Advances and Challenges in Learning from Experience Replay

Daniel Eugênio Neves (daniel.eugenio@sga.pucminas.br)
Pontifícia Universidade Católica de Minas Gerais
Zenilton Kleber Gonçalves do Patrocínio Júnior
Pontifícia Universidade Católica de Minas Gerais
Lucila Ishitani
Pontifícia Universidade Católica de Minas Gerais

Research Article

Keywords: Reinforcement Learning, Deep Reinforcement Learning, Experience Replay, Survey

Posted Date: June 14th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3051202/v1

Additional Declarations: No competing interests reported.

Advances and Challenges in Learning from Experience Replay

Daniel Eugênio Neves, Lucila Ishitani and Zenilton Kleber Gonçalves do Patrocínio Júnior

Instituto de Ciências Exatas e Informática, Pontifícia Universidade Católica de Minas Gerais, Avenida Dom José Gaspar, 500, Belo Horizonte, 30535-901, Minas Gerais, Brazil.

*Corresponding author(s). E-mail(s): daniel.eugenio@sga.pucminas.br;
Contributing authors: lucila@pucminas.br; zenilton@pucminas.br;

Abstract

From the first theoretical propositions in the '50s, inspired by the neuroscience and psychology studies about the learning processes in human beings and animals, to its application in real-world problems on learning to action, Reinforcement Learning (RL) is still being a fascinating, rich, and complex class of machine learning algorithms. In particular, we will start reviewing its fundamental principles and develop a discussion about how a technique called Experience Replay (ER) has been of fundamental importance in making a variety of methods in most of the relevant problems and different domains more data efficient, using agent experiences to improve its performance. We present some of the more relevant methods in the literature, which base most recent research on improving RL with ER. Finally, we bring from the recent literature some of the main trends, challenges, and advances focused on reviewing and discussing ER formal basement and how to improve its propositions to make it even more efficient in different methods and domains.

Keywords: Reinforcement Learning, Deep Reinforcement Learning, Experience Replay, Survey

1 Introduction

Machine learning is an artificial intelligence research field in which mathematical and statistical models applied through computational algorithms seek to provide machines with intelligent behavior and the ability to learn from their experiences. In particular, Reinforcement Learning (RL) research dedicates to developing intelligent computational entities called agents, which try to learn an action policy to interact with the environment and perform a given task, i.e., each agent action on an arbitrary environment state results in a new state and a reward signal and the objective is to learn an optimal policy that maximizes the total expected reward in the long run. If the agent learns an optimal action policy, it will have learned to perform the task. Based on the formal framework of the Markov Decision Process (MDP), different RL methods have been proposed from their early formulations in the '50s to deal with problems involving decision-making sequences. These processes run by defining value functions to measure the agent choices, either using dynamic programming in tabular environments to find exact solutions for discrete action values or approximating these value functions in high-dimensional state spaces with either discrete or continuous action values.

Temporal Difference Learning (TD-learning) was an essential formulation for developing trial-and-error-based learning methods. It allows the agent to adjust itself from a target value based on the reward estimate on a resultant state in the previous interaction, making it possible for many recent algorithms to converge to an optimal policy even if acting sub-optimally, as long as it keeps updating its value function. And this was the grounds for a whole class of off-policy and model-free methods. To cope with some efficiency issues inherent to TD-learning and make TD-based approaches more data efficient, Lin (1992) proposed a fundamental technique called Experience Replay (ER), consisting of reusing previous agent experiences (i.e., previous state transitions) in updating its value function by storing and uniformly sampling them from a replay buffer. This strategy brought unique benefits related to using artificial neural networks in approximating value functions because it decorrelates the agent training data.

A wide range of modern methods employ ER. At the same time, many authors seek to improve it by investigating how to make better use of the replay buffer, how to sample better experiences for agent learning, how to deal with a size-limited buffer (and how big it should be), how to model that buffer and many other questions. Because of its benefits for data-efficient reinforcement learning, research on using and improving ER increased exponentially from 2016, with a disproportionate volume of publications compared to previous years. Therefore, this work aims to review from the early bases until current reinforcement learning methods to formally understand and compare how they use ER and how it makes them more data efficient. Moreover, we seek to contribute to understanding its fundamental ideas and highlight its many theoretical and empirical open problems still being investigated to organize and point out some possible future works and research directions. Moreover,

we are narrowly interested in those works that propose changes or new methods using ER and especially interested in those works that investigate the ER itself and delve into its theoretical issues and empirical investigations.

This work is organized as follows. In Section 2, we present and discuss the theoretical background and some related works. Section 3 presents the Experience Replay foundations. Section 4 presents relevant deep reinforcement learning methods focusing on how they use ER and the differences in their propositions to explore agents' experiences. Section 5 discusses some research challenges and trends and also presents an extensive literature review. Finally, we draw some conclusions in Section 6.

2 Background

Reinforcement Learning uses the formal framework of the Markov Decision Process (MDP) to define the interaction between a learning agent and its environment regarding states, actions, and rewards. An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, so that: $\mathcal{S}$ represents a set of states; $\mathcal{A}$ is a set of actions $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$; $\mathcal{P}(s' \mid s, a)$ is the probability for transiting from state s to s' $(s, s' \in S)$ by taking action $a \in A$; $\mathcal{R}$ is a reward function mapping each state-action pair to a reward in $\mathbb{R}$; and $\gamma \in [0, 1]$ is a discount factor. The agent behavior is represented by a policy π, and the value $\pi(a \mid s)$ represents the probability of taking action a at state s. At each time step t, the agent observes a state $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$ that determines the reward $r_t = \mathcal{R}(s_t, a_t)$ and next state $s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t)$, causing a transition of states $T(s, a, s')$. A discounted sum of future rewards is called return $R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$. The agent goal is to learn (or approximate) an optimal policy π^* that maximizes the long-term expected (discounted) reward value. These processes imply non-deterministic search problems and stochastic decision sequences for selecting actions from observing each state of the environment resulting from a previous decision. This way, each agent action determines the immediate reward and, more importantly, influences subsequent environment states and future rewards. While the immediate reward informs about the result of an action performed in the current state, the long-term expected reward allows evaluation of the action policy by a value function.

Bellman formulated the MDPs as a stochastic version of the optimal control problem and described two value functions applying the concepts of states of a dynamical system. The state-value function $v_\pi(s)$ seeks to estimate the expected total (discounted) reward value when the agent starts in state s and follows the policy π. Describing $v_\pi(s)$, one can note (see Equations 1–4) the expected sum of future rewards for the states reached by adopting the policy π and performing the sequences of states transitions. In (5), it is clear the dynamic nature of the computation of $v_\pi(s)$, and in (6) we have the discount value γ on the total expected reward.

$$v_\pi(s) = \mathbb{E}_\pi[r_1 + r_2 + ... + r_T | s_t = s] \tag{1}$$
$$= \mathbb{E}_\pi[r_t] + \mathbb{E}_\pi[r_{t+1} + r_{t+2} + ... + r_T | s_t = s] \tag{2}$$
$$= \sum_a \pi(s,a) R(s,a) + \mathbb{E}_\pi[r_{t+1} + r_{t+2} + \cdots + r_T | s_t = s] \tag{3}$$
$$= \sum_a \pi(s,a) R(s,a) + \sum_a \pi(s,a) \sum_{s'} T(s,a,s') \mathbb{E}_\pi[r_{t+1} + \cdots + r_T | s_t = s'] \tag{4}$$
$$= \sum_a \pi(s,a) R(s,a) + \sum_a \pi(s,a) \sum_{s'} T(s,a,s') v_\pi(s')] \tag{5}$$
$$= \sum_a \pi(s,a) [R(s,a) + \gamma \sum_{s'} T(s,a,s') v_\pi(s')] \tag{6}$$

In the form presented by Sutton and Barto (2018) in the Equation 7, the state-value function $v_\pi(s)$ clearly demonstrates the notions of probability and transition, making the relationship explicit between the value of a state and the values of its successor states. In turn, the action-value functions $q_\pi(s,a)$ seek to estimate the total expected reward if the agent takes an action a at the state s and follows the policy π, allowing it to evaluate the utility of each possible action at that state (Equation 8).

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a)[r + \gamma v_\pi(s')] \tag{7}$$
$$q_\pi(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s')[\sum_a \pi(s',a') q_\pi(s',a')] \tag{8}$$

In MDPs, a policy π is better or equivalent to another policy π' if $v_\pi(s) \geq v_{\pi'}(s), \forall s \in S$. In all cases, there is always at least one optimal policy π^* that is better than or equal to all others. These policies share the same optimal state-value function $v^*(s) = max_\pi v_\pi(s)$, which is the highest value that can be obtained for each state, and the same optimal action-value function $q^*(s,a) = max_\pi q_\pi(s,a), \forall s, a \in S, A$ (Sutton and Barto, 2018). It is possible to write the optimal action-value function relative to the optimal state-value function so that $q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a]$. Since $v^*(s)$ is an optimal state-value function, Bellman's equation demonstrates that the value of a state under an optimal policy must be equal to the expected return for the best action in that state:

$$v^*(s) = max_{a \in A(s)} q_{\pi*}(s,a) \tag{9}$$
$$= max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v^*(s')]$$

In turn, the Bellman optimality equation for the action-value function can be defined as follows:

$$q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma max_{a'} q^*(S_{t+1}, a') | S_t = s, A_t = a] \\ = \sum_{s',r} p(s', r|s, a)[r + \gamma max_{a'} q^*(s', a')] \tag{10}$$

From v^*, one can find π^* and vice versa, both of which are solutions for MDPs. For each state, one or more actions will produce the maximum value in Bellman's equation, and any policy that maximizes v^* will be optimal. While knowledge of the optimal state-value function v^* makes it possible to search for the optimal policy π^*, knowing the optimal action-value function q^* makes it easy to choose optimal actions. This way, for any state s, the agent only needs to find one action that maximizes the value of q^* because it effectively stores the results for all searches one step ahead. It gives the optimal expected return as a local and immediately available value for each state-action pair. It allows one to select optimal actions without knowing the possible successor states and their respective values. In other words, without knowing the dynamics of the environment.

Bellman's equations allow for finding optimal policies. Still, they are rarely used in practice, as they demand exhaustive searches in the space of states and actions besides assuming that the dynamics of the environment are precisely known, which is not always true. It imposes limitations on a class of methods known as Dynamic Programming (DP) (Szepesvári, 2010), which can converge to optimal policies with exact solutions and provide the basis for understanding several other reinforcement learning methods since many of them consist of attempts to achieve the same results but at a lower computational cost, and without the need for a perfect model of the environment (Sutton and Barto, 2018). The main idea of DP is to use value functions to search for optimal policies, assuming that the environment is described as an MDP, the sets of states, actions, and rewards are finite, and there is a probability function that describes the environment's dynamics. This way, DP can compute the value functions transforming the Bellman equations into updating rules. In this sense, there are four main related algorithms: policy evaluation, policy improvement, policy iteration, and value iteration.

The policy evaluation method is an iterative solution that uses the state-value function. For a sequence of functions $\{v_0, \ldots, v_k\}$ mapping the states to values, the value for v_0 is chosen arbitrarily and updated from the values computed in the subsequent iterations using the Bellman equation for v_π as an update rule, in which the state-value function at iteration $v_{k+1}(s)$ considers the expected discounted return obtained for the next possible state s' in the previous iteration, for every state $s \in S$. It is possible to demonstrate that the sequence of value-functions v_k converges to v_π when $k \to \infty$. At each iteration, to produce v_{k+1} from v_k, the algorithm applies the same operation to each state s, assigning it a new value obtained from the previous values

of the state s', a successor of s, and the expected immediate reward for each possible transition under the policy that is being evaluated. This way, each iteration updates the value of each state to produce a new approximation of the state-value function v_{k+1}:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] \\ &= \sum_a \pi(s|a) \sum_{s',a} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{11}$$

Even having determined the state-value function v_π for a policy π, it is still possible to check whether or not it would be better to select a given action a different from what determined the policy in that state. One way to answer this question is to compute the result for the action-value function $q_\pi(s, a)$, introducing a policy improvement step into the policy evaluation method:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= \sum_{s',a} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{12}$$

The policy will change if $q_\pi(s, a) > v_\pi(s)$, since it will be better to choose the action a in the state s and then follow the policy π instead of follow π all the time. So it is expected that it will be better to select the action a every time the state s is found and that this new policy will be the best overall. Therefore, it is possible to consider changes in all states for all possible actions in a greedy strategy, selecting in each state the best action according to $q_\pi(s, a)$, so that the new policy π' is given by:

$$\begin{aligned} \pi'(s) &= argmax_a q_\pi(s, a) \\ &= argmax_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{13}$$

This is a special case of the policy improvement theorem. Let two policies be π and π' so that, for all $s \in S$, $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, so the policy π' must be as good as or better than π (i.e. π' must have an expected reward greater than or equal to π), where $v_{\pi'}(s) \geq v_\pi(s)$. This result applies particularly to the original policy π and the modified policy π'. If $q_\pi(s, a) > v_\pi(s)$, then the modified policy will be better than the original policy. It demonstrates that, given a policy and its value function, it is possible to evaluate a policy change in a single state for a given action (Sutton and Barto, 2018). Once a policy π has been improved, a policy iteration process produces a sequence of improvements until it reaches an optimal policy π^* and an optimal value function v^*, as each new action policy is guaranteed to be better than the previous one unless this is already the optimal policy. Considering that a finite

MDP has a finite number of policies, this process must converge to an optimal policy and value function in a finite number of iterations.

Even though convergence to the optimal policy and optimal value function is guaranteed, each policy iteration step includes the policy evaluation, which is also iterative, leading to a computation that requires many scans in the space of states. However, truncating the policy evaluation step is possible without losing the policy iteration convergence guarantee. A special case occurs when the policy evaluation stops after a single scan (i.e., after an update step of each state). This method is called value iteration and is a simple update process that combines policy improvement and short policy evaluation steps, as in Equation 14, for all $s \in S$:

$$\begin{aligned} v_{k+1}(s) &= max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \end{aligned} \tag{14}$$

It achieves faster convergence by interposing multiple policy evaluation scans between each policy improvement scan, while its output consists of a deterministic policy $\pi \approx \pi^*$ such that $\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$.

According to Sutton and Barto (2018), Temporal Difference Learning (TD) is one of the most relevant ideas of reinforcement learning. It allows learning to occur directly from the agent experience and without the need for a model of the dynamics of the environment, being able to update estimates based on other learned estimations without having to wait to find a final state, in a clear advantage over DP methods concerning computational efficiency. There are variations of the TD method regarding the number of steps applied to the calculation of the temporal difference, called by the acronym $TD(\lambda)$, where λ is the number of steps. TD(0) is a particular case and updates the estimate of $v(s_t)$ for an iteration t using the observed immediate reward r and the estimate of $v(s_{t+1})$ at iteration $t+1$. Thus, it waits only for the next iteration to form a target value $r + \gamma v(s_{t+1})$ and updates the value of s_t immediately after the transition to s_{t+1}, so that $v(s_t) \leftarrow v(s_t) + \alpha[r + \gamma v(s_{t+1}) - v(s_t)]$, where α is a learning rate, γ is the discount value, and s_t and s_{t+1} are respectively the environment's states at iterations t and $t+1$. The difference $\gamma v(s_{t+1}) - v(s_t)$ is known as the Temporal Difference Error (TDE). Updates by sampling, like those used in TD methods, are different from those used in dynamic programming methods, as they are based on a single successor state and not on a complete probability distribution over all possible successors. Thus, TD methods are independent of a model of the environment, are naturally implemented online and incrementally, and converge to an optimal policy.

Q-Learning (Watkins and Dayan, 1992) is a model-free and off-policy algorithm that applies successive steps to update the estimates for the action-value function $Q(s, a)$ (that approximates the long-term expected discounted reward

value of executing an action from a given state) using TD-learning and minimizing the TD-error (defined by the difference in Equation 15). This function is named Q-function, and its estimated returns are known as Q-values. A higher Q-value indicates that an action a would yield better long-term results in state s. Q-Learning converges to an π^* even if it is not acting optimally every time as long it keeps updating the Q-value estimates for all the pairs of state-action and generates a variation of the usual stochastic approximation conditions through subsequent changes of α, as we can describe in Equation 15.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{15}$$

Applying Q-learning to those problems where the space of states and actions is too large to learn all actions values in all the possible states and/or when these states are multidimensional data, one can achieve a good approximate solution by learning a parameterized value function $Q(s, a, \Theta_t)$ as in Equation 16 (Van Hasselt et al, 2016).

$$\Theta_{t+1} = \Theta_t + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \Theta_t) - Q(s_t, a_t; \Theta_t)]\nabla_{\Theta_t} Q(s_t, a_t; \Theta_t) \tag{16}$$

One can note that the target function in the TD-error calculation consists of a greedy policy defined by the *max* function. This way, the maximum over the estimated values is implicitly used as the estimate of the highest return value, which can lead to considerable maximization bias. For example, in a state s, there may be many actions for which the actual return value $q(s, a)$ is zero, but the estimated values $Q(s, a)$ may be distributed over negative and positive values. In this case, always taking the maximum introduces a clear positive bias on this set of actions. Such maximization bias can lead the agent to choose misguided actions more often in a given state. One way to approach this problem is to use two independent estimates, $Q_1(s, a)$ and $Q_2(s, a)$, of the actual value of $q(s, a)$, $\forall a \in A$. So one can use $Q_1(s, a)$ to determine the action $a^* = argmax_a Q_1(s, a)$ and $Q_2(s, a)$ to provide the estimate $Q_2(s, a^*) = Q_2(s, argmax_a Q_1(s, a))$. This way, it's also possible to perform the same process by reversing the roles of $Q_1(s, a)$ and $Q_2(s, a)$ to obtain a second estimate of reduced bias from $Q_1(s, argmax_a Q_2(s, a))$. This is the approach proposed by van Hasselt (2010) to formulate the method called Double Q-Learning (Equation 17), in which only one of the estimates is updated at each training step based on a probability value. The authors demonstrated that this approach reduced the bias by decomposing the max operation in the target function into action selection and action evaluation, which improved the action-value function updates making it more stable by diminishing the overestimation of the Q-values.

$$Q_1(s, a) \leftarrow Q_1(s, a) + \alpha[r + \gamma Q_2(s_{t+1}, argmax_a Q_1(s_{t+1}, a)) - Q_1(s, a)] \tag{17}$$

To use parameterized value functions, Double Q-Learning learns two value functions using two different sets of weights Θ and Θ' and, at each update

step, one set is used to select the action greedily and the other to evaluate it, as defined in Equations 18 and 19.

$$y_t = r_t + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1}, a, \Theta_t); \Theta'_t) \quad (18)$$

$$\Theta_{t+1} = \Theta_t + \alpha[y_t - Q(s_t, a_t; \Theta_t)]\nabla_{\Theta_t} Q(s_t, a_t; \Theta_t) \quad (19)$$

As an agent interacts with stochastic, non-deterministic, and partially observable environments, *exploration* and *exploitation* are also essential concepts, and how to balance them is a recurring challenge. Exploration refers to the experimentation of the new and the generation of new knowledge but tends to maximize risks to expand the agent's knowledge. Exploitation relates to assimilated knowledge, maximizing efficiency and performance, minimizing risks, and refining the knowledge already acquired. In this way, to increase the accumulated reward value, an agent seeks to select actions they have already tried and produced good results, but it is necessary to try new actions to discover new ones that may provide greater rewards in the future, thus choosing between obtaining rewards quickly or have a chance to select better action in the future. The agent must try various actions and progressively select the ones that seem the best. This dilemma is a complex problem and has not yet been exhausted in the literature.

3 Experience Replay

After an agent has performed a sequence of actions and received a return value, knowing how to assign credit (or discredit) to each state-action pair consists of a difficult problem called *Temporal Credit Assignment*. Temporal Difference Learning (TD) represents one of the main techniques to deal with this problem, despite being a slow process, especially when it involves credit propagation over a long sequence of actions. For example, Adaptive Heuristic Critic – AHC (Sutton, 1992) and Q-Learning (Watkins and Dayan, 1992), which represent the first TD-learning-based methods in RL, are characterized by high convergence times. An effective technique named Experience Replay (ER) was proposed by Lin (1992) to speed up the credit attribution process and, consequently, reduce the convergence time by storing the agent experiences in a replay buffer and uniformly sampling past experiences to update the agent model. One of its main motivations is that relevant algorithms become inefficient when they use trial-and-error experiences only once to adjust the evaluation functions and then discard them. This is because some agents' experiences can be rare, while others can be expensive to obtain, such as those involving penalties. Moreover, using ER with random sampling reduces the effect of the correlation of the data (that represents the environment states). It improves its non-stationary distribution when using neural networks to approximate the value functions because it softens the distribution over many previous experiences (Mnih et al, 2013). The correlation between consecutive observations of the environment's

states can lead the minor updates on the approximate model to cause considerable changes in the policy learned by the agent (Mnih et al, 2015). This way, it can change the data distribution and the relation between the action-value functions in the calculation of the TD-error.

Lin (1992) compared Experience Replay to two other techniques: (i) *Learning Action Models for Planning* and (ii) *Teaching*, regarding shortening the trial-and-error process. In *Learning Action Models for Planning*, the agent does not need to learn a model of actions by itself, as in such cases when there is a perfect environment's model or when it can quickly learn a good one. In *Teaching*, lessons from a teacher (i.e., a human player or maybe another agent) demonstrate how to get from an initial to a final state accomplishing the task goal, and taught lessons store selected actions, state transitions, and received rewards. This way, an agent can repeat these lessons several times, similar to what it could do with its own experiences. The author applied the three techniques in eight reinforcement learning frameworks based on AHC and Q-learning: (i) AHCON (Connectionist AHC-learning); (ii) AHCON-R (AHCON using Experience Replay); (iii) AHCON-M (AHCON using Actions Model); (iv) AHCON-T (AHCON using Experience Replay and Teaching); (v) QCON (Connectionist Q-learning); (vi) QCON-R (QCON with Experience Replay); (vii) QCON-M (QCON with Actions Model); and (viii) QCON-T (QCON with Experience Replay and Teaching). These frameworks seek to learn a policy evaluation function approximated by neural networks and adjusted using TD and backpropagation.

According to Lin (1992), using ER in AHCON-R and QCON-R not improved the agents' performance (compared to AHCON and QCON) but speeded up the convergence in all experiments. In turn, using action models in AHCON-M and QCON-M could speed up the convergence time, but this did not happen during the experiments. Comparing AHCON-T and QCON-T to AHCON-R and QCON-R, there were no significant differences in the lesser complex environments, but in complex ones, AHCON-T and QCON-T were considerably faster. Therefore, the author states that the advantage of using *Teaching* becomes more significant as the task becomes harder and has demonstrated the superiority of Experience Replay over the *Actions Model* when agents need to learn a model of the environment by themselves. However, the latter method is superior if a perfect action model is provided. But it does not seem advantageous, especially in problems with large spaces of states and actions, non-deterministic scenarios, and non-tabular solutions, where there is no way to provide a perfect action model to the agent considering all possible situations.

Experience Replay has some limitations. As it repeats experiences through uniform sampling, if those samples define policies very different from what the agent is learning, it can underestimate the evaluation and utility functions, which affects methods that use neural networks because whenever it adjusts the weights for a given state, this affects the entire model concerning many (or perhaps all) other states. Besides, the memory of experiences does not

differentiate relevant experiences due to the uniform sampling and overwrites many agent experiences due to the buffer size limitation. That points to the need for more sophisticated strategies that can emphasize experiences capable of contributing more to agent learning in the sense of what was proposed by Schaul et al (2016). Another relevant problem is related to the replay buffer size. According to Mnih et al (2015), all the DQN-based methods (some of the most relevant are presented in Section 4) used a fixed replay memory size of 1M transitions. Recently, some research works investigated the effects of small and large buffers (Zhang and Sutton, 2017; Liu and Zou, 2018). In turn, Neves et al (2022) used a small dynamic memory to explore experience replay and the dynamics of the transitions, reducing the required number of experiences to agent learning.

4 Experience Replay in Deep Reinforcement Learning

Deep Q-Network (DQN) (Mnih et al, 2013) and Double Deep Q-Network (DDQN) (Van Hasselt et al, 2016) are two relevant methods based on Q-Learning and Double Q-Learning with ER. These methods achieved state-of-the-art results and human-level performance in learning to play a set of Atari 2600 games emulated in the Arcade Learning Environment (ALE) (Bellemare et al, 2013; Machado et al, 2018), which allows complex challenges for RL agents such as non-determinism, stochasticity, and exploration. To approximate the action-value functions, the authors used Convolutional Neural Networks (CNN) on the representations of environment states obtained from the video game frames without any prior information regarding the games, no manually extracted features, and no knowledge regarding the internal state of the ALE emulator. Thus, agent learning occurred only from video inputs, reward signals, the set of possible actions, and the final state information of each game. The authors attributed their state-of-the-art results mainly to the ability of their CNN to represent the games' states.

Mnih et al (2015) improved DQN by changing the way CNN is used as shown in Algorithm 1. Instead of using the same network (with the same parameters Θ) for approximating both the action-value function $Q(s, a, \Theta)$ and the target action-value function $Q(s', a, \Theta)$, the authors used independent sets of parameters Θ and Θ' for each network. Thus, only the function $Q(s, a, \Theta)$ has its parameters Θ updated by backpropagation. The parameters Θ' are updated directly (i.e., copied) from the values of Θ with a certain frequency, remaining unchanged between two consecutive updates. Thus, only the *forward pass* is carried out, when the network is used with the parameters Θ' to predict the value of the target function. Specifically, at each time-step t, a transition (or experience) is defined by a tuple $\tau_t = (s_t, a_t, r_t, s_{t+1})$, in which s_t is the current state, a_t is the action taken at that state, r_t is the received reward at t, and s_{t+1} is the resulting state after taking action a_t. Recent experiences are stored to construct a replay buffer $\mathcal{D} = \{\tau_1, \tau_2, \ldots, \tau_{N_\mathcal{D}}\}$, in which $N_\mathcal{D}$ is the buffer size.

Algorithm 1: DQN - Deep Q-Networks

input: batch size k, learning rate α, training frequency tf, total iterations number T, ϵ-Greedy probability ϵ, discount factor γ, replay memory capacity n,update target frequency utf

Initialize replay memory $\mathcal{D} \leftarrow \emptyset$, $t \leftarrow 0$;
Initialize Θ at random; Initialize Θ'=Θ;
Initialize $env \leftarrow EnvironmentInstance$; $s_t \leftarrow env.InitialState()$;
for $t \leftarrow 1$ **to** T **do**
    $a_t \leftarrow \epsilon$-Greedy(s_t, ϵ, Θ);
    $s_{t+1}, r_t \leftarrow env.step(s_t, a_t)$;
    $StoreTransition(\mathcal{D}, (s_t, a_t, r_t, s_{t+1}))$;
    if $t \ mod \ tf \ == \ 0$ **then**
        for $k \leftarrow 1$ **to** K **do**
            $\tau = (s_t, a_t, r_t, s_{t+1}) \sim UniformSample(\mathcal{D})$;
            $Q_{target} \leftarrow r_t + \gamma \times max_{a'} Q(s_{t+1}, a', \Theta')$;
            $Q_{value} \leftarrow Q(s_t, a_t, \Theta)$;
            $\mathcal{L}(\Theta) \leftarrow (Q_{target} - Q_{value})^2$;
            $\Delta \leftarrow \Delta + \mathcal{L}(\Theta) \times \bigtriangledown_{\Theta} Q_{value}$;
        $\Theta \leftarrow \Theta + \alpha \times \Delta$;
        if $t \ mod \ utf \ == \ 0$ **then**
            $\Theta' \leftarrow \Theta$;
        $\Delta \leftarrow 0$;
    $s_t \leftarrow s_{t+1}$;

Therefore, a CNN can be trained on samples $(s_t, a_t, r_t, s_{t+1}) \sim U(\mathcal{D})$, drawn uniformly at random from the pool of experiences by iteratively minimizing the following loss function,

$$\mathcal{L}_{DQN}(\Theta_i) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim U(\mathcal{D})} \left[\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a', \Theta') - Q(s_t, a_t, \Theta_i) \right)^2 \right], \quad (20)$$

in which Θ_i are the parameters from the i-th iteration. Instead of using the same network, another one provides the target values $Q(s_{t+1}, a', \Theta')$ used to calculate the TD-error, decoupling any feedback that may result from using the same network to generate its own targets.

Algorithm 2 addresses the trade-off between exploration and exploitation through an ϵ-*Greedy* strategy which selects an action from a homogeneous distribution over the set of possible actions with a given probability (exploration); otherwise, it uses the CNN that approximates the $Q(s, a)$ function to select the action that maximizes the Q-value estimation (exploitation). Generally, the value of the hyperparameter ϵ decreases over time, causing the agent to explore a lot at first but gradually transiting to use more and more of the acquired knowledge.

One can note a bias in DQN, such as in Q-Learning. According to Van Hasselt et al (2016), it can lead to overestimated high action values. Still, these

Algorithm 2: ϵ-Greedy

input : state s, ϵ-Greedy probability ϵ, CNN parameters Θ
output: action a

if $randonNumber > \epsilon$ **then**
    qValues $\leftarrow CNN.Forward(s, \Theta)$;
    a $\leftarrow$ argmax(qValues);
else
    i $\leftarrow$ random($\#actions$);
    a $\leftarrow$ actions[i]
return a;

values would not be (necessarily) a problem if all action values were uniformly higher, which probably does not occur. However, it is more likely that overestimation is common during learning, mainly when the action values are inaccurate. This way, the real problem is if that overestimation is not uniform and rises more often from state-action pairs that lead to suboptimal policies. The authors showed the occurrence of overestimations in DQN and proposed the DDQN based on Double Q-Learning. Since Double Q-Learning learns two value functions using two different sets of weights Θ and Θ', it's possible to compare Q-Learning to Double Q-Learning, rewriting its target value to untangle action selection and evaluation – Equations 21 and 22.

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a, \Theta_t); \Theta_t) \tag{21}$$

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a, \Theta_t); \Theta'_t) \tag{22}$$

The authors demonstrated that DDQN (see Algorithm 3) reduced the bias using the formulation of Double Q-Learning by decomposing the max operation in the target function into action selection and action evaluation, which improved the action-value functions updates making it more stable by diminishing the overestimation of the Q-values. From that, the target value changes from Equation 23 to Equation 24. The update of the target network is performed in the same way as in the DQN, periodically copying the updated weights from the online network, which approximates the evaluation function.

$$Y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a', \Theta') \tag{23}$$

$$Y_t = r_t + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1}, a, \Theta), \Theta') \tag{24}$$

According to Schaul et al (2016), an agent can learn more efficiently from some experiences than others, and some experiences may become more relevant as the agent approximate an optimal policy. Moreover, the experience replay from a uniform sampling does not consider the relevance of the experiences for the agent learning, usually repeating them at the same frequency

Algorithm 3: DDQN - Double Deep Q-Networks

input: batch size k, learning rate α, training frequency tf, total iterations number T, ϵ-Greedy probability ϵ, discount factor γ, replay memory capacity n, update target frequency utf

Initialize replay memory $\mathcal{D} \leftarrow \emptyset$, $t \leftarrow 0$;
Initialize Θ at random; Initialize Θ'=Θ;
Initialize $env \leftarrow EnvironmentInstance$; $s_t \leftarrow env.InitialState()$;
for $t \leftarrow 1$ **to** T **do**
    $a_t \leftarrow \epsilon\text{-Greedy}(s_t, \epsilon, \Theta)$;
    $s_{t+1}, r_t \leftarrow env.step(s_t, a_t)$;
    $StoreTransition(\mathcal{D}, (s_t, a_t, r_t, s_{t+1}))$;
    if $t \ mod \ tf == 0$ **then**
        for $k \leftarrow 1$ **to** K **do**
            $\tau = (s_t, a_t, r_t, s_{t+1}) \sim UniformSampling(\mathcal{D})$;
            $Q_{target} \leftarrow r_t + \gamma \times Q(s_{t+1}, argmax_a Q(s_{t+1}, a, \Theta_t); \Theta'_t)$;
            $Q_{value} \leftarrow Q(s_t, a_t, \Theta)$;
            $\mathcal{L}(\Theta) \leftarrow r_t + \gamma \times Q_{\text{target}} - Q_{\text{value}}$;
            $\Delta \leftarrow \Delta + \mathcal{L}(\Theta) \times \bigtriangledown_\Theta Q_{\text{value}}$;
        $\Theta \leftarrow \Theta + \alpha \times \Delta$;
        if $t \ mod \ utf == 0$ **then**
            $\Theta' \leftarrow \Theta$;
        $\Delta \leftarrow 0$;
    $s_t \leftarrow s_{t+1}$;

they were experienced. Given this, the authors investigated the effects of prioritization of experiences with studies in a proper environment (which presents exploration challenges with rare rewards) resulting in the proposition of the method Prioritized Experience Replay (PER). The authors initially investigated the effects of prioritizing experiences in reducing the number of updating steps a Q-Learning agent needs to learn the Q-function, comparing the results using: (i) a uniform sample; (ii) an oracle that achieves the best results; and (iii) a greedy sampling strategy. This greedy strategy stores the last TD-error value along with each transition in the replay memory and replays the ones with the highest absolute value of TD-error to update the Q-function. They verified that it reduced the number of updating steps compared to the uniform sampling but presented several issues. Because it only updates the TD-error of the replayed transitions, it may not replay those transitions initially associated with low TD-error values for a long time or until they are discarded due to a replay buffer with a size constraint. Moreover, often replaying experiences with high and slowly-decreasing TD-error cause a loss of diversity and may lead the model to overfit, besides being sensitive to noise spikes (as when the rewards are stochastic). Therefore they proposed a stochastic sampling method that interpolates between greedy prioritization and uniform random sampling by defining the sampling probability based on a transition's priority

value. According to Equation 25, the probability of sampling a transition j is

$$P(j) = \frac{p_j^\alpha}{\sum_i p_i^\alpha}, \tag{25}$$

where $p_j > 0$ is the priority of transition j and α defines how much of this value to use (i.e., $\alpha = 0$ to uniform sampling). Nevertheless, experiences' prioritizing introduces bias because it changes the probability distribution on which stochastic updates depend. Therefore, the authors proposed an approach to correct the bias called importance-sampling through a weight w_j (applied in the Q-function update) given by Equation 26,

$$w_j = \left(\frac{1}{N} \times \frac{1}{P(j)}\right)^\beta \tag{26}$$

in which N represents the size of the replay buffer, and that compensates for the non-uniform probabilities $P(i)$ if $\beta = 1$. Based on the hypothesis that it is possible to ignore small values of bias since it impacts more as convergence approaches, the authors exploit the flexibility of annealing the amount of importance-sampling correction over time by (linearly from an initial value) making $\beta = 1$ only at the final of training. Finally, Schaul et al (2016) combined prioritizing replay, stochastic sampling with priority values, and importance sampling to define the method PER (see Algorithm 4). They replaced the uniform sample in DDQN, achieving new state-of-the-art results in learning to play Atari 2600 games in ALE.

Some research works sought to investigate video frames as sequences in the input of the neural network that approximates the value action function in DQN-based methods to improve the perception of movements, speed up the trial and error process, and deal with problems in which agents only observe a reward signal after long sequences of decision-making. For Hausknecht and Stone (2015), mapping states to actions based only on the four previous game states (stacking the frames in a preprocessing step) prevents the DQN agent from achieving the best performance in games that require remembering far-away events from a large number of frames because in those games the future states and rewards depend on several previous states. Therefore, the authors proposed using a long short-term memory (LSTM) in place of the first fully connected layer, just after the series of convolutional layers in the original neural network architecture of DQN, to use the little history better. The authors demonstrate a trade-off between using a non-recurrent network with a long history of observations or a recurrent network with just one frame at each iteration step. They stated that a recurrent network is a viable approach for dealing with observations from multiple states, but it presents no systematic benefits compared to stacking these observations in the input layer of a plain CNN. Moreno-Vera (2019) proposed a similar approach but using DDQN instead of DQN.

Algorithm 4: DDQN with PER using proportional priorization

input: batch size k, learning rate η, discount factor γ, replay memory size N, exponents α and β, total iterations number T, ϵ-Greedy probability ϵ

Initialize replay memory $\mathcal{D} \leftarrow \emptyset$, $t \leftarrow 0$, priority $p_1 \leftarrow 1$;
Initialize Θ at random; Initialize Θ'=Θ;
Initialize $env \leftarrow EnvironmentInstance$; $s_t \leftarrow env.InitialState()$;
for $t \leftarrow 1$ **to** T **do**
  $a_t \leftarrow \epsilon\text{-Greedy}(s_t, \epsilon, \Theta)$;
  $s_{t+1}, r_t \leftarrow env.step(s_t, a_t)$;
  $p_t \leftarrow \max_{i<t} p_i$;
  $StoreTransition(\mathcal{D}, (s_t, a_t, r_t, s_{t+1}), p_t)$;
  if $t \ mod \ tf \ == 0$ **then**
    for $j \leftarrow 1$ **to** k **do**
      $P(j) = \frac{p_j^\alpha}{\sum_i p_i^\alpha}$;
      $\tau = (s_t, a_t, r_t, s_{t+1}) \sim PrioritizedSampling(\mathcal{D}, P(j))$;
      $w_j \leftarrow \frac{N \cdot P(j)^{-\beta}}{max_i w_i}$;
      $Q_{target} \leftarrow Q(s_{t+1}, argmax_a Q(s_{t+1}, a, \Theta), \Theta')$;
      $Q_{value} \leftarrow Q(s_t, a_t, \Theta)$;
      $\mathcal{L}(\Theta) \leftarrow r_t + \gamma \times Q_{target} - Q_{value}$;
      $p_j \leftarrow |\mathcal{L}(\Theta)|$;
      $\Delta \leftarrow \Delta + w_j \times \mathcal{L}(\Theta) \times \bigtriangledown_\Theta Q_{value}$;
    $\Theta \leftarrow \Theta + \eta \times \Delta$;
    if $t \ mod \ utf \ == 0$ **then**
      $\Theta' \leftarrow \Theta$;
    $\Delta \leftarrow 0$;
  $s_t \leftarrow s_{t+1}$;

Wang et al (2016) proposed to use a new deep neural network architecture, called Duelling Networks, in place of classical (well-known) architectures such as CNNs, LSTMs, or MLPs to improve model-free reinforcement learning methods. Their architecture uses two estimators in the same network, one for the state-value function $V(s;\theta,\beta)$ and another for the so-called state-dependent action advantage function $A(s,a;\theta,\alpha)$, defined by two streams of fully connected layers (following the convolutional layers) whose outputs are the (scalar) state-value and a vector of advantages for each action. Equation 27 combines these two outputs and produces the final Q-value estimations, in which θ is the parameters of the convolutional layers, while α and β are the parameters of the two streams of fully-connected layers.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A\left(s,a';\theta,\alpha\right) \right) \tag{27}$$

As the output is also Q-value estimates for each action in the input states, the dueling networks architecture can replace the original neural networks in

other algorithms such as DQN and DDQN, with adaptation only regarding the back-propagation. The authors demonstrated improved experimental results using uniform and prioritized experience replay.

One of the main contributions of Experience Replay (ER) is to reduce non-stationarity and decorrelate the agent's updates, contributing to the stabilization when using deep neural networks to approximate the value functions. Although, the way it stores and samples the agent's experiences using the experience replay memory limits its use to off-policy reinforcement learning algorithms. In place of ER, Mnih et al (2016) proposed using asynchronous gradient descent to optimize deep neural networks and train multiple agents in parallel on multiple environment instances. According to the authors, this parallelism also decorrelates the agents' data because, at each time step, the parallel agents will probably be experiencing various and different states and can explicitly use different exploration policies to maximize their diversity. Moreover, by running different exploration policies in multiple threads, the overall updating changes by multiple actor-learners applying online updates in parallel are likely to be less correlated in time than a single online agent, fulfilling the role of stabilizing undertaken by ER. The authors demonstrated that their approach could be used in both off-policy and on-policy algorithms by presenting multi-threaded asynchronous variants of the methods Q-learning, Sarsa, and Advantage Actor-Critic. Their best-evaluated algorithm called the Asynchronous Advantage Actor-Critic (A3C), surpassed the state-of-the-art (at its publication time) on the Atari 2600 domain in ALE and reduced the training time because that is roughly linear in the number of parallel actor-learners. The authors also evaluated the A3C on the domain of the physics simulator MuJoCo (Todorov et al, 2012).

Bellemare et al (2017) discuss a distributed perspective on the return value in contrast to modeling its expectation and propose an algorithm that applies the Bellman equation to learn from approximate value distributions. Considering that the value function Q estimates the random value return (resulting from the probabilities of the transitions), the authors describe its distributional nature like in Equation 28, where Z is the value distribution, and R (the reward function) is explicitly a random variable. A stationary policy π maps each estate to a probability distribution over the action space.

$$Z(s,a) \stackrel{D}{=} R(s,a) + \gamma Z\left(S', A'\right) \tag{28}$$

The authors define the value Z^π as the sum of discounted rewards along the agent's interaction with the environment, the value functions as vectors in $\mathbb{R}^{S\times A}$, and consider the expected reward function as one such vector. Therefore, they define a Bellman operator τ^π and an optimality operator τ like in Equations 29 and 30, where P is the transition function (as defined in Section 2). From that, instead of the expectation, they consider the full distribution of the variable Z^π, which is what they call *value distribution*. The paper discusses all the theoretical behavior of the distributional analogs of the

Bellman operator in the control setting.

$$\tau^{\pi} Q(s,a) := \mathbb{E}R(s,a) + \gamma \mathbb{E}_{P,\pi} Q(s',a') \tag{29}$$

$$\tau Q(s,a) := \mathbb{E}R(s,a) + \gamma \mathbb{E}_{P} \max_{a'} Q(s',a') \tag{30}$$

The authors present their state-of-the-art results of modeling and applying the distributional value within a DQN agent evaluated on ALE (Bellemare et al, 2013) and demonstrate considerable improvements in the agent performance.

Fortunato et al (2018) approach the relevant problem of exploration and propose a method called NoisyNet to learn neural networks' weight perturbations and use this to drive exploration. The authors based this on the idea that a single change to the weight vector can induce consistent and very complex state-dependent changes in the policy over multiple time steps instead of adding decor-related and state-independent noise, like in ϵ-*greedy*. According to the authors, most exploration heuristics, such as ϵ-greedy and entropy regularization, may not produce large-scale behavioral patterns necessary for efficient exploration in many environments. The methods of *optimism in the face of the uncertain* are often limited to small states-action spaces or linear function approximations. The intrinsic motivation methods augment the environment's reward signal with an additional term to reward novel discoveries, and many research works have proposed different forms of such terms. These methods separate the generalization processes from the exploration, using elements like metrics for intrinsic rewards and importance values, weighted relative to the environment reward, directly defined by the researcher and not learned for the agent's interaction with the environment. Some evolutionary or black-box methods explore the policy space but require many prolonged interactions and usually are not data efficient, requiring a simulator to allow many policy evaluations. The NoiseNet is a neural network that uses the gradient from the agent loss function (by gradient descent) to learn a parameter (alongside the other parameters of the agent) that defines the variance of the agent's neural network weights perturbations sampled from a noise distribution (what the authors called the energy of the injected noise). Therefore, the algorithm injects noise into the parameters and tunes its intensity automatically, defining what the author called the energy of the injected noise. The authors evaluated NoiseNet versions of DQN, Duelling Networks, and A3C (Mnih et al, 2016) on 57 Atari 2600 games in the Arcade Learning Environment (Machado et al, 2018), and their results demonstrated that their agents achieved superhuman performance.

Many relevant improvements in DQN-based methods approach different aspects. While DDQN addresses the overestimation bias of Q-learning and (as a consequence) of DQN, PER improves data efficiency in experiences replaying. The Dueling Network improved the generalization across actions by separately representing state values and action advantages. A3C shifts the bias-variance trade-off by learning from multi-step bootstrap targets and

helps to propagate newly observed rewards faster to earlier visited states. Distributional Q-learning learns a categorical distribution of discounted returns instead of estimating the mean. Noisy DQN uses stochastic network layers for exploration. Given this, Hessel et al (2018) investigated how to combine these different but complementary ideas, using ER, into an ensemble approach called Rainbow, which achieved state-of-the-art on 57 Atari 2600 games in ALE (Bellemare et al, 2013) regarding data efficiency and final performance.

The authors adapted the strategies of PER to use the KL loss of the Distributional Q-Learning, replaced the one-step distributional loss with a multi-step variant, and defined the target distribution by contracting the value distribution in S_{t+n} and shifting it by the truncated n-step discounted return. They combined the multi-step distributional loss with Double Q-Learning and used the greedy strategy to select and evaluate the action in S_{t+n} using the online and the target networks. They also adapted the dueling network architecture for use with return distributions so that the output of a shared state representation layer is fed into a value stream and an advanced stream designed to output distributional values that are combined as in Dueling Networks and then passed through a softmax layer to obtain the normalized parametric distributions used to estimate the returns' distributions. Finally, they replaced all linear layers with equivalent noisy layers and used factorized Gaussian noise (Fortunato et al, 2018) to reduce the number of independent noise variables. An open-source variation of Rainbow is available in the framework for RL agents development called Dompamine (Castro et al, 2018), which differs from the original Rainbow (Hessel et al, 2018) by not including DDQN, dueling heads or noisy networks. It uses the *n-step returns*, which is identified by Fedus et al (2020) as a critical element to improve the agent performance when using a larger replay buffer (i.e., 10 million experiences instead of the classical 1 million limited size). The *n-step returns* updates the Q-value function from an n-step target value rather than one-step so that the target side of Q-learning would be changed from Equation 31 to Equation 32. The authors interpret it as an interpolation between estimating the targets in Monte Carlo (MC) methods as $\sum_{k=0}^{T} \gamma^k r_{t+k}$ (one can find a discussion in Sutton and Barto (2018)) and single-step TD-learning, balancing the low bias but high variance of MC targets and the low variance but high bias of TD(0) (see Section 2).

$$r + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \tag{31}$$

$$\sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \max_a Q\left(s_{t+n}, a\right) \tag{32}$$

Kaiser et al (2019) presented a model-based deep reinforcement learning algorithm with a video prediction model named SimPLe, which performed well after just 102,400 interactions (that correspond to 409,600 frames on ALE and about 800,000 samples from the video prediction model) and compared their results with the ones obtained by Rainbow (Hessel et al, 2018). They

aimed to show that planning with a parametric model allows for data-efficient learning on several Atari video games. In that sense, van Hasselt et al (2019) proposed a broad discussion about model-based algorithms and experience replay, pointing out its commonalities and differences, when to expect benefits from either approach and how to interpret prior works in this context. They set up experiments in a way comparable to Kaiser et al (2019) and demonstrated that in a like-for-like comparison, Rainbow outperformed the scores of the model-based agent with less experience and computation. Rainbow used a total number of 3.2 million replayed samples, and SimPLe used 15.2 million. Łukasz Kaiser et al (2020) presented their final published paper comparing SimPLe and Rainbow on the number of iterations needed to achieve the best results. SimPLe achieved the best game scores on half of the game set. Although, the authors states that one of the SimPLe limitations is that its final scores are on the whole lower than the best state-of-the-art model-free methods.

The DQN-based methods, such as the DDQN and Duelling Networks, either with original ER or PER, achieved state-of-the-art (at the time of their respective publications) in learning to act directly from high dimensional states, interacting with non-deterministic environments in a stochastic way, and approximating the optimal policy from discrete-valued and low dimensional action spaces. However, many interesting problems, such as physical control tasks, have continuous (real-valued) and high-dimensional action spaces ($\mathcal{A} = \mathbb{R}^{\mathcal{N}}$). In these cases, for the DQN to find the action that maximizes the Q-value estimate, an iterative optimization process would be necessary at each step of the agent. Therefore, Lillicrap et al (216) have based on the Deterministic Policy Gradient (DPG) algorithm (Silver et al, 2014) and the DQN to propose an actor-critic and model-free algorithm called Deep Deterministic Policy Gradient (DDPG) that use deep neural networks with ER and can learn over continuous action spaces. According to the authors, DDPG can find policies whose performance is competitive (sometimes better) with those found by a planning algorithm with full access to the dynamics of challenging physical control problems that involve complex multi-joint movements, cartesian coordinates, unstable and rich contact dynamics, and gait behavior. They have evaluated their agent in learning action policies from video-frame pixels and physical control data (such as joint angles), using the same hyperparameters and network architecture across different challenges on simulated physical environments using a physics engine originally proposed for model-based control called MuJoCo (Todorov et al, 2012).

From the derivations of the Bellman's equation presented in Section 2 to define the action-value function in Equation 8, one can note how the target policy could be described as a function $\mu : \mathcal{S} \rightarrow \mathcal{A}$ if this policy is deterministic, to replace the expectation in the target, as described by Lillicrap et al (216), changing from the Equation 33 to Equation 34.

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \tag{33}$$

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1})] \tag{34}$$

As in $\mathcal{MDP}$s, the discounted sum of future rewards R depends on the policy π; the authors denote its distribution over the visited states as ρ^{π}. Therefore, it is possible to learn the function Q^{μ} off-policy using transitions obtained from a different stochastic behavior policy they referenced as β and a distribution ρ^{β}. This way, a Q-value function approximator parameterized by Θ^{Q} could be optimized by minimizing the loss obtained in Equation 35.

$$L(\Theta^{Q}) = \mathbb{E}_{s_t \sim \rho^{\beta}, a_t \sim \beta}[(Q(s_t, a_t|\Theta^{Q}) - r_t + \gamma Q(s_{t+1}, \mu(s_{t+1})|\Theta^{Q})^2] \tag{35}$$

The DPG algorithm applies a parameterized function $\mu(s|\theta^{\mu})$ (the actor) to define the current policy by deterministically mapping states to actions and updating its parameter using the *policy gradient* (i.e, the gradient of the policy's performance) (Silver et al, 2014). This updating technique applies the chain rule to the expected return from the start distribution J concerning the actor parameters, as in Equations 36 and 37. In turn, it learns the Q-value function $Q(s, a)$ (the critic) using the Bellman equation as in Q-Learning (Lillicrap et al, 216).

$$\nabla_{\theta_{\mu}} J \approx \mathbb{E}_{s_t \sim \rho^{\beta}}[\nabla_{\theta_{\mu}} Q(s, a|\theta^{Q})|s = s_t, a = \mu(s_t|\theta^{\mu})] \tag{36}$$

$$= \mathbb{E}_{s_t \sim \rho^{\beta}}[\nabla_{a} Q(s, a|\theta^{Q})|s = s_t, a = \mu(s_t) \nabla_{\theta_{\mu}} \mu(s|\theta^{\mu})|s = s_t] \tag{37}$$

DDPG (see Algorithm 5) applies modifications to DPG to use the contribution of DQN in approximating the Q-value function from high-dimensional states space and use the policy gradient to deal with high-dimensional and continuous action spaces. It also uses a replay buffer with uniform sampling. One change is in the target function updating. Instead of copying the weights directly from the updating to the target neural network (such as in DQN), the authors create a copy of the critic and actor networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$, and use them for estimating target values, then updating these target networks by slowly (for stability) tacking the updating neural networks making $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$, with $\tau \ll 1$. This way, the authors obtained stable targets y_i to consistently train the critic. To deal with learning from low-dimensional physical feature vector observations, whose components may have different units and scales, such as position and velocity, the authors applied the batch normalization technique (Ioffe and Szegedy, 2015) on the state input and all layers of the μ network and the layers of the Q network prior its action input layer. Because it is off-policy, DDPG can deal with exploration (a difficult problem in continuous action spaces) independently from the learning algorithm. While DQN uses an ϵ-greedy approach (see Algorithm 2), Lillicrap et al (216) propose to use an Ornstein-Uhlenbeck (Uhlenbeck and Ornstein, 1930) to create a process $\mathcal{N}$ to add noise to the actor policy and generate temporally correlated exploration, where $\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + \mathcal{N}$.

According to Schaul et al (2016), using a replay buffer presents two main challenges: (i) the selection of which experiences to store; and (ii) the picking of which ones to repeat. They addressed the second when they proposed the PER,

Algorithm 5: DDPG - Deep Deterministic Policy Gradient

input: batch size k, discount factor γ, learning rate α replay memory size N, total iterations number T, total episodes M

Initialize replay memory $\mathcal{D} \leftarrow \emptyset$;
Initialize Θ^{Q}, Θ^{μ} at random; $\Theta^{Q'} \leftarrow \Theta^{Q}$, $\Theta^{\mu'} \leftarrow \Theta^{\mu}$;
Initialize $env \leftarrow EnvironmentInstance$;;
for $m \leftarrow 1$ **to** M **do**
  $\mathcal{N} \leftarrow RandomNoiseProcess()$;
  $s_t \leftarrow env.InitialState()$;
  for $t \leftarrow 1$ **to** T **do**
    $a_t \leftarrow \mu(s_t|\Theta^{\mu}) + \mathcal{N}_t$;
    $s_{t+1}, r_t \leftarrow env.step(s_t, a_t)$;
    $StoreTransition(\mathcal{D}, (s_t, a_t, r_t, s_{t+1}))$;
    $L \leftarrow 0$;
    $P \leftarrow 0$;
    for $j \leftarrow 1$ **to** k **do**
      $\tau = (s_t, a_t, r_t, s_{t+1}) \sim UniformSample(\mathcal{D})$;
      $Q_{target} \leftarrow Q'(s_{t+1}, \mu'(s_{t+1}|\Theta^{\mu'}), \Theta^{Q'})$;
      $Q_{value} \leftarrow Q(s_t, a_t, \Theta^{Q})$;
      $\mathcal{L}(\Theta^{Q}) \leftarrow (r_t + \gamma \times Q_{target} - Q_{value})^2$;
      $L \leftarrow L + \mathcal{L}(\Theta^{Q})$;
      $P \leftarrow P + \nabla_a Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|s = s_t$;
      $\Delta \leftarrow \frac{1}{k} L \times \bigtriangledown_{\Theta^{Q}}$;
    $\Theta^{Q} \leftarrow \Theta^{Q} + \alpha \times \Delta$;
    $\Theta^{\mu} \leftarrow \Theta^{\mu} + \alpha \times \nabla_{\Theta^{\mu}} J \approx \frac{1}{k} P$;
    $\Theta^{Q'} \leftarrow \tau\theta^{Q} + 1(1-\tau)\Theta^{Q'}$;
    $\Theta^{\mu'} \leftarrow \tau\theta^{\mu} + 1(1-\tau)\Theta^{\mu'}$;
    $s_t \leftarrow s_{t+1}$;

assuming that the content of the memory was beyond their control. Novati and Koumoutsakos (2019), Zha et al (2019), and Sun et al (2020) also dealt with the second case seeking to make ER more optimized and data-efficient based on how to sample transitions to improve the current learning policy. In turn, Neves et al (2022) originally approached the first case, investigating how to store transition in a transitions memory, improving data efficiency, but mainly seeking to exploit rare and expensive experiences. For Novati and Koumoutsakos (2019), the accuracy of updates may deteriorate when the policy diverges from past behaviors and can undermine the performance of ER. In place of tuning hyper-parameters to slow down policy changes, they proposed to actively enforce the similarity between the current policy transitions π and past experiences μ used to compute updates, using an approach called Remember and Forget Experience Replay (ReF-ER). This skips gradients computed from experiences that are too unlikely with the current policy transitions

and regulates policy changes within a trust region of the replayed behaviors. Their main objective is to control the similarity between π and μ, classifying experiences either as "near-policy" or "far-policy" based on a ratio ρ of probabilities of selecting the associated action with π and that with μ. Therefore, ReF-ER limits the fraction of far-policy samples in the replay memory and computes gradient estimates only from near-policy experiences. The authors demonstrated that one could apply their approach to any off-policy method with parameterized (i.e., by using DNN) policies and that it allows better stability and agent performance (compared to uniforming sampling) in the main class of methods for continuous actions spaces based on DPG (i.e., DDPG), Q-learning (i.e., NAF in Gu et al (2016)), and off-policy Policy Gradients (off-PG) (Degris et al, 2012).

Zha et al (2019) proposed an Experience Replay Optimization (ERO) framework, which aims to optimize the replay strategy by learning a replay policy (instead of applying a heuristic or rule-based strategy) whose main challenge is dealing with continuous, noised and unstable (regarding the rewards) updating of a large replay memory (usually in the tens of millions of transitions). Its objective is learning to sample experiences that could maximize the expected cumulative reward. While the agent learns a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, ERO learns a policy $\phi : \mathcal{D} \rightarrow \mathcal{B}_{\text{i}}$, where $\mathcal{D}$ is the replay buffer, and $\mathcal{B}_{\text{i}}$ is a batch of transitions sampled from $\mathcal{D}$ at a time step i. ϕ outputs a boolean vector to guide the subset sampling, indirectly teaching the agent by defining what subset it should use to update its value functions. Then, ERO adjusts ϕ according to the return from the environment as a measure of the agent's *performance improvement*. The authors evaluated their approach by applying it to train a DDPG agent on eight continuous control tasks from the OpenAI Gym environment and concluded that it is promising because it could find more "usable" experiences for off-policy agents using ER in different tasks.

Sun et al (2020) proposed the Attentive Experience Replay (AER) to prioritize transitions (at sampling from the replay buffer) containing states more frequently observed by current policy, based on the idea that some states in past experiences may become rarely revisited once the policy is continually updated and may not contribute to or even harm the overall performance of the current policy. The authors consider the similarity between the past transitions' states and the current transition's states as a measurement of frequency and prioritization criteria. To the authors, some experiences in the replay buffer might become irrelevant to the current policy, and others may contain states that the current policy would never visit. Another supposition from Sun et al (2020) is that some transitions from the past might contain states that would never be visited by current policy, and optimization over these states might not improve the overall performance of current policy and can undermine the performances of the frequently visited states. The authors evaluated the results of applying AER in the off-policy algorithms DQN, DDPG, Soft Actor-Critic (SAC) (Haarnoja et al, 2018), and Twin Delayed Deep Deterministic Policy Gradient (TD3) (Fujimoto et al, 2018), and compared with uniform sampling

and PER. They used the ecosystem of tasks from OpenAI Gym (Brockman et al, 2016).

Neves et al (2022) propose a method named *COMPact Experience Replay* (COMPER) to improve the model of experience memory and make ER feasible (and more efficient) using smaller amounts of data. The authors demonstrated that it is possible to produce sets of similar transitions and explore them to build a reduced transitions memory, performing successive updates of their Q-values and learning their dynamics through a Long-short Term Memory (LSTM) network. They also used this same LSTM network to approximate the target value at the TD-learning. According to the authors, this augments the odds of a rare transition being observed compared to a sampling performed on a large replay buffer and makes the updates of the value function more effective. The authors presented a complete analysis of the memories' behavior, along with detailed results for 100,000 frames and about 25,000 iterations with a small experience memory on eight challenging 2600 Atari games in the Arcade Learning Environment (ALE), demonstrating that COMPER can approximate a good policy from a small number of frame observations using a compact memory and learning the similar transitions' sets dynamics using a recurrent neural network.

COMPER (see Algorithm 6) uses ER and TD-learning to update the Q-value function $Q(s, a)$. However, it does not construct just a replay buffer. Instead, COMPER samples transitions from a much more compact structure named *Reduced Transition Memory* ($\mathcal{RTM}$). To achieve that, COMPER first stores the transitions together with estimated Q-values into a structure named *Transition Memory* ($\mathcal{TM}$), which is similar to a traditional replay buffer, except for the presence of the Q-value and the identification and indexing of *Similar Transitions Sets* ($\mathcal{ST}$). After that, the similarities between transitions stored in $\mathcal{TM}$ can be explored to generate a more compact version of it – the $\mathcal{RTM}$. Then, transitions $(s_t, a_t, r_t, s_{t+1}) \sim U(\mathcal{RTM})$ are drawn uniformly from $\mathcal{RTM}$ and used to minimize the following loss function,

$$\mathcal{L}_{COMPER}(\Theta_i) = \mathbb{E}_{\tau_t=(s_t,a_t,r_t,s_{t+1})\sim U(\mathcal{RTM})}\left[\left(r_t + \gamma\, QT(\tau_t, \Omega) - Q(s_t, a_t, \Theta_i)\right)^2\right], \quad (38)$$

in which $Q(s_t, a_t, \Theta_i)$ is a Q-function approximated by a CNN parameterized by Θ_i at i-th iteration. $QT(\tau_t, \Omega)$ is a Q-target function approximated by an LSMT and parameterized by Ω. This function provides the target value and is updated in a supervised way from the $\mathcal{ST}$s stored in $\mathcal{TM}$. Thus, this LSTM is also used to build a model to generate the compact structure of $\mathcal{RTM}$ from $\mathcal{TM}$ while seeking to learn the dynamics of $\mathcal{ST}$s to provide better target values at the next agent update step.

At each training time-step t, the authors define a transition by a tuple $\tau_t = (s_t, a_t, r_t, s_{t+1})$. Two transitions $\tau_{t_1} = (s_{t_1}, a_{t_1}, r_{t_1}, s_{t_1+1})$ and $\tau_{t_2} = (s_{t_2}, a_{t_2}, r_{t_2}, s_{t_2+1}), t_1 \neq t_2$ are similar ($\tau_{t_1} \approx \tau_{t_2}$) when the distance (e.g., Euclidean distance) between τ_{t_1} and τ_{t_2} is lesser than a threshold, i.e., $\mathcal{D}(\tau_{t_1}, \tau_{t_2}) \leq \delta$, in which δ is a distance (or similarity) threshold value. The N transitions that occurred up to a time instant are stored in $\mathcal{TM}$ and can be identified as subsets of similar transitions $\mathcal{ST}$ when the similarity condition

Algorithm 6: COMPER

input: batch size k, learning rate α, training frequency tf, number of states to be observed sn, ϵ-Greedy probability ϵ, update target frequency utf, discount factor γ, similarity threshold value δ

Initialize $TM \leftarrow \emptyset$, $RTM \leftarrow \emptyset$, $t \leftarrow 0$, $\Delta \leftarrow 0$; Initialize Θ, Ω;
Initialize $env \leftarrow EnvironmentInstance$, $run \leftarrow True$, $countframes \leftarrow 0$;
$s_t \leftarrow env.InitialState()$;
a_t, $Q_t \leftarrow \epsilon$-Greedy$\big(s_t, \epsilon, \Theta\big)$;
while *run* **do**
    $s_{t+1}, r_t \leftarrow step(s_t, a_t)$;
    $\mathcal{TM}.StoreTransition([\tau(s_t, a_t, r_t, s_{t+1}), Q_t], \delta)$;
    $s_t \leftarrow s_{t+1}$;
    $t \leftarrow t + 1$;
    if t *mod* tf $==$ *0* **then**
        if $\mathcal{RTM} == \emptyset$ *or* t *mod* utf $==$ *0* **then**
            $\Omega \leftarrow Train(QT, \mathcal{TM})$;
            $\mathcal{RTM} \leftarrow$ProduceRTM$(QT, \mathcal{TM}, \Omega)$;
        for $i \leftarrow 1$ **to** k **do**
            Sample uniformly $\tau' = (s_t, a_t, r_t, s_{t+1})$ from $\mathcal{RTM}$;
            $Q_{target} \leftarrow Forward(\tau', QT, \Omega)$;
            $Q_{\text{value}} \leftarrow Forward(s_t, Q, \Theta)[a_t]$;
            $\mathcal{L}(\Theta) \leftarrow r_t + \gamma \times Q_{\text{target}} - Q_{\text{value}}$;
            $\Delta \leftarrow \Delta + \mathcal{L}(\Theta) \times \bigtriangledown_\Theta Q_{\text{value}}$;
        $\Theta \leftarrow \Theta + \alpha \times \Delta$;
        $\Delta \leftarrow 0$;
    $countframes \leftarrow countframes + env.framesnumber$;
    if *env.ReachedFinalState()* **then**
        $run \leftarrow (countframes \leq sn)$;
        if *run* **then**
            $env.ResetStates()$;
            $s_t \leftarrow env.InitialState()$;
    a_t, $Q_t \leftarrow \epsilon$-Greedy$\big(s_t, \epsilon, \Theta\big)$;

is satisfied. Moreover, they are stored throughout subsequent agent training episodes and identified by a unique index. Therefore, the authors define $\mathcal{TM} = \big\{[T^i, \mathcal{ST}_i] \,|\, i = 1, 2, 3, \ldots, N_{ST}\big\}$, in which N_{ST} is the total number of distinct subsets of similar transitions, T^i is a unique numbered index and $\mathcal{ST}_i$ represents a set of similar transitions and their Q-values. Thus,

$$\mathcal{ST}_i = \big\{ \big[\tau_{i(1)}, Q_{i(k)}\big] \;|\; 1 \leq k \leq N^i_{ST} \big\} \tag{39}$$

in which N^i_{ST} represents the total number of similar transitions in the set $\mathcal{ST}_i$. Thus, $\tau_{i(1)}$ corresponds to some transition $\tau_{t_j}, j \in \{1, \ldots N^i_{ST}\}$ and is the representing transition of similar transitions set $\mathcal{ST}_i$ (e.g., the first one), and $Q_{i(k)}$ is the Q-value corresponding to some transition $\tau_{t_j}, j \in \{1, \ldots N^i_{ST}\}$ such that $\tau_{i(1)} \in \mathcal{ST}_i$ and $\tau_{i(1)} \approx \tau_{i(k)}, 1 \leq k \leq N^i_{ST}$. Therefore, $\mathcal{TM}$ can seem as a set of $\mathcal{ST}$s. A single representative transition for each $\mathcal{ST}$ can be generated together with the prediction of their next Q-value from an explicit model of $\mathcal{ST}$ using the LSTM. This way, from $\mathcal{TM}$, one can produce a $\mathcal{RTM}$ in which τ'_i is the transition that represents all the similar transitions so far

Algorithm 7: COMPER ϵ-Greedy

input : State s, ϵ-Greedy probability ϵ, CNN parameters Θ
output: action a, Q-value Q

```
qValues ← CNN.Forward(s, Θ);
if randonNumber > ε then
    a ← argmax(qValues);
    Q ← qValues[a];
else
    a ← random(#actions);
    Q ← qValues[a];
end
return a, Q;
```

identified in $\mathcal{ST}_i$, so that $\mathcal{RTM} = \{[\tau'_i] \,|\, i = 1, 2, 3, \ldots, N_{ST}\}$. Unlike $\mathcal{TM}$, $\mathcal{RTM}$ does not take care of sets of similar transitions since each τ'_i is unique and represents all the transitions in a given $\mathcal{ST}_i$. According to the authors, it gives the transitions stored in $\mathcal{RTM}$ the chance of having their Q-values re-estimated. Besides, sampling from $\mathcal{RTM}$ increases the chances of selecting rare and very informative transitions more frequently, at the same time, that helps increase diversity (because of variability in each sample).

One could observe in Algorithm 7 that COMPER slightly modifies the ϵ-Greedy algorithm to return the estimation of the Q-value together with the action that maximizes it.

5 Challenges and Trends in Experience Replay

Relevant research works have sought to understand the effects of replay buffer size (either small or large) on the performance of reinforcement learning agents that use ER. For example, since Mnih et al (2015), all the DQN-based methods use a fixed replay memory size of 1 million transitions, and variations in the buffer's size are still understudied in this class of methods (Liu and Zou, 2018). Zhang and Sutton (2017) presented an empirical study about ER, demonstrating that a large replay buffer can harm the agent performance and that its size is a very important hyperparameter neglected in the literature. They proposed a method to minimize the negative influence of a large replay buffer, named Combined Experience Replay (CER), consisting of adding the last transition to the sampled batch before using it in agent training. They hypothesized a trade-off between the data quality and the data correlation as smaller replay buffers make data fresher but highly temporal correlated. At the same time, neural networks often need independent and identically distributed (i.i.d.) data. Data sampling from larger replay buffers tends to be uncorrelated but outdated. A full replay buffer that adopts a FIFO strategy (i.e., as a queue) will impact agent learning. However, if we assume that different transitions

from many stochastic episodes carried out over a non-deterministic environment will be stored and sampled many times from a smaller replay buffer (but not explicitly size-limited), this buffer tends to be less correlated along with the time while the buffer size increases.

According to Fedus et al (2020), it is necessary to investigate the behavior of ER regarding the interrelated effect of variations in its hyperparameters since their effects may not be only individual but come from their joint variation. The authors studied the effect of the relationship between the replay buffer size, associated with a *replay capacity*, and the time that a transition (which represents a policy at some moment in the learning process) remains in memory, which they call the *age of a policy*. Thus, the replay capacity is associated with state-action coverage, while the age of a policy (represented by its respective transition in memory) is related to its distance from the current policy learned (represented by the most recent transitions). It is possible to explore this relationship through an element called *replay ratio*, which refers to the number of agent interaction steps with the environment for each updating step of the value function. For example, DQN (Mnih et al, 2015) performs an update step (from the memory of the transitions) for every 4 iteration steps, which means a replay ratio of 0.25. Thus, their primary objective was to understand how the agent behavior changes as the replay ratio varies. Objectively, the authors defined the age of the policy by the number of value function update steps performed since the storage of the transition (which represents the policy) in the buffer and the replay capacity by the total number of transitions stored. So, with a replay capacity of 1 million transitions and a replay ratio of 0.25, the oldest policy age is 250,000 function update steps. In this relationship, increasing the buffer size also increases the replay capacity and the (possible) age of the policy. In this way, the replay ratio remains constant. However, fixing the possible age of the oldest policy requires more transitions to be stored by the current policy to increase the replay capacity. In other words, it is necessary to decrease the number of function updates per number of interactions with the environment, which increase the number of agent interaction steps to store a more significant number of transitions from the current policy, which results in a decrease in replay ratio. On the other hand, keeping the replay capacity fixed (fixing the buffer size) decreases the age of the oldest policy but also requires more transitions, which will (also) reduce the replay ratio.

Still in Fedus et al (2020), the authors performed experiments using 14 Atari 2600 games in ALE using the Dopamine version of Rainbow (Castro et al, 2018), demonstrating that Rainbow performance consistently increases when the replay capacity increases and generally increases when the age of the oldest policy reduces. When one fixes the policy's age, the performance increases as the replay capacity is more significant and this trend remains regardless of the age value defined, although the magnitude of the increase in performance depends on this value. For the authors, this may occur because a larger state-action coverage capacity can reduce the chances of over-fitting

over a smaller subset of transitions. On the other hand, when fixing the replay capacity, the performance tends to improve when reducing the age of the oldest policy. Based on the idea that the age of a policy distances the current policy, they claimed that this result suggests that learning from policies (sampling transitions) closer to the current policy can increase performance because the agent thus explores transitions with a more significant potential of return (this hypothesis is also explored by (Sun et al, 2020)). An exception observed by the authors occurred when they fixed the replay capacity at 10 million and reduced the age of the oldest policy from 2.5 million to 250 thousand, which, according to them, can be explained by the decrease in the agent score in two games: Montezuma Revenge and PrivateEye, which consist of two environments with very sparse rewards and challenging to explore. In these environments, agents whose samples on definitely newer policies seem unable to find the rewards for depending on the observation of older policies. The authors also noted that increasing the buffer size while maintaining a fixed replay ratio leads to a performance improvement that may vary due to the interaction between the gain in replay capacity and the loss from accumulating older policies. Thus, as the policy age increases, the benefit of increasing the replay capacity generally decreases. One could see that the issue with Montezuma Revenge and PrivateEye games is related to the need to learn longer-term policies, which Neves et al (2022) also sought to address with the use of recurrence on sets of similar transitions and sampling from a smaller memory whose size is not explicitly limited.

Conducting experiments with DQN, the authors observed that its performance did not increase when increasing the replay capacity, either fixing the replay ratio or fixing the age of the policy, contrary to what they observed with the Rainbow version of Dopamine. This version is (basically) the DQN with the addition of PER, C51, and n-steps returns, plus the replacement of the RMSProp optimization method with ADAM (Kingma and Ba, 2015). Compared to the original Rainbow, that one does not include DDQN, dueling, or noisy networks. Therefore, the authors created four DQN variants in which each received the addition of only one of those components to investigate which interacted with the increase in replay capacity (from 1 million to 10 million transitions) to generate the performance gain observed in Rainbow. The results showed that the only independently added component that led to a considerable performance improvement with an increase in the replay capacity in the DQN was the n-steps returns and that this increase also occurs when fixing the policy age instead of fixing the replay ratio. By removing the n-steps from Rainbow, they verified that the agent did not benefit from the increase in replay capacity and that removing other components does not prevent performance gains. It suggests that n-steps returns are the only critical component for performance gain with increased replay capacity. They also observed that the use of n-steps returns when there are few transitions in the buffer worsens the performance of the DQN, suggesting that the performance gain due to its use only occurs when using larger transition buffers. Besides, they also noticed

that only adding PER to the DQN does not significantly increase the performance when the replay capacity is considerably large. One can see in Section 4 how it distributes the weighting obtained as a function of the magnitude of the temporal difference errors in the transition memory and how its possible loss of capacity to contribute to the increase in performance may be related to the buffer size. The authors also carried out experiments with two variations of the DQN using the n-steps returns and offline trained from a buffer of 200 million transitions (corresponding to the total frames used to evaluate agents in the literature) to verify if the performance gain persists while having bigger and bigger buffers, keeping the replay ratio fixed and making the buffer to have more old policies. Those transitions come from another agent and do not have their Q-value estimates updated during the training of the current DQN agent. Still, the authors observed a consistent increase in performance.

Fedus et al (2020) concluded that increasing the replay capacity and reducing the age of the oldest policy increases agent performance and that the n-step returns process is the only element used in Rainbow capable of taking advantage of increasing the replay capacity. Investigating the relationship between n-steps returns and Experience Replay, they observed that the replay capacity can mitigate the variance of n-steps returns, which partially explained the performance increase. The authors highlighted essential aspects regarding the interaction between the learning algorithms and the mechanisms that generate the training data (i.e., the memory of transitions): the distance between the oldest policy and the current policy the agent is learning (which is a classic problem in RL with ER), the state-action coverage space, the correlation between transitions (also explored in Neves et al (2022) and Sun et al (2020), but through another form of exploitation of their similarities), and the cardinality of the distribution support. For the authors, these aspects can be challenging to control separately and independently because typical algorithmic adjustments can affect several of them simultaneously. As the future works, the authors pointed to the investigation of how to untangle those different aspects to obtain agents capable of efficiently scaling in performance with the increase of available data, thus investigating how these aspects of Experience Replay interact with other classes of off-policy and multi-step reinforcement methods.

While Fedus et al (2020) demonstrated that the use of n-steps return is a critical element for Rainbow's performance, ablative studies pointed out by Fujimoto et al (2020) demonstrated that in DQN, this element is PER. According to the authors, although widely used, this prioritization method lacks a critical theoretical foundation, which they formally developed in their work. The expected loss over the distribution of a sample of transitions determines the gradient used for neural network optimization; therefore, when PER causes a bias in this distribution, it effectively modifies this gradient and influences the optimization process. The authors demonstrated that the gradient expected from the minimization of a loss function over a non-uniform distribution is equal to the gradient obtained from another distinct but equivalent loss function minimized over a uniform distribution and that one can use this

relationship to transform any loss function into a prioritized sampling scheme with a new function and vice versa. In this way, that same transformation allows a concrete understanding of the benefits of non-uniform samplings, such as in the PER, also providing a tool for designing new prioritization schemes. In this sense, the authors pointed out three relevant aspects. The first one is that the loss function and the prioritization strategy must be linked, and the design of prioritized sampling methods should not be considered independently of the loss function since this allows verifying the correctness of these methods by transforming the loss in its equivalent on a uniform sampling and verifying if it produces the same results. According to the authors, if using a proper loss function, even PER may not be biased, even if not using its importance sampling calculation (presented in Section 4). The second aspect is the reduction of variance, which allows for a deeper understanding of the prioritization benefits since it is related to the expected gradients. This variance can be reduced by a loss function over uniform sampling and carefully choosing a prioritization scheme defined in conjunction with a corresponding loss function, which means that non-uniform samplings are almost always in favor of a uniform version at the cost of more algorithms and increased computational complexity. A third interesting aspect is that the formulations demonstrated by the authors suggest that some of the benefits obtained by prioritized sampling come from the changes generated in the expected gradient and not from the prioritization itself.

The authors focused their ablative analysis around three loss functions and how they relate to uniform and non-uniform sampling, using a prioritization scheme and comparing their expected gradients. They demonstrated how to carry out the proposed transformations and reduce the gradient variance by applying gradient steps of the same size instead of interspersing larger and smaller steps, pointing out a simple way to minimize the variance of any loss function while keeping the expected gradient unchanged. In this way, the authors took PER as a basis and derived an equivalent loss function for uniform sampling, allowing them to point out corrections and possible improvements in the method. Among other things, they demonstrated that when PER uses mean squared error (MSE), including some subsets with Huber Loss, it optimizes the loss function over TD-error to a power higher two, indicating that it can favor outliers in its estimation of expected target values in the temporal difference rather than learning the mean. According to the authors, the bias in PER may cause its low performance in continuous learning algorithms that depend on the MSE. In addition, they also demonstrated that importance sampling ratios used by PER can be absorbed in the loss function itself, simplifying the algorithm. PER uses importance sampling in weighting the loss function to reduce the bias introduced by prioritization, but when using the MSE and setting the hyperparameter $\beta = 1$ (see Section 4), it is no longer biased. As the expected gradient can absorb the prioritization, PER can be unbiased even without using importance sampling if the expected gradient

is still meaningful. Based on their findings, they proposed a new prioritization scheme called Loss-adjusted Prioritized (LAP) Experience Replay, which simplifies PER by removing the unnecessary importance sampling ratios and setting the minimum priority to one, which reduces bias and the likelihood of dead transitions with near-zero sampling probability. They also proposed an equivalent loss function for uniform sampling called Prioritized Approximation Loss (PAL). It resembles a weighted variant of the Huber Loss function and produces the same expected gradient as LAP. The authors showed that when the variance in the prioritization is minimal, PAL can be used instead of LAP in a simple and computationally efficient way to train neural networks that estimate Q-values and that the loss function and the form of prioritization are closely linked. They pointed out that the loss defined by PAL is never to a power greater than two, meaning there is no more outlier bias from PER.

The authors used the Atari 2600 games and the MuJoCo continuous control tasks through the OpenAI Gym environment to empirically verify the effects produced by LAP and PAL methods. In the first case, they combined their methods with the DDQN and compared them with the original DDQN and the DDQN combined with PER. In the second case, they combined their methods with the TD3 algorithm and compared it with the original TD3, with TD3 combined with PER and SAC algorithms. According to the authors, there was no consistent difference between using LAP or PAL in TD3, which means that prioritization has little benefit in this domain and that the expressive performance gain comes from the change in the expected gradient. Consequently, they showed that it is possible to replace non-uniform sampling by modifying only the loss function and that PER adds little benefit to TD3, consistent with the authors' ablative analysis, which showed that using MSE with PER introduces bias. LAP also produced excellent results in the Atari games, surpassing the performance added by PER in 9 of 10 games, while using PAL led to worse performance in 6 games. According to the authors, this suggested that prioritization plays a more significant role in this domain, considering that games depend on longer observation horizons (with longer-term policy learning) and sometimes have sparse rewards, which does not mean that some improvements do not come from changes in the expected gradient. The authors considered the performance of PAL in MuJoCo tasks particularly interesting because of the method's simplicity. They believed its benefits over the MSE and the original Huber Loss came from its robustness and ability to better approximate the average.

For Fujimoto et al (2020), more research is necessary to understand non-uniform sampling better and propose new prioritization schemes. They also reinforced the sensitivity of deep learning algorithms to small changes since they achieved considerable performance gains in well-known algorithms just by changing the loss function. For them, this suggests that works in the literature that use intense optimization of hyperparameters or algorithmic changes may be showing gains over the original algorithms due to unclear consequences and beyond the proposals of the articles.

Different ways exist to use the agent's experiences to update its value functions, either on-policy in actor-critic methods, off-policy in methods based on Q-learning, or even in evaluating target values in TD-learning. According to Sinha et al (2022), the use of importance weights methods for prioritization can improve the evaluation of the target value for more prolonged traces using TD(γ) and can be used to reduce bias on values computed from off-policy experiments. In this sense, the authors proposed a method to weight experiences based on their likelihood under the stationary distribution of the current policy and justify this with a contraction argument over the Bellman evaluation operator. Their proposal aimed to encourage on-policy sampling behaviors similar to the ReF-ER but without the need to know the policy distribution. For the authors, the Distribution Correction (DisCor) method (Kumar et al, 2020) suggested not using on-policy experiences in this context, which seems to contrast with their proposal. However, DisCor bases its analysis on the optimality operator of Bellman and not on the Bellman evaluation operator. The difference is that the first operator seeks to find the optimal Q-value, while the second seeks to find the Q-value function of the current policy. The authors' objective was to improve the performance of TD learning in the approximation of functions and not use the weights to estimate an advantage function or to reduce the bias in estimating rewards. Indeed, Sinha et al (2022) mixed on-policy and off-policy experiments and sought to balance their variance and bias, respectively, by estimating likelihood-free density ratios and using the ratios learned as a weight for prioritization.

On-policy methods are sometimes more effective in specific domains, such as continuous learning, but using off-policy data produces more efficient sampling, which is critical for exploring environments with rare and expensive experiences. In this sense, replaying experiences based on prioritization schemes increases sampling efficiency, but its use in different methods and domains depends, according to the authors, on the strategy and objective of this prioritization. For example, PER is not very effective in methods based on actor-critic because it bases its prioritization scheme on the magnitude of TD-error from the off-policy experiences stored in the buffer. In contrast, actor-critic methods seek to approximate a Q-value function induced by the current policy, in which it may be better to perform prioritized sampling to reflect on-policy experiences. Given that, prioritization schemes can lead to considerable improvements in sampling in actor-critic methods. According to Sinha et al (2022), it is possible to estimate the value function of a policy by minimizing the expected squared difference between the estimate of the critical function, and the estimate of its target function (actor-critic methods contain the actor function, the critic function, and each of this has its respective target function) over a replay buffer that properly reflects the discrepancy between the two. In this way, one can consider this discrepancy as a priority when it preserves the contraction properties of the Bellman evaluation operator while being measured by the expected quadratic distance under some state-action distribution. In this sense, the authors presented proof that the stationary distribution of

the current policy is the only one in which the Bellman operator is a contraction, which they then define as being a property, and proposed the use of the non-stationary distribution as the underlying distribution of the buffer of repetition, leading to the elaboration of their method of experience replay with likelihood-free weights.

Generally, there is less experience from the current policy, and because of this, its use produces estimates with high variance. On the other hand, more experiences under other policies for the same environment leads to the introduction of bias. Therefore, the method proposed by the authors seeks to obtain their estimates of density rates from a classifier trained to distinguish different types of experiences. For this, they used a smaller buffer, which contains experiences closer to the on-policy experiences, and another larger buffer to store the off-policy experiences, estimating the density rates from these buffers. These rates are then used as importance weights to update the Q-value function, encouraging more updates from more desired state-action pairs under the current policy stationary distribution, which are more present in the smaller buffer. The authors combined their approach with the methods Soft-Actor Critic (SAC) (Haarnoja et al, 2018), Data Regularized Q (DrQ) (Yarats et al, 2021) and the DDQN, and compare with the use of uniform sampling, PER, and Emphasizing Recent Experience (ERE) (Wang and Ross, 2019). To evaluate their approach, they used the environments of ALE, the DeepMind Control Suite (DCS) (Tassa et al, 2018), and the OpenAI Gym tasks, demonstrating considerable improvements due to the introduction of their method.

Schmitt et al (2020) proposed an importance sampling scheme for training actor-critic off-policy agents from a large replay buffer containing at least ten times more experiences than the limit of one million commonly used in the literature. As part of their work, they proposed solutions to improve stability and make the off-policy learning of those agents more efficient, even when they are learning from the experiences of other agents through a shared experience replay process. Their approach obtained state-of-the-art results in training a single agent for 200 million interaction steps on the ALE and DMLab-30 environments and in training concurrently several agents sharing the same repetition buffer. Their algorithm has two main characteristics: mixing transitions from the replay buffer with on-policy transitions and computing what the authors define as a trust region scheme.

According to the authors, combining Experience Replay with actor-critic algorithms is difficult due to their on-policy nature. Although, mixing replay buffer experiences with on-policy transition data using the trust region schemes proposed by them makes it possible that the importance sampling method called V-trace effectively scales to data distributions over which its original formulation would become unstable. V-trace is a technique widely used in training actor-critic agents, which controls the variance commonly observed in naive importance sampling, but at the cost of introducing bias in the estimates. This bias is because its estimate of the v_π value function does not correspond to the expected return value for the policy π but rather for a policy $\tilde{\pi}$, which

is only implicitly related to π and computed biased. Therefore, it can distance too much from π. In this way, the policy gradient is also biased so that, given a value function v^*, the V-trace does not guarantee convergence to a policy π^* in offline training, as the authors demonstrated. Given this, blending on-policy transitions can mitigate the distortion caused by V-trace bias, regularizing the Q-value estimates. A trust region scheme for the off-policy V-trace limits the sampling of off-policy transitions by rejecting highly biased transitions, aiming to provide the agent with an experience replay scheme enriched with experiences with low variance (because of V-trace) and low bias. For this, the authors defined the behavior-relevant function to classify relevant behaviors and a trust-region estimator, which computes expected values from relevant experiences. This approach applied the Kullback–Leibler divergence between the target policy π and the implicit policy $\tilde{\pi}$ and is used for the policy and value estimates.

In addition to the performance improvement compared to state-of-the-art algorithms, the authors showed that uniform sampling, using their approach, obtains results comparable to PER. They also demonstrated that learning using only off-policy experiences and without inserting recent experiences degrades performance, which also happens when using shared experiences without defining trust regions. According to the authors, their ablative experiments exhibited little benefit in using PER in actor-critic methods on the DMLab-30 environments. They highlighted that PER computes priorities on the magnitude of TD-error, which is poorly defined when sharing multi-agent experiences.

According to Kapturowski et al (2019), increasingly complex partially observable domains have demanded considerable advances in the representation of transitions' memories and solutions based on the principles of recurrent neural networks such as LSTM, and its use has increased to overcome the challenges of these environments. Given this, the authors investigated agent training using RNNs with Experience Replay and demonstrated its effects on parameter delay, resulting in representational deviation and recurrent state outdatedness, potentially exacerbated in distributed training environments, which leads to a loss of stability and performance during agent training. From a series of empirical studies on mitigating these issues, the authors presented their proposal called Recurrent Replay Distributed DQM (R2D2), whose significant algorithmic advances led to state-of-the-art results in Atari 2600 games in ALE and equivalent results or higher than those of state-of-the-art in DMLab-30 environment. According to the authors, R2D2 was the first to achieve these results using the same network architecture and the same hyperparameter values.

The authors model the environment as a Partially Observable Markov Decision Process (POMDP), defined by a tuple (S, A, T, R, Ω, O), in which T corresponds to the transition function, Ω is the set of observations potentially received by the agent, and O maps the states to probability distributions

over the observations. Thus, the agent receives an observation $o \in \Omega$ containing only partial information about the underlying state $s \in S$. An action in the environment results in a transition to a state $s' \sim T(.|s, a)$, which results in an observation $o \sim \Omega(.|s, a)$ and a reward $r \sim R(s, a)$. The authors then used an optimized RNN with a technique called Backpropagation Over Time (BPTT) (Werbos, 1990) to learn a representation that disambiguates the real state of the POMDP. In turn, an R2D2 agent is a DDQN agent with n-step returns that uses Prioritized Distributional Experience Replay and whose experiences are generated by 256 agents in parallel and used by a single learning agent. The actors use the Dueling Networks (Wang et al, 2016) architecture (with an additional LSTM layer after the convolutional layers) to approximate the Q-function. Instead of storing the transitions represented by the tuples (s, a, r, s'), the algorithm stores sequences of tuples in the format (s, a, r), with a fixed length ($m = 80$), and underlying sequences that overlap periodically at each predefined time interval, without exceeding the limits of each episode. The authors used invertible value function rescaling for the reward values to generate n-step target values for the Q-value function. They also used a more aggressive prioritization scheme that employs a combination of max and mean absolute n-step TD-errors, as the mean over long sequences tends to hide larger errors, which compresses the range of priorities and limits the prioritization ability of valuable experiences.

According to the authors, to deal with partially observable environments, agents need representations of states that encode information about their trajectory of state-action pairs in addition to their observation of the current state. According to them, the most common way to do this is to use an LSTM trained on complete trajectories stored in the replay buffer so that it can learn relevant long-term dependencies. For this, it is possible to use the zero start state to initialize the network at the beginning of the sampling sequence, which allows independent decorrelated sampling of relatively short sequences, which is essential to obtain robust optimization. However, this forces the recurrent network to learn to retrieve useful predictions from an atypical initial recurrent state, which could limit its ability to rely on its recurrent states to learn to exploit long temporal correlations. Another possible way is to replay the entire episode trajectories, which avoids finding an inadequate initial state. However, the possibility of sequence size variations, which also depend on each environment, the high variance in network updates, and using highly correlated data can bring a series of stability problems. Therefore, the authors proposed and evaluated two training strategies to measure and mitigate the harmful effects of representational deviation and the outdating of recurrent states. After comparing trained agents using each of the strategies in various DeepMind Lab environments, the authors identified the contributions of each of them so that their combination consistently produced the smallest discrepancy in the last states of the sequence together with more robust performance improvements than when used separately.

Finally, the authors evaluated the R2D2 in the 57 games of the ALE and the different tasks of the DMLab-30 environment. They compared their results with those obtained by Ape-X (Horgan et al, 2018) and IMPALA (Espeholt et al, 2018). Unlike R2D2, the hyperparameters were adjusted separately for each environment. The authors pointed out that one of the most significant contributions of the DQN was its ability to generalize over different environments using the same network architecture and the same hyperparameter values, which, according to them, has not been maintained in some works in the literature. According to them, until the date of their publication, no other work had maintained this kind of generalization pattern, using the same architecture and hyperparameters in both ALE and DMLab-30 environments. The authors stated that Rainbow and IQN (Dabney et al, 2018) held state-of-the-art using a single agent in Atari games. In turn, Ape-X held state-of-the-art results using multi-agents. R2D2 obtained better results in these environments than the other methods that used a single agent and quadrupled the results obtained by Ape-X. They also pointed out that other methods had not yet obtained results in so many games (52 of the 57) as R2D2, with above-human performance. Still, like the other methods, R2D2 did not show considerable advances in the games Montezuma's Revenge and Pitfall, which are known for being difficult environments to explore. In turn, the DMLab-30 environment consists of 30 problems in first-person 3D environments and requires long-term memory to obtain reasonable results. According to the authors, while the best-performing algorithms consisted of actor-critic methods trained with some on-policy regime, R2D2 was the first to reach the state-of-the-art using a value-function-based agent.

Rolnick et al (2019) addressed the problem of catastrophic forgetting that occurs when new experiences overwrite old experiences in the multitasking continuous learning scenario. They proposed a method called Continual Learning with Experience Replay (CLEAR), which sought to balance off-policy learning, using behavioral cloning from experience replay, with on-policy learning, in a trade-off that they define between the concepts of stability (from the preservation of achieved knowledge) and plasticity (from the acquisition of new knowledge). According to the authors, the literature often mitigated catastrophic forgetting by using intensive computational resources to try to learn all tasks simultaneously and not sequentially. However, this problem becomes critical as the application of reinforcement learning in continuous learning problems grows in industry and robotics, with scenarios where rare (and difficult to obtain) experiences may be more common, making simultaneous learning unfeasible. It demands the agent to be able to learn one task at a time in a sequence that is not under the agent's control and whose limits are not unknown. For the authors, this training paradigm eliminates the possibility of simultaneous learning, in which one learns from several tasks, increasing the possibility of catastrophic forgetting. However, efforts to prevent catastrophic forgetting have concentrated on approaches that seek to protect the parameters of neural networks inferred in a given task when the agent learns

another task, which is motivated by the concepts of consolidation of synapses expected from neuroscience. For Rolnick et al (2019), many possibilities of using Experience Replay in the scenario of catastrophic forgetting have been ignored in the literature, while the works that used and widely investigated the repetition of experiences did so with a focus on the efficiency of the use of data for agent learning.

To ensure the expected stability from off-policy learning in CLEAR, the authors introduced a method of behavioral cloning between past policies and current policies. Based on ablative studies on three DeepMind Lab tasks, they evaluated the effects of applying their method to reduce the damage caused by catastrophic forgetting and to verify its behavior in different balances on the trade-off between stability and plasticity, concluding that behavioral cloning is just as crucial to CLEAR performance as using on-policy experiences. They used 900 million frames observed by the agent, varying the size of the replay buffer from 450 million to 5 million experiences. Only in the latter case was some loss of performance observed. The authors also conducted experiments to evaluate the performance of CLEAR varying balances between off-policy and on-policy learning and how these variations impact stability and plasticity, concluding that a trade-off with a percentage of 50/50 led to better results on the DMLab-30 environment, and 75/25 was the best balance on the Atari 2600. Finally, the authors evaluated CLEAR compared to two state-of-the-art methods of reducing catastrophic forgetting that assume task boundaries are known. The authors demonstrated that CLEAR achieved equivalent or better results than both methods, despite being simple and agnostic about task boundaries.

According to Rolnick et al (2019), in those continuous learning cases in which storing experience in a replay buffer, is prohibitive, better approaches are in the methods focused on protecting parameters when passing from one task to another. In scenarios where the type and boundaries between tasks can be somehow shared, exploiting this can reduce the computational cost or even accelerate agent learning. However, in many cases, such as when the action space changes from one task to another, trying to address past policy distributions, whether through behavior cloning, off-policy learning, weight protection, or another strategy to prevent catastrophic forgetting, could lead to considerable performance losses. Developing algorithms that selectively forget or protect specific learned behaviors would be necessary in those cases.

5.1 Research on Experience Replay and Some Directions for Future Works

A more systematic literature review can show the researcher's interest in Reinforcement Learning using Experience Replay over the last years and point out future directions. We conducted six restriction levels on querying over five indexing databases: CAPES[1], ACM-DL[2] (Full-text collection and

[1] https://www-periodicos-capes-gov-br.ezl.periodicos.capes.gov.br/index.php
[2] https://dl.acm.org/

Guide to Computing Literature), IEEE Explore[3], ScienceDirect[4], and Scopus[5]. From that, we picked up some relevant works that are recent and closely related to the subjects we discussed in this survey. Appendix A presents the detailed search results, methodology, and applied criteria. We could verify that many works in the literature apply well-known RL methods to approach different classes of complex and exciting problems such as: (i) geographical routing-decision process to assign sensing tasks to mobile users, (ii) anomaly detection in smart environments, (iii) cellular-connected unmanned aerial vehicles network, (iv) nonlinearities and uncertainties of biochemical reactions in wastewater treatment process control, (v) robotic lever control, (vi) handover decision in 5G Ultradense Networks, and (vii) automation of software test (Zhang and Qiu, 2022; Tao and Hafid, 2020; Fährmann et al, 2022; Koroglu and Sen, 2022; Crowder et al, 2021; Li et al, 2022b; Wu et al, 2022; Remman and Lekkas, 2021; Rosenbauer et al, 2020). However, we are more interested in those works that propose changes or new methods using Experience Replay. Moreover, we are more concerned with works investigating the Experience Replay and delving into its theoretical issues and empirical investigations. Table 1 summarizes some methods found in the literature.

Optimal control is a recurrent problem in the literature that approaches complex nonlinear systems, such as robotics, autonomous driving, and domains like biochemical reactions (Yang et al, 2022). Many research works use variations of policy gradient-based reinforcement learning methods with novel mechanisms of experience replay, such as in Wang et al (2019), whose proposal was to transform adaptive cruise control problems (ACC) into optimal tracking control problems and handled by a novel model-free Adaptive Dynamic Programming (ADP) approach called ADPER. Similarly, Yang and He (2020) proposed a decentralized event-triggered control (ETC) strategy based on Adaptive Critic Learning (ACL) and using experience replay, and Zhou et al (2022) proposed an attention-based actor-critic algorithm with Prioritized Experience Replay (PER) to improve the convergence time on robotic motion planning problems, changing the LSTM-based advantage actor-critic algorithm by using an encoder attention weight and initializing the networks using PER. Kim et al (2020) proposed a motion planning algorithm for robot manipulators using a twin delayed deep deterministic policy gradient, which applies hindsight experience replay. Prianto et al (2020) approached the path planning for multi-arm manipulators by proposing a method based on the Soft Actor-Critic (SAC) algorithm with hindsight experience replay to improve exploration in high-dimensional problems. Sovrano et al (2022) approached the complex problem of autonomously driving in rule-dense environments by partitioning the experiences buffer into clusters labeled by explanations about rulesets. Cui et al (2023) proposed an experience replay approach called Double Bias Experience Replay (DBER) with a new loss function (which is a challenging environment modeling problem in these domains) and applied to

[3] https://ieeexplore.ieee.org/Xplore/home.jsp
[4] https://www.sciencedirect.com/
[5] https://www.elsevier.com/pt-br/solutions/scopus

Table 1 New methods, problems, and propositions

Method	Base Method	Domain	Propositions
ADPER (Wang et al, 2019)	Actor-Critic	Adaptive Cruise Control (ACC) in driving scenarios	Transform the Adaptive Cruise Control (ACC) into a novel Model-free Adaptive Dynamic Programming using a new ER mechanism.
Decentralized ETC (Yang and He, 2020)	Actor-Critic A2C	Event-triggered Control (ETC) in non-linear systems	Model Decentralized ETC in groups of optimal policies with ACL and ER
ERTS-Q (Jiang et al, 2021)	Q-learning	Maintain Car and Maze	Adaptive tree structure integrated with ER for Q-learning to virtually model sequence of states from transitions
Attention-based AC (Zhou et al, 2022)	Actor-Critic A2C	2D Robotic motion planning in dynamic scenarios	Using an attention mechanism to encoding features in place of classically using an LSTM and combining it PER.
MSER (Yang and Peng, 2021)	DDPG	Robot Trajectory Planning	A meta-learning-based experience replay buffer separation to approach the computacional complexity present in PER
DSEGR (Zhang and Qiu, 2022)	DDPG	Routing protocol in unmanned aerial vehicles (UAVs) with ad roc networks	Learning routing decision processes using DDPG and PER from predictions of residual energy of neighbor nodes.
DNPER-DDPG (Kang et al, 2021)	DDPG	Classical control in Open AI Gym	A double network prioritized experience replay mechanism to reduce local optimal policy incidence and speed up convergence by changing prioritization criteria
DRL-QER (Wei et al, 2022)	DQN	Atari 2600 games	A quantum-inspired ER which chooses experiences according to its complexity and replayed times.
Adaptation of DDQN with PER (Fährmann et al, 2022)	DDQN	Anomaly detection in smart environment	An adaptation of DDQN with PER for learning decision-making function from from multivariate sequential time series data
FER (Kong et al, 2021)	DDPG TD3 SAC	Mujoco Gym and CARLA simulator	Employing a half-normal sampling probability window to changes replay probability of newer experiences in the replay buffer at uniforme sampling
ERO-ATSC ERO-ATSC+TSC (Zha et al, 2019)	DDPG ERO	Pendulum control in continuous action value space	ER optimization by using dual memory, an alternating transition selection control (ATSC) and a transition storage control (TSC) to mitigate catastrophic forgetting
EIT (Sovrano et al, 2022)	Distributed DQN	Power allocation and spectrum sharing interference in wireless networks	An experienced-instance transfer strategy for exploring historical and current knowledge using ER
XAER (Sovrano et al, 2022)	DQN TD3 SAC	Rule-dense and exception-ridden environments in autonomous driving	Partitioning the replay buffer into clusters labeled on a per-explanation basis to sampling experiences in a curricular and task-oriented manner.
FPER (Ma et al, 2022)	Dueling Networks DDPG	OpenAI Gym	Method to fresher experiences in the buffer on PER with improved experience replacement strategy
LSER (Chen et al, 2022a)	DDPG	Online recommendations from users' dynamics in recommendation systems	An state-aware experience replay model using a locality-sensitivity method to selectively discover most relevant experiences
LiDER (Du et al, 2022)	A3C	Atari 2600 games	Proposes a framework to refresh replayed experiences by leveraging the agent's current policy in ER
RAMDP (Kim et al, 2020)	TD3 DDPG	Motion and task planning for robot manipulators in manufacturing industry	Defines a Robot Arm Markov Decision Process and approach by proposing an algorithm using hindsight experience replay and TD3
DBER (Cui et al, 2023)	DQN, QR-DQN Dueling DDQN	Autonomous driving	An ER approach with a double bias strategy and a new loss function
DPCER Li and Ji (2021)	MP-DQN, DQN DDPG	Distribuited reinforcement learning in simulated RoboCup soccer	A distributed training framework with a parallel curriculum experience replay to identify the difficulty level of subtasks collected in parallel
SAC-based path planning algorithm (Prianto et al, 2020)	SAC	Path planning for multi-arm manipulators in high-dimensional problems	Proposes a SAC-based algorithm with hindsight experience replay and configuration space augmentation
R-T Experience Prioritized Parameter (Gao et al, 2021)	DDPG	Open AI Gym	Proposes a prioritization parameter that combines past experiences rewards with TD-errors to improve data efficiency on sampling and storing transitions.
HER with sampling rate decay (Vecchietti et al, 2022)	DDPG	Robot control in multigoal tasks with sparse reward in Open AI Gym	A variable sampling rate decay strategy for sampling in hindsight experience replay to increase the number of experiences and reduce the original method bias.
Imp-PER (Zhang et al, 2020)	DDQN DDPG	Atari 2600 games Mujoco	A self-adaptive priority correction algorithm to reduce bias in PER and approach outdated experiences by defining importance weights
SLER (Li et al, 2021)	DQN	Problems in learning multiple tasks on StarCraft II and GridWorld	A continual learning algorithm and an experience replay module to approach catastrophic forgetting using less system memory compared with standard dual ER.

the classical off-policy algorithms DQN and DDQN, and also to the Quantile Regression DQN (QR-DQN). Li and Ji (2021) presented a distributed training framework with parallel curriculum experience replay to approach sparse rewards in distributed training on a simulated environment to robots.

Many works explored novel data structures to make Experience Replay even more data-efficient by modeling the behavior of the transition and exploring predictions about states and rewards (Jiang et al, 2021). Others proposed different forms to define the importance of the samples (Kong et al, 2021), new

prioritization criteria (Gao et al, 2021), and changes in value function approximation updating to speed up the convergence time by avoiding the incidence of optimal local policy (Kang et al, 2021). In Wei et al (2022), one can find a new (quantum-inspired) experience replay paradigm that considers the complexity and the replayed times of each experience to achieve a better balance between exploration and exploitation, which is a fundamental choice and a complex problem.

Catastrophic forgetting is another well-known problem affecting training stability and agent performance, especially in continuous action spaces. It is mainly related to buffer size limitation and is sensitive to experience sampling and storage strategies. Recent works addressed this problem by proposing strategies to improve control over the mechanisms for storing, selecting, retaining, and forgetting in experience replay (Osei and Lopez, 2023), including using transfer learning about past experiences (Anzaldo and Andrade, 2022). According to (Li et al, 2021), their Self-generated Long-term Experience Replay (SLER) approach improved the dual experience replay algorithm, applied in continuous learning tasks to mitigate catastrophic forgetting by reducing its impact on computer memory-consuming growth. (Li et al, 2022a) proposed a method to cluster and replay experiences by a divide-and-conquer framework based on time division to explore experiences that may not be prioritized during sampling or even forgotten due to limited transition memory.

As discussed in Section 4, a (relatively) recent, theoretically relevant, and still little-explored question refers to the trade-off regarding how recent and closer to current policy (and potentially more biased) or how outdated (but more diverse, possibly rare, or expensive to obtain) the experiences in the memory should be to contribute to the agent learning process. Some works approached these questions by changing the priority measure in PER. Ma et al (2022) proposed to increase the probability of sampling more recent experiences with a novel strategy for replacing experiences in the memory buffer, while (Zhang et al, 2020) presented a novel self-adaptive priority correction algorithm called Importance-PER to reduce bias. Instead of approaching the sampling strategy, Du et al (2022) proposed a framework to *refresh* experiences by moving the agent back to past states, executing sequences of actions following its current policy, and storing and reusing new experiences from this process if it turned out better than what the agent previously experienced. (Liu et al, 2022) proposed a dynamic experience replay strategy based on Mult-armed Bandit, which combines multiple priority weighted criteria to measure the importance of the experiences and adjust its weights from one episode to another. Yang and Peng (2021) proposed the Meta-learning-based Experience Replay (MSER) applied in DDPG to deal with the computational complexity in PER and its need for careful hyperparameter adjustments. They divided the experiences memory buffer into a successful experience buffer and a failure experience buffer and uniformly sampled from those buffers according to a ratio learned by a neural network.

Hindsight Experience Replay (HER) and Aggressive Rewards to Counter Bias in HER (ARCHER) are two strategies to deal with the classic and challenging problem of sparse rewards but also present some problems. HER considers every failure a success for an alternative (virtual) goal and uses these goals by uniform sampling. Although it introduces bias when not taking variable importance at different training moments and not considering the relevance of goals to the agent learning. (Vecchietti et al, 2022) investigated the impact of a variable sampling rate on training performance and proposed a sample decay strategy that decreases the number of hindsight experiences during training. Manela and Biess (2021) proposed to improve HER by prioritizing virtual goals and reducing bias by removing misleading samples. (Manela and Biess, 2022) presented an algorithm combining curriculum learning with HER to learn sequential object manipulation tasks for multiple goals and sparse feedback by exploiting the recurrent structure inherent in many object manipulation tasks. (Chen et al, 2022b) approached the problem of dealing with sparse rewards in online recommender systems that use reinforcement learning. They proposed a state-aware experience replay model to allow the agent to selectively discover the relevant experiences using locality-sensitive hashing that retain the most meaningful experience at scale and that replays more valuable experiences with a higher chance.

Multi-agent and cooperative robot tasks also can take advantage of experience replay and Yu et al (2023) approached the problem of processing environmental information while increasing the number of participants by proposing a hybrid attention module integrated with Multi-agent Deep Deterministic policy gradient (MADDPG) and prioritized experience replay. In turn, Nicholaus and Kang (2022) proposed a technique to use experience replay that introduces additional strength to the exploration-exploitation trade-off in these scenarios.

6 Conclusions

This work demonstrates that Experience Replay is a fundamental idea with many theoretical and empirical open problems still being investigated to understand its contributions and propose improvements and new applications with different reinforcement learning methods to solve complex problems in many research fields. Automation, robotics, autonomous driving, trajectory planning, and optimization are among the many application areas that lead to the proposition of new methods of reinforcement learning as well as new approaches and techniques of experience replay so that these methods can become even more efficient in using data from transitions experienced by agents. New schemas for prioritization of experiences and importance sampling, techniques to avoid catastrophic forgetting, dealing with sparse rewards, and improving memory efficiency in multi-agent environments are among the research explicitly dedicated to improving experience replay.

Acknowledgments. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. It has also received partial funding from Brazilian Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and from Pontifical Catholic University of Minas Gerais (PUC Minas).

Appendix A Indexing Database Query Results

To query the indexing databases, we defined six query strings as follows, whose criteria transform the least restrictive search into a fully restrictive one regarding the subject of the research works and its relation with Experience Replay and Reinforcement Learning. We set the range for all the databases from the earliest year until the search date. Table A1 presents the counting of results returned by applying these queries to each database after removing duplicates.

- **Q1** = Search for ("Experience Replay" AND ("Reinforcement Learning" OR "Deep Reinforcement Learning")) in all (any) sections.
- **Q2** = Search for ("Experience Replay" AND ("Reinforcement Learning" OR "Deep Reinforcement Learning")) in the document title, keywords, and abstract.
- **Q3** = Search for ("Reinforcement Learning" OR ("Experience Replay" OR "Deep Reinforcement Learning")) in the document title AND "Experience Replay" in the keyword OR "Experience Replay" in the abstract.
- **Q4** = Search for ("Reinforcement Learning" OR ("Experience Replay" OR "Deep Reinforcement Learning")) in the document title AND "Experience Replay" in the keyword AND "Experience Replay" in the abstract.
- **Q5** = Search for "Experience Replay" in the document title AND ("Reinforcement Learning" OR "Deep Reinforcement Learning") in the keywords AND ("Reinforcement Learning" OR "Deep Reinforcement Learning") in the abstract.
- **Q6** = Search for "Experience Replay" in the document title AND (("Reinforcement Learning" OR "Deep Reinforcement Learning") AND "Experience Replay") in the keywords AND ("Reinforcement Learning" OR "Deep Reinforcement Learning" AND "Experience Replay") in the abstract.

The number of publications overgrew after 2015, with numbers much higher than in previous years and continuously growing until 2022. We could observe this behavior in all levels of research criteria restrictions. We analyzed the results for each search to understand their distribution by publication type, year, journal, and publisher. We also identified and grouped the keywords over the whole set and by year of publication to get an initial idea of the problems associated with the main terms of our searches. After this initial analysis, we applied the following criteria, in the order presented, to select up to 40 research papers to read:

1. Only works published after 2018;

Table A1 Number of results for each query. In some cases, it was not possible to execute the query because of limitations in the search form (indicated by *).

Database	Q1	Q2	Q3	Q4	Q5	Q6
ACMDL-Full-text	471	170	41	8	5	5
ACMDL-Guide	1577	543	150	31	20	15
CAPES	1000	656	104	61	43	*
IEEE Xplore	427	*	309	84	45	33
ScienceDirect	1099	97	84	*	20	*
Scopus	5128	652	626	231	132	88

2. Works remaining in each base after applying the query with the most restrictive criteria, starting in Q6;
3. Works published in high-impact journals;
4. Works published in high-impact conferences; and
5. If there are still eligible works and the maximum number has not been reached, return item two and consider the following decreasing restriction query (Q5...Q1).

A1 - ACM-DL

Table A2 shows the results of applying each query on the ACM-DL Full-Text Collection and the ACM-DL Guide to Computing Literature. Figures A1 and A2 illustrate the distribution of results from Q1 (the least restrictive query) over the years on the ACM-DL Full-Text Collection and the ACM-DL Guide to Computing Literature, respectively.

Table A2 Distribution of the publication types on the ACM-DL Full-Text Collection and the ACM-DL Guide to Computing Literature.

	Results by Level of Restriction											
	ACM-DL Full Text						**ACM-DL - Guide**					
Queries	Q1	Q2	Q3	Q4	Q5	Q6	Q1	Q2	Q3	Q4	Q5	Q6
Total	**471**	**170**	**41**	**8**	**5**	**5**	**1577**	**543**	**149**	**31**	**20**	**15**
Inproceeding	346	155	36	7	4	4	668	257	83	14	14	6
Article	106	15	5	1	1	1	848	286	57	17	6	9
Proceedings	16	0	0	0	0	0	16	0	0	0	0	0
Inbook	2	0	0	0	0	0	32	0	1	0	0	0
Book	1	0	0	0	0	0	2	0	1	0	0	0
PhD Thesis	0	0	0	0	0	0	11	0	4	0	0	0

A2 - CAPES

Table A3 shows the results of applying each query on CAPES. Figure A3 illustrates the distribution of results from Q1 (the least restrictive query) over the years on CAPES.

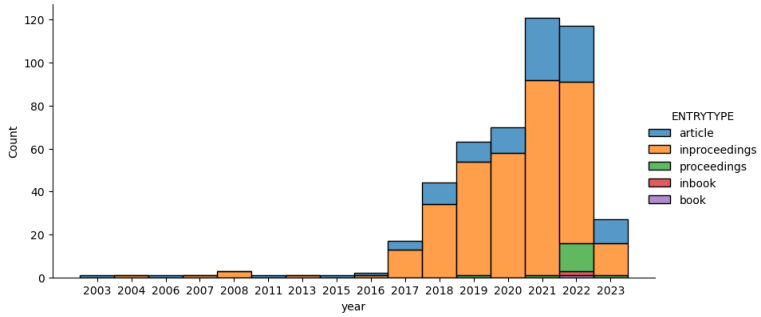

Fig. A1 Q1- Types of publications over the years on the ACM-DL Full-Text Collection.

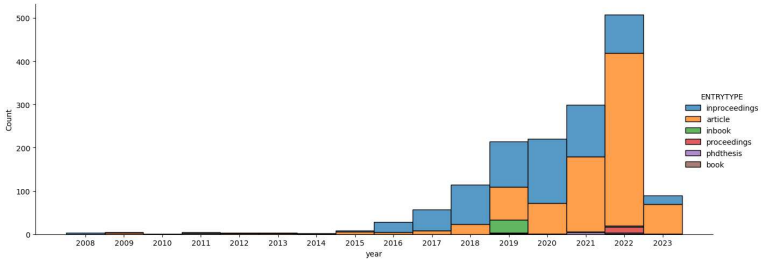

Fig. A2 Q1- Types of publications over the years on the ACM-DL Guide to Computing Literature.

Table A3 Distribution of the publication types on CAPES. In some cases, it was not possible to execute the query because of limitations in the search form (indicated by *).

	Results by Level of Restriction					
Queries	Q1	Q2	Q3	Q4	Q5	Q6
Total	**998**	**654**	**104**	**61**	**5**	*
Inproceedings	0	29	6	5	4	*
Article	998	615	94	54	1	*
Others	0	4	1	1	0	*

A3 - IEEE Xplore

Table A4 shows the results of applying each query on IEEE Xplore. Figure A4 illustrates the distribution of results from Q1 (the least restrictive query) over the years on IEEE Xplore.

A4 - Science Direct

Table A5 shows the results of applying each query on Science Direct. Figure A5 illustrates the distribution of results from Q1 (the least restrictive query) over the years on Science Direct.

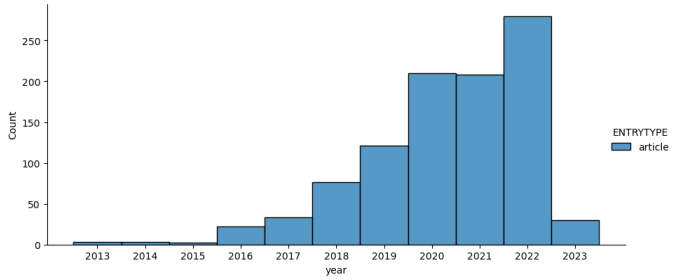

Fig. A3 Q1- Types of publications over the years on CAPES.

Table A4 Distribution of the publication types on IEEE Xplore. In some cases, it was not possible to execute the query because of limitations in the search form (indicated by *).

	Results by Level of Restriction					
Queries	Q1	Q2	Q3	Q4	Q5	Q6
Total	**427**	*****	**309**	**84**	**45**	**33**
Inproceedings	256	*	176	48	30	21
Article	171	*	133	36	15	12

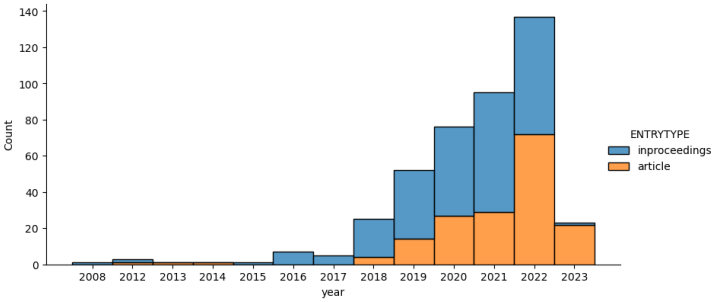

Fig. A4 Q1- Types of publications over the years on IEEE Xplore.

A5 - Scopus

Table A6 shows the results of applying each query on Scopus. Figure A6 illustrates the distribution of results from Q1 (the least restrictive query) over the years on Scopus.

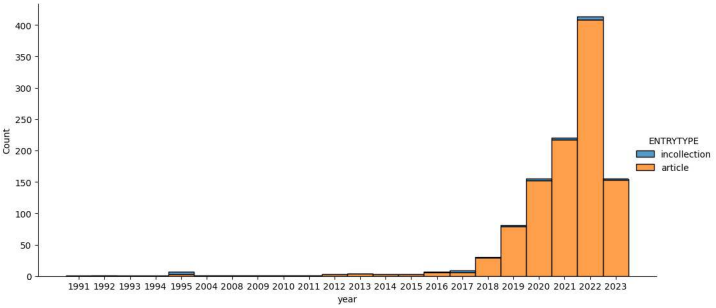

Fig. A5 Q1- Types of publications over the years on Direct Science.

Table A5 Distribution of the publication types on Science Direct. In some cases, it was not possible to execute the query because of limitations in the search form (indicated by *).

	Results by Level of Restriction					
Queries	Q1	Q2	Q3	Q4	Q5	Q6
Total	**1099**	**97**	**84**	*****	**20**	*****
Article	1072	0	83	*	20	*
Incollection	27	1	1	*	0	*

Table A6 Distribution of the publication types on Scopus.

	Results by Level of Restriction					
Queries	Q1	Q2	Q3	Q4	Q5	Q6
Total	**5128**	**851**	**626**	**231**	**132**	**88**
Article	2762	453	342	127	61	45
Conference	2185	363	260	96	66	40
Book	19	1	0	0	0	0

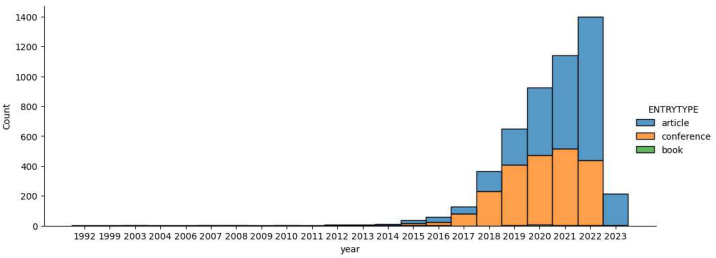

Fig. A6 Q1- Types of publications over the years on Scopus.

References

Anzaldo A, Andrade ÁG (2022) Experience replay-based power control for sum-rate maximization in multi-cell networks. IEEE Wireless Communications Letters 11(11):2350–2354. https://doi.org/10.1109/LWC.2022.3202904

Bellemare MG, Naddaf Y, Veness J, et al (2013) The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research 47:253–279. https://doi.org/10.1613/jair.3912

Bellemare MG, Dabney W, Munos R (2017) A distributional perspective on reinforcement learning. In: International Conference on Machine learning, PMLR, pp 449–458

Brockman G, Cheung V, Pettersson L, et al (2016) Openai gym. arXiv preprint arXiv:160601540 https://doi.org/10.48550/arXiv.1606.01540

Castro PS, Moitra S, Gelada C, et al (2018) Dopamine: A research framework for deep reinforcement learning. arXiv preprint arXiv:181206110 https://doi.org/10.48550/arXiv.1812.06110

Chen X, Yao L, McAuley J, et al (2022a) Locality-sensitive state-guided experience replay optimization for sparse rewards in online recommendation. In: SIGIR 2022 - Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. Association for Computing Machinery, Inc, p 1316 – 1325, https://doi.org/10.1145/3477495.3532015

Chen X, Yao L, McAuley J, et al (2022b) Locality-sensitive state-guided experience replay optimization for sparse rewards in online recommendation. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp 1316–1325, https://doi.org/10.1145/3477495.3532015

Crowder DC, Abreu J, Kirsch RF (2021) Hindsight experience replay improves reinforcement learning for control of a mimo musculoskeletal model of the human arm. IEEE Transactions on Neural Systems and Rehabilitation Engineering 29:1016–1025. https://doi.org/10.1109/TNSRE.2021.3081056

Cui J, Yuan L, He L, et al (2023) Multi-input autonomous driving based on deep reinforcement learning with double bias experience replay. IEEE Sensors Journal 23(11):11,253–11,261. https://doi.org/10.1109/JSEN.2023.3237206

Dabney W, Ostrovski G, Silver D, et al (2018) Implicit quantile networks for distributional reinforcement learning. In: Proceedings of the 35th International Conference on Machine Learning, PMLR, pp 1096–1105

Degris T, White M, Sutton RS (2012) Off-Policy Actor-Critic. In: 29th International Conference on Machine Learning

Du Y, Warnell G, Gebremedhin A, et al (2022) Lucid dreaming for experience replay: refreshing past states with the current policy. Neural Computing and Applications 34(3):1687–1712. https://doi.org/10.1007/s00521-021-06104-5

Espeholt L, Soyer H, Munos R, et al (2018) Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In: 35th International Conference on Machine Learning, PMLR, pp 1407–1416

Fährmann D, Jorek N, Damer N, et al (2022) Double deep q-learning with prioritized experience replay for anomaly detection in smart environments. IEEE Access 10:60,836–60,848. https://doi.org/10.1109/ACCESS.2022.3179720

Fedus W, Ramachandran P, Agarwal R, et al (2020) Revisiting fundamentals of experience replay. In: International Conference on Machine Learning, PMLR, pp 3061–3071

Fortunato M, Azar MG, Piot B, et al (2018) Noisy networks for exploration. In: 6th International Conference on Learning Representations

Fujimoto S, van Hoof H, Meger D (2018) Addressing function approximation error in actor-critic methods. In: Dy J, Krause A (eds) Proceedings of the 35th International Conference on Machine Learning, vol 80. PMLR, pp 1587–1596

Fujimoto S, Meger D, Precup D (2020) An equivalence between loss functions and non-uniform sampling in experience replay. In: Larochelle H, Ranzato M, Hadsell R, et al (eds) Advances in Neural Information Processing Systems, vol 33. Curran Associates, Inc., pp 14,219–14,230

Gao J, Li X, Liu W, et al (2021) Prioritized experience replay method based on experience reward. In: 2021 International Conference on Machine Learning and Intelligent Systems Engineering (MLISE), IEEE, pp 214–219, https://doi.org/10.1109/MLISE54096.2021.00045

Gu S, Lillicrap T, Sutskever I, et al (2016) Continuous deep q-learning with model-based acceleration. In: Proceedings of The 33rd International Conference on Machine Learning, PMLR, pp 2829–2838

Haarnoja T, Zhou A, Abbeel P, et al (2018) Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the 35th International Conference on Machine Learning, PMLR, pp 1861–1870

van Hasselt H (2010) Double Q-learning. In: Advances in Neural Information Processing Systems 23 (NIPS 2010), pp 2613–2621

van Hasselt HP, Hessel M, Aslanides J (2019) When to use parametric models in reinforcement learning? In: Wallach H, Larochelle H, Beygelzimer A, et al (eds) Advances in Neural Information Processing Systems, vol 32. Curran Associates, Inc.

Hausknecht M, Stone P (2015) Deep recurrent q-learning for partially observable mdps. In: AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15)

Hessel M, Modayil J, Van Hasselt H, et al (2018) Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, https://doi.org/10.1609/aaai.v32i1.11796

Horgan D, Quan J, Budden D, et al (2018) Distributed prioritized experience replay. In: International Conference on Learning Representations

Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Bach F, Blei D (eds) Proceedings of the 32nd International Conference on Machine Learning, vol 37. PMLR, pp 448–456

Jiang WC, Hwang KS, Lin JL (2021) An experience replay method based on tree structure for reinforcement learning. IEEE Transactions on Emerging Topics in Computing 9(2):972–982. https://doi.org/10.1109/TETC.2018.2890682

Łukasz Kaiser, Babaeizadeh M, Miłos P, et al (2020) Model-based reinforcement learning for atari. In: International Conference on Learning Representations

Kaiser L, Babaeizadeh M, Milos P, et al (2019) Model-based reinforcement learning for atari. arXiv preprint arXiv:190300374 https://doi.org/10.48550/arXiv.1903.00374

Kang C, Rong C, Ren W, et al (2021) Deep deterministic policy gradient based on double network prioritized experience replay. IEEE Access 9:60,296–60,308. https://doi.org/10.1109/ACCESS.2021.3074535

Kapturowski S, Ostrovski G, Quan J, et al (2019) Recurrent experience replay in distributed reinforcement learning. In: International Conference on Learning Representations

Kim M, Han DK, Park JH, et al (2020) Motion planning of robot manipulators for a smoother path using a twin delayed deep deterministic policy gradient with hindsight experience replay. Applied Sciences 10(2):575. https://doi.org/10.3390/app10020575

Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: Bengio Y, LeCun Y (eds) 3rd International Conference on Learning Representations

Kong SH, Nahrendra IMA, Paek DH (2021) Enhanced off-policy reinforcement learning with focused experience replay. IEEE Access 9:93,152–93,164. https://doi.org/10.1109/ACCESS.2021.3085142

Koroglu Y, Sen A (2022) Fast witness generation for readable gui test scenarios via generalized experience replay. IEEE Access 10:116,224–116,240. https://doi.org/10.1109/ACCESS.2022.3218902

Kumar A, Gupta A, Levine S (2020) Discor: Corrective feedback in reinforcement learning via distribution correction. In: Larochelle H, Ranzato M, Hadsell R, et al (eds) Advances in Neural Information Processing Systems, vol 33. Curran Associates, Inc., pp 18,560–18,572

Li C, Li Y, Zhao Y, et al (2021) Sler: Self-generated long-term experience replay for continual reinforcement learning. Applied Intelligence 51(1):185–201. https://doi.org/10.1007/s10489-020-01786-1

Li M, Huang T, Zhu W (2022a) Clustering experience replay for the effective exploitation in reinforcement learning. Pattern Recognition 131. https://doi.org/10.1016/j.patcog.2022.108875

Li Y, Ji J (2021) Parallel curriculum experience replay in distributed reinforcement learning. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems, p 782–789

Li Y, Aghvami AH, Dong D (2022b) Path planning for cellular-connected UAV: A DRL solution with quantum-inspired experience replay. IEEE Transactions on Wireless Communications 21(10):7897–7912. https://doi.org/10.1109/TWC.2022.3162749

Lillicrap TP, Hunt JJ, Pritzel A, et al (216) Continuous control with deep reinforcement learning. In: International Conference on Representation Learning

Lin LJ (1992) Self-improving reactive agents based on reinforcement learning, planning, and teaching. Machine Learning 8(3-4):293–321. https://doi.org/10.1007/BF00992699

Liu R, Zou J (2018) The effects of memory replay in reinforcement learning. In: 2018 56th Annual Allerton Conference on Communication, Control, and Computing, pp 478–485, https://doi.org/10.1109/ALLERTON.2018.8636075

Liu X, Zhu T, Jiang C, et al (2022) Prioritized experience replay based on multi-armed bandit. Expert Systems with Applications 189:116,023. https://doi.org/10.1016/j.eswa.2021.116023

Ma J, Ning D, Zhang C, et al (2022) Fresher experience plays a more important role in prioritized experience replay. Applied Sciences 12(23):12,489. https://doi.org/10.3390/app122312489

Machado MC, Bellemare MG, Talvitie E, et al (2018) Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. Journal of Artificial Intelligence Research 61:523–562. https://doi.org/10.1613/jair.5699

Manela B, Biess A (2021) Bias-reduced hindsight experience replay with virtual goal prioritization. Neurocomputing 451:305–315. https://doi.org/10.1016/j.neucom.2021.02.090

Manela B, Biess A (2022) Curriculum learning with hindsight experience replay for sequential object manipulation tasks. Neural Networks 145:260–270. https://doi.org/10.1016/j.neunet.2021.10.011

Mnih V, Kavukcuoglu K, Silver D, et al (2013) Playing atari with deep reinforcement learning. In: Deep Learning Workshop NIPS 2013

Mnih V, Kavukcuoglu K, Silver D, et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533. https://doi.org/10.1038/nature14236

Mnih V, Badia AP, Mirza M, et al (2016) Asynchronous methods for deep reinforcement learning. In: Balcan MF, Weinberger KQ (eds) Proceedings of The 33rd International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 48. PMLR, pp 1928–1937

Moreno-Vera F (2019) Performing deep recurrent double q-learning for atari games. In: 2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI), IEEE, pp 1–4

Neves DE, Ishitani L, do Patrocínio Júnior ZKG (2022) When less may be more: Exploring similarity to improve experience replay. In: Intelligent Systems: 11th Brazilian Conference, BRACIS 2022, Part II, Springer, pp 96–110, https://doi.org/10.1007/978-3-031-21689-3_8

Nicholaus IT, Kang DK (2022) Robust experience replay sampling for multi-agent reinforcement learning. Pattern Recognition Letters 155:135–142. https://doi.org/10.1016/j.patrec.2021.11.006

Novati G, Koumoutsakos P (2019) Remember and forget for experience replay. In: Chaudhuri K, Salakhutdinov R (eds) Proceedings of the 36th International Conference on Machine Learning, vol 97. PMLR, pp 4851–4860

Osei RS, Lopez D (2023) Experience replay optimisation via ATSC and TSC for performance stability in Deep RL. Applied Sciences 13(4):2034. https://doi.org/10.3390/app13042034

Prianto E, Kim M, Park JH, et al (2020) Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay. Sensors 20(20):5911. https://doi.org/10.3390/s20205911

Remman SB, Lekkas AM (2021) Robotic lever manipulation using hindsight experience replay and shapley additive explanations. In: 2021 European Control Conference (ECC), IEEE, pp 586–593, https://doi.org/10.23919/ECC54610.2021.9654850

Rolnick D, Ahuja A, Schwarz J, et al (2019) Experience replay for continual learning. Advances in Neural Information Processing Systems 32

Rosenbauer L, Stein A, Pätzel D, et al (2020) XCSF with experience replay for automatic test case prioritization. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, pp 1307–1314, https://doi.org/10.1109/SSCI47803.2020.9308379

Schaul T, Quan J, Antonoglou I, et al (2016) Prioritized experience replay. In: Proceedings of the International Conference on Representation Learning

Schmitt S, Hessel M, Simonyan K (2020) Off-policy actor-critic with shared experience replay. In: III HD, Singh A (eds) Proceedings of the 37th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 119. PMLR, pp 8545–8554

Silver D, Lever G, Heess N, et al (2014) Deterministic policy gradient algorithms. In: Xing EP, Jebara T (eds) Proceedings of the 31st International Conference on Machine Learning, vol 32. PMLR, Bejing, China, pp 387–395

Sinha S, Song J, Garg A, et al (2022) Experience replay with likelihood-free importance weights. In: Firoozi R, Mehr N, Yel E, et al (eds) Proceedings of The 4th Annual Learning for Dynamics and Control Conference, vol 168. PMLR, pp 110–123

Sovrano F, Raymond A, Prorok A (2022) Explanation-aware experience replay in rule-dense environments. IEEE Robotics and Automation Letters 7(2):898–905. https://doi.org/10.1109/LRA.2021.3135927

Sun P, Zhou W, Li H (2020) Attentive experience replay. Proceedings of the AAAI Conference on Artificial Intelligence 34(04):5900–5907. https://doi.org/10.1609/aaai.v34i04.6049

Sutton RS (1992) Reinforcement learning architectures. In: Proceedings ISKIT'92 International Symposium on Neural Information Processing

Sutton RS, Barto AG (2018) Reinforcement Learning: An Introduction, 2nd edn. The MIT Press, Cambridge

Szepesvári C (2010) Algorithms for Reinforcement Learning, vol 4. Morgan & Claypool Publishers, https://doi.org/10.2200/S00268ED1V01Y201005AIM009

Tao X, Hafid AS (2020) Deepsensing: A novel mobile crowdsensing framework with double deep q-network and prioritized experience replay. IEEE Internet of Things Journal 7(12):11,547–11,558. https://doi.org/10.1109/JIOT.2020.3022611

Tassa Y, Doron Y, Muldal A, et al (2018) Deepmind control suite. arXiv preprint arXiv:180100690 abs/1801.00690. https://doi.org/10.48550/arXiv.1801.00690

Todorov E, Erez T, Tassa Y (2012) Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 5026–5033, https://doi.org/10.1109/IROS.2012.6386109

Uhlenbeck GE, Ornstein LS (1930) On the theory of the brownian motion. Phys Rev 36:823–841. https://doi.org/10.1103/PhysRev.36.823

Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, https://doi.org/10.1609/aaai.v30i1.10295

Vecchietti LF, Seo M, Har D (2022) Sampling rate decay in hindsight experience replay for robot control. IEEE Transactions on Cybernetics 52(3):1515–1526. https://doi.org/10.1109/TCYB.2020.2990722

Wang B, Zhao D, Cheng J (2019) Adaptive cruise control via adaptive dynamic programming with experience replay. Soft Computing 23(12):4131 – 4144. https://doi.org/10.1007/s00500-018-3063-7

Wang C, Ross KW (2019) Boosting soft actor-critic: Emphasizing recent experience without forgetting the past. arXiv preprint arXiv:190604009 abs/1906.04009. https://doi.org/10.48550/arXiv.1906.04009

Wang Z, Schaul T, Hessel M, et al (2016) Dueling network architectures for deep reinforcement learning. In: Balcan MF, Weinberger KQ (eds) Proceedings of The 33rd International Conference on Machine Learning, vol 48. PMLR, pp 1995–2003

Watkins CJCH, Dayan P (1992) Q-learning. Machine Learning 8(3):279–292. https://doi.org/10.1007/BF00992698

Wei Q, Ma H, Chen C, et al (2022) Deep reinforcement learning with quantum-inspired experience replay. IEEE Transactions on Cybernetics 52(9):9326–9338. https://doi.org/10.1109/TCYB.2021.3053414

Werbos P (1990) Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78(10):1550–1560. https://doi.org/10.1109/5.58337

Wu DF, Huang C, Yin Y, et al (2022) State aware-based prioritized experience replay for handover decision in 5g ultradense networks. Wireless Communications and Mobile Computing 2022. https://doi.org/10.1155/2022/5006770

Yang J, Peng G (2021) DDPG with meta-learning-based experience replay separation for robot trajectory planning. In: 2021 7th International Conference on Control, Automation and Robotics (ICCAR), pp 46–51, https://doi.org/10.1109/ICCAR52225.2021.9463493

Yang R, Wang D, Qiao J (2022) Policy gradient adaptive critic design with dynamic prioritized experience replay for wastewater treatment process control. IEEE Transactions on Industrial Informatics 18(5):3150–3158. https://doi.org/10.1109/TII.2021.3106402

Yang X, He H (2020) Adaptive critic learning and experience replay for decentralized event-triggered control of nonlinear interconnected systems. IEEE Transactions on Systems, Man, and Cybernetics: Systems 50(11):4043–4055. https://doi.org/10.1109/TSMC.2019.2898370

Yarats D, Kostrikov I, Fergus R (2021) Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In: International Conference on Learning Representations

Yu L, Huo S, Wang Z, et al (2023) Hybrid attention-oriented experience replay for deep reinforcement learning and its application to a multi-robot cooperative hunting problem. Neurocomputing 523(C):44–57. https://doi.org/10.1016/j.neucom.2022.12.020

Zha D, Lai KH, Zhou K, et al (2019) Experience replay optimization. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization, pp 4243–4249, https://doi.org/10.24963/ijcai.2019/589

Zhang H, Qu C, Zhang J, et al (2020) Self-adaptive priority correction for prioritized experience replay. Applied Sciences 10(19):6925. https://doi.org/10.3390/app10196925

Zhang S, Sutton RS (2017) A deeper look at experience replay. In: 31st Conference on Neural Information Processing Systems (NIPS 2017)

Zhang Y, Qiu H (2022) DDQN with prioritized experience replay-based optimized geographical routing protocol of considering link stability and energy prediction for uanet. Sensors 22(13):5020. https://doi.org/10.3390/s22135020

Zhou C, Huang B, Hassan H, et al (2022) Attention-based advantage actor-critic algorithm with prioritized experience replay for complex 2-d robotic motion planning. Journal of Intelligent Manufacturing 34(1):151–180. https://doi.org/10.1007/s10845-022-01988-z