H. S. Barjouei, H. Ghorbani, N. Mohamadian, D. A. Wood, S. Davoodi, J. Moghadasi, and H. Saberi, "Prediction performance advantages of deep machine learning algorithms for two-phase flow rates through wellhead chokes," *Journal of Petroleum Exploration and Production*, vol. 11, no. 3, pp. 1233–1261, 2021.

[62] M. Sewak, "Deep q network (dqn), double dqn, and dueling dqn," in *Deep Reinforcement Learning*. Springer, 2019, pp. 95–108.

[63] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[64] S. Khodadadian, T. T. Doan, J. Romberg, and S. T. Maguluri, "Finite sample analysis of two-time-scale natural actor-critic algorithm," *IEEE Transactions on Automatic Control*, 2022.

[65] Y. Xiao, X. Lyu, and C. Amato, "Local advantage actor-critic for robust multi-agent deep reinforcement learning," in *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 2021, pp. 155–163.

[66] K. Zhou, W. Wang, T. Hu, and K. Deng, "Application of improved asynchronous advantage actor critic reinforcement learning model on anomaly detection," *Entropy*, vol. 23, no. 3, p. 274, 2021.

[67] H. Yoo, B. Kim, J. W. Kim, and J. H. Lee, "Reinforcement learning based optimal control of batch processes using monte-carlo deep deterministic policy gradient with phase segmentation," *Computers & Chemical Engineering*, vol. 144, p. 107133, 2021.

[68] Q. Shi, H.-K. Lam, C. Xuan, and M. Chen, "Adaptive neuro-fuzzy pid controller based on twin delayed deep deterministic policy gradient algorithm," *Neurocomputing*, vol. 402, pp. 183–194, 2020.

[69] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[70] A. Stooke and P. Abbeel, "Accelerated methods for deep reinforcement learning," *arXiv preprint arXiv:1803.02811*, 2018.

[71] Y. Lin, M. Wang, X. Zhou, G. Ding, and S. Mao, "Dynamic spectrum interaction of uav flight formation communication with priority: A deep reinforcement learning approach," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 3, pp. 892–903, 2020.

[72] S. Y. Luis, D. G. Reina, and S. L. T. Marín, "A multiagent deep reinforcement learning approach for path planning in autonomous surface vehicles: The ypacaraí lake patrolling case," *IEEE Access*, vol. 9, pp. 17 084–17 099, 2021.

[73] E. Bøhn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2019, pp. 523–533.

[74] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *arXiv preprint arXiv:1802.10592*, 2018.

[75] Y. Chu, Z. Wei, G. Sun, H. Zang, S. Chen, and Y. Zhou, "Optimal home energy management strategy: A reinforcement learning method with actor-critic using kronecker-factored trust region," *Electric Power Systems Research*, vol. 212, p. 108617, 2022.

[76] Y. L. E. Nuin, N. G. Lopez, E. B. Moral, L. U. S. Juan, A. S. Rueda, V. M. Vilches, and R. Kojcev, "Ros2learn: a reinforcement

learning framework for ros 2," *arXiv preprint arXiv:1903.06282*, 2019.

[77] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa," *IEEE Access*, vol. 8, pp. 54 074–54 084, 2020.

[78] M. Thabet, "Imagination-augmented deep reinforcement learning for robotic applications," Ph.D. dissertation, University of Manchester, 2022.

[79] S. Alaniz, "Deep reinforcement learning with model learning and monte carlo tree search in minecraft," *arXiv preprint arXiv:1803.08456*, 2018.

[80] J. Jia and W. Wang, "Review of reinforcement learning research," in *2020 35th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, 2020, pp. 186–191.

[81] E. Akanksha, N. Sharma, K. Gulati *et al.*, "Review on reinforcement learning, research evolution and scope of application," in *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE, 2021, pp. 1416–1423.

[82] A. K. Mondal and N. Jamali, "A survey of reinforcement learning techniques: strategies, recent development, and future directions," *arXiv preprint arXiv:2001.06921*, 2020.

[83] H. Moussaoui, M. Benslimane, and N. El Akkad, "A novel brain tumor detection approach based on fuzzy c-means and marker watershed algorithm," in *International Conference on Digital Technologies and Applications*. Springer, 2021, pp. 871–879.

[84] D. Zhang, B. Chen, and S. Li, "Sequential conditional reinforcement learning for simultaneous vertebral body detection and segmentation with modeling the spine anatomy," *Medical Image Analysis*, vol. 67, p. 101861, 2021.

[85] H. Moussaoui, M. Benslimane, and N. El Akkad, "Image segmentation approach based on hybridization between k-means and mask r-cnn," in *WITS 2020*. Springer, 2022, pp. 821–830.

[86] L. Khrissi, N. El Akkad, H. Satori, and K. Satori, "Clustering method and sine cosine algorithm for image segmentation," *Evolutionary Intelligence*, vol. 15, no. 1, pp. 669–682, 2022.

[87] L. KHRISSI, N. EL AKKAD, H. SATORI, and K. SATORI, "An efficient image clustering technique based on fuzzy c-means and cuckoo search algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, 2021.

[88] L. Khrissi, N. E. Akkad, H. Satori, and K. Satori, "Color image segmentation based on hybridization between canny and k-means," in *2019 7th Mediterranean Congress of Telecommunications (CMT)*. IEEE, 2019, pp. 1–4.

[89] L. Khrissi, N. El Akkad, H. Satori, and K. Satori, "Image segmentation based on k-means and genetic algorithms," in *Embedded Systems and Artificial Intelligence*. Springer, 2020, pp. 489–497.

[90] H. Moussaoui, N. El Akkad, and M. Benslimane, "Moroccan carpets classification based on svm classifier and orb features," in *International Conference on Digital Technologies and Applications*. Springer, 2022, pp. 446–455.

[91] L. KHRISSI, N. EL AKKAD, H. SATORI, and K. SATORI, "A performant clustering approach based on an improved sine cosine algorithm," 2022.

[92] L. M. Gladence, C. Vakula, M. P. Selvan, T. Samhita *et al.*, "A research on application of human-robot interaction using artifical intelligence," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 9, pp. 784–787, 2019.

[93] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley, "Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review," *Robotics*, vol. 10, no. 1, p. 22, 2021.

[94] C. Yu, J. Liu, S. Nemati, and G. Yin, "Reinforcement learning in healthcare: A survey," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–36, 2021.

[95] X. Wu, J. Chi, X.-Z. Jin, and C. Deng, "Reinforcement learning approach to the control of heavy material handling manipulators for agricultural robots," *Computers and Electrical Engineering*, vol. 104, p. 108433, 2022.

[96] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," *arXiv preprint arXiv:1912.10944*, 2019.

[97] T. Wu, P. Zhou, K. Liu, Y. Yuan, X. Wang, H. Huang, and D. O. Wu, "Multi-agent deep reinforcement learning for urban traffic light control in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 8243–8256, 2020.

[98] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 7153–7162.

[99] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2019.

[100] R. R. O. Al-Nima, T. Han, and T. Chen, "Road tracking using deep reinforcement learning for self-driving car applications," in *International Conference on Computer Recognition Systems*. Springer, 2020, pp. 106–116.

[101] A. Folkers, M. Rick, and C. Büskens, "Controlling an autonomous vehicle with deep reinforcement learning," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 2025–2031.

[102] J. Duan, S. Eben Li, Y. Guan, Q. Sun, and B. Cheng, "Hierarchical reinforcement learning for self-driving decision-making without reliance on labelled driving data," *IET Intelligent Transport Systems*, vol. 14, no. 5, pp. 297–305, 2020.

Hanae Moussaoui Is currently a Phd student at faculty of sciences and techniques of fes. LISA Laboratory of ENSA- FES. Her research interests include Artificial intelligence, Machine Learning, Image Processing,Image classification, Deep learning

Nabil El Akkad Dr. Nabil EL AKKAD is an Associate Professor, Department of Computer Science, Sidi Mohammed Ben Abdellah University - Fez - Morocco. He received the PhD degree from Faculty of sciences, Sidi Mohammed Ben Abdellah University - Fez - Morocco. He is currently a professor of computer science at National School of Applied Sciences (ENSA)of Fez,He is a member of the LISA Laboratory. His research interests include Artificial intelligence, Camera self-calibration, Machine Learning, Cryptography, 3D Reconstruction, Image Processing, Data Mining, Image Segmentation, Image classification.

Mohamed Benslimane Dr. Mohamed Benslimane is a Professor, Department of Computer Science, Sidi Mohammed Ben Abdellah University - Fez - Morocco. He is currently a professor of computer science at the Superior School of Technology (EST)of Fez. He is a member of the LTI Laboratory at (EST) of Fez.