PONTIFICAL CATHOLIC UNIVERSITY OF MINAS GERAIS

Graduate Program in Informatics

Daniel Eugênio Neves

WHEN LESS MAY BE MORE: EXPLORING SIMILARITY TO IMPROVE EXPERIENCE REPLAY IN REINFORCEMENT LEARNING

Belo Horizonte

2024

Daniel Eugênio Neves

WHEN LESS MAY BE MORE: EXPLORING SIMILARITY TO IMPROVE EXPERIENCE REPLAY IN REINFORCEMENT LEARNING

Thesis presented to the Graduate Program in Informatics of the Pontifical Catholic University of Minas Gerais, as a partial requirement for obtaining the title of Doctor in Informatics.

Advisor: Prof. Dr. Zenilton Kleber Gonçalves do Patrocínio Júnior

Co-advisor: Profa. Dra. Lucila Ishitani

Belo Horizonte

2024

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Neves, Daniel Eugênio
N518w When less may be more: exploring similarity to improve experience replay in reinforcement learning / Daniel Eugênio Neves. Belo Horizonte, 2024.
175 f. : il.

Orientador: Zenilton Kleber Gonçalves do Patrocínio Júnior
Coorientadora: Lucila Ishitani
Tese (Doutorado) – Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática

1. Inteligência artificial. 2. Similaridade (ciência da informação). 3. Redes neurais (Computação). 4. Aprendizado do computador. 5. Algoritmos. 6. Markov, Processos de. 7. Processo estocástico. I. Patrocínio Júnior, Zenilton Kleber Gonçalves do. II. Ishitani, Lucila. III. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática. IV. Título.

CDU: 681.3.092
Ficha catalográfica elaborada por Fabiana Marques de Souza e Silva - CRB 6/2086

Daniel Eugênio Neves

WHEN LESS MAY BE MORE: EXPLORING SIMILARITY TO IMPROVE EXPERIENCE REPLAY IN REINFORCEMENT LEARNING

Thesis presented to the Graduate Program in Informatics of the Pontifical Catholic University of Minas Gerais, as a partial requirement for obtaining the title of Doctor in Informatics.

Prof. Dr. Zenilton Kleber Gonçalves do Patrocínio Júnior – PUC Minas
(Advisor)

Profa. Dra. Lucila Ishitani – PUC Minas
(Co-advisor)

Prof. Dr. Luis Enrique Zárate – PUC Minas
(Board Member)

Profa. Dra. Cristiane Neri Nobre – PUC Minas
(Board Member)

Prof. Dr. Luís Fabrício Wanderley Góes – University of Leichester
(Board Member)

Prof. Dr. Ricardo Poley Martins Ferreira – UFMG
(Board Member)

Belo Horizonte, December 20, 2024

From the first steps to this great leap in my life, for their example and guidance, I dedicate this work to my parents. From the first glance exchanged to the trust in companionship and love that I carry with me to this day, I dedicate this work to my beloved wife. From the first games through all the adventures, laughter, and difficulties shared, I dedicate this work to every friend I had and still have. For every letter on the paper, for every curiosity awakened, for the excitement of learning and every dream of growth realized, for their passion and dedication to their craft, I dedicate this work to the good teachers who have inspired me throughout these 47 years of paths traveled.

ACKNOWLEDGMENTS

Even in the loneliest moments, I never felt completely alone, and I thank God for that. My faith and hope strengthened my resilience.

In the most difficult moments, I never lacked strength, courage, and perseverance because my parents taught me these things through words and examples, and I am eternally grateful to them for that. Father and mother, all my love to you.

At every moment of doubt or weakness, my beloved wife was my shining light. My love, simply thinking of you brings back my sparkle because there is no inspiration in my life greater than you.

To my admired advisors, mentors, and professors of many years, Zenilton and Lucila, how could I not express my immense gratitude, respect, and deep affection for you? Thank you very much for everything.

My dear friends, who shared precious moments of joy, confidences, and encouragement, were my sigh and breath, you are fantastic.

I would also like to thank the Coordination for the Improvement of Higher Education Personnel (CAPES) and the National Council for Scientific and Technological Development (CNPq) for their financial support and encouragement of research, which is necessary for technological and scientific innovations.

Some people struggle for a day and are good;
Others struggle for a year and are better;
Some struggle for many years and are very good;
But there are those who struggle for their whole lives, and these are indispensable.

Bertold Brecht

ABSTRACT

Experience Replay (ER) improves data efficiency in Deep Reinforcement Learning by allowing the agent to revisit past experiences that could contribute to the current policy learning. However, ER has some limitations regarding reusing past experiences whose improvements have been recurrent in the literature. This research proposes and evaluates the COMPact Experience Replay (COMPER) and the COMPact Memory for Continuous ACTions (COMCACT), which are two model-free and off-policy reinforcement learning methods that seek to improve ER data efficiency and reduce the required number of experiences for agent training regarding the total accumulated rewards in the long run. They use temporal difference learning (TD-learning) with predicted target values for sets of similar transitions and a new experience replay approach based on a compact memory of transitions and slight changes in the loss function. These methods can approximate good policies from a considerably smaller number of observations of the environment to achieve similar results obtained by essential methods in the literature, which generally use millions of observations and agent training steps, both in problems with discreet and continuous actions space. The main hypotheses behind this work are two-fold. First, it is possible to produce sets of similar transitions and explore them to build a reduced memory, making successive updates of their Q-values to learn their dynamics through a deep neural network (DNN) and use this DNN to approximate the target value at the TD-learning; this value should improve the update of the Q-value function. Second, using a compact memory augments the odds of a rare transition being observed compared to a sampling performed on a large FIFO-like buffer, making updates of the value function more effective. We present an assessment of two possible neural network architectures for the target function with a complete analysis of the memories' behavior. To evaluate our propositions, we present detailed results for 100,000 video frames and about 25,000 iterations with a small experience memory on eight challenging 2600 Atari games on the Arcade Learning Environment (ALE) and compare these results with those obtained by a Deep Q-Network (DQN) agent with the same experimental protocol on the same set of games used as a baseline. We also conduct a detailed analysis of the COMPER's components to verify and understand how they operate and how they could improve data efficiency. COMCACT share the same experience mechanisms of COMPER but adapted for dealing with continuous action space. We evaluate it on problems of physical control of multijoint humanoids on the set of Mujoco's environments and in tasks of classical pendulum control. We present detailed results from running only 50,000 iterations in all the tasks of these environments and compare them with those obtained by a Deep Deterministic Policy Gradient (DDPG) agent. Finally, we demonstrate that (i) both

COMPER and COMCACT can approximate good action policies from a small number of frame observations or training iteration steps using a compact memory and that (ii) they achieve better or at least equivalent results than other methods in the literature, which use considerable more environment observations.

Keywords: Deep reinforcement learning, data-efficient experience replay, similar transition sets, compact transitions memories, recurrence on target value prediction.

RESUMO

O método Experience Replay (ER) melhora a eficiência dos dados no Aprendizado por Reforço Profundo ao permitir que o agente revisite experiências passadas que poderiam contribuir para o aprendizado da política atual. No entanto, o ER possui algumas limitações quanto à forma de armazenamento e amostragem de experiências cujas melhorias têm sido recorrentes na literatura. Esta pesquisa propõe e avalia o COMPact Experience Replay (COMPER) e o COMPact Memory for Continuous ACTions (COMCACT), que são dois métodos de aprendizado por reforço livre de modelo (*model free*) e independente da política (*off policy*). Estes métodos buscam melhorar a eficiência no uso dos dados para repetição de experiências e reduzir o número necessário de experiências para o treinamento do agente. Eles utilizam o aprendizado por diferença temporal (*TD-learning*) com valores alvo preditos para conjuntos de transições similares e uma nova abordagem de repetição de experiências baseada em uma memória compacta de transições e pequenas mudanças na função de perda. Esses métodos podem aproximar boas políticas a partir de um número consideravelmente menor de observações do ambiente para atingir resultados semelhantes aos obtidos por métodos essenciais na literatura, que geralmente requerem milhões de observações e passos de treinamento do agente, tanto em problemas com espaço de ações discretas quanto contínuas. As principais hipóteses investigadas neste trabalho são duas. Primeiro, é possível produzir conjuntos de transições similares e explorá-los para construir uma memória reduzida, fazendo atualizações sucessivas de seus valores Q para aprender sua dinâmica por meio de uma rede neural profunda (DNN) e usar esta DNN para aproximar o valor alvo para o *TD-learning*; este valor deve melhorar a atualização da função de valor Q. Segundo, usar uma memória compacta aumenta as chances de uma transição rara ser observada em comparação com uma amostragem realizada a partir de um grande buffer do tipo FIFO (*first in, first out*), tornando as atualizações da função de valor mais eficazes. Apresentamos uma avaliação de duas possíveis arquiteturas de rede neural para a função alvo com uma análise completa do comportamento das memórias. Para avaliar nossas proposições, apresentamos resultados detalhados para 100.000 quadros de vídeo e cerca de 25.000 iterações com uma pequena memória de experiências em oito jogos desafiadores do Atari 2600 no emulador Arcade Learning Environment (ALE) e comparamos esses resultados com aqueles obtidos por um agente Deep Q-Network (DQN) com o mesmo protocolo de experimentos no mesmo conjunto de jogos, que é utilizado como linha de base. Também conduzimos uma análise detalhada dos componentes do COMPERpara verificar e entender como eles operam e como podem melhorar a eficiência dos dados. O COMCACT compartilha os mesmos mecanismos de experiências do COMPER mas adaptados para lidar com espaços

de ações contínuos. Nós o avaliamos em problemas de controle físico de humanoides multiarticulares no conjunto de ambientes do Mujoco e em tarefas clássicas de controle de pêndulos. Apresentamos resultados detalhados da execução de apenas 50.000 iterações em todas as tarefas desses ambientes e os comparamos com aqueles obtidos por um agente Deep Deterministic Policy Gradient (DDPG). Finalmente, demonstramos que (i) tanto COMPER quanto COMCACT podem aproximar boas políticas de ação a partir de um pequeno número de observações de quadros ou etapas de iteração de treinamento usando uma memória compacta e que (ii) eles alcançam resultados melhores ou pelo menos equivalentes aos resultados de outros métodos na literatura, que usam consideravelmente mais observações do ambiente.

Palavras-chave: Aprendizado por reforço profundo, repetição de experiência com eficiência de dados, conjuntos de transições semelhantes, memórias de transições compactas, recorrência na predição do valor alvo.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ACL Adaptive Critic Learning

ADP Adaptive Dynamic Programming

ACER Asynchronous Curriculum Experience Replay

AER Attentive Experience Replay

A3C Asynchronous Advantage Actor-Critic

ADAM Adaptive Moment Estimation

AHC *Adaptive Heuristic Critic*

AHCON *Connectionist AHC-learning*

AHCON-M AHCON using Actions Models

AHCON-R AHCON using *Experience Replay*

AHCON-T AHCON using *Experience replay* and *Teaching*

AHP Analytic Hierarchy Process

ALE *Arcade Learning Environment*

ARCHER Aggressive Rewards to Counter Bias in HER

ARIMA Autoregressive Integrated Moving Average

BPTT Backpropagation Over Time

CLEAR Continual Learning with Experience Replay

COMCACT *COMP*act Memory for *C*ontinous *ACT*ions

COMPER ***COMP****act* ***E****xperience* ***R****eplay*

CNN Convolutional Neural Networks

CTER Curiosity-tuned Experience Replay

DBER Double Bias Experience Replay

DDPG Deterministic Policy Gradient

DDQN *Double Deep Q-Networks*

DisCor Distribution Correction

DNN Deep Neural Network

DCS DeepMind Control Suite

DP Dynamic Programming

DPER-VDP3G Diffusion Prior Driven Neural Representation

DPG Deterministic Policy Gradient

DQN *Deep Q-Networks*

DrQ Data Regularized Q

DRQN *Deep Recurrent Q-Network*

ER *Experience Replay*

ERO Experience Replay Optimization

FIFO First in, first out

FIOU First-In, Useless-Out

HAER-MADDPG Hybrid Attention Module Multi-agent DDPG

HER Hindsight Experience Replay

LAP Loss-adjusted Prioritized Experience Replay

LSTM *Long Short-Term Memory*

MDP *Markov Decision Process*

MLP Multi-layer Perceptron

MSE Mean Squared Error

MSER Metalearning-Based Experience Replay

off-PG off-policy Policy Gradients

PAL Prioritized Approximation Loss

PEPCRL-MVP Progression Cognition Reinforcement Learning with Prioritized Experience for Multi-Vehicle Pursuit

PER Prioritized Experience Replay

TM Transitions Memory

POMDP *Partially-Observable Markov Decision Process*

QCON *Connectionist Q-learning*

QCON-M QCON whit Actions Model

QCON-T QCON whith *Experience Replay* and *Teaching*

QCON-R QCON with *Experience Replay*

QR-DQN Quantile Regression DQN

QRNN *Q-Recurrent Neural Network*

R2D2 Recurrent Replay Distributed DQN

ReF-ER Remember and Forget Experience Replay

RL Reinforcement Learning

RMSProp Root Mean Square Propagation

RNN *Recurrent Neural Network*

RTM Reduced Transitions Memory

SAC Soft Actor-Critic

SLER Self-generated Long-term Experience Replay

SPER Segmented Prioritized Experience Replay

ST Simmilar Transitions

TD Temporal Difference

TDE *Temporal Difference Error*

TD-Learning *Temporal Difference Learning*

TD3 Twin Delayed Deep Deterministic Policy Gradient

TF *Train Frequency*

TM Transition Memory

TMI Transition Memory Index

UAVs Unmanned Aerial Vehicles

UANET UAV Ad Hoc Network

UF *Update Frequency*

XAER Explanation-Awareness Experience Replay

UOS-RL Collaborative Multi-Agent RL

SARSA State-Action-Reward-State-Action

CONTENTS

1 INTRODUCTION

In common sense, “learning” can refer to an individual’s ability to do something new based on their discoveries or to do something better based on their experience, which increases throughout their learning process. In this sense, learning is closely related to intelligence, and an intelligent individual is said to be capable of learning. In the history of science, some have dedicated themselves to understanding these concepts in light of the human and biological sciences. At the same time, there are those who one day imagined machines capable of learning and, therefore, intelligent machines. Although not biological individuals, these machines would be endowed with “artificial intelligence” and, to perform complex tasks and learn through their experience, would then need a “machine learning” learning process.

Machine Learning is an area of artificial intelligence research within Computer Science in which mathematical and statistical models applied to computational algorithms seek to provide machines with intelligent behavior and the ability to learn from their experiences. With many areas of application in industry and academia, the different machine learning methods are capable of addressing complex problems, such as natural language understanding, pattern recognition and computer vision, control in automation and robotics systems, data analysis and decision-making support, information retrieval, generation of multimedia content from examples and feedback, among many others.

In particular, this work deepens into a class of machine learning methods called Reinforcement Learning, in which a computational entity, called an agent, seeks to learn the best way to interact with a given environment through actions to achieve an objective. Each action affects the environment, and the agent observes this effect to evaluate how good or bad this action was or how good its choices of actions could be in the long run to bring it closer or further away from its objective.

Even after decades of research and advances, there are still many questions to answer and many limitations to overcome in this class of methods. Some limitations are critical while the problems to address and their applications become increasingly complex. One example is the time required to train an agent, which is directly related to convergence capacity to a good action policy. If the demand for long training runs can limit the application of reinforcement learning methods and harm their effectiveness, how to effectively reduce this time is an important question.

1.1 Context and Motivation

Interaction with the environment permeates human learning throughout life, at its different stages, enabling persons to learn from the simplest things, such as crawling and playing, to the most complex, such as developing a dialogue, defending a thesis, investigating a phenomenon, or discussing the learning process itself. From this perspective, interactions occur essentially through the actions of the persons inserted in the environment, whose effects, as well as its consequences, produce a range of information about both the persons and the environment, which they can use to evaluate their subsequent actions and conduct them towards one or more objectives, whether it is walking from one point to another or convincing someone about an idea. Similarly, interaction with the environment is an essential factor in learning for the animals, which can involve simpler mechanisms and objectives, such as feeding, to complex social relationships involving communication and hierarchical organization, for example. Therefore, it is clear that, in one way or another, interaction permeates the learning process of any individual who needs to learn to relate to the environment and to make good decisions, primarily based on the results of this interaction.

While the concept of "learning" can be closely related to the understanding of what "intelligence" is, interacting, observing, making decisions, and acting towards a goal are all factors inherent to what can be called the "learning process". In this sense, the interesting point is that in this process, as highlighted by Sutton and Barto (2018), there is no explicit need for a teacher's intervention. Thus, focusing on learning obtained through interaction with the environment, reinforcement learning methods are inspired by the fundamental ideas that guide many theories about intelligence and learning. They address problems related to "learning to act" and go beyond supervised learning models, with examples of expected inputs and outputs, or unsupervised learning, which are directly dependent on data sets, to address problems related to learning complex behaviors, such as locomotion, decision making, environments exploration, interaction between agents and planning.

The challenges to advancing the state-of-the-art in reinforcement learning are numerous, and many problems still require better solutions or even some solution, as seen in the literature. While scientific production on the subject has increased exponentially in the last decade, the understanding of human cognitive processes has also significantly advanced in its research areas since the works of the mid-20th century, which propositions inspired reinforcement learning in its first formulations up to the present day. Thus, developing algorithms and mathematical models inspired by these propositions are far from exhausting their potential for development and discovery in the face of increasingly complex problems to solve in the real world.

In this sense, it is possible to formulate some intriguing questions. For example,

relevant works in the literature proposed using a technique called Temporal Difference Learning to adjust the agent's action policies approximation models. Later, the fundamental formulations presented by Lin (1992), adopted in several other works of equal relevance, showed that the replay of the agent's experiences and their use to adjust the value function models from the temporal difference error contributes to accelerating its learning and reducing its convergence time to an optimal action policy. Given this, one could ask: could there be, beyond merely repeating experiences drawn from a buffer under some weighting or probabilities, some relationship or pattern within these experiences that would enable projecting one step ahead? This would allow for predictions that are not just based on mathematical expectations but instead identify which experiences might contribute most effectively to enhancing the agent's learning—not in the moment of sampling but in the near future. This question can be based on the impression that every human being has that they will become capable of dealing better with possible future situations, even if they have not yet experienced them, due to their experience with similar situations. Moreover, this ability seems to be related not only to patterns of events but also to patterns of successive events over time.

The above exercise motivated this research, which investigates some interesting hypotheses about the mechanisms of experience replay and its effects on agent learning and demonstrates that, like humans, reinforcement learning agents, which learn from past experiences, learn more from similar past experiences.

1.2 Research Problem

Reinforcement Learning (RL) research is dedicated to developing intelligent computational entities called agents that 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 value, 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.

After an agent has performed a sequence of actions and received a reward, knowing how to assign credit (or discredit) to each state-action pair consists of a complex problem called *Temporal Credit Assignment* (LIN, 1992). 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; DAYAN, 1992), which represent the first TD-learning-based methods in RL, are characterized by high convergence times. Therefore, since the assignment of temporal credit is related to the agent's effectiveness, more efficient methods

for approaching this problem can also reduce the convergence time in the training of these agents, which, in turn, is a critical factor for the application (and democratization) of reinforcement learning in increasingly complex problems.

Lin (1992) proposed an important and effective technique called Experience Replay (ER) to accelerate the credit attribution process and reduce the convergence time by storing the agent's experiences in a replay buffer and uniformly sampling past experiences to update the agent's value functions used to estimate the expected reward. One of its main motivations is that relevant algorithms become inefficient when they use trial-and-error experiences only once to make updates and soon discard them. This is because some agents' experiences can be rare and potentially produce relevant updates, while others can be expensive, such as those involving penalties. Moreover, using ER with random sampling reduces the effect of the correlation of the data (which represents the environment states) and improves its nonstationary 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 generate considerable changes in the policy learned by the agent (MNIH et al., 2015). So, it can change the data distribution and the relation between the action-value functions in calculating the Temporal Difference Error (TD error) used to update the agent's models.

Lin (1992) compared Experience Replay with two other techniques called *Learning Action Models for Planning* and *Teaching*, and evaluated these three techniques applied to eight reinforcement learning methods, namely AHC, *Q-learning*, and three extensions of each. Based on the results, the author demonstrated how Experience Replay is an effective method to accelerate the credit assignment process and, consequently, reduce convergence time, outperforming the other methods in all scenarios in which the agent needs to learn the best action policy by itself without relying on a model of the environment's dynamics and without the intervention of a "teacher" who provides examples of how to act.

However, Experience Replay has some limitations. As it repeats experiences through uniform sampling, if these samples define policies that are very different from what the agent is learning, it can underestimate its value functions. This affects methods that use neural networks because whenever the weights for a given state are adjusted, 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 size of the replay buffer. Using an arbitrary and fixed replay buffer size implies a considerable potential for sampling old

experiences that are distant from the current policy of actions in the case of large buffers or introducing bias in very small buffers, as discussed in the recent literature (ZHANG; SUTTON, 2017; LIU; ZOU, 2018).

In summary, ER has three main limitations: (i) the memory of experiences does not differentiate relevant experiences due to the uniform sampling; (ii) experiences are often overwritten due to the buffer size limitation, leading to the loss of potentially relevant experiences; and (iii) the arbitrally and fixed size of the buffer makes ER less data-efficient in reusing experiences to update the value functions. From that, it is possible to identify three exciting challenges: (i) the picking of what experiences to repeat, (ii) the selection of which one to store, and (iii) how to deal with the threshold regarding buffer sizing—the more prominent researches in the literature regarding improving ER proposed solutions for the first challenge. One of the pioneers was Schaul et al. (2016), which presented an effective method to prioritize experiences to replay. On the other hand, motivated by the reflections presented in Section 1.1, this work addresses the last two challenges, seeking to overcome the limitations regarding the fixed buffer size and make ER more data-efficient by avoiding losing information (regarding expected reward and value function updates) from rare and expensive experiences which could be discarded or simple not sampled for replay. Therefore, it was necessary to define a strategy to store and organize the experiences so that it is possible to estimate the potential future reward based on their entire current set before sampling or discarding them.

1.3 Questions and Hypotheses

Considering that there is a trade-off between data quality and data correlation, smaller replay buffers with a fixed size make data fresher (i.e., close to the current action policy) but highly temporal correlated. They are also biased and could make revisiting rare or expensive experiences even less likely, mainly if they could be more relevant in the future than when the agent experienced them. At the same time, neural networks often need independent and identically distributed (i.i.d.) data. On the other hand, experiences sampled from larger replay buffers tend to be uncorrelated but more likely to be outdated (i.e., very distant from the current action policy). Moreover, besides its capacity to retain rare or expensive experiences for a long time, it reduces the probability of revisiting these experiences under a uniform sample on a large set of diverse experiences.

Therefore, a replay buffer adopting a FIFO strategy (i.e., as a queue) with a fixed arbitrary size will impact agent learning. However, suppose we assume that (i) different experiences from many stochastic episodes carried out over a nondeterministic environment will be stored and sampled many times from a compact replay buffer but not explicitly size-limited, and (ii) that sample size is the unique fixed and ever minor than

the buffer size. In that case, this buffer will start with size one (i.e., the size corresponding to one experience, the first one) and grow as the agent experiences new actions and new state transitions in the environment. From that, suppose we (i) sample and remove sets of experiences from time to time, reducing that incrementally-sized buffer on the sample size (ii) but not discarding the set sampled before learning the implicit dynamics of its contained experiences regarding the variations on the expected long-run reward estimated at the time every single experience was observed. Finally, suppose (iii) we do not want to store new batches of experiences, always of the same size, generated periodically under a given number of agent interactions in the same queue as all other previous experiences, as done in the literature that uses the traditional replay buffer, because it can arbitrarily increase the buffer by inserting experiences that are the same or very similar to previously stored ones. Instead, (iv) we could define no more a buffer but a memory of experiences, composed by sets of similar experiences (intuitively like in humans), and store the ones for which there is no similar experience (in the memory) in a new unitary set for each, together with their respective estimations of the long-run reward. Regarding the others whose similar experiences already exist in some set, (v) we could insert them in that corresponding set, together with their respective estimations of long-run rewards, and use only one experience to represent a set of similar experiences and an estimation of the long-run reward learned from the dynamics of the set.

Considering the above suppositions, we could think about a compact memory of experiences, which is capable, at the same time, of (i) being fresher but not so biased, (ii) incrementally growing but not linearly at the number of interactions of the agent and (iii) not so large that faster makes lesser likely to resemble rare and expensive experiences and make possible to learning its dynamics to improve the estimations of the long-run reward used to update the agents' value functions.

Therefore, the main hypotheses behind this work are two-fold. First, it is possible to produce sets of similar transitions and explore them to build a reduced memory. One can also perform successive updates of their Q-values, learning their dynamics through a deep neural network (DNN) and using this DNN to approximate the target value at the TD-learning. Second, it augments the odds of a rare transition being observed compared to a sampling performed on the entire set, making updates of the value function more effective.

To support these main hypotheses, we derivate them as follows:

Question 1 – *Is it possible that the iterative training process of an agent with reinforcement learning, which uses temporal difference learning, leads to the repetition of state transitions but with different return values and associated with varying estimates of the value function? Thus, would it be possible to define sets of similar transitions based*

on the state and action components of their tuples and establish some relationship or a pattern of behavior between the transitions of a given set, in addition to the similarity relationship, so that one could explore it to evaluate this set as a whole and produce a single transition that represents all the transitions contained in it?

Training an agent with temporal difference learning in a non-deterministic environment may lead to repeating transitions or observing similar transitions, with different estimations of the expected long-term reward value over consecutive training episodes in the episodic tasks (each episode starts in the initial state and finishes when the game is over at a final state, so the task restarts and another episode is carried, and so on) and at least similar transitions in the continuous tasks (which consist in iterative processes anyway, over the same environment and possible actions set). From that, consider that a transition at a time step t is represented by a tuple (s_t, a_t, r_t, s_{t+1}) and the long-term expected reward estimated by the agent's value function is represented by Q_t. From that, s_t is the current environment's state, and a_t is the action the agent chose from its observation of s_t; r_t is an immediate reward value the agent received for taking a_t on s_t; and s_{t+1} is the resulting state (i.e., by taking a_t on s_t, the agent causes a state transition of the environment to s_{t+1}). As the agent learns from an iterative process, state-action pairs leading to the same or similar s_{t+1} may be evaluated more than once, composing tuples in which the elements s_t, a_t and s_{t+1} may be the same or similar between similar transitions, while r_t and Q_t will probably be different. Thus, similar transitions can be considered as those in which the tuples (s_t, a_t and s_{t+1}) are equal or similar, with differences only in the resulting values of r and Q. In this way, it is possible to compose a set of similar transitions associated with their respective values of r and Q. Therefore, if successive transitions are inserted into a set of similar transitions and related to the values of Q in the time step t in which they occur during subsequent iterations, that set can be represented by any of its transitions.

Hypothesis 1 – It is possible to learn the dynamics of the variations of Q over similar subsequent transitions to generate a model that, given a transition, estimates the resulting Q in the next step when a similar transition occurs. Therefore, the entire set of similar transitions and different Q can be represented and replaced by only one similar transition and its estimate of Q for a possible occurrence in the future.

Question 2 – *Is it possible to obtain a single set of transitions, smaller than the set of all transitions already experienced up to a step t of the learning process (as is in the traditional buffer) but capable of representing all of them so that this set is more informative for a value function updating step from the temporal difference error?*

Consider that (i) there are sets of transitions grouped as similar transitions whose immediate return values and the value function estimate associated with them are

variables, and (ii) these sets are contained in the set of all transitions ever performed. If there is a single transition obtained from each set of similar transitions, in which (i) the state and action elements are equal or similar to those of all similar transitions in its source set, (ii) the immediate reward value is approximated from the values in the set, and (iii) the estimated long-term reward value associated with it is predicted based on the analysis of the values of all similar transitions in its source set, then (i) a new set, composed by these unique transitions, will be smaller than the set of all transitions and, at the same time, (ii) will be able to represent it, since it will contain representations of all pairs of states and actions already observed, present in the tuples of transitions. Moreover, (iii) this smaller set will have a representation of all immediate return values and estimates of the value function, represented respectively by the approximated value for r' and by the predicted value para Q', which are associated with each unique transition obtained from their respective similar transitions sets and that represent it. Thus, when sampling transitions from this smaller set, composed of unique transitions representing their sets of similar transitions, the chances of a rare or expansive transition being observed increase compared to sampling on the entire set of transitions already performed. Also, in this sense, if sampling is performed based on a probability distribution or weights based on the values of r' and Q' instead of the original values of r and Q, to favor transitions that will possibly produce a better return value, these values of r' and Q' already bring in themselves information about the expected values for a similar transition in the future.

Hypothesis 2 – It is possible that sampling on this smaller set will produce more effective value function updates from the temporal difference error compared with sampling on the large set of all transitions, similar or not, the agent already experienced.

Question 3 – *Consider that there is a large set containing all transitions ever experienced by an agent and another set obtained from it, which is considerably smaller but still capable of representing all transitions ever experienced. Could using that smaller set help to reduce the convergence time for an optimal action policy in reinforcement learning methods that use temporal difference learning?*

Consider the following statements: (i) experience replay is an effective method for reducing convergence time; (ii) sampling transition experiences that define policies that are very different from the current policy may lead to underestimates of the agent's value function; (iii) even sampling that uses some probability distribution technique or weights to prioritize experiments, if performed over a fixed-size buffer and under a sliding time window, may still waste rare or expensive-to-obtain but highly informative transitions.

Hypothesis 3 – It is possible to state that sampling over a smaller set but still capable of representing a large set of experiences allows for effective experience replay

and increases the probability of sampling transitions representing experiences that are rare or expensive to obtain under a reduced distribution space. Thus, using a smaller set but with a great representation of the agent's universe of experiences can contribute to reducing the convergence time in reinforcement learning methods with temporal difference learning by providing more informative transition samples for calculating the temporal difference error and, consequently, reducing the training time required for agent learning.

1.4 Objectives

The research carried out to develop this thesis had general and specific objectives, as described below.

1.4.1 General Objective

The general objective of this work was to make the technique called Experience Replay more data efficient in using the agents' experience through the investigation and proposal of new strategies that could overcome the main limitation identified in the literature.

1.4.2 Detailed Objectives

The following objectives were defined from the research problem and the hypotheses presented in Sections 1.2 and 1.3, respectively.

a) Propose and validate a grouping strategy and memory structure to produce and store sets of similar transitions so that it is possible to obtain, from each set, a single tuple containing the elements that define a transition, which represents all other transitions and which contains predicted values for the expected reward in the long run according to Hypothesis 1.
b) Propose and validate a method for generating a compact transition memory from sets of similar transitions, which is smaller than the set of all transitions already performed by the agent and which, at the same time, represents all of them, according to Hypothesis 2.
c) Propose and validate a new method to approximate the target function in the temporal difference learning framework so that (i) this method learns the dynamics of updates in the estimates of the expected long-run rewards from the sets of similar transitions to predict it in a future where a similar transition occurs; (ii) produce more effective computations of the temporal difference error and (iii) obtain more

effective updates for the Q-value function from the gradients produced by the new proposed loss, which is computed from a single target value for each similar transition.

d) According to Hypothesis 3, investigate the effects of using a compact memory that is dynamically sized – meaning it grows based on the dynamics of the environment and is not explicitly sized as the traditional buffer – for sampling single transitions obtained from similar transition sets. The aim is to enhance the agent's learning process and reduce convergence time.

f) Evaluate the proposals' efficacy in speeding up the agent's learning on different problems involving environments and tasks in discrete and continuous action and state values spaces.

1.5 Research Work Justifications

Demanding long training times can limit the application of reinforcement learning methods. How to effectively reduce this time is still an important and recurring issue in the literature, and it was the main objective of this work.

Lin (1992) proposed the Experience Replay to effectively accelerate convergence by overcoming many issues in approximating the value function from experiences with deep neural networks, among many other contributions. However, its FIFO-like memory buffer and uniform sampling still present limitations, which many relevant works in the literature seek to overcome by exploring new sampling strategies, such as using some schema to prioritize experiences or guiding the sampling process. However, the literature has neglected aspects of fundamental importance, such as buffer size, the selection of which experiences should be stored and how, and possible similarities or implicit patterns between experiences that produce better or worse estimates of expected reward value in the long term. The first research works to highlight this gap were this one and that presented by Zhang and Sutton (2017), which we found in our literature review and share similar questions about the size of the buffer of experiences but not the other aspects we proposed to investigate. That gap and its evidence in the literature demonstrate the relevance of the hypotheses raised here. From that, the objectives and expected results, now corroborated by empirical results, justified the development of this work.

1.6 Contributions

This work presents a rich and substantial discussion that deeps relevant questions about experience replay. It hypothesized about subjacent behavior patterns of the agents' experiences and strategies regarding structuring and sizing the replay memory

and discussed its effects on data-efficient experience replay. These discussions defined a consistent problem, intriguing motivations, and a set of relevant hypotheses and objectives that led to the formal proposition and evaluation of a new strategy of experience replay and two new reinforcement learning methods. These methods achieved better or at least comparable results to those obtained by relevant off-policy and model-free methods in the literature, although expending considerably fewer agent training steps, both on discrete and continuous-valued action spaces.

Its theoretical contributions relativized assumptions we tend to assume as truth (in any case), such as the assumption that an agent always experiences completely different states. This assumption is valid if one considers exactly equal states (mainly because of intrinsically non-deterministic problems or even in high-dimensionality states); however, this work proved that it is not strictly true if we consider similar states. Another known valid assumption is that related experiences can lead to non-stationarity and, in this way, harm the learning process in deep reinforcement learning. It is true if we use such experiences directly in a back-forward optimization step on a deep neural network. However, it is not true that exploring any relationship can harm the process; on the contrary, it can reveal behaviors or patterns that improve the process, as proved in this work. Besides, this research hypothesized about the adverse effects of arbitrarily defining a large or a small experience memory, formally presented a strategy to construct a compact but dynamic memory, and empirically corroborated the hypothesis that sampling from a small and dynamic sized memory could make experience replay more data efficient. Moreover, our primary motivation was thinking that an agent can learn from their experiences but could learn more from similar experiences (like our intuition about humans in many moments in their lives), and it proved to be true.

Besides the theoretical contributions, this work has the following specific contributions:

- An extensive and structured literature review and a taxonomy that organizes the many research works and the different RL methods that employ Experience Replay, focusing on how they improve and apply ER strategies, demonstrating their specificities and contributions. Moreover, the text is organized in a facet-oriented way, allowing different perspectives of reading, whether based on the fundamental problems of RL or with a view to different applications of RL with ER.
- A formal proposition of a new data-efficient reinforcement learning method that seeks to reduce the required number of experiences for agent training. It uses temporal difference learning with predicted target values from sets of similar transitions and a new experience replay approach based on a compact transitions memory.

- A new method called COMPact Experience Replay (COMPER), that expends about only one hundred thousand video-frame observations to achieve better or at least similar results to the ones achieved by other relevant methods in the literature, that generally demands millions of video-frames, to train an agent on the Atari 2600 games.
- A new method called COMPact Memory for Continuous ACTions (COMCACT), that achieves better or at least similar results to the ones achieved by another relevant method in an experimental setup and using only fifty thousand training iterations to learn physically controlling multijoint humanoids while the literature trains the agents over millions of iteration steps.

1.7 Text Overview

Chapter 2 presents a detailed theoretical background. Chapter 3 presents the literature review. Chapter 4 presents the proposed method of experience replay with compact memory. Chapters 5 and 6 present the evaluation methodology and the empirical results for two new reinforcement learning methods with compact memory. Chapter 7 deepens on the components of the proposed method. Chapter 8 brings the conclusions and points out some directions for future works.

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 of 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. A policy π represents the agent's behavior, and the value $\pi(a \mid s)$ represents the probability of taking action a in 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 the 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 aims to learn (or approximate) an optimal policy π^* that maximizes the expected long-term (discounted) reward value. These processes imply nondeterministic search problems and stochastic decision sequences for selecting actions from observing each state of the environment resulting from a previous decision. In this way, each agent's 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 using the concept 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 2.1–2.4) the expected sum of future rewards for the states reached by adopting the policy π and performing the sequences of state transitions. In (2.5), it is clear the dynamic nature of the computation of $v_\pi(s)$, and in (2.6) we have the discount value γ on the total expected reward.

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[r_1 + r_2 + ... + r_T \,|\, s_t = s] && (2.1)\\
&= \mathbb{E}_\pi[r_t] + \mathbb{E}_\pi[r_{t+1} + r_{t+2} + ... + r_T \,|\, s_t = s] && (2.2)\\
&= \sum_a \pi(s,a)R(s,a) + \mathbb{E}_\pi[r_{t+1} + r_{t+2} + \cdots + r_T \,|\, s_t = s] && (2.3)\\
&= \sum_a \pi(s,a)R(s,a) + \sum_a \pi(s,a) \sum_{s'} T(s,a,s')\mathbb{E}_\pi[r_{t+1} + \cdots + r_T \,|\, s_t = s'] && (2.4)\\
&= \sum_a \pi(s,a)R(s,a) + \sum_a \pi(s,a) \sum_{s'} T(s,a,s')v_\pi(s') && (2.5)\\
&= \sum_a \pi(s,a) \left[R(s,a) + \gamma \sum_{s'} T(s,a,s')v_\pi(s') \right] && (2.6)
\end{aligned}
$$

In the form presented by Sutton and Barto (2018) in Equation 2.7, the state-value function $v_\pi(s)$ 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 in the state s and follows the policy π, allowing it to assess the utility of each possible action at that state (Equation 2.8).

$$v_\pi(s) = \sum_a \pi(a \,|\, s) \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma v_\pi(s') \right] \tag{2.7}$$

$$q_\pi(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \left[\sum_a \pi(s',a')q_\pi(s',a') \right] \tag{2.8}$$

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, at least one optimal policy π^* 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; 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:

$$
\begin{aligned}
v^*(s) &= max_{a \in A(s)} q_{\pi*}(s,a) && (2.9)\\
&= max_a \sum_{s',r} p(s', r \,|\, s, a)[r + \gamma v^*(s')]
\end{aligned}
$$

In turn, the Bellman optimality equation for the action-value function can be

defined as follows:

$$q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma max_{a'} q^*(S_{t+1}, a') \mid S_t = s, A_t = a] \\ = \sum_{s',r} p(s', r \mid s, a)[r + \gamma max_{a'} q^*(s', a')] \tag{2.10}$$

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 with their respective values or, in other words, 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. They also assume that the dynamics of the environment are precisely known, which is not always true. This way, using these equations imposes limitations on a class of methods known as Dynamic Programming (DP) (SZEPESVáRI, 2010). The DP-based methods 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; 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. In 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 arbitrarily chosen and updated from the values computed in 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

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 in evaluation. In this way, each iteration updates the value of each state to produce a new approximation of the state-value function v_{k+1}:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(s \mid a) \sum_{s',a} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{2.11}$$

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:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',a} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{2.12}$$

The policy will change if $q_\pi(s, a) > v_\pi(s)$, as it will be better to choose the action a in the state s and then follow the policy π instead of following 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:

$$\begin{aligned} \pi'(s) &= argmax_a q_\pi(s, a) \\ &= argmax_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= argmax_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{2.13}$$

This is a special case of the policy improvement theorem. Let two policies be π and π' so that, for all $s \in S$, $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, 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. Given a policy and its value function, it is possible to evaluate a policy change in a single state for a given action (SUTTON; 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 value function and policy in a finite number of iterations.

Although convergence to the optimal policy and the 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 for 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 2.14, for all $s \in S$:

$$\begin{aligned} v_{k+1}(s) &= max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \end{aligned} \tag{2.14}$$

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

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 without the need for a model of the dynamics of the environment. It can update estimates based on other learned estimations before reaching a final state, which is a clear advantage over DP methods concerning computational efficiency. There are variations of the TD method regarding the number of steps applied to calculate 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). Sampling-based updatings, like those used in TD methods, are distinct 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; 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 2.15). 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 as 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 2.15.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.15}$$

Applying Q-learning to those problems where the space of states and actions is too large to learn all the actions' values in all possible states 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 2.16 (HASSELT; GUEZ; SILVER, 2016).

$$\Theta_{t+1} = \Theta_t + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a, \Theta_t) - Q(s_t, a_t, \Theta_t)]\nabla_{\Theta_t} Q(s_t, a_t, \Theta_t) \tag{2.16}$$

One can see that the target function in the TD-error calculation consists of a greedy policy defined by the *max* function. In this way, the maximum over the estimated values is implicitly used to estimate 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 in this set of actions. Such a 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, we 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 is 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 Hasselt (2010) to formulate the method called Double Q-Learning (Equation 2.17), 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 bias by decomposing the max operation in the target function into action selection and evaluation, which improved the update of the action-value function, making it more stable by diminishing the overestimation of the

Q-values.

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha[r_t + \gamma Q_2(s_{t+1}, argmax_a Q_1(s_{t+1},a)) - Q_1(s,a)] \tag{2.17}$$

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 2.18 and 2.19.

$$y_t = r_t + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1}, a, \Theta_t), \Theta'_t) \tag{2.18}$$

$$\Theta_{t+1} = \Theta_t + \alpha[y_t - Q(s_t, a_t, \Theta_t)]\nabla_{\Theta_t} Q(s_t, a_t, \Theta_t) \tag{2.19}$$

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 the knowledge assimilated, the maximization of efficiency and performance, the minimization of risks, and the refinement of the knowledge already acquired. In this way, to increase the accumulated reward value, an agent seeks to select actions already experienced and which produced good results, but it is necessary to try new actions to discover new ones that may provide greater rewards, thus choosing between obtaining rewards quickly or having a chance to select better action in the future. The agent must try various actions and progressively pick the best ones. This dilemma is a complex problem and has not yet been exhausted in the literature.

Bellemare, Dabney and Munos (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 2.20, 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.

$$Z(s,a) \overset{D}{=} R(s,a) + \gamma Z\left(S', A'\right) \tag{2.20}$$

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 of those vectors. Therefore, they define a Bellman operator τ^π and an optimality operator τ like in Equations 2.21 and 2.22, where P is the transition function (as defined in Section 2). Instead of expectation, they consider the full

distribution of the variable Z^π, which they call *value distribution*. Moreover, they also discuss the theoretical behavior of the distributional analogs of the Bellman operator in the control setting.

$$\tau^\pi Q(s,a) := \mathbb{E}R(s,a) + \gamma \mathbb{E}_{P,\pi} Q(s',a') \tag{2.21}$$

$$\tau Q(s,a) := \mathbb{E}R(s,a) + \gamma \mathbb{E}_{P} \max_{a'} Q(s',a') \tag{2.22}$$

Finally, 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 agent performance.

2.1 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; DAYAN, 1992), which represent the first TD-learning-based methods in RL, are characterized by high convergence times. An effective technique called Experience Replay (ER) was proposed by Lin (1992) to speed up the credit attribution process and consequently reduce convergence time by storing 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, such as those involving penalties. Moreover, using ER with random sampling reduces the effect of the correlation of the data (which represents the environment states) and improves its nonstationary 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 generate considerable changes in the policy learned by the agent (MNIH et al., 2015). So it can change the data distribution and the relation between the action-value functions in calculating 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 grasp 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 and accomplish the task goal. Those lessons store selected actions, state transitions, and attained 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 did not improve the agent 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. When comparing AHCON-T and QCON-T with AHCON-R and QCON-R, no significant differences existed in the less complex environments. However, in the complex environments, 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 more demanding and has demonstrated the superiority of ER 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. Nevertheless, it does not seem advantageous, especially in problems with large spaces of states and actions, nondeterministic scenarios, and nontabular 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 size of the replay buffer. According

to (MNIH et al., 2015), all DQN-based methods (some of the most relevant are presented in Section 2.2) used a fixed replay memory size of 1M transitions. Recently, some research works investigated the effects of small and large buffers (ZHANG; SUTTON, 2017; LIU; ZOU, 2018). In turn, this research work uses a small dynamic memory to explore the replay of the experience and the dynamics of the transitions, reducing the number of experiences required for agent learning.

2.2 Experience Replay in Deep Reinforcement Learning

Looking at the history of reinforcement learning, some of the most recent and significant improvements have arisen from the possibility of approximating value functions from multidimensional data. At this point, adopting artificial neural networks was a promising proposition deeply investigated in the early related literature, including Lin (1992). However, the proposal of using Convolutional Neural Networks to approximate the action-value functions together with Experience Replay was a game-changing approach presented by Mnih et al. (2013). From that, and because ER reduces nonstationarity and decorrelates the agent's updates, contributing to the stabilization when using deep neural networks, the recent research in the literature has been working based on these two fundamental findings and bringing many relevant discovering mainly in this, but not only, relevant aspects: (i) approximating value function; (ii) reducing bias; (iii) composing better value functions; (iv) improving data efficiency; and (v) dealing with continuous-valued action spaces. Therefore, this section starts the discussions regarding ER in Deep RL, supported by some fundamental works and other new studies about these relevant aspects.

2.2.1 Variations on Convolutional Neural Networks in Q-learning and Double Q-learning-based approaches

Deep Q-Network (DQN) (MNIH et al., 2013) and Double Deep Q-Network (DDQN) (HASSELT; GUEZ; SILVER, 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 how CNN is used, as shown in Algorithm 1. Instead of using the same network (with the same parameters Θ) to approximate 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 performed 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 reward received at t, and s_{t+1} is the state resulting 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. 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,

$$\mathcal{L}_{DQN}(\Theta_i) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim U(\mathcal{D})} \left[\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a', \Theta') - Q(s_t, a_t, \Theta_i) \right)^2 \right], \quad (2.23)$$

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.

The 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 estimation of the Q-value (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 Hasselt, Guez and Silver (2016), it can lead to overestimated high action values. Still, these values would not be 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 action values are inaccurate. In 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

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$\big(s_t, \epsilon, \Theta\big)$;
    $s_{t+1}, r_t \leftarrow env.step(s_t, a_t)$;
    $StoreTransition(\mathcal{D}, (s_t, a_t, r_t, s_{t+1}))$;
    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**
            $\Theta' \leftarrow \Theta$;
        $\Delta \leftarrow 0$;
    $s_t \leftarrow s_{t+1}$;

DDQN based on Double Q-Learning. Since Double Q-Learning learns two value functions using two different sets of weights Θ and Θ', it is possible to compare Q-Learning to Double Q-Learning, rewriting its target value to untangle action selection and evaluation – Equations 2.24 and 2.25.

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a, \Theta_t), \Theta_t) \quad (2.24)$$

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a, \Theta_t), \Theta'_t) \quad (2.25)$$

The authors demonstrated that DDQN (see Algorithm 3) reduced bias using the

Algorithm 2: ϵ-Greedy

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

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

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**
    $a_t \leftarrow \epsilon$-Greedy$\big(s_t, \epsilon, \Theta\big)$;
    $s_{t+1}, r_t \leftarrow env.step(s_t, a_t)$;
    $StoreTransition(\mathcal{D}, (s_t, a_t, r_t, s_{t+1}))$;
    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**
            $\Theta' \leftarrow \Theta$;
        $\Delta \leftarrow 0$;
    $s_t \leftarrow s_{t+1}$;

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. The target value changes from Equation 2.26 to Equation 2.27. 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.

$$Y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a', \Theta') \tag{2.26}$$

$$Y_t = r_t + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1}, a, \Theta), \Theta') \tag{2.27}$$

Some research works sought to investigate video frames as sequences in the input of the neural network that approximate 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 pre-processing step) prevents the DQN agent from achieving the best performance in games that require remembering faraway 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

Long Short-term Memory (LSTM) instead of the first fully connected layer, just after the series of convolutional layers in the original DQN neural network architecture, 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. However, 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 using DDQN instead of DQN.

Wang et al. (2016) proposed using a new deep neural network architecture called Dueling Networks instead of classical (well-known) architectures such as CNNs, LSTMs, or MLPs to improve model-free reinforcement learning methods. This architecture can generalize learning across actions without changes to the underlying reinforcement learning algorithm. 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 2.28 combines these two outputs and produces the final Q-value estimations, in which α and β are the parameters of the two streams of fully connected layers, and θ represents the parameters of the convolutional layers.

$$Q(s, a, \theta, \alpha, \beta) = V(s, \theta, \beta) + \left(A(s, a, \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A\left(s, a', \theta, \alpha\right) \right) \tag{2.28}$$

As the output is also Q-value estimates for each action in the input states, the dueling network architecture can replace the original neural networks in other algorithms such as DQN and DDQN, with adaptation only regarding backpropagation. The authors demonstrated improved experimental results using uniform and Prioritized Experience Replay (PER) (SCHAUL et al., 2016).

2.2.2 Dealing with Continuous-valued Action Spaces in Off-policy RL

The DQN-based methods, such as the DDQN and Dueling 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 nondeterministic 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. (2016) 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 uses 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 in different challenges in simulated physical environments through a physics engine originally proposed for model-based control called MuJoCo (TODOROV; EREZ; TASSA, 2012).

From the derivations of the Bellman equation presented in Section 2 to define the action-value function in Equation 2.8, 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. (2016), changing from Equation 2.29 to Equation 2.30.

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \tag{2.29}$$

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1})] \tag{2.30}$$

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}. Thus, an approximator for the Q-value function parameterized by Θ^{Q} could be optimized by minimizing the loss obtained in Equation 2.31.

$$L(\Theta^{Q}) = \mathbb{E}_{s_t \sim \rho^{\beta}, a_t \sim \beta}[(Q(s_t, a_t \mid \Theta^{Q}) - r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}) \mid \Theta^{Q})^2] \tag{2.31}$$

The DPG algorithm applies a parameterized function $\mu(s \mid \theta^{\mu})$ (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 update technique applies the chain rule to the expected return from the start distribution J concerning the actor parameters, as in Equations 2.32 and 2.33. In turn, it learns the Q-value function $Q(s, a)$ (the critic) using the Bellman equation as in

Q-Learning (LILLICRAP et al., 2016).

$$\nabla_{\theta_\mu} J \approx \mathbb{E}_{s_t \sim \rho^\beta}[\nabla_{\theta_\mu} Q(s, a \mid \theta^Q)|s = s_t, a = \mu(s_t \mid \theta^\mu)] \tag{2.32}$$

$$= \mathbb{E}_{s_t \sim \rho^\beta}[\nabla_a Q(s, a \mid \theta^Q)|s = s_t, a = \mu(s_t) \nabla_{\theta_\mu} \mu(s \mid \theta^\mu)|s = s_t] \tag{2.33}$$

DDPG (see Algorithm 4) applies modifications to DPG to use the contribution of DQN in approximating the Q-value function from the high-dimensional states space and to 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 updating of the target function. 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, $Q'(s, a \mid \theta^{Q'})$ and $\mu'(s \mid \theta^{\mu'})$, and use them to estimate the target values, then update these target networks by slowly (for stability) tackling 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 train the critic network consistently. 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; SZEGEDY, 2015) to the state input and all layers of the μ network and the layers of the Q network before its action input layer. Because it is off-policy, DDPG can deal with exploration (a difficult problem in continuous action spaces) independently of the learning algorithm. While DQN uses an ϵ-greedy approach (see Algorithm 2), Lillicrap et al. (2016) use an Ornstein-Uhlenbeck process (UHLENBECK; ORNSTEIN, 1930) to add a noise $\mathcal{N}$ to the actor policy and generate temporally correlated exploration, where $\mu'(s_t) = \mu(s_t \mid \theta_t^\mu) + \mathcal{N}$.

2.2.3 Improving Sample Data Efficiency in Experience Replay

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 approximates 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 they occurred. 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 Prioritized Experience Replay (PER) method. The authors initially investigated the effects of prioritizing experiences in reducing the number of updating steps that 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

Algorithm 4: 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**
      $\tau = (s_t, a_t, r_t, s_{t+1}) \sim UniformSample(\mathcal{D})$;
      $Q_{target} \leftarrow Q'(s_{t+1}, \mu'(s_{t+1}|\Theta^{\mu'}), \Theta^{Q'})$;
      $Q_{value} \leftarrow Q(s_t, a_t, \Theta^{Q})$;
      $\mathcal{L}(\Theta^{Q}) \leftarrow (r_t + \gamma \times Q_{target} - Q_{value})^2$;
      $L \leftarrow L + \mathcal{L}(\Theta^{Q})$;
      $P \leftarrow P + \nabla_a Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|s = s_t$;
      $\Delta \leftarrow \frac{1}{k} L \times \bigtriangledown_{\Theta^{Q}}$;
    $\Theta^{Q} \leftarrow \Theta^{Q} + \alpha \times \Delta$;
    $\Theta^{\mu} \leftarrow \Theta^{\mu} + \alpha \times \nabla_{\Theta^{\mu}} J \approx \frac{1}{k} P$;
    $\Theta^{Q'} \leftarrow \tau\theta^{Q} + 1(1-\tau)\Theta^{Q'}$;
    $\Theta^{\mu'} \leftarrow \tau\theta^{\mu} + 1(1-\tau)\Theta^{\mu'}$;
    $s_t \leftarrow s_{t+1}$;

memory and replays the ones with the highest absolute value of the TD-error to update the Q-function. They verified that it reduced the number of update 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, replaying experiences with high and slowly decreasing TD-errors often causes a loss of diversity. That may lead the model to overfit, besides being sensitive to noise spikes (e.g., when the rewards are stochastic). Therefore, they proposed a stochastic sampling method that combines greedy prioritization and uniform random sampling by defining the sample probability based on a transition's priority value. According to Equation 2.34, the probability of sampling a transition j is

$$P(j) = \frac{p_j^{\alpha}}{\sum_i p_i^{\alpha}}, \tag{2.34}$$

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 2.35,

$$w_j = \left(\frac{1}{N} \times \frac{1}{P(j)}\right)^{\beta} \tag{2.35}$$

in which N represents the size of the replay buffer and compensates for the nonuniform 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 5). They replaced the uniform sample in DDQN, achieving new state-of-the-art results in learning to play Atari 2600 games in ALE.

According to Schaul et al. (2016), the use of 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, assuming that the content of the memory was beyond their control. Novati and Koumoutsakos (2019), Zha et al. (2019), and Sun, Zhou and Li (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, this research work originally approaches the first case, investigating how to store the 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 the updates can deteriorate when the policy diverges from past behaviors and can undermine the performance of the ER. Instead of tuning hyperparameters to slow down policy changes, they actively reinforced the similarity between 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 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 their approach could be applied to any off-policy method with parameterized policies (i.e., by using Deep Neural Network

Algorithm 5: 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 $\Theta' = \Theta$;
Initialize $env \leftarrow EnvironmentInstance$; $s_t \leftarrow env.InitialState()$;
for $t \leftarrow 1$ **to** T **do**
  $a_t \leftarrow \epsilon$-Greedy$\big(s_t, \epsilon, \Theta\big)$;
  $s_{t+1}, r_t \leftarrow env.step(s_t, a_t)$;
  $p_t \leftarrow \max_{i<t} p_i$;
  $StoreTransition(\mathcal{D}, (s_t, a_t, r_t, s_{t+1}), p_t)$;
  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**
      $\Theta' \leftarrow \Theta$;
    $\Delta \leftarrow 0$;
  $s_t \leftarrow s_{t+1}$;

– DNN) and that it allows for better stability and agent performance (compared to uniform sampling) in the main class of methods for continuous action 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; WHITE; SUTTON, 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. They concluded their proposal is promising because it could find more "usable" experiences for off-policy agents using ER in different tasks.

Sun, Zhou and Li (2020) proposed the Attentive Experience Replay (AER) to prioritize transitions (at sampling from the replay buffer) containing states more frequently observed by the 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 considered the similarity between past and current transition states as a measure of frequency and prioritization criteria. For 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, Zhou and Li (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 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; HOOF; MEGER, 2018), and compared with uniform sampling and PER. They used the OpenAI Gym task ecosystem (BROCKMAN et al., 2016).

One of the main contributions of Experience Replay (ER) is to reduce nonstationarity and decorrelate the agent's updates, contributing to the stabilization when using deep neural networks to approximate the value functions. However, how 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 several agents in parallel in multiple instances of the environment. 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 off-policy and on-policy algorithms by presenting multithreaded asynchronous variants of Q-learning, Sarsa, and Advantage Actor-Critic methods. 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 A3C on the

MuJoCo physics simulator domain (TODOROV; EREZ; TASSA, 2012).

2.2.4 Combining Benefits in Ensemble Methods

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

The authors adapted the PER strategies to use the KL loss of the Distributional Q-Learning, replaced the one-step distributional loss with a multistep 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 multistep distributional loss with Double Q-Learning and used the greedy strategy to select and evaluate the action in S_{t+n} using the target and online 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 Dopamine (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 2.36 to Equation 2.37. The authors interpret it as an interpolation between estimating targets in Monte Carlo (MC) methods as $\sum_{k=0}^{T} \gamma^k r_{t+k}$ (a discussion can be found in Sutton and Barto (2018)) and single-step TD-learning, balancing the low bias but high variance of the MC targets and the low

variance but high bias of TD(0) (see Section 2).

$$r + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \tag{2.36}$$

$$\sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \max_a Q\left(s_{t+n}, a\right) \tag{2.37}$$

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, Hasselt, Hessel and Aslanides (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). They 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. 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. However, the authors state 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.

3 LITERATURE REVIEW

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 a fascinating, rich, and complex class of machine learning algorithms. In Chapter 2, we reviewed its fundamental principles. We discussed how a technique called Experience Replay (ER) has been of fundamental importance in making various methods in most relevant problems and domains more data efficient, using agent experiences to improve its performance. In this chapter, we present some of the more recent methods in the literature and bring 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.

3.1 Challenges and Trends in Experience Replay

Table 1 – Main challenges in Reinforcement Learning with Experience Replay (a)

Problem	Description
Replay Buffer Size	It is related to complex challenges, like: (i) FIFO-like buffers can lead to a loss of experiences because of limited size; (ii) Uniform sampling in FIFO-like buffers may overwrite many relevant experiences because it does not differentiate experiences; (iii) Small buffers produce fresher but biased samples; (iv) Large buffers produce more diverse but outdated samples. We consider research works aiming to overcome the limitations that arise from the buffer size.
Exploration Efficiency	Imposes a difficult problem (especially in continuous action spaces) in regulating trade-offs between expanding knowledge by exploring actions and refining knowledge by exploiting it. It is closely related to bias and diversity and directly impacts the agent's performance. We consider works that concentrate on improving or proposing strategies for better exploration and exploitation.
Sampling Efficiency	Some experiences contribute more to the agent's learning than others, and some experiences may become more relevant as the agent approximates an optimal policy. The same batch of samples may be more important than others. Not exploring this in some way may negatively impact relevant aspects of the agent's learning. We consider works that focus on making relevant sampling.

Table 2 – Main challenges in Reinforcement Learning with Experience Replay (b)

Problem	Description
Data Efficiency	Using the agents' experiences in a data-efficient way implies thinking about extracting information directly from their data (such as using RNN), modeling transitions and memories (such as using many buffers, compacting memories, or proposing new data structures), and organizing and storing the experiences. We consider works that directly exploit the transition's data (i.e., its data values) or the memory structure.
Catastrophic Forgetting	Leads to a loss of information and instability in agent training when new experiences overwrite old experiences in the multitasking or multi-agent learning scenario. It is closely related to the buffer size and data efficiency. We consider works that mainly overcome the limitations that arise from the buffer limitation in a shared scenario.
Sparse Rewards	This is more of a fundamental and complex problem in TD-Learning-based RL than a closely ER-related problem. It imposes on the agent to learn long-term policies until rewarded or to explore an arbitrarily large space of experiences because the immediate rewards are fundamental to guiding the optimal policy approximation. However, there are notable results in approaching this problem by employing strategies of ER.

The challenges we found in the literature, from the early propositions until the most recent research, allowed us to identify a set we could consider essential problems, such as the ones Experience Replay proposed to solve. However, there are different classes of relatively recent issues arising from previously proposed approaches to open problems, such as those that PER proposed to address at the expense of possible bias or the many methods that suffer from catastrophic forgetting by benefiting from the ER, for example. This section identifies some relevant general problems in the Experience Replay domain (despite the many benefits of each approach in the literature), as presented in Tables 1 and 2, and selects some to bring exciting and in-depth discussions of the literature.

3.1.1 Replay Buffer Size

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 DQN-based methods use a fixed replay memory size of one million transitions, and variations in the buffer's size are still understudied in this class of methods (LIU; ZOU, 2018). Zhang and Sutton (2017) presented an empirical

study on ER, demonstrating that a large replay buffer can harm 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 called 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 adopting a FIFO strategy (i.e., as a queue) will impact agent learning. However, according to Neves, Ishitani and Patrocínio Jr (2022), if we assume that different transitions from many stochastic episodes carried out over a nondeterministic 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 concerning the interrelated effect of variations in its hyperparameters, since their effects may not only be individual but come from their joint variation. The authors studied the relationship between the size of the replay buffer, the *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 step of updating 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 potential age of the oldest policy requires storing more transitions 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 increases 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 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 improves when replay capacity increases which is generally related to the age reduction of the oldest policy. When one fixes the policy's age, the performance increases with the growth of replay capacity, and this trend remains regardless of the value defined for the policy's age. However, 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 overfitting over a smaller subset of transitions. On the other hand, when the replay capacity is fixed, the performance tends to improve with the reduction of the age of the oldest policy. Based on the idea that the age of an old policy distances it from the current policy and that the age depends on the replay ratio and replay capacity, the authors claimed that the results of their experiments suggest 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 (Sun, Zhou and Li (2020) also explored this hypothesis). 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, because of the sparse rewards that demand learning long-term policies, the agent could not accumulate rewards when the authors reduced the potential age of the policies and concentrated the transition samples in those closer to the current policy. 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 age of the policy 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, Ishitani and Patrocínio Jr (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.

When conducting experiments with DQN, the authors observed that its performance did not increase with the growth of the replay capacity, either by fixing the replay ratio or by 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-step returns, plus the replacement of the RMSProp optimization method with ADAM (KINGMA; 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-step returns. This increase also occurs when the policy age is fixed instead of the replay ratio. By removing the n-step return 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-step returns are the only critical component for performance gain with increased replay capacity. They also observed that the use of n-step 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. They also noticed that only adding PER to the DQN does not significantly increase performance when the replay capacity is considerably large. One can see in Section 2.2 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-step 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 have older 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 expanding the replay capacity. Investigating the relationship between n-step returns and Experience Replay, the authors observed that the replay capacity can mitigate the variance of n-step returns, which partially explains 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, Ishitani and Patrocínio Jr (2022) and Sun, Zhou and Li (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. In future works, the authors pointed to studying 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.

3.1.2 Exploration Efficiency

Fortunato et al. (2018) approached the relevant problem of exploration, proposing a method called NoisyNet to learn neural networks' weight perturbations and using this to drive exploration. The authors based this on the idea that a single change in 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, such as 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 (which 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, Dueling 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.

3.1.3 Sampling Efficiency

While Fedus et al. (2020) demonstrated that the use of n-step return is a critical element for Rainbow's performance, ablative studies in Fujimoto, Meger and Precup (2020) demonstrated that in DQN, this element is PER. According to Fujimoto, Meger and Precup (2020), 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 nonuniform 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 transformation allows a concrete understanding of the benefits of nonuniform samplings, such as in the PER, and provides 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 2.2). The second aspect is the reduction of variance, which allows 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. 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 nonuniform 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 demonstrated that the 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. However, it is no longer biased when using the MSE and setting the hyperparameter $\beta = 1$ (see Section 2.2). 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 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 the 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 nonuniform sampling by modifying only the loss function. PER adds little benefit to TD3, which is 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 a 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 simplicity of the method. They believed that its benefits over the MSE and the original Huber Loss came from its robustness and ability to approximate better the average.

For Fujimoto, Meger and Precup (2020), more research is necessary to understand better nonuniform sampling and propose new prioritization schemes. They also reinforced the sensitivity of deep learning algorithms to minor 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 importance weights methods for prioritization can improve the evaluation of the target value for more prolonged traces using TD(γ) and be used to reduce bias on values computed from off-policy experiences. 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; GUPTA; LEVINE, 2020) suggested not using on-policy experiences in this context, which contrasts with their proposal. However, DisCor bases its analysis on the Bellman optimality operator instead of the Bellman evaluation operator. The difference is that the first operator seeks to find the optimal Q-value, while the second aims 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 to 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 experiences and sought to balance their variance and bias by estimating likelihood-free density ratios and using the learned ratios as weights for prioritization, respectively.

3.1.4 Data Efficiency

On-policy methods are sometimes more effective in specific domains, such as continuous learning. However, 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. However, its use in different methods and domains depends 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. Therefore, 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 these 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, having more experiences under other policies in 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 experience. 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; KOSTRIKOV; FERGUS, 2021) and DDQN, and compared with the use of uniform sampling, PER, and Emphasizing Recent Experience (ERE) (WANG; ROSS, 2019). To evaluate their approach, they used the ALE environments, 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, Hessel and Simonyan (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.

Still according to Schmitt, Hessel and Simonyan (2020), combining Experience Replay with actor-critic algorithms is difficult due to their on-policy nature. Despite that, they proposed trust region schemes for mixing replay buffer experiences with on-policy transition data. That allowed the importance sampling method called V-trace to scale 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 off-line 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 performance improvement compared to state-of-the-art algorithms, the authors showed that uniform sampling obtains results comparable to those obtained with PER. They also demonstrated that learning using only off-policy experiences 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 the 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 a recurrent neural network with Experience Replay. They 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 modeled 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 performs 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 DQN 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 authors used the Dueling Network (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 the 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, compressing the range of priorities and limiting 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 and 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 for 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 the use of highly correlated data can bring about 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 strategy in various DeepMind Lab environments, the authors identified 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 R2D2 in the 57 games of 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 DQN was its ability to generalize over different environments using the same network architecture and hyperparameter values. 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 the 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 achieved 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 some methods had not yet obtained above-human performance in many games (52 of the 57) such as R2D2. Still, like the other methods, R2D2 did not show considerable advances in Montezuma's Revenge and Pitfall games, which are known for being difficult environments to explore. 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.

3.1.5 Catastrophic Forgetting

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 chance 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 approach 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 tasks' types and boundaries 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.

3.1.6 Sparse Rewards

Andrychowicz et al. (2017) presented a technique to deal with sparse rewards, one of the significant and challenging problems in Reinforcement learning, because it imposed on the agent to learn long-term policies (until it receives a reward) or to explore an arbitrarily large space of experiences, considering that the immediate rewards are fundamental to guiding the optimal policy's approximation. Despite impacting all methods of RL in different domains, this is a particular issue when dealing with continuous action spaces in robotics-related problems. According to the authors, a common challenge, especially in robotics, is engineering a reward function that reflects the task and can guide policy optimization. Many approaches dedicate efforts to formulating complicated cost functions for problems like staking a brick on top of another, which limits the application of RL because it requires domain-specific knowledge. Motivated by the way human beings learn almost as much from not-so-good results as from the good or the desired ones, they proposed to bring the reinforcement learning this same idea whenever there are multiple goals to achieve, i.e., achieving each state could be a separate goal. Therefore, their approach was training universal policies, taking input from the current state and a goal state, and replaying each episode with a different goal than the one the agent was trying to achieve, and that was one of the goals achieved in the episode.

After experiencing some episodes, their algorithm called Hindsight Experience Replay (HER) stores every transition resulting from the agent's experiences in the replay buffer, along with the original goal used for this episode and a subset of other goals.

As the current goal influences the agent's actions but not the environment dynamics, replaying each trajectory with an arbitrary goal is possible using an off-policy method. According to the authors, one relevant aspect is the strategy they defined to choose the additional goals used for the replay. In the simplest version, they replay each trajectory with the goal achieved in the final state of the episode. However, they have experimentally evaluated different types and quantities of additional goals. In all cases, they also replay each trajectory with the original goal. The authors argue that one can see HER as a form of implicit curriculum because the goals used for replay naturally shift from simple to achieve even by a random agent to more difficult ones. However, HER does not require having any control over the distribution of initial environment states. The experimental results demonstrate that the HER algorithm learns with extremely sparse rewards and performs better with sparse rewards than with shaped ones.

3.2 Research and Applications on ER and Some Directions for Future Works

A 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 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 C presents detailed search results, methodology, and applied criteria. Although we are more interested in works that propose changes or new methods using ER, mainly the ones that investigate it, delving into its theoretical issues and empirical investigations, we could verify that many works in the literature apply well-known RL methods to approach different classes of complex and exciting problems. We present these works in Subsection A and highlight some of them, which, besides focusing on applicated RL, brought interesting approaches in adapting RL and ER methods to make them more suitable to their problems and application domain.

Optimal control is a recurrent problem in the literature that approaches complex non-linear systems, such as robotics, autonomous driving, and domains such as biochemical reactions (YANG; WANG; QIAO, 2022). Many research works use variations of policy gradient-based reinforcement learning methods with novel mechanisms of experience replay, such as in (WANG; ZHAO; CHENG, 2019), whose proposal was to

[1]https://www-periodicos-capes-gov-br.ezl.periodicos.capes.gov.br/index.php
[2]https://dl.acm.org/
[3]https://ieeexplore.ieee.org/Xplore/home.jsp
[4]https://www.sciencedirect.com/
[5]https://www.elsevier.com/pt-br/solutions/scopus

transform adaptive cruise control problems into optimal tracking control problems and handled by a novel model-free Adaptive Dynamic Programming (ADP) approach called ADPER. Similarly, Yang and He (2020) presented 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 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) introduced a motion planning algorithm for robot manipulators using a twin delayed deep deterministic policy gradient, which applies the Hindsight Experience Replay (HER) formulated by Andrychowicz et al. (2017). 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, Raymond and Prorok (2022) approached the complex problem of autonomously driving in rule-dense environments by partitioning the experiences buffer into clusters labeled by explanations about rulesets, defining a method called Explanation-Awareness Experience Replay (XAER). Cui et al. (2023) presented their experience replay approach, Double Bias Experience Replay (DBER), and a new loss function (a challenging environment modeling problem in these domains), applied to the classical off-policy algorithms DQN and DDQN and also to the Quantile Regression DQN (QR-DQN). Li and Ji (2021) proposed a distributed training framework with parallel curriculum experience replay to approach sparse rewards in distributed training in a simulated environment to robots. Hu et al. (2023) described and evaluated the Asynchronous Curriculum Experience Replay (ACER), which uses multiple threads to update priorities and increases the diversity of experiences. Regarding the memory of experiences, they introduce a temporary pool to improve learning from fresher experiences and change the memory buffer approach from FIFO (First-In, First-Out) to FIOU (First-In, Useless-Out) to enhance learning from old experiences. The authors' main objective was to overcome what they identified as the limitations of PER to seek safe autonomous motion control of unnamed aerial vehicles in complex, unknown environments. Liu et al. (2024) approached the challenges of insufficient data, dynamic uncertainties, long time delay, and slow time-varying of thermal processes in the domain of coordinated control of coal-fired power generation systems. They proposed a DDPG-based method called DPER-VDP3G with a dual-prioritized experience replay and a value distribution strategy to reduce nonuniform sampling bias, remove redundant data, enhance sample diversity, and improve the accuracy of the cost function.

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; HWANG; LIN, 2021). Others proposed different forms to define the importance of samples (KONG; NAHRENDRA; PAEK, 2021), new prioritization criteria (GAO et al., 2021), and changes in the updates of the value function approximation 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 paradigm of replay of experiences (inspired by quantum theory) that considers the complexity and the times replayed of each experience to achieve a better balance between exploration and exploitation, which is a fundamental choice and a complex problem. According to Wang et al. (2024), the imbalanced class distribution may affect the performance of deep reinforcement learning with ER applied to classification problems. Therefore, they approached the problem of customers' credit scoring in P2P lending by modeling it as discrete-time finite Markov decision processes and proposed a balanced stratified prioritized ER strategy to optimize the loss function of a DQN model. Their objective was to balance the numbers of minority and majority experience samples in the mini-batch (according to the class representation) and select more important experience samples for replay based on the principles of PER. The authors defined the concepts and measures of majority and minority experience samples and stored the samples in minority and majority experience replay buffers. They calculated the TD-error for samples from each buffer, calculated the probabilities for prioritization based on the respective TD-errors, and used two value functions and two target functions, one for each sample.

Catastrophic forgetting is another well-known problem affecting training stability and agent performance, especially in continuous action spaces in multi-agent scenarios. It is mainly related to the buffer size limitation and is sensitive to 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; LOPEZ, 2023), including using transfer learning about past experiences (ANZALDO; 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, Huang and Zhu (2022) proposed a method to cluster and replay experiences using 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 2.2, 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 to replace 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 Multi-armed Bandit, which combines multiple priority weighted criteria to measure the importance of experiences and adjust their weights from one episode to another. Yang and Peng (2021) introduced the Metalearning-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) (LANKA; WU, 2018) 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. However, it introduces bias when not taking variable importance at different training moments and not considering the relevance of goals to agent learning. Vecchietti, Seo and Har (2022) showed that an essential factor in learning multi-goal tasks with the HER is the (relative) rate of hindsight experience used in each training epoch, and it replaces real experience with hindsight experience at a fixed rate during the entire training process. However, their results suggest that hindsight experiences are more relevant at the beginning of training, such as when a robot learns the basic sensing skills and subtasks necessary to achieve the goal. They proposed adjusting the rate of the hindsight experience using a variable sampling rate between the real and 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 that combines 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. (2022a) approached the problem of dealing with sparse rewards in online recommendation systems that use reinforcement learning. They defined a state-aware experience replay model to allow the agent to selectively discover the relevant experiences using locality-sensitive hashing that retains the most meaningful experience at scale and replays more valuable experiences with a higher chance. Dong, Li and Gong (2023) proposed the Curiosity-tuned

Experience Replay (CTER) method and a curiosity mechanism that generates an intrinsic reward based on a predicted curiosity value to deal with sparse rewards in command decision modeling for simulated wargaming scenarios. This mechanism also provides an adaptive exploration strategy, a novel prioritized replay, and a more efficient strategy to update the memory of experiences. To improve exploration, they introduced decaying and normalizing factors to guide the agent to explore feasible paths under sparse rewards and a curiosity-adjusted partially greedy exploration to control the ϵ-greedy policy adaptively according to the current level of curiosity of the experience memory. Regarding the ER, they elaborated a curiosity-augmented sampling technique to prioritize experiences by considering both the TD-error and the curiosity. For storing the experiences, they presented a K-segmented curiosity-balanced memory updating approach, which aims to balance the oldness and usefulness of the ER buffer.

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

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 (TAO; HAFID, 2020; FÄHRMANN et al., 2022; KOROGLU; SEN, 2022; CROWDER; ABREU; KIRSCH, 2021; LI; AGHVAMI; DONG, 2022; WU et al., 2022; REMMAN; LEKKAS, 2021; ROSENBAUER et al., 2020).

In unmanned aerial vehicles (UAVs), each node can communicate with the others through a routing protocol in UAV ad hoc networks (UANETs). However, UAV routing protocols face the challenges of high mobility and limited node energy, leading to unstable links and sparse network topology due to premature node death. Therefore, Zhang and Qiu (2022) proposed the Deep Reinforcement-Learning-based Geographical Routing Protocol for UANETs to consider link stability and energy prediction called DSEGR. They use the Autoregressive Integrated Moving Average (ARIMA) model to predict the residual energy of neighbor nodes and a link stability evaluation indicator. They modeled the packet forward process as an MDP and used a DDQN with PER to learn the routing decision process. They also designed a reward function to obtain a better convergence rate,

and the Analytic Hierarchy Process (AHP) was used to analyze the weights of the factors considered in the reward function. Finally, the authors conducted simulation experiments with DSEGR to analyze network performance, and the results demonstrate that their proposal outperforms others in packet delivery ratio and has a faster convergence rate.

According to Shi et al. (2024), UAVs equipped with mobile edge computing servers have become an emerging technology that provides computing resources for mobile devices to effectively relieve the computational pressure of massive data in 6G wireless networks. Therefore, they investigated a Multi-UAV Collaborative Assisted Mobile Edge Computing architecture to optimize the computational costs and attempt to reduce consumption of the limited onboard energy, which jointly optimizes the UAV trajectories and the scheduling strategies for the mobile devices offloading. They converted this non-convex optimization problem with high-dimensional continuous actions into an MDP and proposed the UAVs-assisted Offloading Strategy based on Collaborative Multi-Agent RL (UOS-RL). Due to the highly dynamic variation of the environment, they also presented an experience prioritization mechanism to improve the training efficiency in this scenario. The simulation results demonstrate that the proposed PER-UOS-RL algorithm outperforms existing works relating computational cost.

According to Panda et al. (2024), microgrids are self-supporting generation sources that incorporate renewable energy sources. However, managing the batteries' charge-discharge level is essential while considering the devices' smoother long-term efficiency and reliability. An RL-based strategy can provide instructions for generating pulse width modulation signals in grid-connected inverters. These inverters possess bidirectional power exchange capabilities, enabling them to regulate the direction and magnitude of power flow between the battery and the utility grid, and the agents are trained on real-world sensor readings in practical scenarios to govern the inverter's operations, thereby managing the battery's charging and discharging processes. According to the authors, previous approaches used Deep Q-learning-based methods with PER. Therefore, they justified and proposed investigating the use of Distributional RL with PER in a residential PV-microgrid setup, exploring various algorithms. They focused on energy management to reduce net power imported from the grid, paying particular attention to formulating the penalty function so that the battery does not operate at extreme limits; in addition, a complicated reward function can cause slower convergence in the learning process. They (i) benchmarked different algorithms with PER, (ii) analyzed the deep distributional and Q-learning algorithm training performance with varied discretized action space, random experience replay, and penalty without ToU-induced corrective action, and (iii) analyzed battery operation performance.

The batch process produces relatively few high-value-added products, such as fine chemicals, polymers, and pharmaceuticals. RL is a potential alternative to traditional

control methods, such as model predictive control, which seriously affects control performance when the process model is inaccurate. Therefore, Xu, Ma and Tao (2024) proposed a batch process controller based on Segmented Prioritized Experience Replay (SPER) Soft Actor-Critic (SAC) because it can obtain a more robust control strategy than other RL methods to accurately deal with their complex nonlinear dynamics and unstable operating conditions. SPER is a sampling experience method that the authors designed to improve the efficiency of ER in tasks with long episodes and multiple phases. They also proposed a novel reward function to deal with the sparse reward. They showed the effectiveness of their SPER-SAC-based controller by comparing it with other RL-based control methods.

To address the many problems involving multi-vehicle pursuit, such as autonomous police vehicles that pursue suspects, Li et al. (2024) proposed a multi-agent reinforcement learning algorithm called Progression Cognition Reinforcement Learning with Prioritized Experience for Multi-Vehicle Pursuit (PEPCRL-MVP) in urban multi-intersection dynamic traffic scenes. They used a prioritization network to assess state transitions in the global experience replay buffer according to each agent's parameters, whose mechanisms introduce diversity into the multi-agent learning process and improve collaboration and task-related performance. Furthermore, they employed an attention module to extract critical features from dynamic urban traffic environments and used it to develop a progression cognition method to adaptively group pursuing vehicles. Each group efficiently targeted one evading vehicle. The authors used a simulator on unstructured roads in an urban area and concluded that PEPCRL-MVP is superior to other state-of-the-art methods.

The recent literature approaches the problems discussed in Section 3.1 in continuous action space, multi-agents, robots and humanoids, and control in complex nonlinear systems such as in the application works presented we presented here. Most proposals are mainly concerned with strategies for sample and data efficiency, and the path the researchers are seeking is clear: speed up and improve the training process in increasingly complex environments and, for that, determine how to explore and exploit the agent's experiences is still a critical question. Moreover, many problems arise from the fundamental issue regarding the limited buffer size, for which simple solutions based on finding an arbitrary or heuristic-measured size seem insufficient, and this aspect needs more attention in the literature. Thinking about a good direction for future work, we suggest investigating the memory of experiences from the perspective of a dynamic-sized structure and looking at the experiences themselves beyond the old, well-known structure that represents a transition in a past time. It is essential to ask how we could work with a more elastic and flexible memory in its structure or how we could explore the relations and dynamics between the agent's experiences and turn this information as relevant as

the number of or the priority of the experiences the agent is replaying, in the sense of discussed by Zhang and Sutton (2017) and Neves, Ishitani and Patrocínio Jr (2022).

3.3 Structured Summary of Literature

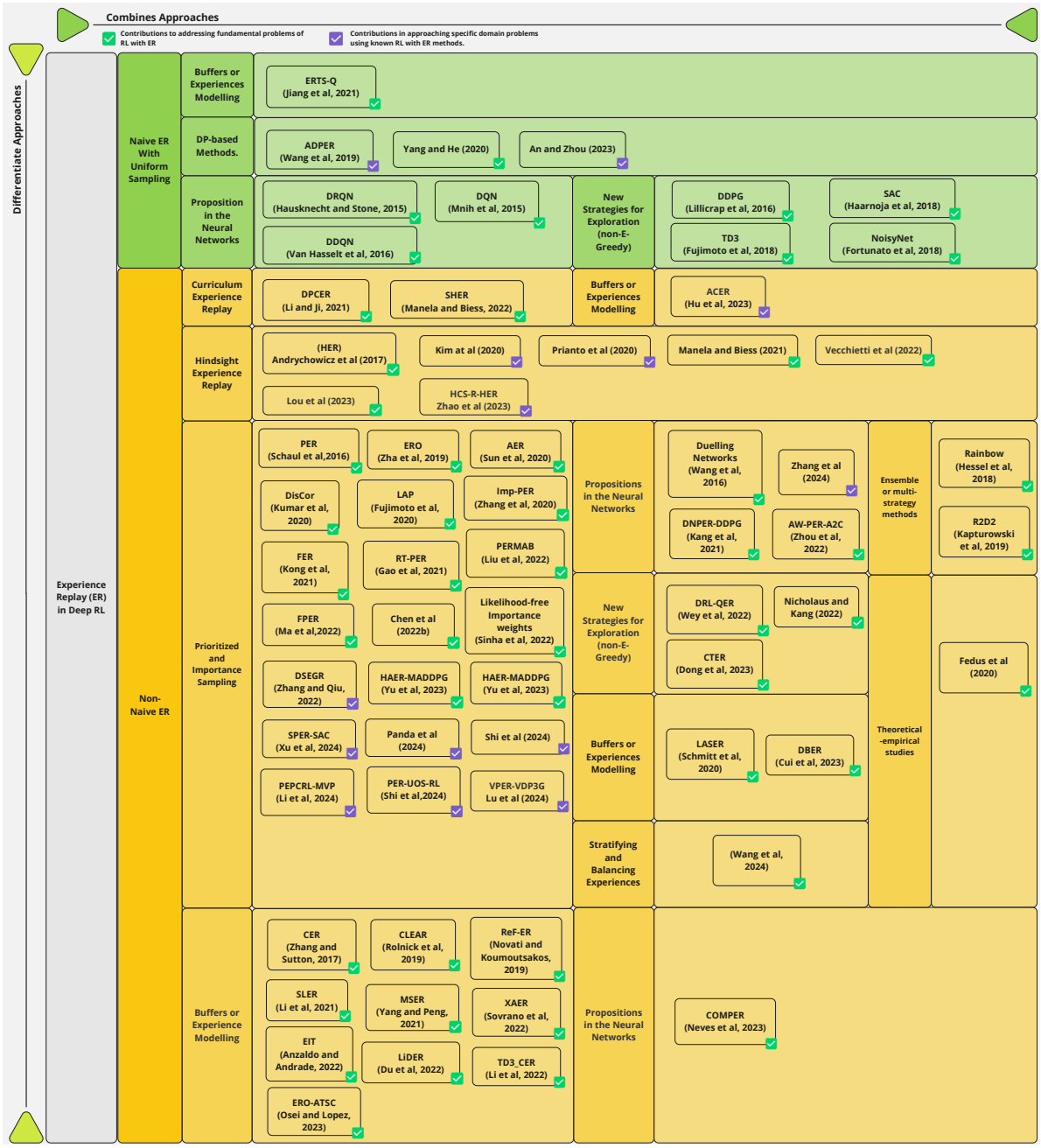


Figure 1 – Research works organization according to strategies and architectures.

This section summarizes the research works, methods, and challenges discussed in this extensive review, whose focus was to contribute to Experience Replay. The main algorithmic strategies and the proposed architectures were the first facets used to subdivide and group the research works, as presented in Figure 1. All methods address ER

in some relevant aspect. The first distinction between research works within the taxonomy is at the individual level, which groups them according to the following criterion. Some works focus on investigating and proposing strategies for specific ER problems related to its formulation and its different methods. These are identified with a green mark in Figure 1. Other works study and present innovative ways and improvements to existing ER methods applied to complex real-world problems and are identified with a purple mark in Figure 1. These markings are at the inferior right corner of each box that delimits an article and its authors. The second distinction criterion concerns the strategies of ER each work is approaching, identified by the larger label blocks in the chart vertically, from top to bottom (e.g., naive or non-naive ER? What form of prioritization or relevance does it focus on?). Lastly, the third criterion is the algorithm modifications or architectural changes that each work proposes and uses (e.g., a new neural network architecture, new exploration strategies, or new data structures for the memory of the experience). This criterion can identify more than one distinction for the same work since it may bring contributions in different but combined aspects. This is shown by staking label blocks horizontally from left to right.

The second facet focuses on the research domain, the main problem each work addresses (Tables 1 and 2), and the proposed approach, organized in Tables 3 to 20. Each table contains the research works corresponding to the larger label blocks in Figure 1 in chronological order of publication. Tables 3 to 6 present the works that employ the original uniform sampling from a FIFO-like replay buffer but propose relevant architectural and algorithm improvements on methods strongly based on ER. Tables 7 to 9 present works whose propositions mainly focus on Curriculum and Hindsight ER strategies. Tables 10 to 17 present the works whose propositions approach some strategy of experiences prioritizing or importance sampling and those works that combine them with some architectural or algorithm propositions (e.g., PER with changes in the neural network and a new method for exploration). Tables 18 to 20 present research works that aim to improve ER in sample and data efficiency by directly focusing on architectural and algorithm improvements (e.g., by proposing new data structures for the memory of experiences, compacting or extending the memory, or applying recurrent neural networks).

Table 3 – Naive ER With Uniform Sample plus Buffers or Experiences Modeling

Reference	Domain	Addressing	Approach Summary
ERTS-Q (JIANG; HWANG; LIN, 2021)	Continuous action space in mountain car and mobile robot	Data Efficiency	Defines a tree structure with ER. Establishes a virtual model for subsequent and diferent states. Aggregates states with highly similar variations.

Table 4 – Naive ER With Uniform Sample in DP-Based Methods

Reference	Domain	Addressing	Approach Summary
ADPER (WANG; ZHAO; CHENG, 2019)	Countinuous action space for optimal control in nonlinear Systems	Data Efficiency	Transforms adaptive cruise control into optimal tracking control in a model-free adaptive dynamic programming approach.
(YANG; HE, 2020)	Optimization in uncertain nonlinear interconnected systems	Sample Efficiency Data Efficiency	Proposes a Decentralized Event-Triggered Control (ETC) strategy. Obtains optimal ETC laws of auxiliary subsystems. Updates the weight vectors in the critic networks using ER.
(AN; ZHOU, 2023)	Control in complex nonlinear systems with a cart–pole and a quadrotor unmanned aerial vehicle	Data Efficiency	Proposes a data–driven approach that combines Adaptive Dynamic Programming (ADP) with an actor-critic structure.

Table 5 – Naive ER With Uniform Sample plus Propositions on DNN

Reference	Domain	Addressing	Approach Summary
DQN (MNIH et al., 2015)	Discrete action space on Atari 2600	Data Efficiency	Approximates the action-value function using a CNN elaborated to video frames features extraction and mapping to a Q-values estimates distribution over the agents' actions in the output layer.
DRQN (HAUSKNECHT; STONE, 2015)	Discrete action space on Atari 2600	Data Efficiency	Learns long-term policy in sparsed rewards problems. Slightly changes the network from DQN by including a recurrence layer after the CNN layers and before the last dense layer.
DDQN (HASSELT; GUEZ; SILVER, 2016)	Discrete action space on Atari 2600	Overestimation bias	Mitigates Q-values overestimation bias. Approximates the action-value functions using CNN like in DQN, although using the Double Q-Learn formulation instead of Q-Learn.

Table 6 – Naive ER With Uniform Sample plus Propositions on DNN plus Exploration

Reference	Domain	Addressing	Approach Summary
DDPG (LILLICRAP et al., 2016)	Continuous action space on physics tasks	Data Efficiency	Presents an actor-critic, model-free and off-policy algorithm based on DPG and using ER to learning policies in a competitive level of planning algorithm.
NoisyNet (FORTUNATO et al., 2018)	Discrete action space on Atari 2600	Exploration Efficiency	Adds learned noise to the network weights to induce stochasticity of the agent's policy and aid efficient exploration.
TED3 (FUJIMOTO; HOOF; MEGER, 2018)	Continuous action space on OpenAI Gym	Data Efficiency	Modifies DDPG to form the algorithm TD3, which considers the interplay between function approximation errors in both policy and value updates.
SAC (HAARNOJA et al., 2018)	Continuous control benchmark tasks	Exploration Efficiency	Proposes an off-policy actor-critic algorithm based on the maximum entropy reinforcement learning framework. Entropy is a measure of randomness in the policy. Increasing entropy results in more exploration.

Table 7 – Non-Naive ER - Curriculum Experience Replay

Reference	Domain	Addressing	Approach Summary
DPCER (LI; JI, 2021)	Continuous-valued action space in Distributed RL	Sample Efficiency Data Efficiency Sparse Rewards	Presents a distributed training framework with a parallel curriculum experience replay to identify the difficulty level of subtasks collected in parallel.
SHER (MANELA; BIESS, 2022)	Ball-throwing in authors' handcraft environments	Sample Efficiency Data Efficiency Sparse Rewards	Proposes a Multigoal-based RL that combines curriculum learning with HER. Exploits the recurrent structure inherent in many object manipulation tasks.

Table 8 – Non-Naive ER - Curriculum Experience Replay plus Buffers or Experience Modeling

Reference	Domain	Addressing	Approach Summary
ACER (HU et al., 2023)	Autonomous motion control of unmanned aerial vehicles	Sample Efficiency	Proposes an asynchronous curriculum experience replay to increase the diversity of experiences. Uses multithreads to update and assign priorities.

Table 9 – Non-Naive ER - Hindsight Experience Replay

Reference	Domain	Addressing	Approach Summary
SAC-based path planning algorithm (PRIANTO et al., 2020)	Multi-arm manipulators in high-dimensional problems	Application Sample Efficiency	Proposes a SAC-based algorithm with hindsight experience replay and configuration space augmentation.
(KIM et al., 2020)	Motion and task planning for robot manipulators in manufacturing industry	Application Sample Efficiency	Defines a Robot Arm Markov Decision Process and approaches it by proposing an algorithm using hindsight and TD3.
(MANELA; BIESS, 2021)	Ball-throwing in authors' handcraft environments	Sample Efficiency Data Efficiency	Proposes the techniques IBS and Filtered HER. Prioritizes virtual goals and reduces bias by removing misleading samples. Exploits the recurrent state–space structure.
HER with sampling rate decay (VECCHIETTI; SEO; HAR, 2022)	Robot control in multigoal tasks with sparse reward in OpenAI Gym	Exploration Efficiency Sample Efficiency	Defines a variable sampling rate decay strategy for sampling in Hindsight ER to increase the number of experiences and reduce the original method bias.
RHER and SGES (LUO et al., 2023)	Single-object and multi-object tasks from OpenAI Gym.	Exploration Efficiency Aparse Rewards	Proposes a self-guided continual RL framework and a self-guided exploration strategy to decompose a sequential task into several subtasks with increasing complexity.
HCS-R-HER (ZHAO; DU; WANG, 2023)	Hierarchical task control of continuous action space in robots motion.	Exploration Efficiency Aparse Rewards	Proposes a data enhancement algorithm and a two-tier structure containing upper-level and lower-level controllers for decision-making and atomic operations.

Table 10 – Non-Naive ER - Prioritized and Importance Sampling

Reference	Domain	Addressing	Approach Summary
PER (SCHAUL et al., 2016)	Discrete action space on Atari 2600	Sample Efficiency	Prioritizes experiences according to the magnitude of the TD-Error. Combines stochastic sampling priority values and importance sampling strategy.
ERO-ATSC ERO-ATSC+TSC (ZHA et al., 2019)	Pendulum control in continuous action space	Catastrophic Forgetting Data Efficiency	Proposes an ER optimization strategy by using dual memory, an alternating transition selection control, and a transition storage control to mitigate catastrophic forgetting.
AER (SUN; ZHOU; LI, 2020)	Discrete action space on Atari 2600	Data Efficiency	Prioritizes transitions containing states more frequently observed by current policy (by similarity) as an heuristic for sampling.
DisCor (KUMAR; GUPTA; LEVINE, 2020)	Continuous and discrete action spaces in multi -task on Atari 2600	Data Efficiency Sample Efficiency	Proposes modifications in the ADP algorithm to change the buffer's data distribution and reweight the transitions used for training to deal with delayed rewards.
LAP (FUJIMOTO; MEGER; PRECUP, 2020)	Continuous and discrete action spaces on Mujoco and Atari 2600	Sample Efficiency	LAP simplifies PER. Removes unnecessary importance sampling ratios and reduces bias and the likelihood of near-zero-sampling probability experiences. PAL is a LAP-equivalent loss for uniform sampling. It resembles a weighted variant of the Huber loss and computes the same expected gradient as LAP.
Imp-PER (ZHANG et al., 2020)	Discrete and continuous action space on Mujoco and Atari 2600	Data Efficiency Sample Efficiency	Proposes a self-adaptive priority correction algorithm, which predicts the sum of real TD-error of all data in the buffer to define an importance weight. Applies a truncated importance sampling with a self-adaptive truncation threshold.
FER (KONG; NAHRENDRA; PAEK, 2021)	Continuous action space on Mujoco and CARLA	Sample Efficiency	Employs a half-normal sampling probability window that allocates a higher sampling probability to newer experiences in the buffer. Proposes a general and local size adjustment schemes that determine the standard deviation of the half-normal sampling window.
RT-PER (GAO et al., 2021)	Continuous action space in Pendulum control on OpenAI Gym	Sample Efficiency	Combines the reward value with the TD-error to form an experience prioritization parameter and use it as a standard to update and store the memory pool based on the priority.
PERMAB (LIU et al., 2022)	Discrete action space on a Multi-armed bandit mechanism.	Sample Efficiency	Proposes a dynamic ER strategy that can adaptively combine multiple priority criteria to measure the experience's importance. The weight of each assessing criterion can be adaptively adjusted according to their respective contribution to the agent's performance.
FPER (MA et al., 2022)	Discrete and continuous action spaces on OpenAI Gym	Sample Efficiency	Modifies PER based on experience freshness, using a freshness discounted factor as a new hyperparameter and a novel experience replacement strategy in the buffer.
(SINHA et al., 2022)	Online recommendation	Sample Efficiency Sparse Rewards	Proposes a locality-sensitive hashing method to selectively retain the most meaningful experience at scale, together with a reward-driven strategy to prioritize more valuable experiences to deal with sparse rewards.

Table 11 – Non-Naive ER - Prioritized and Importance Sampling (cont.)

Reference	Domain	Addressing	Approach Summary
(CHEN et al., 2022b)	Discrete and continuous action space on Atari 2600 and DeepMind Control Suite	Sample Efficiency	Reweights experiences based on their likelihood under the stationary distribution of the current policy. Uses a likelihood-free density ratio estimator to balance bias from on-policy and variance from off-policy experiences by using the learned ratios as the prioritization weights.
HAER-MADDPG (YU et al., 2023)	Collaborative Multi-robot in a predator-prey game	Sample Efficiency	Uses a hybrid attention-oriented ER in a multi-agent DDPG algorithm, which gives more attention to the states and actions of other robots.
DSEGR (ZHANG; QIU, 2022)	Link stability and energy prediction in UANETs	Sample Efficiency	Models the packet forward as an MDP and uses a DDQN with PER to learn the routing decision process. Designs a reward function for better convergence.
PER-UOS-RL (SHI et al., 2024)	Computational cost optimization in Mobile Edge Computing servers.	Sample Efficiency	Converts the non-convex optimization and high-dimensional continuous action space problem into an MDP. Based on collaborative Multi-Agent RL, proposes a PER mechanism to improve the training efficiency in this scenario.
(PANDA et al., 2024)	Management of batteries' charge-discharge level in renewable energy sources	Sample Efficiency	Investigates and compares the use of different Distributional RL algorithms with PER in a residential PV-microgrid setup. Formulates a non-trivial penalty and reward function and benchmark it based on the battery operations' performance analyses.
SPER-SAC (XU; MA; TAO, 2024)	Industry batch process control	Sample Efficiency Sparse Rewards	Proposes a controller based on Segmented PER (SPER) with SAC to accurately deal with complex nonlinear dynamics and unstable operating conditions, improving the performance of ER in tasks with long episodes and multiple phases. Defines a novel reward function to deal with sparse rewards.
PEPCRL-MVP (LI et al., 2024)	Multi-vehicle Pursuit (MVP) in urban multi-intersection dynamic traffic	Sample Efficiency	Proposes a multi-agent RL with PER and a prioritization network to assess experiences according to each agent's parameters. It employs an attention module to extract critical features from the dynamics of the environments and uses this information to develop a progression cognition method to adaptively group pursuing vehicles.
VPER-VDP3G (LIU et al., 2024)	Coordinated control of coal-fired power generation systems.	Sample Efficiency Data Efficiency	Proposes a dual-prioritized experience replay and a value distribution strategy to reduce nonuniform sampling bias, remove redundant data, enhance sample diversity, and improve the accuracy of the cost function.

Table 12 – Non-Naive ER - Prioritized and Importance Sampling plus Proposition on DNNs

Reference	Domain	Addressing	Approach Summary
Dueling Network Architectures (WANG et al., 2016)	Discrete action space on Atari 2600.	Data Efficiency	Uses two streams in the network to approximate the state-value function and a state-dependent action advantage function, whose outputs are the state-value and a vector of advantages for each action. Combining these outputs produces final Q-value.
DNPER-DDPG (KANG et al., 2021)	Continuous action space on OpenAI Gym	Sample Efficiency	Proposes a double network PER mechanism in which the minimum estimates of the action-value functions are used to update the actor policy. Uses the Q-values from the two networks and the immediate reward as the criteria for prioritization. Divides the importance of the samples to improve the convergence time.
AW-PER-A2C (ZHOU et al., 2022)	Robotic motion planning	Sample Efficiency	Represents the features of the robot and the obstacles using an encoder attention weight and combines online and offline A2C with PER.
DDQN-PER (ZHANG et al., 2023)	Power consumption planning and collaborative charging control strategy for electric vehicles	Sample Efficiency Sparse Rewards	Proposes a charging control strategy based on DDQN with PER, which employs an LSTM to address the uncertainties caused by power requirements.

Table 13 – Non-Naive ER - Prioritized and Importance Sampling plus Strategies for Exploration

Reference	Domain	Addressing	Approach Summary
DRL-QER (Wei et al., 2021)	Discrete-valued action space on Atari 2600	Exploration Efficiency	Adaptively replays experiences according to their complexity and a replaying count to balance exploration and exploitation. Formulates the experiences in quantum representations. An operation of preparation reflects the relationship between the TD-errors and the importance of the experiences. In contrast, an operation of depreciation ensures the transitions' diversity.
(NICHOLAUS; KANG, 2022)	Multi-Agent Cooperation	Exploration Efficiency	Presents a technique that finds suitable experiences in a multi-agent scenario by implicitly introducing additional strength to the exploration-exploitation trade-off. It filters the experiences by computing the similarities between the observed state and previous states in the experiences. This value is used as a hyperparameter to decide which experiences will suit.
CTER (DONG; LI; GONG, 2023)	Wargaming decision modeling Continuous action space on OpenAI Gym	Sample Efficiency Sparse Rewards Exploration Efficiency	Proposes an assembled curiosity-based intrinsic reward that applies decaying and normalizing factors to guide the agent in exploring feasible paths under sparse rewards. Develops a curiosity-adjusted partially greedy exploration to control the ϵ-greedy policy adaptively. Proposes a novel curiosity-augmented sampling technique to prioritize the experience replay and a novel K-segmented curiosity-balanced experience memory updating approach.

Table 14 – Non-Naive ER - Prioritized and Importance Sampling plus Buffers or Experiences Modeling

Reference	Domain	Addressing	Approach Summary
LASER (SCHMITT; HESSEL; SIMONYAN, 2020)	Discrete-valued action space on Atari 2600	Sample Efficiency	From the analyses of the bias-variance trade-offs in the form of importance sampling for actor-critic, called V-trace, proposes a new trust region scheme. It mixes experience samples with on-policy experience that scales effectively to data distributions where V-trace becomes unstable.
DBER (CUI et al., 2023)	Visual an sensor data on multi-input autonomous driving	Sample Efficiency	Proposes a double-bias experience replay and a new loss function to balance negative and positive loss. Stores nonuniform samples using a biased storage strategy, which learns the weight of samples on a large buffer for positive samples and a smaller buffer for negative samples to separate the types of learning samples. The learning weight of negative samples experience is degraded in the process of loss function calculation.

Table 15 – Non-Naive ER - Prioritized and Importance Sampling plus Stratifying and Balancing Experiences

Reference	Domain	Addressing	Approach Summary
DQN-BSPER (WANG et al., 2024)	Class-imbalanced customer credit score in P2P lending.	Sample Efficiency	Proposes a balanced stratified prioritized ER strategy to optimize the loss function of a DQN model. The objective is to balance the numbers of minority and majority experience samples in the mini-batch (according to the class representation) and select more important experience samples based on the principles of PER.

Table 16 – Non-Naive ER - Prioritized and Importance Sampling in Essemble or Multi-strategy Methods

Reference	Domain	Addressing	Approach Summary
Rainbow (HESSEL et al., 2018)	Discrete action space on Atari 2600	Sample Efficiency Data Efficiency Exploration Efficiency	Adapts PER to use the KL loss of the Distributional Q-Learning. Replaces the one-step distributional loss with a multi-step variant. Defines the target distribution by contracting the value distribution and shifting it by the truncated n-step discounted return. Combines the multi-step distributional loss with Double Q-Learning. Uses the greedy strategy to select and evaluate the action using the online and the target networks. Adapts the dueling architecture for return distributions, uses noisy layers, and factorizes Gaussian to reduce the number of independent noise variables.
R2D2 (KAPTUROWSKI et al., 2019)	Discrete action space on Atari 2600	Data Efficiency	Investigates RNNs with ER and demonstrates its effect on the parameter lag, which leads to representational drift and recurrent state staleness, mainly in the distributed training setting. Presents an empirical study into the effects of several approaches with RNN, mitigating these effects. Combines RNN to learn representations, DDQN with n-step returns, dueling networks with an additional LSTM layer, Prioritized Distributional ER, and experiences from agents training in parallel.

Table 17 – Non-Naive ER - Prioritized and Importance Sampling - Theoretical-empirical studies

Reference	Domain	Addressing	Approach Summary
(FEDUS et al., 2020)	Discrete action space on Atari 2600	Replay Buffer Size Sample Efficiency	Presents a systematic analysis of ER in Q-learning methods, focusing on replay capacity, replay ratio, and n-step returns. Demonstrates that greater capacity increases the performance of some algorithms but not others and that the theoretically ungrounded, uncorrected n-step returns are uniquely beneficial. In contrast, other techniques confer limited benefits for sifting through larger memory.

Table 18 – Non-Naive ER - Buffers or Experience Modeling

Reference	Domain	Addressing	Approach Summary
CER (ZHANG; SUTTON, 2017)	Discrete action space on Atari 2600	Buffer Size Data Efficiency	Proposes a new hyperparameter for the buffer size and demonstrates that both a small and a large buffer can heavily hurt the learning process. Remediates the negative influence of a large buffer by adding the current transition to the sampled batch and using the corrected batch to train the agent.
CLEAR (ROLNICK et al., 2019)	DeepMind Lab tasks	Catastrophic Forgetting	Introduces a method that reduces catastrophic forgetting in multi-task RL by leveraging off-policy learning, behavioral cloning from replay to enhance stability, and on-policy learning to preserve plasticity, which does not require any knowledge of the tasks being learned.
ReF-ER (NOVATI; KOUMOUTSAKOS, 2019)	Continuous action space on OpenAI Gym	Data Efficienncy	Computes the similarity between the current transition and past experiences to skip gradients from those too unlikely with the current policy. It regulates policy changes within a trust region of the replayed behaviors, limits the fraction of far-policy samples in the buffer, and computes gradient estimates only from near-policy experiences.
SLER (LI et al., 2021)	Multiple tasks in sequence.	Catastrophic Forgetting	Proposes a short-term ER to retain current task experience and a long-term ER to keep all past experiences to achieve continual learning. A sample model generates the simulated state sequence of the previous tasks for knowledge retention. This is combined with the experience of the new task, generating the simulation of the experience for all previous tasks to alleviate forgetting.
MSER (YANG; PENG, 2021)	Trajectory planning of robot manipulator	Sample Efficiency	Proposes a meta-learning-based ER buffer separation method that divides the original buffer into a successful experience buffer and a failure experience buffer, with samples randomly from the two buffers according to a ratio learned through a neural network.
XAER (SOVRANO; RAYMOND; PROROK, 2022)	Discrete and continuous navigation in rule-dense and exception-ridden environments	Sample Efficiency	Presents a method for partitioning the buffer into clusters labeled on a per-explanation basis in discrete and continuous navigation environments compatible with modular rulesets. Converts rule-based explanations into case-based explanations and samples experiences in a curricular and task-oriented manner.
EIT (ANZALDO; ANDRADE, 2022)	Power allocation algorithms	Data Efficiency Catastrophic Forgetting	Proposes a strategy based on a dual memory to mitigate catastrophic forgetting. It exploits previously trained and current data knowledge of the distributional RL model and evaluates the effects of additional training by reusing knowledge from the source model to train new models.

Table 19 – Non-Naive ER - Buffers or Experience Modeling (cont.)

Reference	Domain	Addressing	Approach Summary
LiDER (DU et al., 2022)	Discrete action space on Atari 2600	Sample Efficiency	Refreshes the experiences in the buffer in a multi-agent training by moving the agent back to a past state, executing a sequence of actions by following its current policy, and storing and reusing the new experiences if they are better than previously experienced.
TD3_CER (LI; HUANG; ZHU, 2022)	Continuous action space on Mujoco in OpenAI Gym	Exploration Efficiency	Exploits the experiences by clustering and replaying them using a divide-and-conquer framework that divides the training process into several periods. It clusters the transitions at the end of each period and defines a conditional probability density function to ensure that all transitions are sufficiently replayed in the current training.
ERO-ATSC (OSEI; LOPEZ, 2023)	Continuous action space in OpenAI Gym	Sample Efficiency Data Efficiency Catastrophic Forgetting	Proposes an ER optimization method combining a replay policy network, dual memory, and an alternating two transition selection control mechanism. The first is a hybrid of ER optimization and dual memory, and the second integrates a transition storage control into the first one.

Table 20 – Non-Naive ER - Buffers or Experience Modeling plus Propositions on DNNs

Reference	Domain	Addressing	Approach Summary
COMPER (NEVES; ISHITANI; PATROCÍNIO JR, 2022)	Discrete action space in Atari 2600	Replay Buffer Size Data Efficiency	Proposes a method to reduce the convergence time regarding the total number of observations or agent training iterations. Uses TD-Learning with predicted target values for similar transitions sampled from a Compact Transitions Memory, which an LSTN generates by reducing a Transition Memory modeled as sets of similar transitions. This LSTM learns the similar transition sets dynamics regarding its respective observed Q-values to predict the Q-target value at the Q-value function update.

4 EXPERIENCE REPLAY WITH COMPACT MEMORY

Recalling Section 1.2, we stated that Experience Replay (ER) has three main limitations: (i) the memory of experiences does not differentiate relevant experiences due to the uniform sampling; (ii) experiences are often overwritten due to the buffer size limitation, leading to the loss of potentially relevant experiences; and (iii) the arbitrally and fixed size of the buffer makes ER less data efficiency in reusing experiences to update the value functions. From that, we identified three exciting challenges: (i) the picking of what experiences to repeat, (ii) the selection of which one to store, and (iii) how to deal with the threshold regarding buffer sizing.

To address the last two challenges in discrete and continuous action problems, we propose and evaluate two new methods called COMPact Experience Replay (COMPER) and COMPact Memory for Continuous ACTions (COMCACT). Both methods seek to reduce the required number of experiences for agent training regarding the total accumulated rewards in the long run. They use temporal difference learning with predicted target values for sets of similar transitions and a new experience replay approach based on two memories of transitions. Their main differences are that COMPER implements a DQN-based strategy and COMPact Memory for Continuous ACTions implements a DDPG-based strategy.

In Section 4.1, we formally present the formulations and components of COMPER. In Section 5, we present the methodology of experiments and the empirical results that demonstrate how the proposed method proves the hypothesis and how the research developed as part of this work fulfilled all the objectives presented by making ER more data-efficient using smaller amounts of data. This section presents its foundations and their formal definitions.

The empirical results demonstrate that COMPER can approximate a good policy from a few video-frame observations in learning to play digital games, and COMCACTcan speed up learning with few interactions in learning control in physical problems.

4.1 COMPact Experience Replay (COMPER)

COMPER 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 called *Reduced Transition Memory* ($\mathcal{RTM}$). The goals behind that are two-fold: (i) allowing rare (and possibly expensive) experiences to be sampled more frequently; and (ii) decreasing the amount of memory needed by a large buffer without hampering the convergence time. To achieve that, COMPER first

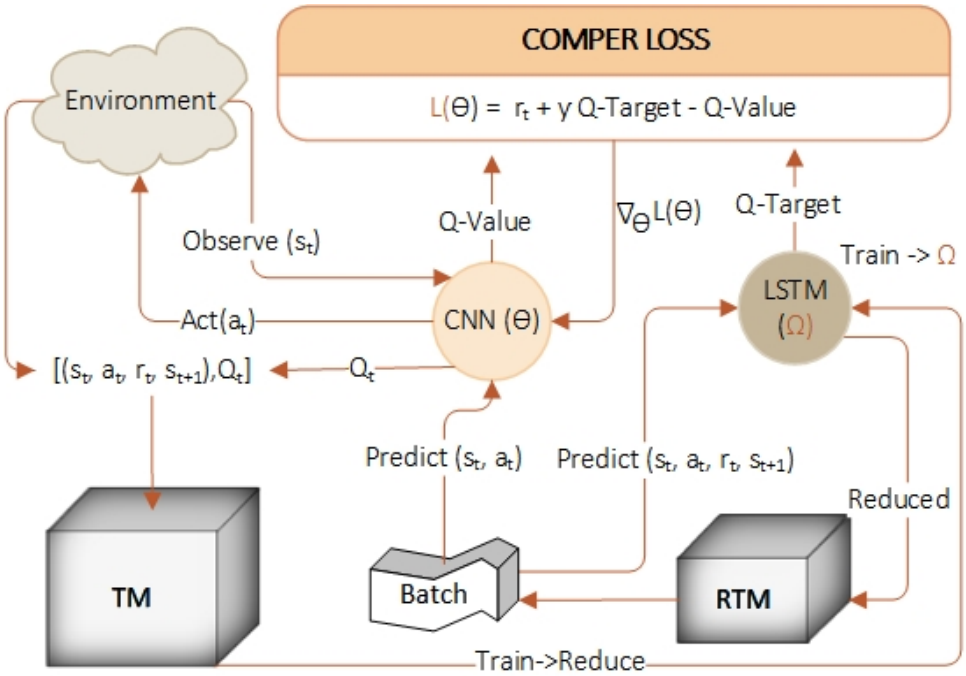


Figure 2 – General COMPER flow and main components

stores 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 (as explained in Section 4.1.1) to generate a more compact (but still very helpful) 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,

$$\mathcal{L}_{COMPER}(\Theta_i) = \mathbb{E}_{\tau_t=(s_t,a_t,r_t,s_{t+1})\sim U(\mathcal{RTM})}\left[(r_t + \gamma\, QT(\tau_t, \Omega) - Q(s_t, a_t, \Theta_i))^2\right], \qquad (4.1)$$

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 another DNN and parameterized by Ω. This function provide the target value and is updated in a supervised way from the $\mathcal{ST}$s stored in $\mathcal{TM}$. Thus, this DNN is also used to build a model to generate the compact structure of $\mathcal{RTM}$ from $\mathcal{TM}$, while seeks to learn the dynamics of $\mathcal{ST}$s to provide better target values at the next agent update step. Figure 2 illustrates how COMPER components interact with each other.

4.1.1 Modeling the Transition Memory

During training, an agent experiences successive transitions through its interactions with the environment, along with respective rewards. So, these transitions can be stored

in a $\mathcal{TM}$ to compose a history of the expected Q-values (analogously to an "*extended*" replay buffer). During agent training runs, each episode starts at an initial state and finishes at a final one. After that, another episode begins. So, transitions with similar state representations may occur and be stored more than once over subsequent episodes. Moreover, we can use sampled transitions (during Q-value function updates steps, when TD-error is computed) and reinsert them in $\mathcal{TM}$ along with its new Q-value estimates. This way, probably several transitions in $\mathcal{TM}$ could be similar in their components (previous state, selected action, immediate reward, and resulting state) but with different Q-value estimates. The criterion (presented ahead) adopted to consider transitions as similar ones is decisive for modeling $\mathcal{TM}$ as a set of $\mathcal{ST}$s. That allows COMPER to predict the Q-values associated with those sets through a DNN and use this estimate as the target value in the next Q-function update step.

4.1.1.1 Similar Transitions Sets

At each training time-step t, we define a transition 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. Consider two transitions $\tau_{t_1} = (s_{t_1}, a_{t_1}, r_{t_1}, s_{t_1+1})$ and $\tau_{t_2} = (s_{t_2}, a_{t_2}, r_{t_2}, s_{t_2+1}), t_1 \neq t_2$. We say that those transitions are similar ($\tau_{t_1} \approx \tau_{t_2}$) when the 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 $\mathcal{D}$ could be any distance measure, e.g., Euclidean distance, and δ is a distance (or similarity) threshold value. Let be N the total number of transitions that occurred up to a time instant. Those N transitions are stored in $\mathcal{TM}$ and can be identified as subsets of similar transitions $\mathcal{ST}$ when the similarity condition is satisfied. Moreover, they are stored throughout subsequent agent training episodes and identified by a unique index. Therefore, we can define $\mathcal{TM} = \{[T^i, \mathcal{ST}_i] \,|\, i = 1, 2, 3, \ldots, N_{ST}\}$, 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,

$$\mathcal{ST}_i = \left\{ \left[\tau_{i(1)}, Q_{i(k)}\right] \mid 1 \leq k \leq N^i_{ST} \right\} \tag{4.2}$$

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 a DNN.

4.1.1.2 Reduced Transition Memory

From $\mathcal{TM}$, one can produce a $\mathcal{RTM}$ in which τ'_i is the transition that represents all the similar transitions so far 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 about sets of similar transitions, since each τ'_i is unique and represents all the transitions in a given $\mathcal{ST}_i$. 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 increasing diversity (because of variability in each sample).

4.1.2 Identifying and Indexing Similar Transitions

We identify similar transitions by defining a *Transition Memory Index* ($\mathcal{TMI}$) as part of $\mathcal{TM}$. At each training time-step t we insert a transition τ_t together with its estimated Q-value (Q_t) into $\mathcal{TM}$ and check the $\mathcal{TMI}$ for a similar transition given a minimum distance (or similarity) threshold δ. If there is a similar transition, $\mathcal{TMI}$ will return its unique numerical identifier T^i. Otherwise, we insert τ_t together with Q_t into $\mathcal{TMI}$ and get the corresponding T^i. So, a given T^i is used to find the first $\tau_{i(1)}$ similar transition in $\mathcal{TM}$ and update its Q-value, or to insert τ_t as the first similar transition, together with its first Q_t, identified by T^i. The set of subsequent transitions identified as similar under the same identifier T^i are what we call $\mathcal{ST}_i$. But we do not need to store the entire $\mathcal{ST}_i$. We can use its corresponding T^i to locate its representing transition tuple and update its associated Q-value, considering that its previous value contributions were learned by the Q-target network. Algorithm 6 shows details of that procedure. To implement $\mathcal{TMI}$, we used a library for efficient similarity search and clustering of dense vectors called Faiss (DOUZE et al., 2024), which we describe in Appendix A.1.

Algorithm 6: $\mathcal{TM}$-StoreTransition

input: transition $[\tau_t, Q_t]$, distance (or similarity) threshold value δ

```
T^i ← TMI.GetIndex(τ_t, δ);
if T^i == 0 then
    T^i ← TMI.UpdateIndex(τ_t);
    TM.Insert(T^i, [τ_t, Q_t]);
else
    TM.Update(T^i, Q_t);
```

COMPER (see Algorithm 7) uses two memories, named Transitions Memory ($\mathcal{TM}$) and Reduced Transitions Memory ($\mathcal{RTM}$). A CNN approximates the action-value function Q to estimates the Q-values (Q_{value}) from a sample of transitions stored in $\mathcal{RTM}$. Another component named QT uses a DNN both to produce the $\mathcal{RTM}$ from $\mathcal{TM}$ and to predict the Q-target value (Q_{target}).

Algorithm 7: COMPact Experience Replay - COMPER

input: batch size k, learning rate α, trainging 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$\big(s_t, \epsilon, \Theta\big)$;

The agent observes the current environment state (s_t) at each iteration (t) and selects an action (a_t). Then, it performs a_t on s_t, receives an immediate reward (r_t), observes a new state (s_{t+1}) – that becomes the current state s_t to the next iteration, and an estimate of Q_{value}. These elements make up the transition τ_t which is indexed by the transitions memory index ($\mathcal{TMI}$) and stored in $\mathcal{TM}$. QT is updated and used to produce $\mathcal{RTM}$ from $\mathcal{TM}$ at a frequency defined by the hyper-parameter utf. In turn, Q is updated at a frequency defined by the hyper-parameter tf, and this process consists of: (i) performing a uniform sampling of transitions from $\mathcal{RTM}$; (ii) predicting the Q_{target} using QT; and (iii) estimating the Q_{value} using Q. The TD-error is calculated from these estimates and used to update Q through backpropagation.

First, if $\mathcal{RTM}$ is empty or depending on utf, QT is updated, that is, its DNN

is trained, and then it is used to update $\mathcal{RTM}$. So, a batch of transitions (with size k) is sampled from $\mathcal{RTM}$ and QT is used to predict the Q_{target} values for each sampled transition. Then, Q is used to estimate the Q_{values} from the state s_t and action a_t for each sampled transition. The TD-error is calculated from Q_{value} and Q_{target}. A gradient vector as a function of TD-error is then obtained and accumulated in a vector Δ. After iterating over all the sampled transitions (which is always done in parallel), the parameters Θ are updated at a learning rate α.

4.2 COMPact Memory for Continuous ACTions (COMCACT)

The propositions in COMPER approach relevant issues in the literature regarding data efficiency when, for example, it composes a smaller but not explicitly size-limited replay buffer. Moreover, it demonstrates that agents could learn not only from past experiences but also, and many times better, from past and similar experiences (showing that they occur) and that exploiting these similarities can improve the agent learning process. However, it was necessary to understand better how it contributes to the performance improvements in reinforcement learning and if it could aggregate some benefit in data efficiency in other methods, domains, and different problems. Therefore, we conducted a detailed analysis of its components to verify and understand how they operate and how they would improve data efficiency in environments with continuous-valued actions. From that, and based on the proposition of Lillicrap et al. (2016) for the Deep Deterministic Policy Gradient (DDPG), we develop a method called COMPact Memory for Continuous ACTions - COMCACT.

In COMCACT we employ an actor-critic strategy, in which we define a predicted target value to make additional updating steps for the model that approximate the critic function model, aiming to speed up the agent training process. We evaluated it by comparing its results with the results of a DDPG agent. Both agents were trained over a set of challenges on the environments of Mujoco (TODOROV; EREZ; TASSA, 2012) and Classical Control problems in the platform of Gymnasium (BROCKMAN et al., 2016).

By comparing DQN, COMPER and DDPG in Section 2, one can note that its Q-value functions (the critic, in DDPG) are equivalent and use the same formulation, which we can define as $Q(s, a, \theta)$. Moreover, the three methods update that function using temporal difference learning, in which the difference is in the target function. DDPG first updates the critic function using its target function $Q'(s, a, \theta')$ and then updates that target function from the critic function. In COMCACT we define the same TD-error calculation and loss as in COMPER and apply a step between the update of the critic and the update of its respective target function in which we use the Q-target function as defined in COMPER first to update the critic and, consequently, improve the update

of its target function in the next step without introducing some instability. In synthesis, COMCACT strategies of COMPER for ER combined with a new algorithm to update the critic function from the TD-error between its estimates and the estimates from a third target function (see Algorithm 8).

We address the problem of exploration introducing noise into actor policy at each agent interaction step with the environment. We sample this noise from a function independent of the agent learning that applies the Ornstein-Uhlenbeck process (UHLENBECK; ORNSTEIN, 1930) (which generates temporally correlated exploration to improve efficiency in physical control problems) such as proposed by Lillicrap et al. (2016). Just after the agent performs the current action, we compose the tuple of the transition in the same way as in COMPER and store it in $\mathcal{TM}$, considering the similar transition sets ($\mathcal{ST}$). After that, the following steps are executed: (1) update the actor and the critic using their target functions and minimizing the TD error, such as in DDPG, but using uniform samplings of transitions stored in the $\mathcal{RTM}$; (ii) update the Q-target function (similar to COMPER) by (uniform) sampling a maximum of 50,000 similar transitions removed from the $\mathcal{TM}$, training the LSTM, and finally updating the $\mathcal{RTM}$ (also similar to COMPER); and (iii) update the targets functions of the actor and the critic such as in DDPG. This new approach is named COMCACT. The details regarding the neural network architectures and the hyperparameters are in Appendix B.

Algorithm 8: COMPact Memory for Continuous ACTions - COMCACT

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

Initialize $TM \leftarrow \emptyset$, $RTM \leftarrow \emptyset$; Initialize Θ^Q, Θ^μ, Ω 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**
    $a_t \leftarrow \mu(s_t \mid \Theta^\mu) + \mathcal{N}_t$;
    $s_{t+1}, r_t, Q_t \leftarrow env.step(s_t, a_t)$;
    $Q_t \leftarrow Q(s_t, a_t, \Theta^Q)$;
    $\mathcal{TM}.StoreTransition([\tau(s_t, a_t, r_t, s_{t+1}), Q_t], \delta)$;
    $L \leftarrow 0$;
    $P \leftarrow 0$;
    $TransitionList \leftarrow \emptyset$;
    for $j \leftarrow 1$ **to** k **do**
      if $\mathcal{RTM} \neq \emptyset$ **then**
        $\tau = (s_t, a_t, r_t, s_{t+1}) \sim UniformSample(\mathcal{RTM})$;
      else
        $\tau = (s_t, a_t, r_t, s_{t+1}) \sim UniformSample(\mathcal{TM})$;
      $Q_{target} \leftarrow Q'(s_{t+1}, \mu'(s_{t+1} \mid \Theta^{\mu'}), \Theta^{Q'})$;
      $Q_{value} \leftarrow Q(s_t, a_t, \Theta^Q)$;
      $\mathcal{L}(\Theta^Q) \leftarrow (r_t + \gamma \times Q_{target} - Q_{value})^2$;
      $L \leftarrow L + \mathcal{L}(\Theta^Q)$;
      $P \leftarrow P + \nabla_a Q(s, a \mid \theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s \mid \theta^\mu)|s = s_t$;
      $\Delta \leftarrow \frac{1}{k} L \times \bigtriangledown_{\Theta^Q}$;
      $TransitionList.add(\tau)$;
    $\Theta^Q \leftarrow \Theta^Q + \alpha \times \Delta$;
    $\Theta^\mu \leftarrow \Theta^\mu + \alpha \times \nabla_{\Theta^\mu} J \approx \frac{1}{k} P$;
    $\Omega \leftarrow Train(QT, \mathcal{TM})$;
    $\mathcal{RTM} \leftarrow$ProduceRTM$(QT, \mathcal{TM}, \Omega)$;
    for each $\tau \in TransitionList$ **do**
      $Q_{target} \leftarrow QT(\tau, QT, \Omega)$;
      $Q_{value} \leftarrow Q(s_t, a_t, \Theta^Q)$;
      $\mathcal{L}(\Theta^Q) \leftarrow r_t + \gamma \times Q_{\text{target}} - Q_{\text{value}}$;
      $\Delta \leftarrow \Delta + \mathcal{L}(\Theta^Q) \times \bigtriangledown_\Theta \Theta^Q{}_{\text{value}}$;
    $\Theta^Q \leftarrow \Theta^Q + \alpha \times \Delta$;
    $\Theta^{Q'} \leftarrow \tau\theta^Q + 1(1-\tau)\Theta^{Q'}$;
    $\Theta^{\mu'} \leftarrow \tau\theta^\mu + 1(1-\tau)\Theta^{\mu'}$;
    $s_t \leftarrow s_{t+1}$;

5 COMPER – METHODOLOGY AND EMPIRICAL RESULTS

To evaluate COMPER concerning the objectives presented in Section 1.4 and verify if its results support the hypotheses discussed in Section 1.3, we conducted experiments on various tasks in an environment of off-policy and model-free learning to Atari 2600 games with discrete-valued actions. The details about deep neural networks and hyperparameters are in the Appendix A.

5.1 The Arcade Learning Environment

The Arcade Learning Environment (ALE) (BELLEMARE et al., 2013; MACHADO et al., 2018) consists of a development platform and a set of challenges for RL agents involving problems such as non-determinism, stochasticity, and environments with different values and ranges for awarding rewards. According to Machado et al. (2018), relevant works in the literature generally use different methodologies, making it challenging to analyze and compare their results. Therefore, the authors proposed a very well-defined methodology for agent evaluation using ALE, and presented a benchmarking with the experimental results for DQN (MNIH et al., 2015) and SARSA(λ)+ blob-PROST (LIANG et al., 2016) agent. From that, we highlight the following propositions as a premise for our experiments:

- Agents should be evaluated over the training data and not in "test mode", because we are interesting to evaluate their learning processes, not their gaming performance.
- The agent's performance should be evaluated on different training checkpoints considering the last episode rewards.
- Agents interact with the game environment in an episodic way. An episode begins by putting the environment in its initial state and ends in a natural final state for the game.
- The main measure of the agent performance consists of the undiscounted sum of the rewards for each episode.
- Non-determinism and stochasticity are fundamental issues of reinforcement learning and are explored in ALE using two techniques called *stick actions* and *frame skipping*. We use both with the same configuration as Machado et al. (2018).
- Three approaches are commonly used in the literature to represent the states of the environment: (i) *color averaging*, (ii) *frame pooling*, and (iii) *frame stacking*. ALE implements color averaging. Frame stacking is not implemented directly by ALE and is an algorithm decision. Color averaging and frame pooling may remove the

most interesting form of partial observation in ALE, and frame stacking reduces the degree of partial observability. We use color averaging and make experiments using single frames (that is, not stacked) and stacked frames.

- There are differences in the literature on how to summarize and measure the results and how the agent performance is evaluated based on these results. These questions lead to an important methodological decision, which refers to when each training episode must end. We consider only the signal of "game over" as the end of an episode, as strongly recommended by Machado et al. (2018).

- How results are measured and summarized can hamper or improve the analysis of an agent learning process. Machado et al. (2018) recommends analyzing results on a fixed number of the last episodes at different times, or checkpoints, during agent training to evaluate the learning progress. The authors also pointed out that a total number of 200 million frames has been adopted in the literature, which implies a great computational cost and a lot of time in experiment execution. Therefore, their methodology allows any researcher to report results earlier during experiments.

- Training an agent over a given number of episodes before evaluating it can yield misleading results, as the duration of each episode can vary widely between different games and as a function of the agent's performance. A more interesting approach is to measure the amount of training data in terms of the total number of environment states (i.e., the video frames), which makes reproducing experiments and comparing the results more accessible.

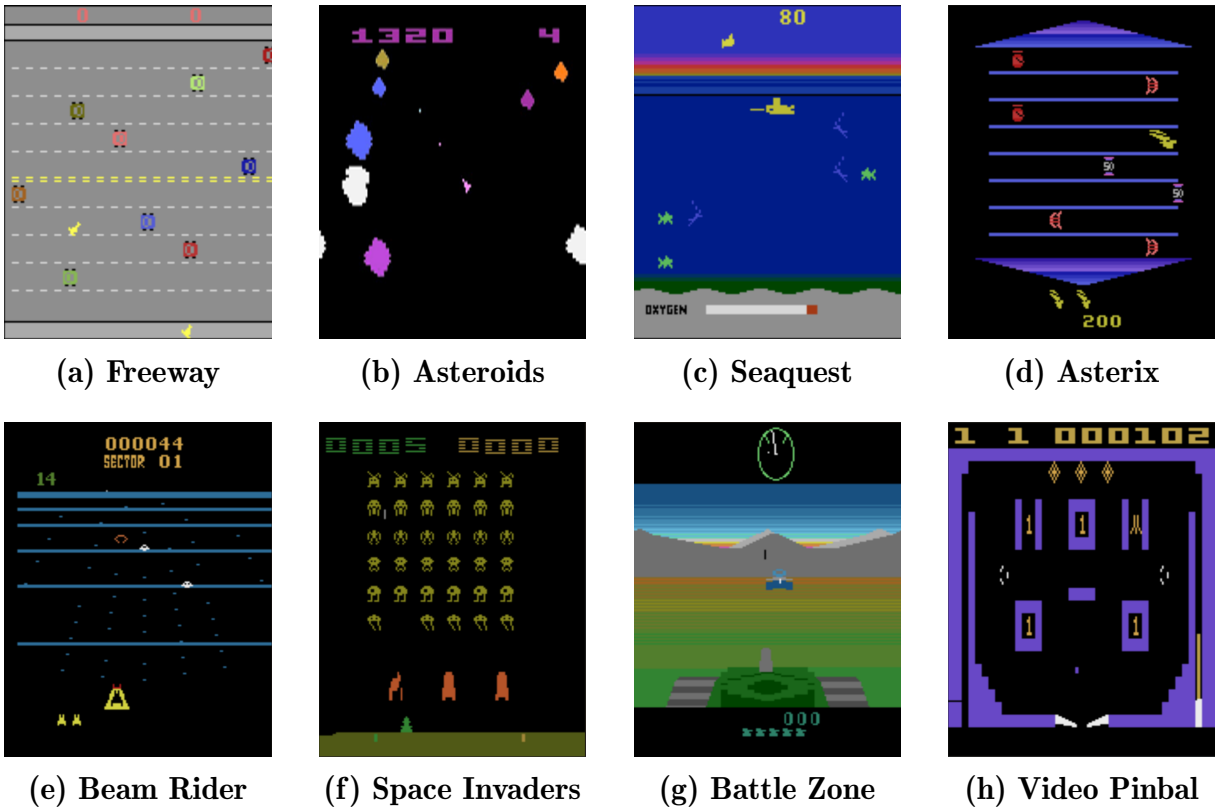
- Interrupting an episode as soon as the total number of frames is reached or waiting for the end of the episode does not represent a clear choice in the literature. We end the training based on a total number of frames and without interrupting the episode until the game over signal as recommended by Machado et al. (2018).

- To evaluate the agent's generalization capacity, only a smaller subset of games should be used to select and adjust hyperparameters, and the agent should be evaluated using the defined values (and without further tuning) over the entire set of games.

- In the ALE, the agent can access a vector with all possible actions independently of the game or the specific actions for each game. Since we pursue generalization, we always use the complete set of actions, and the agent must learn which of them are helpful or not in each game.

5.1.1 Games used for Agent Training and Evaluation.

This section briefly describes the games illustrated in Figure 3, and we used them in the experiments.



(a) Freeway **(b) Asteroids** **(c) Seaquest** **(d) Asterix**

(e) Beam Rider **(f) Space Invaders** **(g) Battle Zone** **(h) Video Pinbal**

Figure 3 – Atari 2600 - Emulated on ALE

In the game Freeway (Figure 3a), a chicken must cross an avenue with many vehicles moving in different positions and speeds until it reaches the other bank without being run over. The chicken starts at the button-side promenade and runs toward the top promenade. The possible actions are to go up, down, right, or left, and there is a gameplay time counting. The agent receives one point as a reward only when the chicken reaches out to the other side by completely crossing the avenue and avoiding all the vehicles, and this configures a sparse-rewarding scenario. On the other hand, every time a vehicle hits the chicken, it is thrown back to the bottom promenade without receiving any penalty or reward. As the agent depends on the reward to evaluate his policy, these rewarding mechanics make Freeway a challenging game.

In Asteroids (Figure 3b), the agent controls a spaceship and fires at various asteroids that move in different directions and shrink in size when hit, and the objective is to destroy as many as possible, avoiding collision. Each hit is rewarded, and each collision reduces the number of agent's lives. The reward is inversely proportional to the size of

the asteroid. The spaceship can maneuver in all directions, rotate 360 degrees to the left or right, and fire a single projectile at a time from the tip of its triangular shape. As asteroids decrease in size and increase in distance, while the projectile's velocity remains constant, the agent must carefully determine both the position of the spaceship and the timing of their shot to ensure the projectile intercepts the asteroid at some point along its path. When the spaceship's movement is interrupted, or its direction is suddenly changed, the game simulates a slight sliding effect, allowing for adjustments in position. This game presents several exploration challenges. For instance, the agent must decide which asteroid to target at any given moment and the optimal movements to make.

In the game Seaquest (Figure 3c), a submarine must rescue divers from the ocean. Many obstacles and opponents move horizontally, on different rows, appearing on one side of the video and disappearing on the other. The possible actions are to go up, down, right, left, or fire. There is a counting of remaining oxygen when the submarine submerges and is restarted when the submarine emerges. The agent needs to identify the position of each obstacle or opponent to destroy or avoid them, as well as each diver, which it must collect to bring to the surface, in a very complex dynamic. There are differences in the value of penalties or rewards for each element that hits the submarine and is destroyed by the submarine's firing. The highest reward is obtained after the submarine has collected the most significant number of divers and brought them to the surface in the shortest possible time, represented by the remaining oxygen supply. The remaining oxygen is converted into rewards and added to the total reward. It is noticed that the tasks of making a trajectory between obstacles, reaching divers, and then taking them to the surface to obtain a more significant reward require learning a long-term action policy.

In Asterix (Figure 3d), the agent must collect objects like helmets, fish, and apples, each worth different points. The player starts with three lives and earns points for every object collected, with bonus lives awarded at specific scores. The game becomes progressively challenging as objects move faster. These elements move in horizontal lines and are basically of two types: the ones that penalize and the others that reward and are never present simultaneously in the same line. Thus, the agent needs to locate which line each type of element moves in and intercept a rewarding one at the correct line, avoiding that which penalizes. This game is challenging because all these elements appear and disappear randomly at different lines and positions in short periods and a relatively small game scenario. Thus, the expected total reward seems to be related more to the number of observed rewarding elements than to learning a long-term action policy, as in the case of the game Freeway, imposing more challenge to the agent's stochastic decision-making capacity.

Beam Rider (Figure 3e) is another spaceship game; the scenario is in perspective, and the opponents appear in the upper, more distant portion of the scene and can move

in any direction. The agent remains in the bottom portion, moving only from left to right and shooting at opponents, who also shoot at it. The perspective is reinforced by increasing and decreasing the opponents' sizes as they approximate or distantiate from the spaceship the agent controls. At the beginning of the game, fewer opponents are simultaneously present in the scene, but their numbers increase gradually. More distant and smaller enemies are more difficult to hit due to a bounding box collision and the time the fired projectile spends to reach the target, which is ever-moving. Thus, at the beginning of the game, the agent is less likely to be hit quickly, letting him more time to learn the complex dynamic of avoiding fire, learning trajectories, and deciding what opponent to fire first; the closer, the more dangerous, but the easier to hit. This dynamics presents a good scenario for learning to plan.

In Space Invaders (Figure 3f), enemies at the top fire against the agent's spaceship at the bottom, which can hide behind four barriers disposed side-by-side horizontally in the line above the line of the agent's spaceship, and both the enemies and the agent only move horizontally. Many enemies are already covering the middle-top portion of the screen, from side to side, at the beginning of the gameplay. Therefore, as the enemies and the agent move and fire projectiles at all times, the chances of the agent hitting an opponent and receiving a reward are very high at all times, as well as the chances of being hit and penalized by losing a life, especially when it is still exploring at the beginning of the learning process. Therefore, the big challenge seems more related to learning to survive and planning what time to hide, move, and fire. Because the chances of an agent being hit very early are so high, it requires the agent to observe a lot of frames until they learn to equalize "survival" and "hit the target".

Battlezone (Figure 3g) dynamics are similar to Beam Rider, with the enemies getting closer or more distant to the agent and in size increasing and decreasing in response. It is one of the first 3D arcade games for first-person shots. The agent controls a war tank that is ever-moving forwarding and can rotate one hundred and eighty degrees, firing against the enemies, which are tanks, missiles, and UFOs. These enemies become progressively more challenging as the game progresses. A small black vertical line on the screen defines the agent's tank's crosshairs, located in the line above the enemy line and lighting up when an enemy is in the crosshairs. A circular radar that divides the scene into quadrants marks the location of enemies on the map and is displayed in the upper portion of the game screen above the battle scene. Points are awarded for destroying enemy tanks and UFOs. Bonus points are given for shooting UFOs.

Video Pinball (Figure 3h) simulates a classic pinball machine. The agent must throw the ball and control the flippers to keep it in the scene for as long as possible while bouncing it toward different elements. The scenario includes several typical elements of a pinball machine, such as flippers, bumpers, spinners, and passages, and each element hit

by the ball produces a different reward value in addition to bonus elements and multipliers that increase the score. The agent is penalized when the ball passes directly through the flippers without being able to bounce it back. In addition to the ball's trajectory directly depending on the position and angle at which it hits the flippers when they bounce it again toward the game elements, some of these elements repel the ball, whose direction also depends on the location and angle at which the ball hit them. In this way, the ball's trajectories bring an extra load of non-determinism and randomness, imposing a complex problem of balancing exploration and exploitation on the agent. There is also a considerable variation in the intervals between one reward and another, as the agent may reach a bonus element as soon as he hits the ball or bounces the ball several times without reaching any rewarding element.

5.2 General Parameters of ALE and the Evaluation Protocol

Table 21 – Parameters used to set up the environment and evaluate the methods

Hyperparameter	**Value**	**Description**
Action set	Full	18 actions are always available to the agent.
Frame skip	5	Each action lasts 5 time steps.
Stochasticity (ς)	0.25	We used $\varsigma = 0.25$ for all experiments.
Lives signal used	False	We did not use the game-specific information about the number of lives the agent has at each time step.
Number of episodes used for evaluation	3 or 10	We report the average score obtained in the last 3 or 10 episodes used for learning.
Number of episodes used for evaluation in the continual learning	3 or 5	In the continual learning setting, we report the average score obtained in the last 3 or 5 episodes at the end of each tercile. See Subsection 5.3 for details.
Number of frames used for learning (T)	100,000	We report the scores after 100,000 frames and at the end of each tercile of episodes.
Number of trials	5	COMPER and DQN were evaluated in 5 trials.

In Table 21, we present the parameters used to set up the Arcade Learning Environment (ALE), to evaluate the agent's learning, and to summarize the results. In Table 22, we present the parameters related to the exploration (and exploitation) strategy and the discount factor.

5.3 Learning to play Atari 2600 games with discrete-valued actions

Machado et al. (2018) defined a limit of 200 million frames for each run of agent training. Four checkpoints were defined for 10, 50, 100, and 200 million frames. The

Table 22 – Discount factor and exploration rate used by COMPER and DQN

Hyperparameter	Value	Description
Discount factor (γ)	0.99	The discount factor applied in the update rule on the expected long-term reward.
Initial exploration rate (ϵ)	1.0	Probability of a random action will be taken at each time step.
Final exploration rate (ϵ)	0.001	Probability of a random action will be taken at each time step.
Final exploration frame	90,000	Number of frames over which ϵ is linearly annealed.

reward average and standard deviation were computed over the last 100 episodes preceding each checkpoint.

We define a limit of 100,000 frames for each training run and log the results at every 100 iterations and the end of each episode. We divided the number of episodes obtained in each run to split the results into 3 checkpoints. From that, we calculated the average scores (with standard deviations) for the last 5 episodes preceding each checkpoint. That was necessary because of the smaller number of episodes since we reduced the total number of frames, which is 0.0005% of the total number used in Machado et al. (2018). We have also redefined the decay rate of ϵ, so that it starts at 1.0 and drops to 0.01 in 90,000 frames (instead of 1 million frames used in Machado et al. (2018)).

In COMPER, the weights Ω in $QT(\tau_t, \Omega)$ are updated through backpropagation over sets of similar transitions every 100 agent steps, using a similarity threshold $\delta = 10^{-4}$. In turn, the update of $Q(s_t, a_t, \Theta)$ is performed every 4 agent steps, similar to DQN in Machado et al. (2018). Moreover, we start weight updating after the first 100 steps in COMPER and 1,000 steps in DQN (instead of 50,000 as in Machado et al. (2018)). Finally, we evaluate COMPER by representing the states of the environment either with a single frame in a matrix with dimensions of $84 \times 84 \times 1$ or with frames stacked in a matrix of $84 \times 84 \times 4$. In both cases, we just used the luminance values for every two frames using the color averaging method of ALE. We implemented our DQN agent as defined in Mnih et al. (2015) and only adopted new hyperparameters values for the total number of frames, the ϵ value decay, the results log frequency, the learning start, and the target function update frequency to accomplish our experiments. Finally, we evaluated and compared the use of a Long-short Term Memory (LSTM) and a Multi-layer Perceptron (MLP) to model DNN QT, which is used to produce the $\mathcal{RTM}$ and to predict the Q-target value. We adjusted the hyperparameters in just 3 training runs in the game Space Invaders both for COMPER and DQN. We then use the best values to run the experiments on all other games.

5.3.1 Evaluation Results

We report results for eight challenging games from ALE in five training runs of COMPER and DQN with 100,000 frames and about 25,000 (single frame) and 6,000 (stacked frames) iterations. Moreover, we report the results for COMPER using two different neural network architectures, an LSMT and an MLP, to approximate the Q-target function. We used an 8-core CPU (1 thread per core) and 48 GB of RAM to perform our experiments and spent about 9 to 12 hours on each training run using stacked frames and 24 to 36 hours using single frame.

Table 23 – Average scores of the last 3* and 10 training episodes. The best results are in bold, and the standard deviation is in parentheses

	10^5 Frames (10 or 3* last episodes)					10^7 Frames (100 last episodes)
	COMPER (Single)		COMPER (Stacked)		DQN	DQN
	LSTM	MLP	LSTM	MLP	CNN	CNN
Freeway*	19.53 (9.77)	8.4 (10.9)	**22.33 (0.0)**	9.46 (11.59)	0.0 (0.0)	13.8 (8,1)
Asteroids	**827.8 (142.93)**	781.18 (108.18)	362.4 (52.05)	442.8 (59.52)	237.8 (47.57)	301.4 (14.3)
Battle Zone*	**4,733.33 (2,080.6)**	3,200 (2,049.24)	4.400 (2,689.59)	1,066.66 (1,289.27)	1,666.66 (1,135.29)	2,428.3 (200.4)
Video Pinball*	**7,353.26 (7,366.18)**	3,225.46 (3,828.35)	2,524.73 (84)	6,827.13 (3,605.71)	6,936.13 (2,992.37)	4,009.3 (271.9)
Seaquest	**279.6 (24.21)**	52.4 (59.21)	73.2 (3.71)	44.4 (16.12)	132.0 (10.2)	311.15 (36.9)
Asterix	239.0 (58.3)	248.0 (98.2)	**258.0 (130.64)**	207.0 (48.23)	182.0 (22.05)	1732.60 (314.6)
Beam Rider*	509.06 (89.83)	504.53 (94.87)	416.53 (123.7)	465.33 (73.79)	**563.20 (89.26)**	693.9 (111.0)
Space Invaders	182.2 (68.27)	131.3 (75.13)	110.7 (53.58)	167.3 (76.92)	**199.80 (49.78)**	211.6 (14.8)

Table 24 – Professional Game Tester - Average reward achieved from 20 episodes played after 2 hours of training in each game

Professional Game Tester			
Freeway	**Asteroids**	**Battle Zone**	**Video Pinball**
26.6	13,157	37,800	17,298
Seaquest	**Asterix**	**Beam Rider**	**Space Invaders**
20,182	850	5,775	1,652

We evaluated COMPER using single frame and stacked frames and compared the results using an LSTM and an MLP to approximate the Q-target function. Table 23 presents the average scores (and standard deviation, in parentheses) for five runs of COMPER and DQN, calculated for the last 3 or 10 training episodes with a limit of 100,000 frames from ALE. The last right column presents the results from Machado et al.

(2018) with 10 million frames for comparison. For the Freeway, Battle Zone, Video Pinball, and Beam Rider, we have computed only the last three episodes because of the fewer total episodes we obtained compared with the other games. We also defined checkpoints at the end of every tercile on the total frames number throughout the training episodes. Tables 25 and 26 present the average scores (and standard deviation) for the last 3 and 5 episodes before each of these checkpoints. COMPER obtains the best results on the games Freeway, Asteroids, Battle Zone, Video Pinball, Seaquest, and Asterix. Whether using single frame or stacked frames, the best results were obtained by approximating the Q-target function through an LSTM. The large standard deviation for the Video Pinball with single frame is due to a training run for which the agent has not scored in the last three episodes. The results for Beam Rider seem to be statistically equivalent, while COMPER results for Space Invaders were not so good. Regarding the evaluation on the checkpoints, we highlight the continuous and stable evolution of COMPER with LSTM on the games Freeway, Asteroids, Seaquest, Asterix, and Beam Rider, while the adoption of MLP decreases the COMPER performance, especially for the games Freeway, Battle Zone and Seaquest. It is essential to highlight that these results were obtained using the value of 10^{-4} as the similarity threshold. Section 5.3.2 presents a study on the effects of variations in this value, concluding that more restrictive values produce better results. The best example was Space Invaders, which, by changing the threshold from 10^{-4} to 10^{-5}, obtained an average reward of **223.1** with an 81.4 standard deviation, thus passing to appear on the list of games with the best results overall. Table 24 adapts the results obtained by a professional human game tester (MNIH et al., 2015). He trained by playing each game for around 2 hours. After that, their results were measured by the average reward from around 20 episodes of each game lasting a maximum of 5 minutes each.

Table 25 – COMPER with LSTM - Average scores of the last 3* and 5 last episodes of each tercile. The best results for each tercile are in bold and the standard deviation in parentheses

	End of First Tercile			**End of Second Tercile**			**End of Third Tercile**		
	COMPER (Single)	COMPER (Stacked)	DQN	COMPER (Single)	COMPER (Stacked)	DQN	COMPER (Single)	COMPER (Stacked)	DQN
Freeway*	**4.26** (2.38)	4.0 (0.0)	0.0 (0.0)	12 (6.70)	**16.0** (0.0)	0.0 (0.0)	19.53 (10.92)	**22.33** (0.0)	0.0 (0.0)
Asteroids	**885.20** (124.95)	440.8 (85.75)	376.00 (81.47)	**902.40** (175.34)	334.80 (50.56)	386.00 (51.51)	**846.0** (237.37)	341.20 (78.55)	233.60 (0.0)
Battle Zone*	1,933.33 (862.81)	**3,400.00** (1,673.32)	3,200.00 (2,089.65)	**4,333.33** (2,392.11)	2,333.33 (408.24)	2,066.66 (1,038.16)	**4,733.33** (2,326.17)	4,400.00 (3,003.70)	1,666.66 (1,269.29)
Video Pinball*	10,157.73 (6,695.87)	2,565.73 (1,442.02)	**11,513.86** (7,672.34)	**8,747.33** (8,058.55)	2,665.46 (2,844.67)	4,454.66 (1,127.78)	**8,544.60** (10,671.12)	2,524.73 (2,764.71)	3,936.13 (3,345.57)
Seaquest	**143.2** (17.52)	76.8 (15.59)	78.40 (40.52)	**356.0** (91.82)	80.0 (22.44)	152.00 (23.50)	**222.4** (38.32)	72.0 (4.0)	123.20 (15.88)
Asterix	268.0 (82.28)	258.0 (82.88)	**290.00** (58.99)	228.0 (30.33)	210.0 (24.49)	**234.00** (18.55)	232.0 (52.15)	**256.0.0** (165.92)	166.00 (45.87)
Beam Rider*	**417.33** (119.26)	352.53 (102.05)	381.33 (46.38)	419.46 (49.30)	354.93 (116.59)	**475.20** (56.12)	509.06 (100.43)	416.53 (137.81)	**563.20** (89.26)
Space Invaders	139.8 (42.24)	109.6 (42.84)	**140.80** (59.94)	**201.6** (30.58)	128.4 (31.54)	174.00 (35.57)	193.8 (63.08)	129.2 (75.74)	**213.80** (68.90)

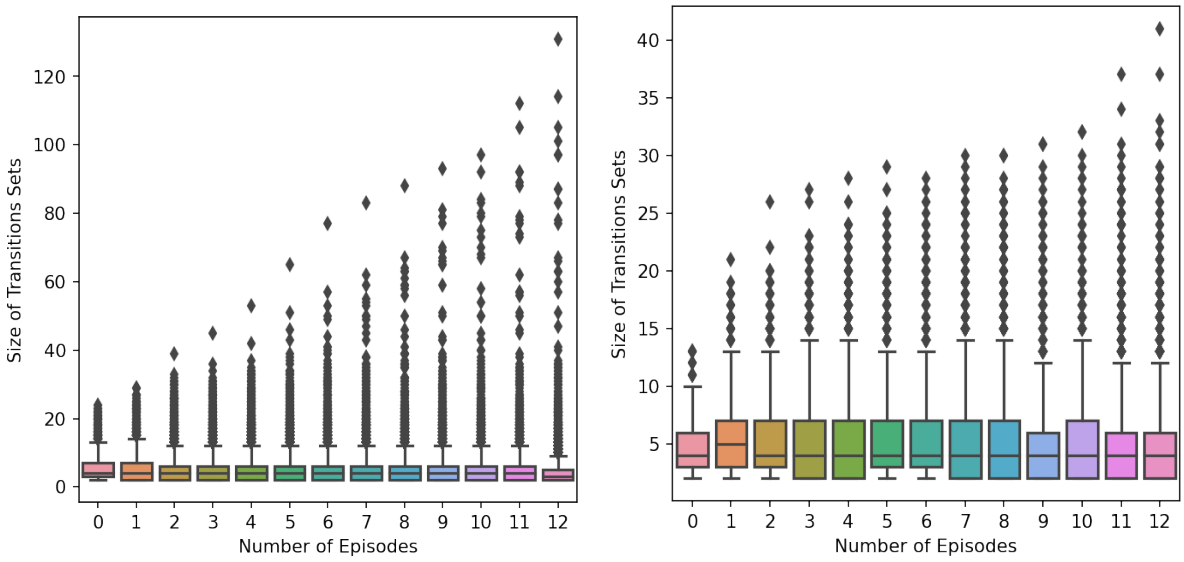
Table 26 – COMPER with MLP - Average scores of the last 3* and 5 last episodes of each tercile. The best results for each tercile are in bold and the standard deviation in parentheses

	End of First Tercile			End of Second Tercile			End of Third Tercile		
	COMPER (Single)	COMPER (Stacked)	DQN	COMPER (Single)	COMPER (Stacked)	DQN	COMPER (Single)	COMPER (Stacked)	DQN
Freeway*	**2.13** (2.92)	2.0 (2.73)	0.0 (0.0)	6.0 (8.21)	**6.13** (8.39)	0.0 (0.0)	8.4 (11.50)	**9.46** (12.96)	0.0 (0.0)
Asteroids	**883.2** (105.93)	379.6 (96.69)	376.00 (81.47)	**871.2** (288.55)	402.4 (141.02)	386.00 (51.51)	**837.2** (168.31)	420.8 (87.05)	233.60 (0.0)
Battle Zone*	**3,266.66** (2,465.31)	2,200.00 (1,425.94)	3,200.00 (2,089.65)	**3,066.66** (1,876.75)	1,533.33 (1,345.77)	2,066.66 (1,038.16)	**3,200.00** (2,693.61)	1,066.66 (1,441,44)	1,666.66 (1,269.29)
Video Pinball*	9,114.66 (5,096.14)	4,075.40 (2,050.22)	**11,513.86** (7,672.34)	**7,330.33** (9,133.18)	7,303.80 (5,064.68)	4,454.66 (1,127.78)	3,225.46 (4,280.22)	**6,827.13** (4,031.30)	3,936.13 (3,345.57)
Seaquest	67.2 (85.20)	65.6 (36.17)	**78.40** (40.52)	75.2 (101.36)	71.2 (38.71)	**152.00** (23.50)	40.8 (47.19)	39.2 (18.41)	**123.20** (15.88)
Asterix	266.0 (106.91)	212.0 (56.74)	**290.00** (58.99)	**294.0** (45.05)	260.0 (66.33)	234.00 (18.55)	**242.0** (82.88)	204.0 (47.74)	166.00 (45.87)
Beam Rider*	**404.80** (91.35)	390.13 (83.35)	381.33 (46.38)	415.46 (150.63)	437.06 (82.57)	**475.20** (56.12)	504.53 (106.06)	465.33 (82.49)	**563.20** (89.26)
Space Invaders	**188.4** (70.35)	188.2 (88.03)	140.80 (59.94)	132.0 (47.43)	165.4 (63.03)	**174.00** (35.57)	109.4 (77.91)	181.6 (89.83)	**213.80** (68.90)

It is worth saying that Freeway, Seaquest, and Asteroids are complex games. The first two present difficulties in agent rewarding related to long-term policy learning needs, while the latter deals with a hard exploration task and complex dynamics (MACHADO et al., 2018). Asterix seems to present a high degree of randomness in the rewarding elements' positions at each time step, making the rewards accumulation strongly related to the number of interactions with the game environment. In turn, there are a lot of simultaneous rewarding elements in Space Invaders screen from the first frames until long after the game starts. Thus, the agent has many chances to hit an enemy (and be hit), and the number of frames does not seem to impact the total reward accumulated up to 100,000 frames, contributing more towards reducing the standard deviation. One should also note that COMPER scores better when using single frame (instead of stacked frames) and that training runs with single frame were the ones that produced more similar transitions. Moreover, memory size from which the agent sampled transitions is considerably lesser than those used in the literature. Therefore, the agent can learn from smaller memories, and similar transition sets help capture the dynamics regarding the expected long-term rewards, which an LSTM seems to better model (compared to an MLP).

To verify the behavior of $\mathcal{TM}$, we evaluated the number of similar transitions throughout the training episodes, and the sizes of these similar sets. Figures 4 and 5 illustrate the average sizes of the similar transitions sets for Freeway and Beam Rider. We can verify three main behaviors: (i) new sets were created in different moments; (ii) some of these sets are removed from the memory and do not occur again, while others remain being updated until the last training episode; and (iii) when using single frame, the number of similar transitions is larger than when using stacked frames. This behavior

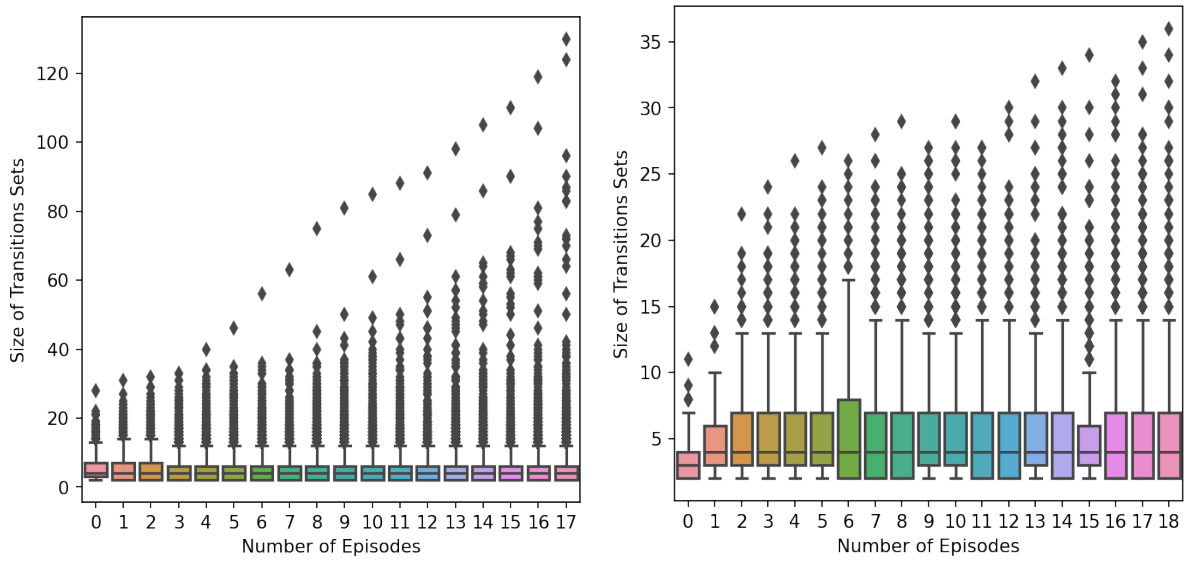
is observed in all games. Figures 6a and 6b illustrate the sizes of the similar transition sets for all the games with single and stacked frames, while Figures 7a and 7b show the occurrence of similarities. This information is relevant because the number of similarities implies the number of times the Q-value for similar transitions was updated instead of reinserting a new tuple in the memory, and that helps to maintain its size small and under control. As the number of observed frames increases throughout the successive episodes, we can see that the size of $\mathcal{TM}$ varies but keeps an upper limit because of the reduction step applied when similar transitions are removed to train the Q-target network and update $\mathcal{RTM}$. The small size of $\mathcal{RTM}$ is verified directly by the total number of unique transitions at the end of each run. And, even sampling from such a small number of transitions, COMPER manages to achieve good results. Figures 8 and 9 illustrate these behaviors.



(a) Freeway($84 \times 84 \times 1$) **(b) Freeway($84 \times 84 \times 4$)**

Figure 4 – Freeway - Sizes of Similar Transitions Sets

A detailed analysis shows the following. For Freeway, COMPER did not score until the end of the second training episode. However, once it received the first point, it began to accumulate rewards continuously, shown in Tables 25 and 26, and Figures 10 and 11. In turn, DQN was somehow unable to get any score with only 100,000 frames. For Asteroids, agents got high scores in the first episodes and kept values close to the average until the end of the runs when they were only exploiting, and COMPER obtained the best results even at the end of each tercile. For Battle Zone, COMPER did not reduce the average scores at the end of each tercile and maintained the best results compared to DQN. Video Pinball presented a lesser stable behavior for both agents, but COMPER obtained better average scores at the end of the runs and explicit learning progress at the



(a) Beam Rider($84 \times 84 \times 1$) **(b) Beam Rider($84 \times 84 \times 4$)**

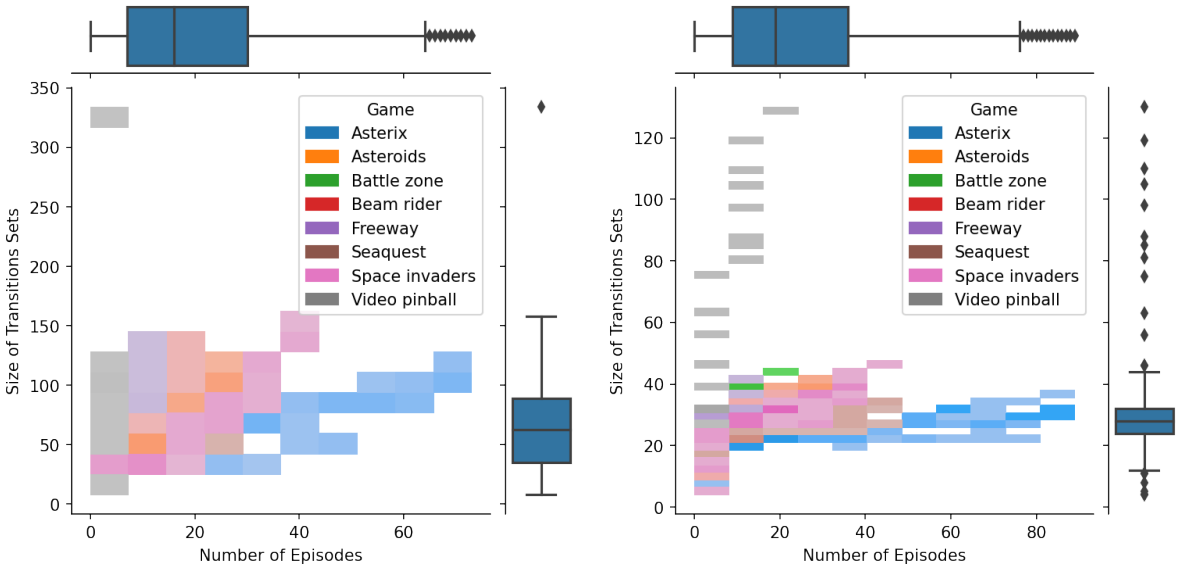
Figure 5 – Beam Rider - Sizes of Similar Transitions Sets

first run with single frame and at the last with stacked frames (Figures 12 and 13). For Seaquest, COMPER presented increasing scores from the first to the third tercile at all the runs, both with single and stacked frames and obtained the best result at the last tercile. For Asterix, COMPER scores did not decrease at the end of the terciles, and we highlight the third run with single frame and the second with stacked frames (Figures 14 and 15). Moreover, COMPER reached the last episodes with better results than DQN. For Beam Rider, COMPER and DQN obtained very close average values and similar behaviors throughout the learning process. Finally, the same did not happen for Space Invaders; COMPER and DQN obtained close average scores throughout the first terciles, but COMPER got worse results than the DQN for the last tercile. Complete test results are available at the COMPER research repository[1], which contains all the training log files for COMPER and DQN, with Jupyter notebooks used for analysis.

5.3.2 The Importance of the Similarity Threshold

The core of COMPact Experience Replay and COMCACT are (i) the similar transition sets, (ii) the compact transitions memory, and (iii) the Q-target function. However, computing similar transitions implies the number, size, and even (probably) the subjacent behavior of the similar transition sets. This implies the efficacy of the Q-target function and the degree of compacting the transitions memory. All these depend on a small essential element: the similarity threshold hyperparameter.

[1]https://github.com/DanielEugenioNeves/COMPER-RELEASE-RESULTS



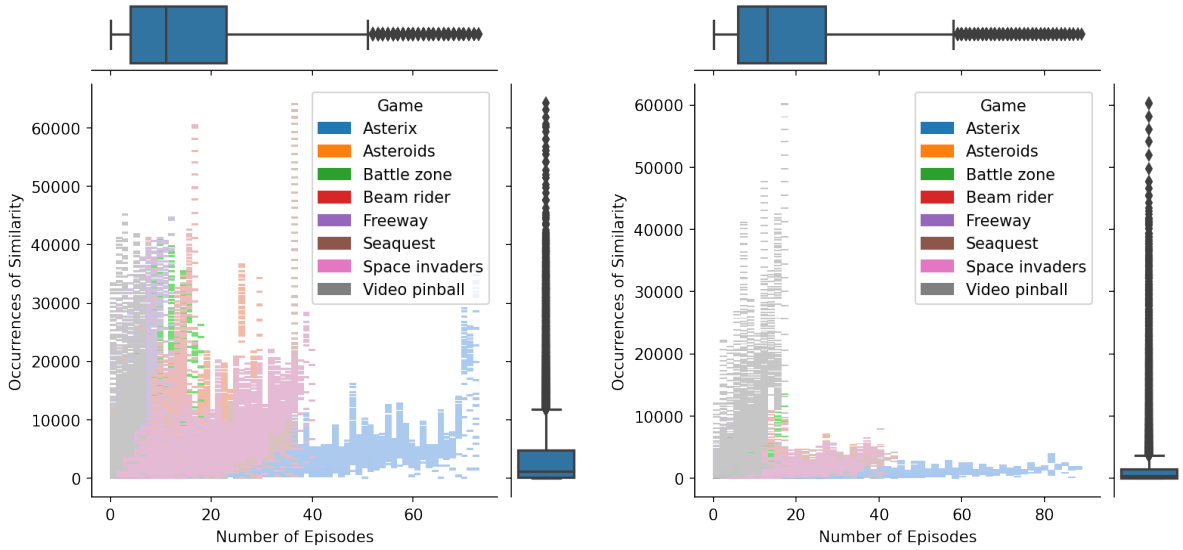
(a) Single Frame **(b) Stacked Frames**

Figure 6 – Sizes of the Similar Transitions Sets Throughout Episodes

To evaluate the impact of varying the similarity threshold, we selected from Table 23 those games on which COMPER obtained relatively worse results: Asterix, Beam Rider, and Space Invaders. We ran new experiments according to the same definitions and methodology from Section 5.3. We used the single-frame version and the LSTM to approximate the Q-target function, such as the original formulation of COMPER. Previously, we used a similarity threshold value equal to 10^{-4}. Therefore, for the results discussed in this section, we used a more permissive value of 10^{-3} and a more restrictive value of 10^{-5}. Table 27 summarizes the results. The first two columns present the average reward values for the last $3*$ and 10 episodes from 5 trials for each game when using the values 10^{-3} (in salmon) and 10^{-5} (in blue) for the similarity threshold. The last two columns present the results achieved by COMPER when we ran the previous experiments with 10^{-4} (in yellow) as a baseline. The average rewards for each trial are in Table 28.

Table 27 – Average Rewards from Different Similarity Threshold Values

	Averages from 5 Trials			
	10^{-3}	10^{-5}	10^{-4}	
Frames mode:	Single	Single	Single	Stacked
Asterix	219.0 (140.98)	223.1 (81.4)	239.0 (58.3)	**258.0 (130.64)**
Beam Rider*	437.86 (194.22)	402.13 (109.19)	**509.06 (89.83)**	416.53 (123.7)
Space Invaders	163.4 (83.71)	**223.1 (81.4)**	182.2 (68.27)	110.7 (53.58)



(a) Single Frame **(b) Stacked Frames**

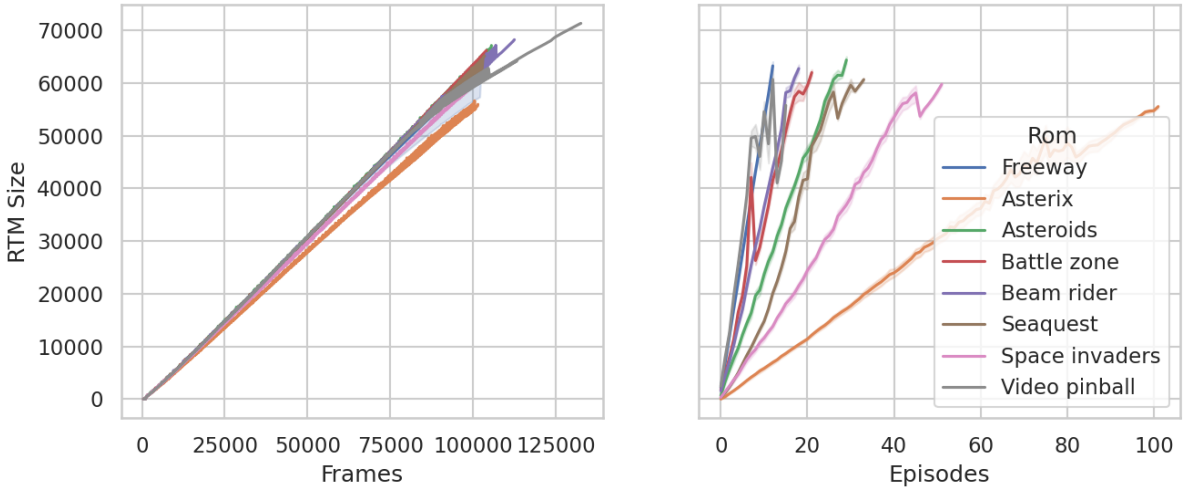
Figure 7 – Occurrences of Similarity Throughout Episodes

Table 28 – Average Rewards from Different Similarity Threshold Values by Trial

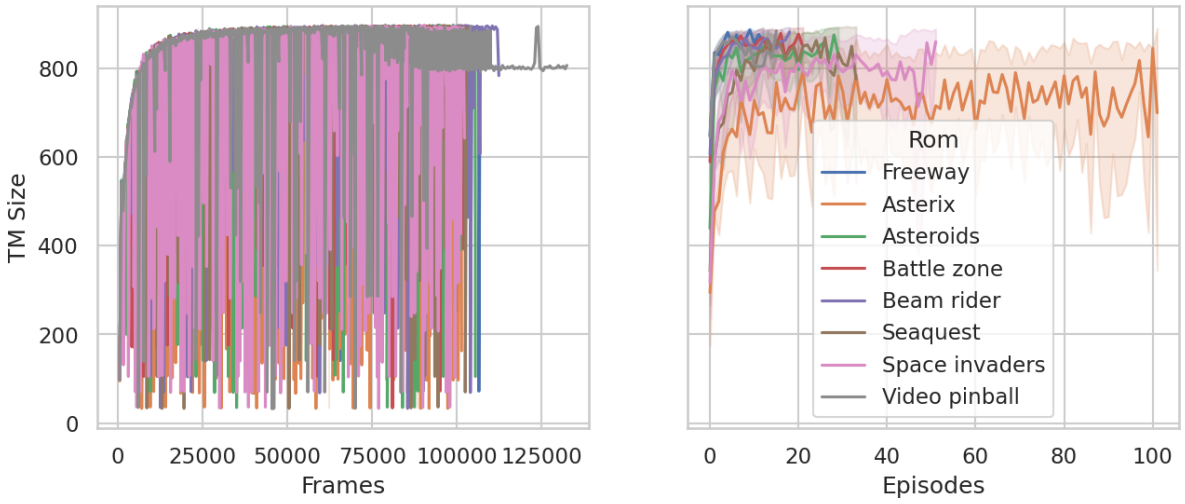
	Trial 1		Trial 2		Trial 3		Trial 4		Trial 5	
	10^{-3}	10^{-5}	10^{-3}	10^{-5}	10^{-3}	10^{-5}	10^{-3}	10^{-5}	10^{-3}	10^{-5}
Asterix	145.0	205.0	100.0	210.0	170.0	440.0	185.0	105.0	495.0	170.0
Beam Rider*	396.0	558.66	542.66	440.0	117.33	440.0	425.33	337.33	708.0	234.0
Space Invaders	234.5	182.0	23.0	315.0	251.5	327.0	169.0	135.0	130.0	156.5

Based on the results, one can note that a more permissive threshold value, such as 10^{-3}, tends to produce worse results in general, mainly leading to less stable behavior in the final of the trials, with outliers that may lead to a higher average, as in the unique case of Beam Rider, but with a higher standard deviation value. And, even in this case, it did not overcome the baseline. On the other hand, less permissive values, such as 10^{-4} from the baseline and 10^{-5} from the new experiments, lead to more stable behavior and, consequently, better results in general. It is necessary to highlight that Space Invader, the game with worse results in the previous experiments when it surpassed neither the baseline DQN nor the benchmark DQN (MACHADO et al., 2018), has now overcome both by using a similarity threshold of 10^{-5}, i.e., even more restrictive.

We hypothesize that a more permissive similarity threshold produces worse sets of similar transitions because it permits to be in the same set, experiences from policies that are (ii) very different, (ii) some old and some fresher, (iii) or that represent diverse behaviors, when what one expects is that the experiences in a set are not simple numerically similar, but similar in its behavior and representing similar actions policies.



(a) $\mathcal{RTM}$ - Sizes for Single Frame

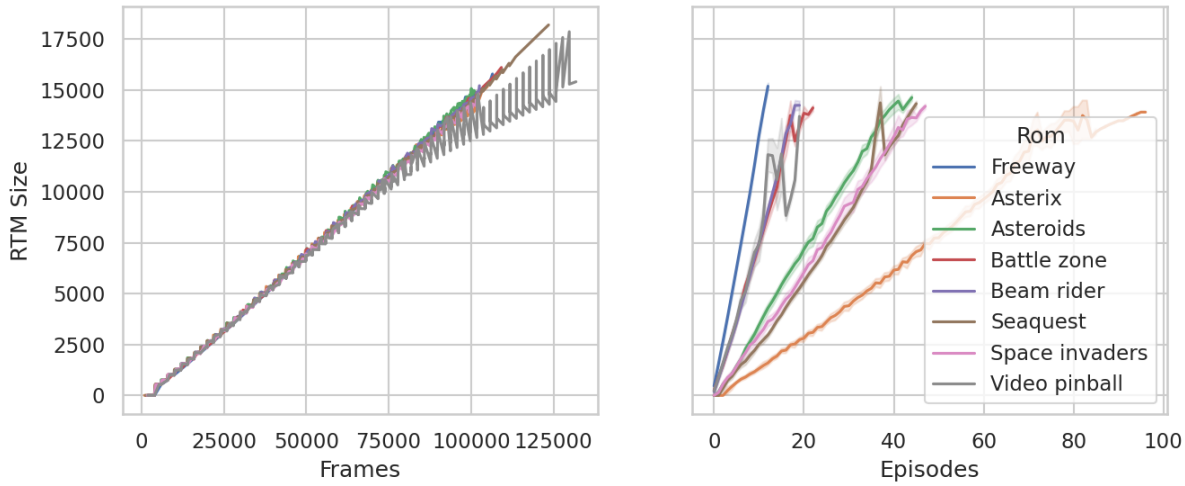


(b) $\mathcal{TM}$ - Sizes for Single Frame

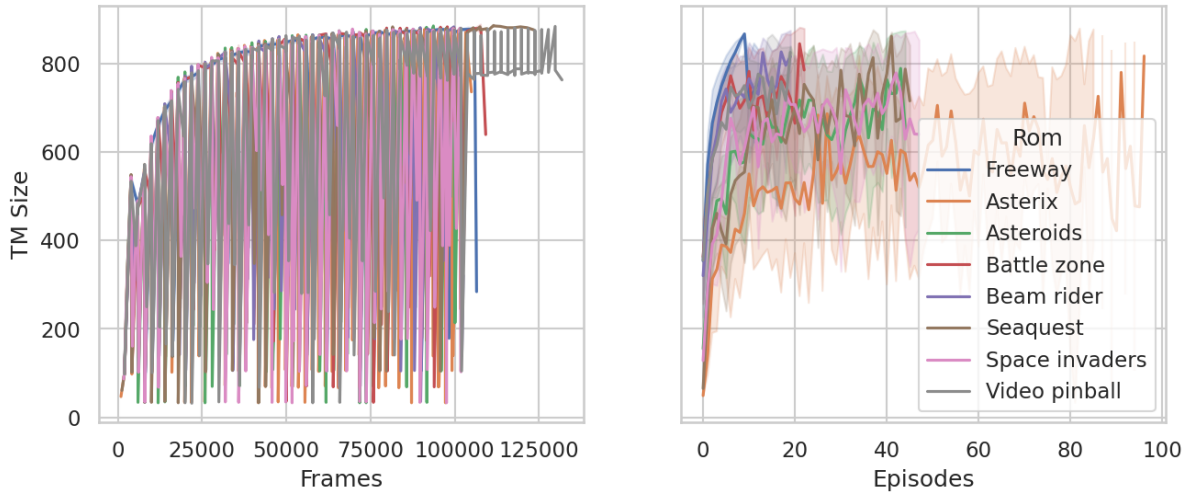
Figure 8 – Behavior of Transition Memories with Single Frame

5.4 Considerations

From the experiments, it is possible to attest that COMPER can approximate good policies on Atari 2600 games on the Arcade Learning Environment (ALE) from a considerably smaller number of frame observations to achieve similar results obtained by DQN-based methods in the literature, which generally demand millions of video frames. The results demonstrate that COMPER achieves excellent performance on challenging games with fewer observations of the environment and a smaller memory than those used in the literature. We highlight that COMPER scores better when using single frame instead of stacked frames and that the training runs with single frame generated the greatest amount of similar transitions. Therefore, using COMPER agents can learn from



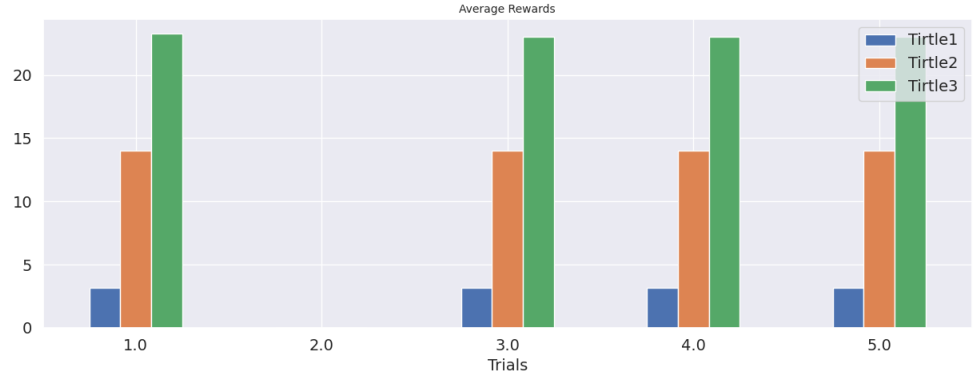
(a) $\mathcal{RTM}$ - Sizes for Stacked Frames



(b) $\mathcal{TM}$ - Sizes for Stacked Frames

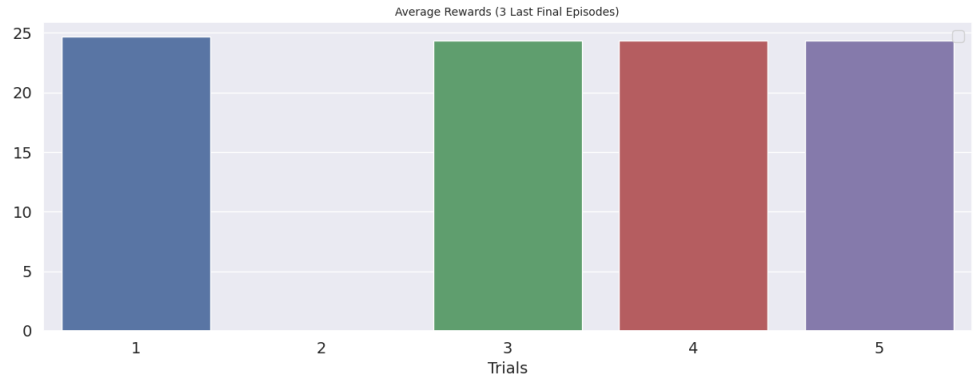
Figure 9 – Behavior of Transition Memories with Stacked Frames

smaller memories, and similar transition sets help capture the dynamics regarding the expected long-term rewards, which LSTM seems to better model than MLP.



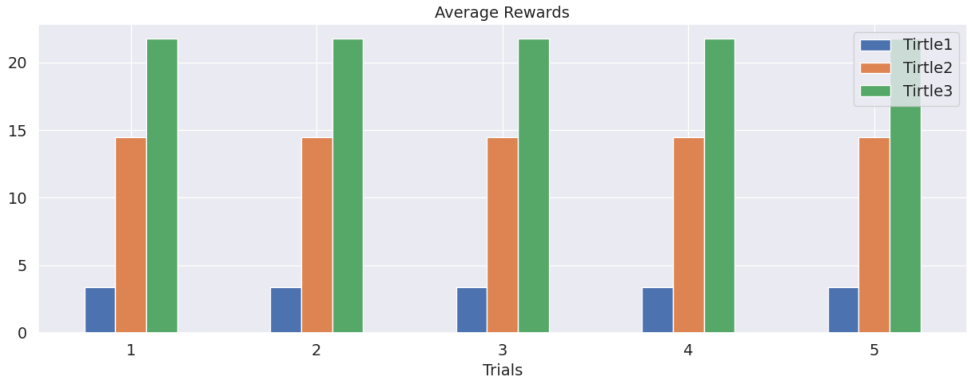
(a) Freeway – Rerwars by Tercile – Single Frame

(b) Freeway – Rewards by Tercile and 3 Last Episodes – Single Frame

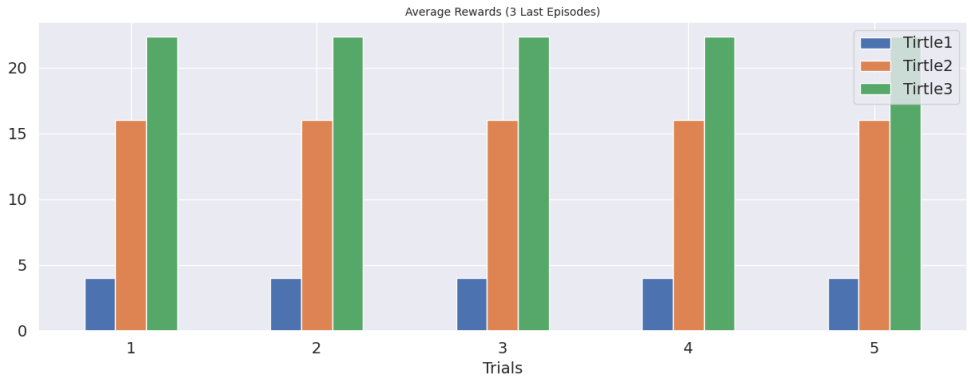


(c) Freeway – Rewards by Trial and 3 Last Episodes – Single Frame

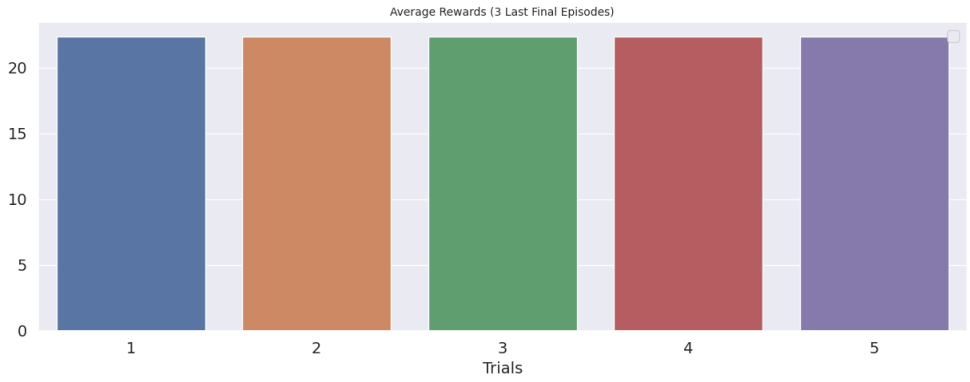
Figure 10 – Freeway - Average Rewards Using Single Frame



(a) Freeway – Rewards by Tercile – Stacked Frames



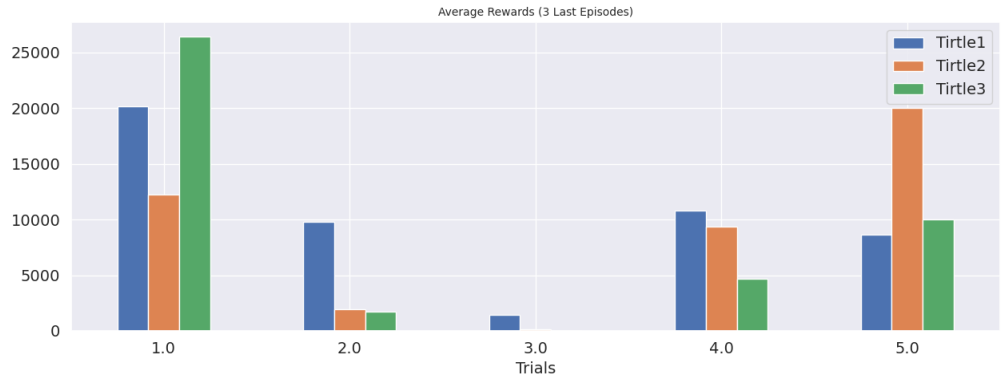
(b) Freeway – Rewards by Tercile and 3 Last Episodes – Stacked Frames



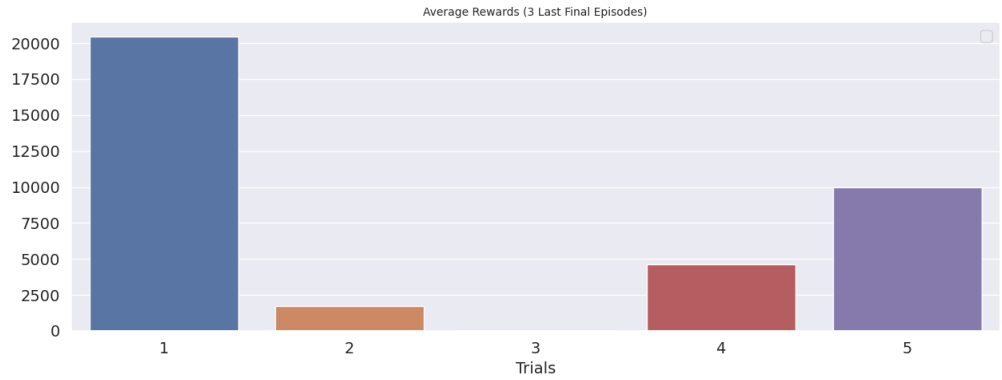
(c) Freeway – Rewards by Trial and 3 Last Episodes – Stacked Frames

Figure 11 – Freeway - Average Rewards Using Stacked Frames

(a) Video Pinball– Rewards by Tercile – Single Frame



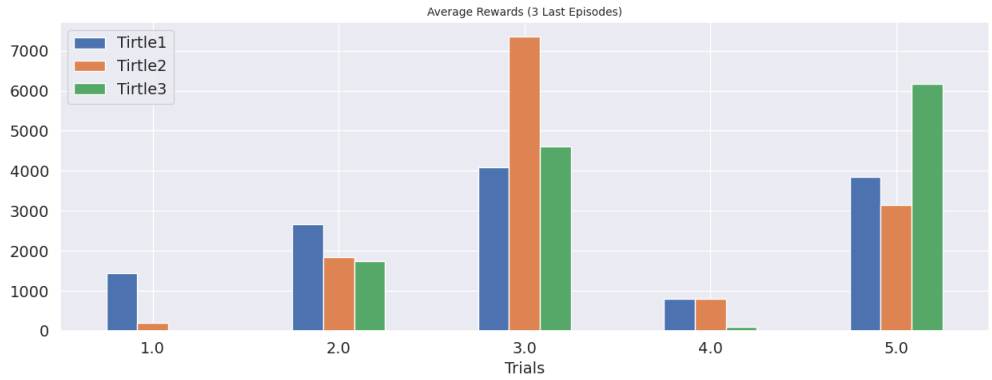
(b) Video Pinball – Rewards by Tercile and 3 Last Episodes – Single Frame



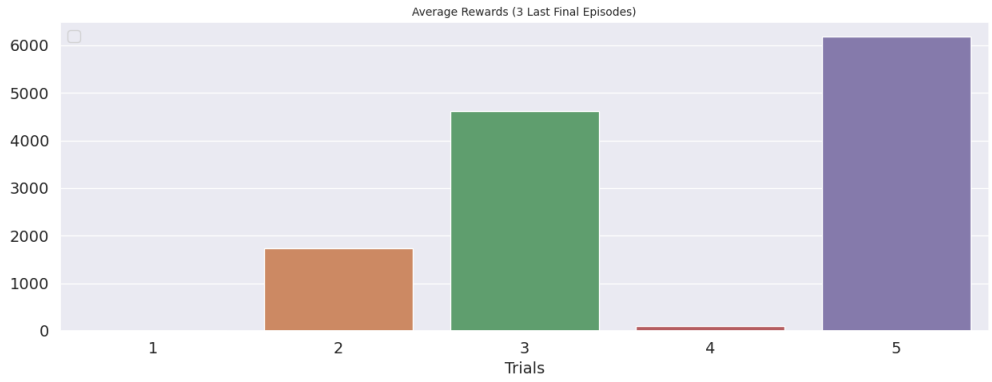
(c) Video Pinball – Rewards by Trial and 3 Last Episodes – Single Frame

Figure 12 – Video Pinball - Average Rewards Using Single

(a) Video Pinball– Rewards by Tercile – Stacked Frames



(b) Video Pinball – Rewards by Tercile and 3 Last Episodes – Stacked Frames

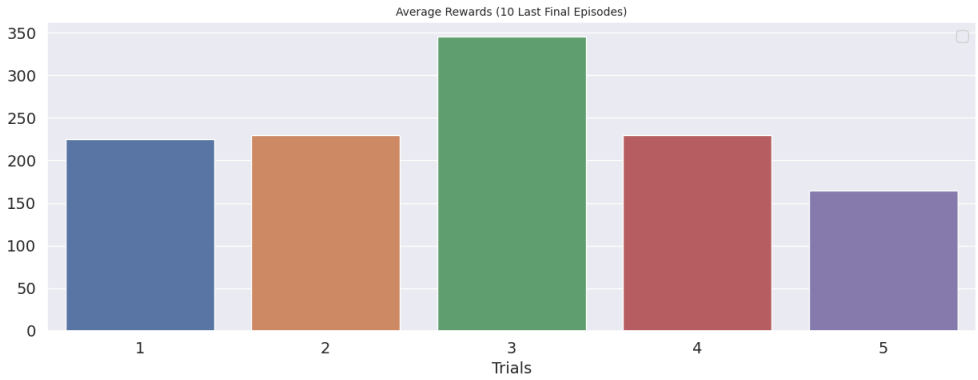


(c) Video Pinball – Rewards by Trial and 3 Last Episodes – Stacked Frames

Figure 13 – Video Pinball – Average Rewards Using Stacked Frames

(a) Asterix – Rewards by Tercile – Single Frame

(b) Asterix – Rewards by Tercile and 3 Last Episodes – Single Frame

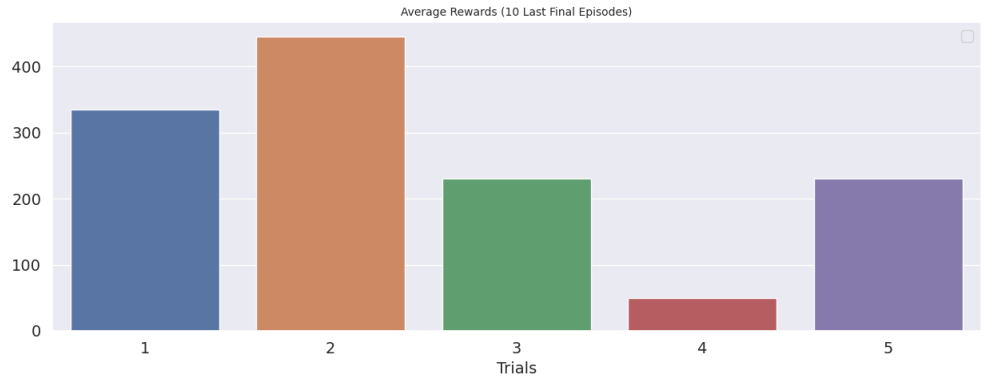


(c) Asterix – Rewards by Trial and 3 Last Episodes – Single Frame

Figure 14 – Asterix - Average Rewards Using Single

(a) Asterix – Rewards by Tercile – Stacked Frames

(b) Asterix – Rewards by Tercile and 3 Last Episodes – Stacked Frames



(c) Asterix – Rewards by Trial and 3 Last Episodes – Stacked Frames

Figure 15 – Asterix - Average Rewards Stacked Frames

6 COMCACT – METHODOLOGY AND EMPIRICAL RESULTS

To evaluate COMCACT concerning the objectives presented in Section 1.4 and verify if its results support the hypotheses discussed in Section 1.3, we conducted experiments in various tasks in two different environments to learn physical control with continuous-valued actions under physics simulations. The details about deep neural networks and hyperparameters are in Appendix B.

6.1 The Environments of Mujoco and the Classical Control Tasks

This section briefly describes the environments illustrated in Figure 16, and we used them in the experiments. However, each environment and its controllable robot or humanoid are specifically defined and have different parameters and behaviors. The details about mathematical formulations regarding the definitions of actions, states, reward functions, sets of parameters, angles, force, torque, friction, and the physics engine working are present in the documentation of Gymnasium (TOWERS et al., 2023).

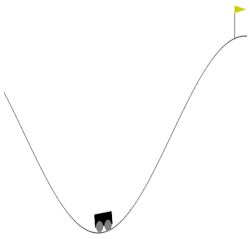
The Mountain Car is a deterministic MDP consisting of a car placed stochastically at the bottom of a sinusoidal valley. The only possible action is applying acceleration values to the car in either direction, and the goal is to reach the top of the right hill. The observation is an array in which the two elements correspond to the car's position along the x axis and its velocity, whose values are in $[-\infty, \infty]$. The action is a 1-position array representing the directional force applied to the car, whose value is clipped in the range $[-1, 1]$ and multiplied by a power of 0.0015. The collisions at either end are inelastic, with the velocity set to 0 upon collision with the wall. The position is clipped to the range $[-1.2, 0.6]$, and the velocity is clipped to the range $[-0.07, 0.07]$. Therefore, the transition dynamics are defined by Equations 6.1 and 6.2, where *force* is the action clipped to the range $[-1, 1]$ and *power* is a constant 0.0015 (TOWERS et al., 2023).

$$velocity_{t+1} = velocity_t + force \times power - 0.0025 \times \cos(3 \times position_t) \tag{6.1}$$

$$position_{t+1} = position_t + velocity_{t+1} \tag{6.2}$$

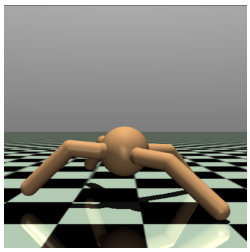
The agent receives a negative reward of $-0.1 \times action^2$ each time-step it takes actions of large magnitude. If the mountain car reaches the goal, a positive reward of $+100$ is added to the negative reward for that time-step. The default behavior is to terminate the episode if the car's position is greater than or equal to 0.45 (the goal position on top of the right hill) or truncate the episode if the number of iterations reaches 999 time-steps.

The environment of the Pendulum consists of a pendulum attached at one end to a fixed point, and the other is free. The pendulum starts in a random position, and the goal

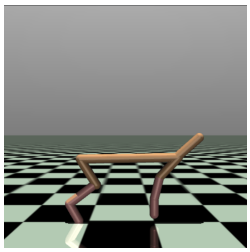


(a) Mountain Car

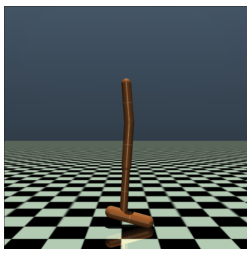
(b) Pendulum



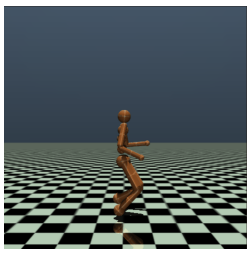
(c) Ant



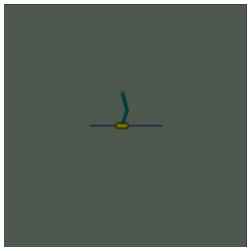
(d) HalfCheetah



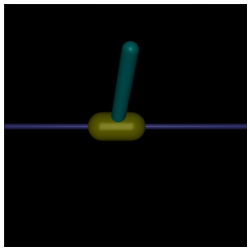
(e) Hopper



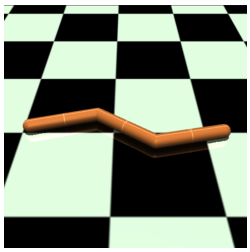
(f) Humanoid



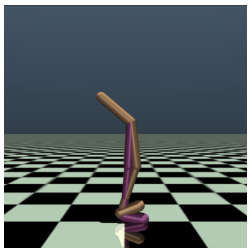
(g) Inv. D. Pend.



(h) Inv. Pend.



(i) Swimmer



(j) Walker2d

Figure 16 – Environments of Classical Control and Mujoco

is to apply torque on the free end to swing it into an upright position, with its center of gravity right above the fixed point. The state observation is an array with three positions representing: (i) the coordinate $x = \cos(\theta)$ in the range $[-1.0, 1.0]$; (ii) the coordinate $y = sin(\theta)$ in the range $[-1.0, 1.0]$; (ii) angular velocity in $[-8.0, 8.0]$. The action is an array with one position representing the torque applied to the free end of the pendulum whose value is in $[-2, 2]$. The reward function is defined by Equation 6.3, where θ is the pendulum's angle normalized between $[-\pi, \pi]$ (with 0 being upright), and tau(τ) is the torque which defines a positive counter-clockwise (TOWERS et al., 2023).

$$r = -(\theta^2 + 0.1 \times \theta dt^2 + 0.001 \times \tau^2) \tag{6.3}$$

Based on the above equation, the minimum reward that can be obtained is $-(\pi^2 + 0.1 \times 8^2 + 0.001 \times 2^2) = -16.2736044$, while the maximum reward is zero (pendulum is upright with zero velocity and no torque applied). The starting state is a random angle in $[-\pi, \pi]$ and a random angular velocity in $[-1, 1]$. The episode truncates at 200 time-steps.

In the environment of Ant, a 3D robot with one torso (free rotational body) and four legs, each with two body parts. The goal is to coordinate the four legs to move in the right direction by applying torques on the eight hinges connecting the two body parts of each leg and the torso (nine body parts and eight hinges). The action space is an array with 8 positions representing the torques applied at the hinge joints. The observations consist of positional values in $[-\infty, \infty]$ for different body parts of the ant, followed by the velocities of those individual parts (their derivatives), with all the positions ordered before all the velocities and composing an array of 27 positions (default configuration). All observations start in state $[0.0, 0.0, 0.75, 1.0, 0.0, \ldots, 0.0]$ with noise added to the values for stochasticity. Note that the initial z coordinate is selected to be slightly high, indicating a standing-up ant. The initial orientation is designed to make it face forward as well. The reward consists of (a) a reward for health with a fixed value at every time step the ant is healthy, (b) a reward for moving forward, and (c) a negative reward for penalizing the ant if it takes actions that are too large. The total reward is $(a) + (b) - (c)$. It is possible to parameterize the environment to consider a contact cost (d) to penalize the ant if the external contact force is too large, changing the reward function to $(a) + (b) - (c) - (d)$. The ant is said to be unhealthy if any of the state space values are no longer finite or if the z-coordinate of the torso is not in a parameterized healthy range whose default is $[0.2, 1.0]$ The default behavior is to terminate the episode if the ant is unhealthy or truncate the episode if the number of iterations reaches 1,000 time-steps. It is possible to disable, via parameter, the termination when unhealthy.

The HalfCheetah is a 2-dimensional robot with 9 body parts and 8 joints connecting them (including two paws). The goal is to apply torque on the joints to make the cheetah

run forward (from left to right) as fast as possible. The torso and head of the cheetah are fixed, and the torque can only be applied on the other 6 joints over the front and back thighs (connecting to the torso), shins (connecting to the thighs), and feet (connecting to the shins). The action space is represented by an array containing the torque values applied at the hinge joints. The observations consist of positional values of different body parts of the cheetah, followed by the velocities of those individual parts (their derivatives), with all the positions ordered before all the velocities. The first 8 values in the state are positional, and the last 9 values are velocity, and all observations start in the state $[0.0, ..., 0.0]$. A uniform noise from a parameterized interval is added to the positional values. A standard normal noise with a mean of 0 and standard deviation on the superior limit of the noise values is added to the initial velocity values of all zeros. The episode truncates when the length exceeds 1000 steps. The agent receives a positive reward based on the distance moved forward and a negative reward for moving backward. The reward consists of two parts: (a) the reward for moving forward and (b) a control cost to penalize the cheetah if it takes actions that are too large. The total reward is $(a) - (b)$. The episode truncates when it reaches 1,000 time steps.

The Hopper is a two-dimensional, one-legged figure that consists of four main body parts: (i) the torso at the top, (ii) the thigh in the middle, (iii) the leg at the bottom, and (iv) a single foot on which the entire body rests. The goal is to make hops that move in the forward (left to right) direction by applying torques on the three hinges connecting the four body parts. An array with three positions represents an action whose values are in [-1,1] and defines the torques applied at the hinge joints. The state observations consist of positional values of different body parts of the hopper, followed by the velocities of those individual parts (their derivatives), with all the positions ordered before all the velocities. By default, the observation is an array with 11 positions. All observations start in the state [0.0, 1.25,..., 0.0] with a uniform noise in a parameterized range added to the values for stochasticity. The reward consists of three parts: (a) a fixed valuated reward every time-step that the hopper is healthy, (b) a reward based on the distance moved forward, and (c) a negative reward if the hopper takes too large actions. The total reward returned is $(a) + (b) - (c)$. The hopper is said to be unhealthy if: (i) an element of observation is no longer contained in the closed interval specified by the argument that defines the healthy state range; (ii) the height of the hopper is no longer in a parameterized healthy range (usually meaning that it has fallen); (iii) the angle is no longer in a parameterized healthy range. If defined (using the specific parameter) to terminate when unhealthy (which is the default), the episode ends when any of the following happens: (i) truncate the episode at 1,000 time-steps; or (ii) the hopper is unhealthy. Otherwise, the episode ends only when truncated.

In Humanoid, a 3D bipedal robot designed to simulate a human has a torso

(abdomen) with legs and arms. The legs consist of three parts, and the arms of two parts (representing the knees and elbows, respectively). The goal is to walk forward as fast as possible without falling over. The action is represented by an array with 17 positions whose values are in $[-1, 1]$ and defines the torques applied at the hinge joints. The observations consist of positional values of different humanoid body parts, followed by the velocities of those individual parts (their derivatives), with all the positions ordered before all the velocities. By default, the state's observations are arrays with 376 values in $[-\infty, +\infty]$ and do not include the torso's x- and y-coordinates, which may be included using a specific parameter. In this observation space, 44 elements are positional values. Additionally, after all the position and velocity-based values, the observation contains (i) the mass and the inertia of a single rigid body relative to the center of mass (this is an intermediate result of transition), with a shape 14×10 ($nbody \times 10$), adding to another 140 elements in the state space; (ii) the center of mass based velocity, with shape 14×6 ($nbody \times 6$), adding another 84 elements in the state space; (iii) a constraint force generated as the actuator force, with shape 1×23 ($nv \times 1$), adding another 23 elements to the state space; (iv) the center of mass based external force on the body, with shape 14×6 ($nbody \times 6$), adding another 84 elements in the state space. The term $nbody$ stands for the number of bodies in the robot, and nv stands for the number of degrees of freedom. The (x, y, z) coordinates for translation, while the orientations are used for rotations expressed as quaternions. All observations start in the state $[0.0, 0.0, 1.4, 1.0, 0.0, \ldots, 0.0]$ with a uniform noise in a parameterized range added to the positional and velocity values for stochasticity. The initial z coordinate is intentionally selected to be high, thereby indicating a standing-up humanoid. The initial orientation is designed to make it face forward as well. The reward consists of four parts: (a) a reward for health at every time-step that the humanoid is alive; (b) a reward for walking forward; (c) a control cost to penalize the humanoid if it has too large of a controlled force; and (d) a negative reward if the external contact force is too large. The total reward is $(a) + (b) - (c) - (d)$. The humanoid is said to be unhealthy if the z-position of the torso is no longer contained in the specified closed interval. If parameterized to terminate when unhealthy (which is the default), the episode ends when any of the following happens: (i) truncation when the episode reaches 1,000 time-steps or (ii) termination when the humanoid is unhealthy. Otherwise, the episode ends only when truncated.

The environment of the Inverted Double Pendulum involves a cart that can move linearly and be pushed left or right, with a pole fixed on it and a second pole fixed on the other end of the first one (leaving the second pole as the only one with one free end). The goal is to balance the second pole on top of the first pole by applying continuous forces on the cart. The actions are represented by a 1-element array whose value is continuous in $[-1, 1]$ and defines the force applied to the cart (with magnitude representing the amount

of force and sign representing the direction). The state space consists of positional values of different body parts of the pendulum system, sine, and cosine of the angles of the joints, followed by the velocities of those individual parts (their derivatives), and with all the positions ordered before all the velocities, represented by an 11-element array. There is physical contact between the robot and the environment and one constraint force for contacts for each degree of freedom. All observations start in the state $[0.0, ..., 0.0]$ with a uniform noise in the range of $[-0.1, 0.1]$ added to the positional values (cart position and pole angles) and standard normal force with a standard deviation of 0.1 added to the velocity values for stochasticity. The reward consists of three parts: (a) a reward of $+10$ for each time step that the second pole is upright; (b) a distance penalty, which is a measure of how far the tip of the second pendulum (the only free end) moves; and (c) a negative reward for penalizing the agent if it moves too fast. The total reward is the $(a) - (b) - (c)$. The episode ends when any of the following happens: (i) truncate the episode when it reaches 1,000 time-steps; (ii) any of the state space values are no longer finite; or (iii) the y coordinate of the tip of the second pole is less than or equal to 1.

The Inverted Pendulum involves a cart that can moved linearly, with a pole fixed on it at one end and having another end free. The cart can be pushed left or right, and the goal is to balance the pole on the top by applying force on the cart to make the inverted pendulum stand upright (within a certain angle limit) as long as possible. The agent takes a 1-element array for actions whose values are continuous in $[-3, 3]$ and represents the numerical force applied to the cart (with magnitude representing the amount of force and sign representing the direction). The states are represented by an array consisting of 4 positional values of different body parts of the pendulum system, followed by the velocities of those individual parts (their derivatives), with all the positions ordered before all the velocities. The position of the car is measured in meters, the angles in radians, the velocity in meters/second, and the angular velocity in radians/second. All observations start in a state $[0.0, 0.0, 0.0, 0.0]$ with a uniform noise in the range of $[-0.01, 0.01]$ added to the values for stochasticity. The agent receives a reward of $+1$ for each time step that the pole is upright. The episode ends when any of the following happens: (i) truncate the episode when it reaches 1,000 time-steps; (ii) any of the state space values are no longer finite; or (iii) the absolute value of the vertical angle between the pole and the cart is greater than 0.2 radians.

The Swimmer consists of three or more segments and one less rotor joint. One rotor joint connects exactly two links to form a linear chain. The swimmer is suspended in a two-dimensional pool and always starts in the same position (subject to some deviation drawn from a uniform distribution). The goal is to move as fast as possible towards the right by applying torque on the rotors and using the fluid's friction. An action is represented by an array of two positions with values in $[-1, 1]$, which define the torques

applied in the rotors. By default, an observation consists of (i) an angle Θ_i of a segment i with respect to the axis and (ii) the derivative Θ'_i with respect to time (angular velocity). By default, the observation is an array with 8 positions with values in $[-\infty, \infty]$, containing the values of the angles, the velocity along the x-axis, and the angular velocities. It does not include the x and y coordinates of the front tip, which can be included by setting the specific parameter. All observations start in the state $[0, .., 0]$ with a uniform noise for stochasticity. The episode truncates when it reaches 1,000 time steps.

The Walker 2D is a two-dimensional and two-legged figure that consists of seven main body parts: (i) a single torso at the top (with the two legs splitting after the torso); (ii) two thighs in the middle below the torso; (iii) two legs in the bottom below the thighs; and (iv) two feet attached to the legs on which the entire body rests. The goal is to walk in the forward (right) direction by applying torques on the six hinges connecting the seven body parts. The actions are represented by an array with 6 positions and values in $[-1, 1]$, defining the torques applied at the hinge joints. Observations consist of positional values of different body parts of the walker, followed by the velocities of those individual parts (their derivatives), with all the positions ordered before all the velocities. By default, an observation is represented by an array with 17 positions and values in $[-\infty, +\infty]$. It does not include the x coordinate of the torso, which can be included by setting the specific parameter. All observations start in the state $[0.0, 1.25, ..., 0.0]$, with a uniform noise added to the values for stochasticity. The reward consists of three parts: (a) a reward for health with a fixed value at every time-step that the walker is alive; (b) a reward of walking forward; (c) a control cost for penalizing the walker if it takes actions that are too large. The total reward returned is $(a) + (b) - (c)$. The walker is said to be unhealthy if any of the following happens: (i) any of the state space values are no longer finite; (ii) the height of the walker is not in a parameterized healthy range; or (iii) the absolute value of the angle of the torso is not in the parameterized closed interval. If defined to terminate when unhealthy, the episode ends when any of the following happens: (i) truncation of the episode when it reaches 1,000 time-steps or (ii) the walker is unhealthy. Otherwise, the episode ends only at 1,000 time steps.

6.2 Learning physical control with continuous-valued actions

We evaluated the DDPG and COMCACT agents in ten environments of the Gymnasium's task set (BROCKMAN et al., 2016), whose states and actions are composed of continuous values, and using the low-dimensional state description. These environments represent both the states and the actions in vectors of different sizes and bring specific values for each task, representing positions, speeds, joint angles, angular speeds, torque, and acceleration, among others. In addition, each environment utilizes different reward

functions, transition dynamics, and end-of-episode conditions. The first two are the Mountain Car Continuous and Pendulum and are part of the Classical Control. The other eight are environments of Mujoco (TODOROV; EREZ; TASSA, 2012), which stands for multi-joint dynamics with contact: Ant, Half Cheetah, Hopper, Humanoid, Inverted Double Pendulum, Inverted Pendulum, Swimmer, and Walker 2D. A relevant aspect is the two possible ending episode conditions: (i) the occurrence of a natural final state generally defined by some event that indicates a bad condition of health of the agent or (ii) the truncation of the episode at a fixed total number of iterations. Therefore, we report the results obtained from ten trials of the DDPG agent and ten trials of the COMCACT agent in each environment so that five trials considered the truncate condition to interrupt an episode for each agent in each environment, and the other five assessed the termination condition. Because we are interested in verifying the agents' learning progress using a smaller amount of data, we always limit the total number of iterations to 50,000. We log the accumulated reward for every 100 iterations for each trial, also computing the average rewards for the last 100, 50, and 10 episodes to evaluate the learning progress. The performance progress is measured once every 5,000 steps by running the deterministic policy without action noise for ten trials and logging the final average reward. We ran the experiments using a PC gamer with an AMD Ryzen 7 processor with 16 cores (8 physical and 8 logical), 64 GB RAM, and an Nvidia 1060 with 6 GB RAM.

6.2.1 Evaluation Results

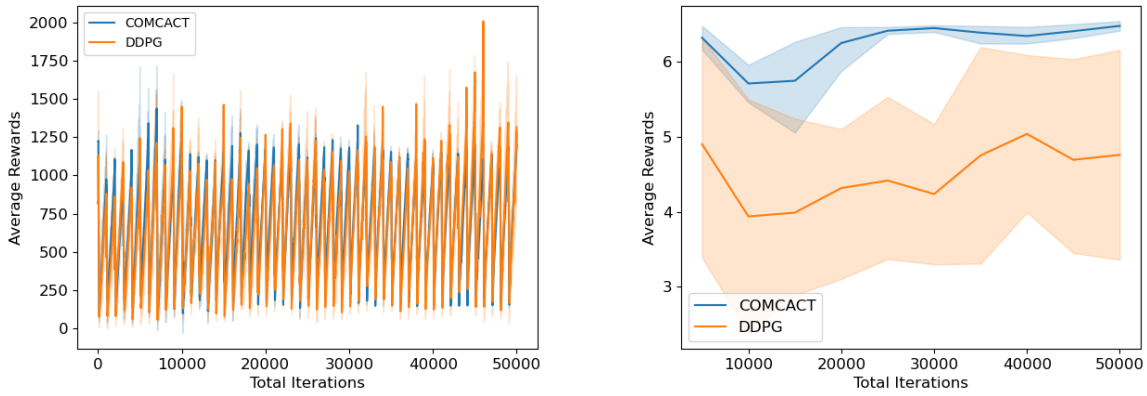
The first set of graphics represents the agent's learning progress during training and evaluation, considering the episode truncation signal. The second set does the same regarding the learning progress but considering the episode termination signal.

6.2.1.1 Results considering only the episode truncation signal

Based on the results, considering the episode's truncation, COMCACT improves efficiency and accelerates the agent's learning in some complex tasks, even in the first 50,000 iterations, which is mainly true of the set of Mujoco's environments, which are more complicated in terms of states and actions than the Classical Control environments, as presented in Section 6.1. Figures 18 to 26 present two line graphs each. One compares the average reward from five trials for COMCACT and DDPG along 50,000 training iterations, and the other compares their results in evaluation mode. We highlight the improvements in the Hopper-v4, Walker2d-v4, Ant-v4, Swimmer-v4, and Inverted Double Pendulum-v4, shown in Figures 17, 18, 19, 20, and 24, respectively. Regarding the HalfCheetah-v4, COMCACT presents slight performance improvement when training but slightly worse results when being evaluated despite showing more stable growth in reward earnings than

the DDPG, as one can observe in Figure 21. Until $\approx 40,000$ iterations, both COMCACT and DDPG present the same behavior. However, from that point ahead, COMCACT is still in the same way while DDPG improves, as one can verify in Figure 22.

Regarding the Classical Control, one can note in Figure 23 that the DDPG does not learn to act in the environment of Mountain Car Continuous-v4. COMCACT presents the best rewards when training and continuously increases its performance when evaluated, though starting with poor results. The training and evaluating logs for the LSTM demonstrate overfitting in the Q-target function at the COMPER side of COMCACT at the beginning of the agent training but not after 30,0000 iterations. The results in Figure 23 demonstrate the impact of this initial overfitting, and possibly increasing the interval between the Q-target function updating should improve the results. But even if this interval is not modified, the evaluation curve demonstrates a rapid improvement in the agent performance, which could be verified by increasing the total training iterations. This overfitting does not occur in the Inverted Pendulum-v4 and Pendulum-v1, but DDPG presents better performance, as one can verify in Figures 25 and 26.

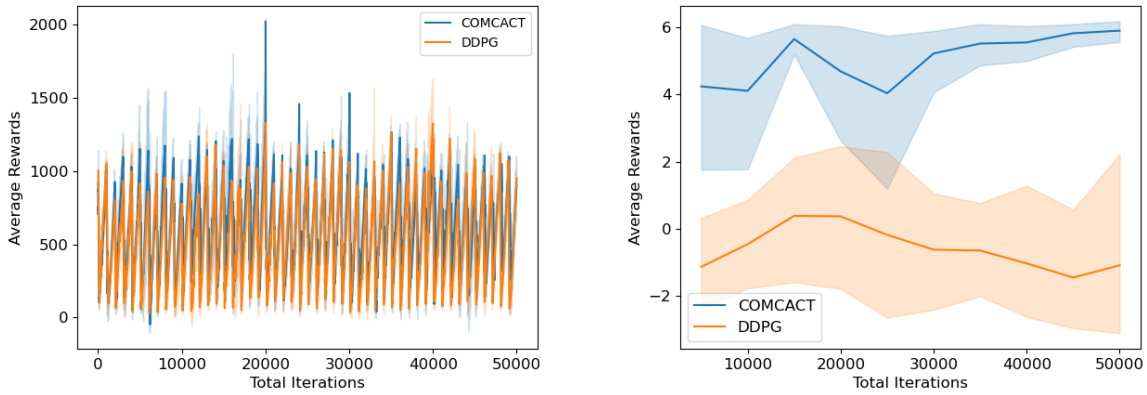


(a) Truncate – Train **(b) Truncate – Evaluate**

Figure 17 – Results for Hopper-v4 considering the episode truncation signal

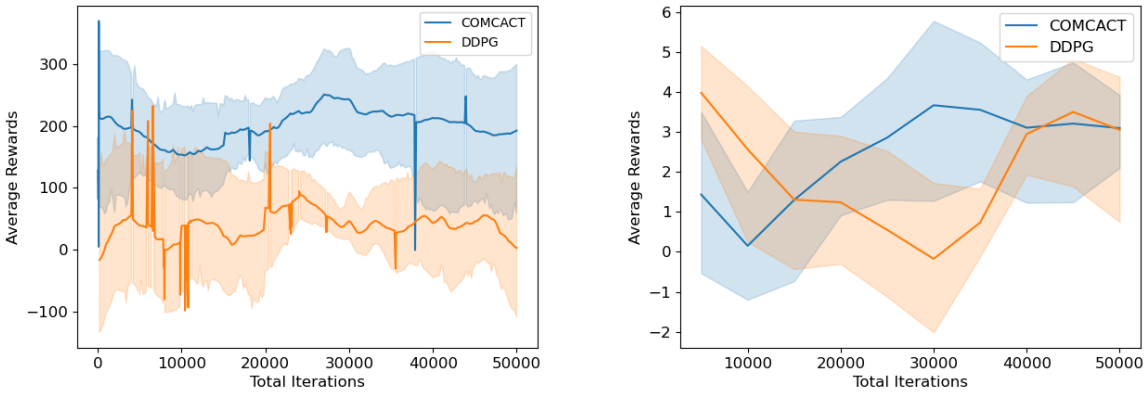
6.2.1.2 Results considering only the episode termination signal

Considering the episode termination signal, one can see that COMCACT contributes to improving the agent performance in the set of Mujoco's environments of Hopper-v4, Walker2d-v4, Ant-v4, Swimmer-v4, and Humanoid-v4, and in the Classical Control environments of Inverted Double Pendulum-v4 and Inverted Pendulum-v4 (see Figures 27, 28, 29, 30, 32, 34, and 35). Regarding the HalfCheetah-v4, as illustrated in Figure 31, COMCACT and DDPG present similar performance when training, but the evaluation results demonstrate slightly better performance of DDPG. As shown in



(a) Truncate – Train **(b) Truncate – Evaluate**

Figure 18 – Results for Walker2d-v4 considering the episode truncation signal



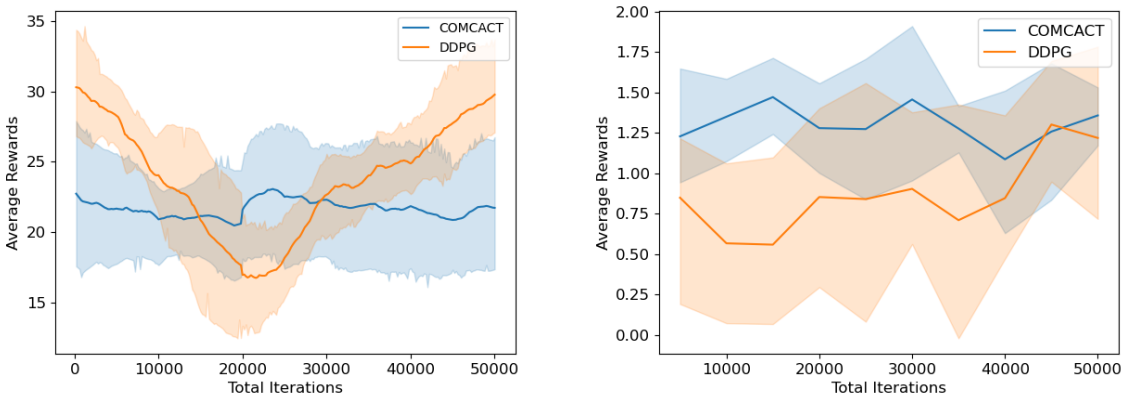
(a) Truncate – Train **(b) Truncate – Evaluate**

Figure 19 – Results for Ant-v4 considering the episode truncation signal

Figure 33, COMCACT performs worse than DDPG for both training and evaluating in the environment of Mountain Car Continuous-v0. In Pendulum-v1, COMCACT performs worse than DDPG when training, but we consider similar performances when evaluating both agents, as displayed in Figure 36.

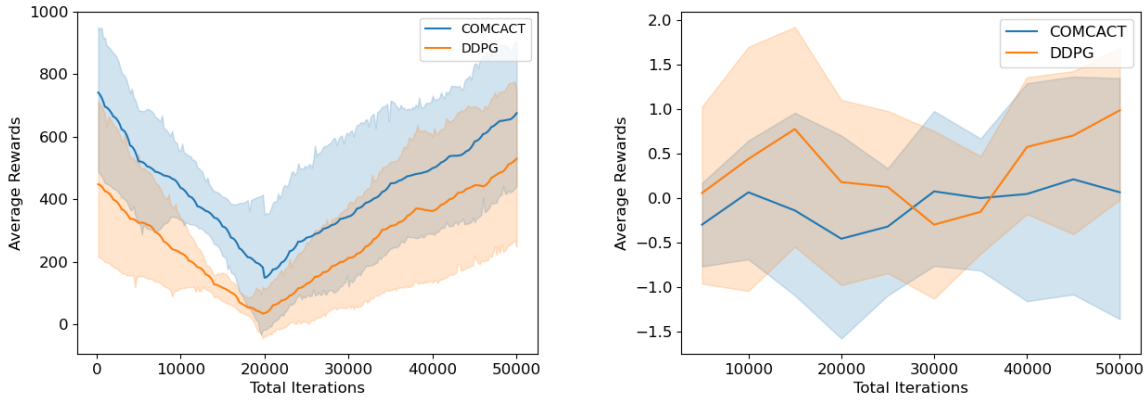
6.3 Considerations

We proposed and evaluated the COMCACT aiming to bring the advantages verified on COMPER regarding more data-efficient experience replay to problems involving continuous-evaluated actions. The results demonstrate that COMCACT clearly and considerably improves the agent performance in two large set environments: Mujoco and



(a) Truncate – Train **(b) Truncate – Evaluate**

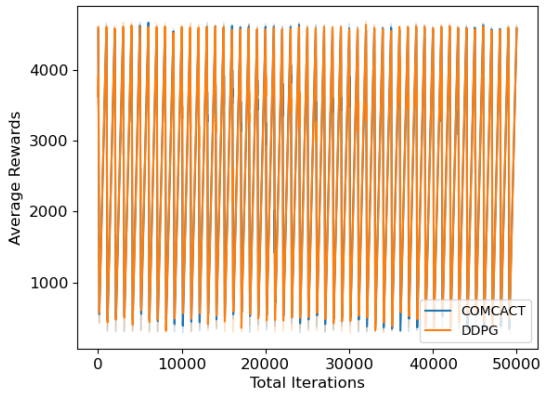
Figure 20 – Results for Swimmer-v4 considering the episode truncation signal



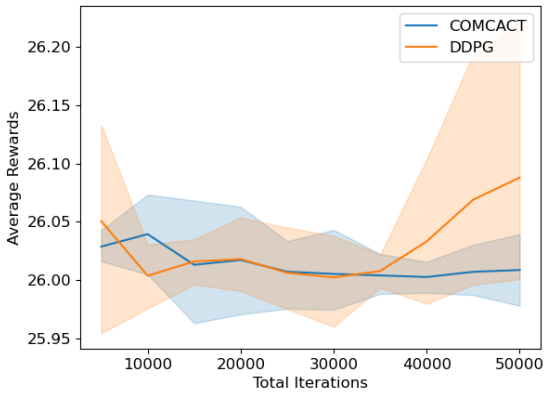
(a) Truncate – Train **(b) Truncate – Evaluate**

Figure 21 – Results for Half Cheetah-v4 considering the episode truncation signal

Classical Control. Some results may indicate possible future works to explore better hyperparameter tunning, including the similarity threshold and the limit of iterations. Anyway, this work demonstrates that it is possible to improve the performance of model-free and off-policy agents in problems with continuous-valued states and action representations using the propositions of COMPER to improve data efficiency in learning from experience replay.

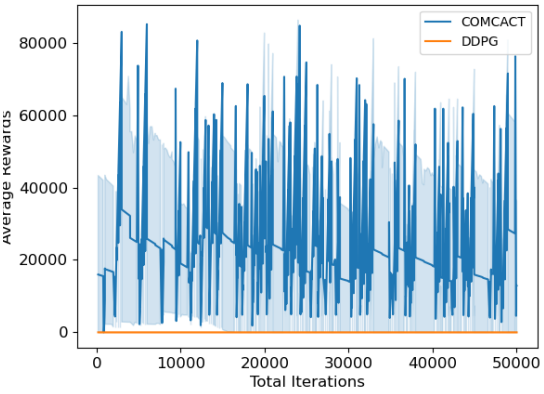


(a) Truncate – Train

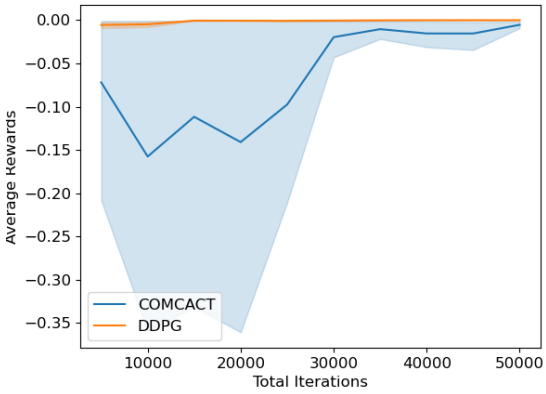


(b) Truncate – Evaluate

Figure 22 – Results for Humanoid-v4 considering the episode truncation signal

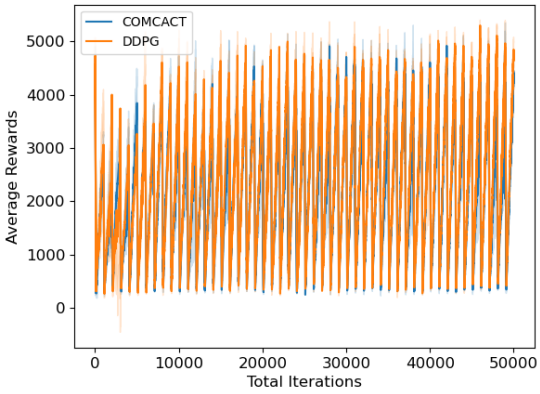


(a) Truncate – Train

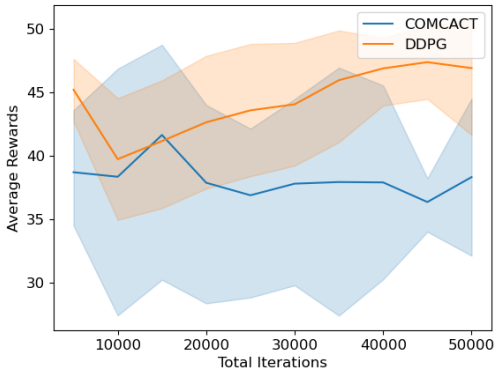


(b) Truncate – Evaluate

Figure 23 – Results for Mountain Car-v0 considering the episode truncation signal

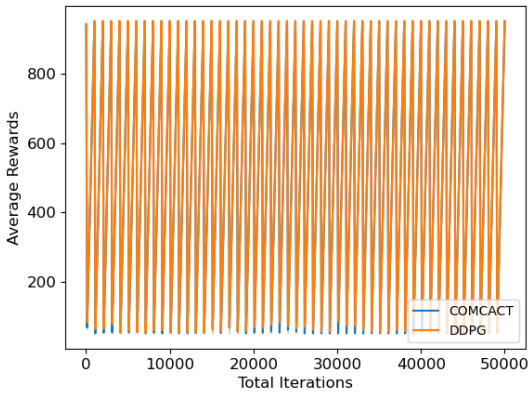


(a) Truncate – Train

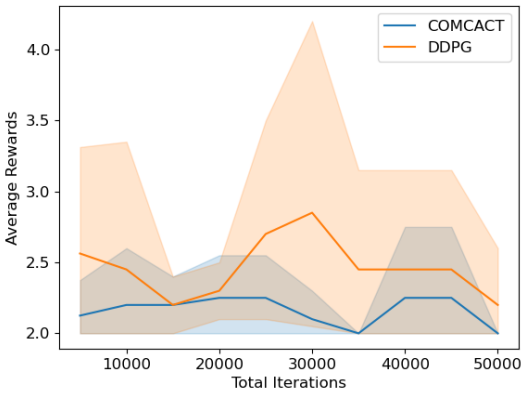


(b) Truncate – Evaluate

Figure 24 – Results for Inverted Double Pendulum-v4 considering the episode truncation signal

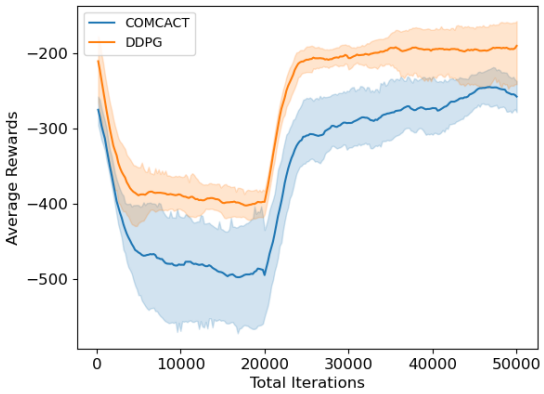


(a) Truncate – Train

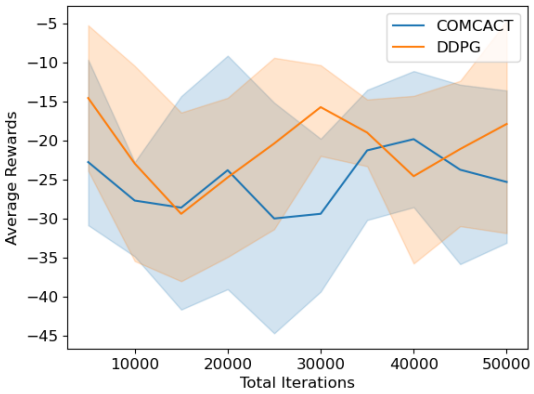


(b) Truncate – Evaluate

Figure 25 – Results for Inverted Pendulum-v4 considering the episode truncation signal

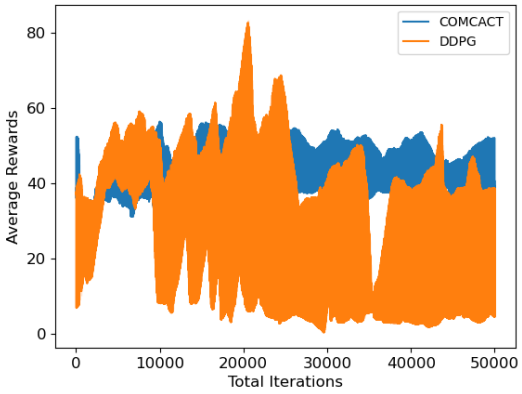


(a) Truncate – Train

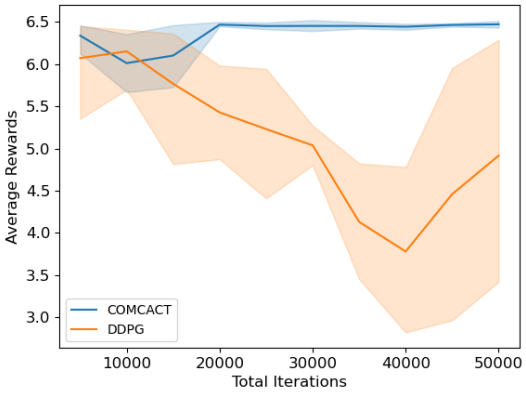


(b) Truncate – Evaluate

Figure 26 – Results for Pendulum-v1 considering the episode truncation signal

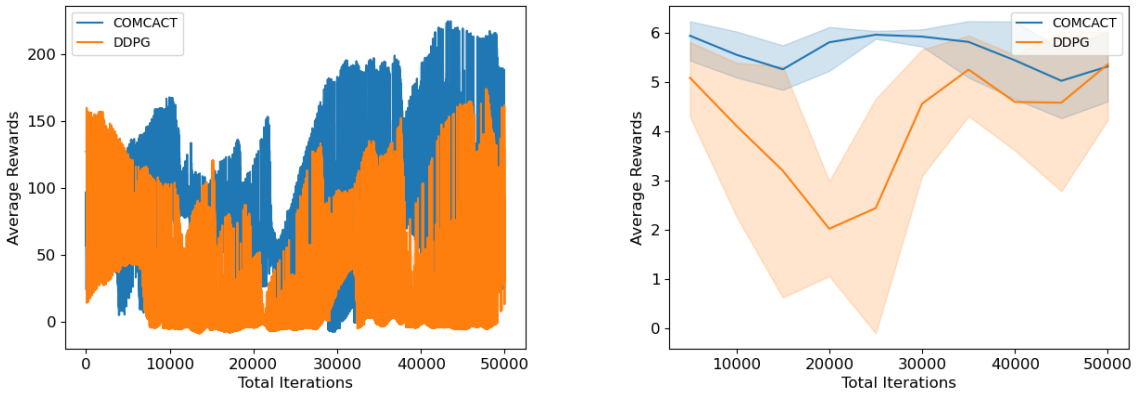


(a) Done Signal – Train



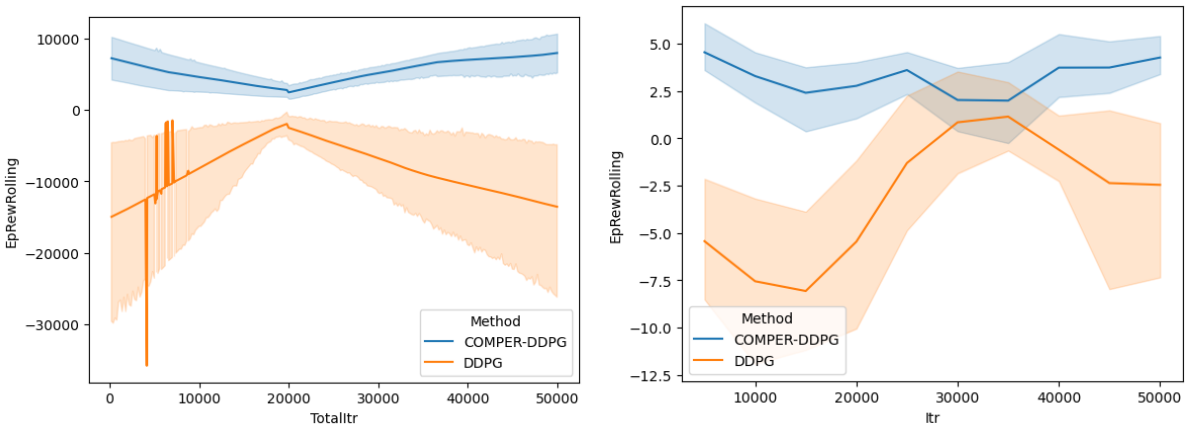
(b) Done Signal – Evaluate

Figure 27 – Results for Hopper-v4 considering the episode done signal.



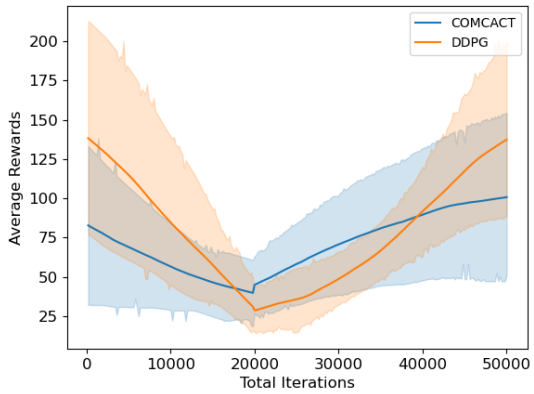
(a) Done Signal – Train (b) Done Signal – Evaluation

Figure 28 – Results for Walker2d-v4 considering the episode done signal

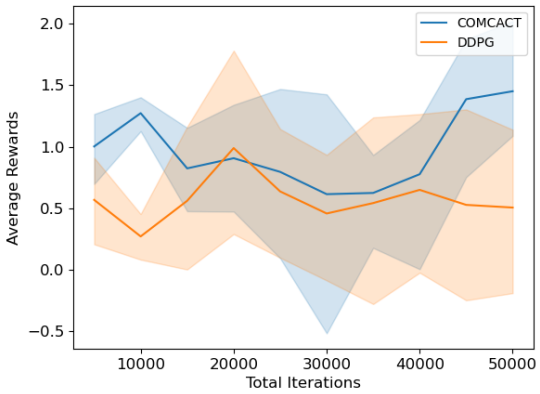


(a) Done Signal – Train (b) Done Signal – Evaluation

Figure 29 – Results for Ant-v4 considering the episode done signal

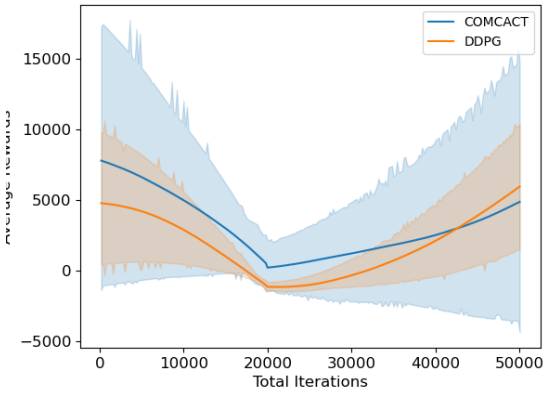


(a) Done Signal – Training

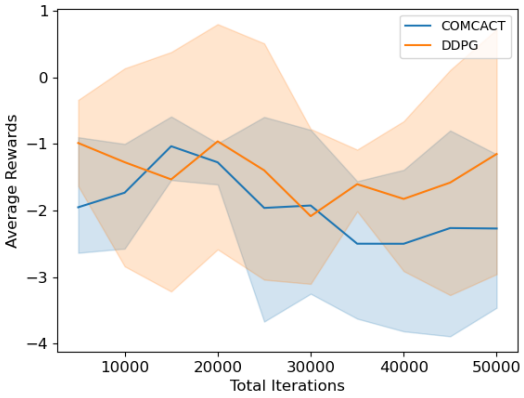


(b) Done Signal – Evaluation

Figure 30 – Results for Swimmer-v4 considering the episode done signal

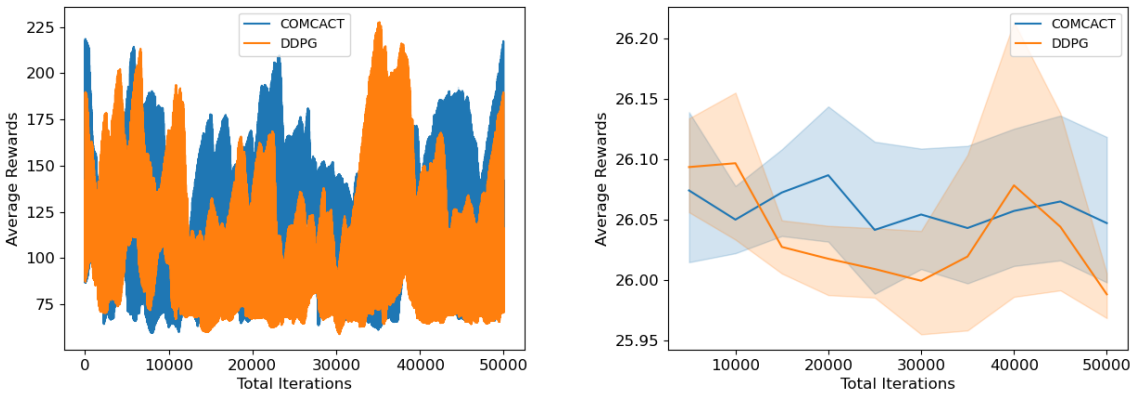


(a) Done Signal – Train



(b) Done Signal – Evaluation

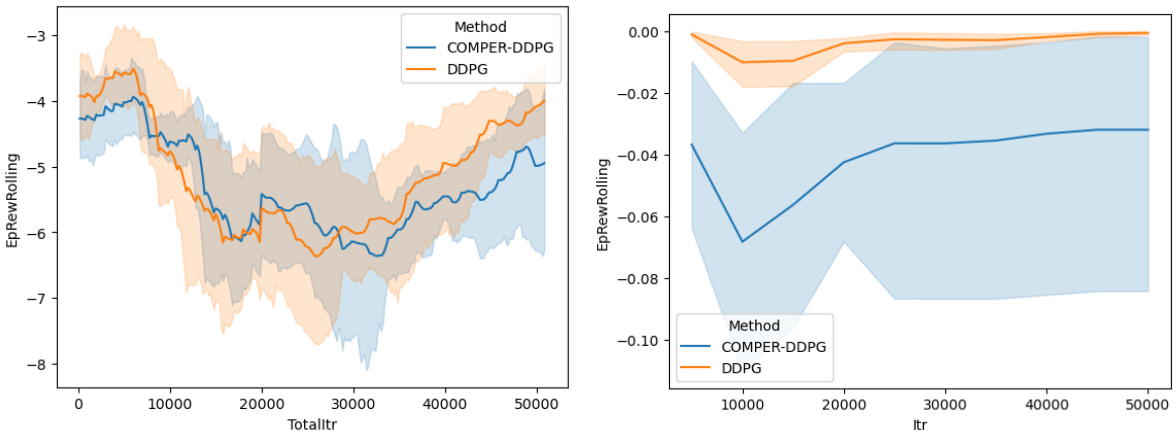
Figure 31 – Results for Half Cheetah-v4 considering the episode done signal



(a) Done Signal – Train

(b) Done Signal – Evaluation

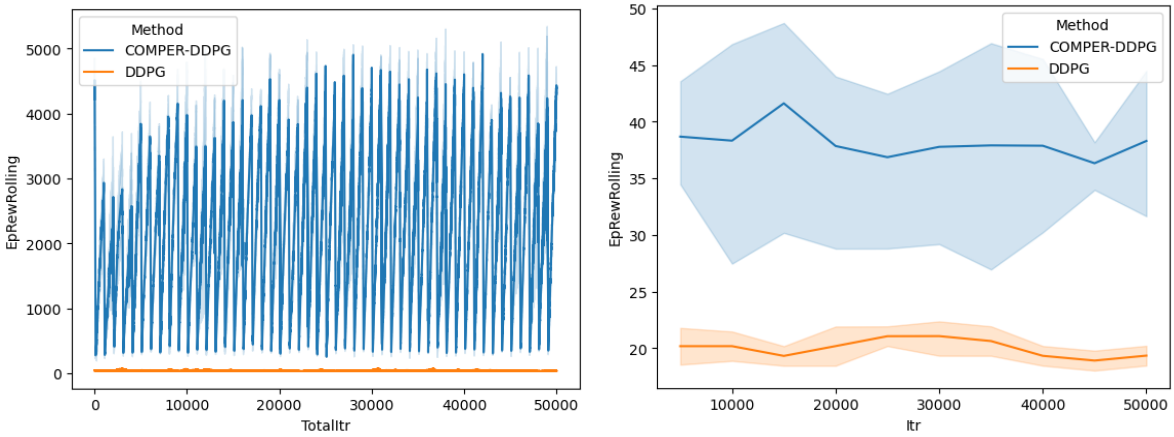
Figure 32 – Results for Humanoid-v4 considering the episode done signal



(a) Done Signal – Train

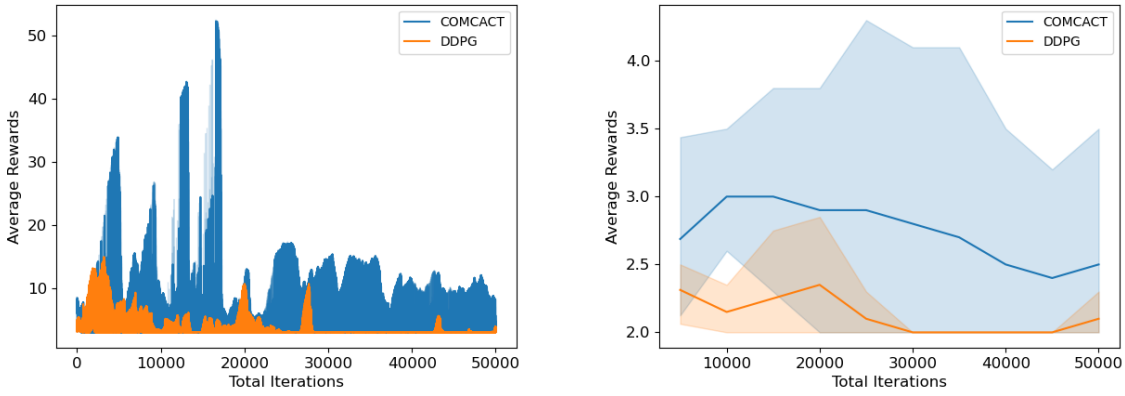
(b) Done Signal – Evaluation

Figure 33 – Results for Mountain Car Continuous-v0 considering the episode done signal



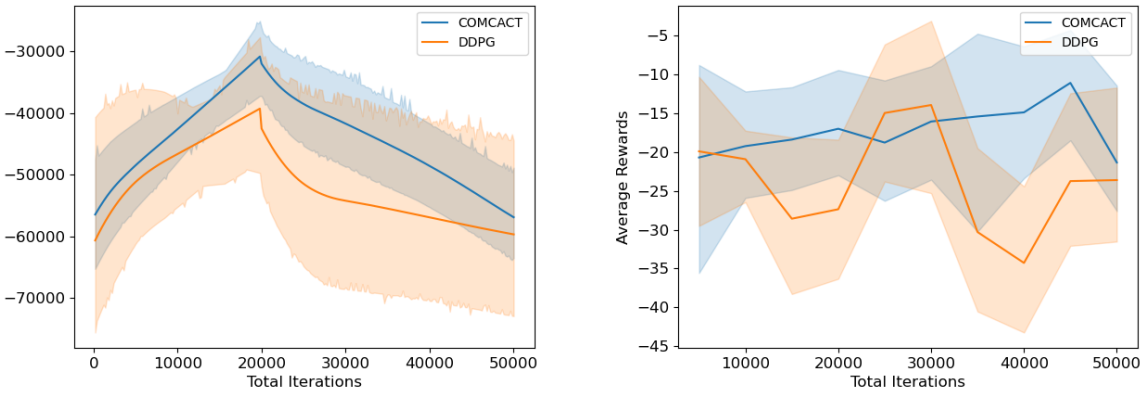
(a) Done Signal – Train (b) Done Signal – Evaluation

Figure 34 – Results for Inverted Double Pendulum-v4 considering the episode done signal



(a) Done Signal – Train (b) Done Signal – Evaluation

Figure 35 – Results for Inverted Pendulum-v4 considering the episode done signal



(a) Done Signal – Train **(b) Done Signal – Evaluation**

Figure 36 – Results for Pendulum-v1 considering the episode done signal

7 ABLATIVE STUDY ON COMPER'S COMPONENTS

The COMPER uses a compact transitions memory and predicts target values to temporal difference learning using recurrence over sets of similar transitions, seeking to make ER more efficient using smaller amounts of data. However, it is not clear what each component's specific contribution is to the method's effectiveness. We conducted a detailed analysis, isolating these components to verify and understand how they operate and contribute to improving data efficiency. We show that removing any of these components impairs the agent's performance, but it still obtains better results than a DQN agent we used as a baseline.

7.1 Contributions of Reduced Memory and Q-Target Funcion

Two main components operate together in COMPER: (i) a reduced transition memory and (ii) a recurrent target network, which implies a change in the loss function for TD-error calculation. The propositions developed in COMPER approach relevant issues in the literature regarding data efficiency when, for example, it composes a smaller but not explicitly size-limited replay buffer. Moreover, it demonstrates that agents could learn not only from past experiences but also from past similar experiences (showing that they occur) and that exploiting these similarities can improve the agent learning process. This section aims to understand how each component (the reduced memory and the recurrent target network) contributes to performance improvements and determine whether using only one could aggregate some benefit in data efficiency. Therefore, we isolated these components and analyzed their impact on agent performance. Thus, a COMPER agent is trained and compared to those obtained by removing each component separately. We also compare its results with those obtained by a DQN agent.

Our objective was to independently assess (i) the RTM and (ii) the Q-target function regarding its relevance to COMPER since each of these components responds to one of the two primary hypotheses that supported the method proposition as discussed in Section 1.3. We made surgical changes from implementing the original method to generating two other versions, isolating the use of each component. We called COMPER-F (see Figure 37) the version of COMPER that uses the mechanisms to reduce $\mathcal{TM}$ to $\mathcal{RTM}$ and samples uniformly from RTM but does not use either the Q-target function approximated by the LSTM or the loss computation of COMPER. In place, it uses the Convolutional Neural Network (CNN) and the same loss function as DQN. By COMPER-M (see Figure 38), we called the version which uses the Q-target function approximated by the LSTM and the loss computation of COMPER but updates the Q-value function by uniformly sampling directly from $\mathcal{TM}$ instead from $\mathcal{RTM}$.

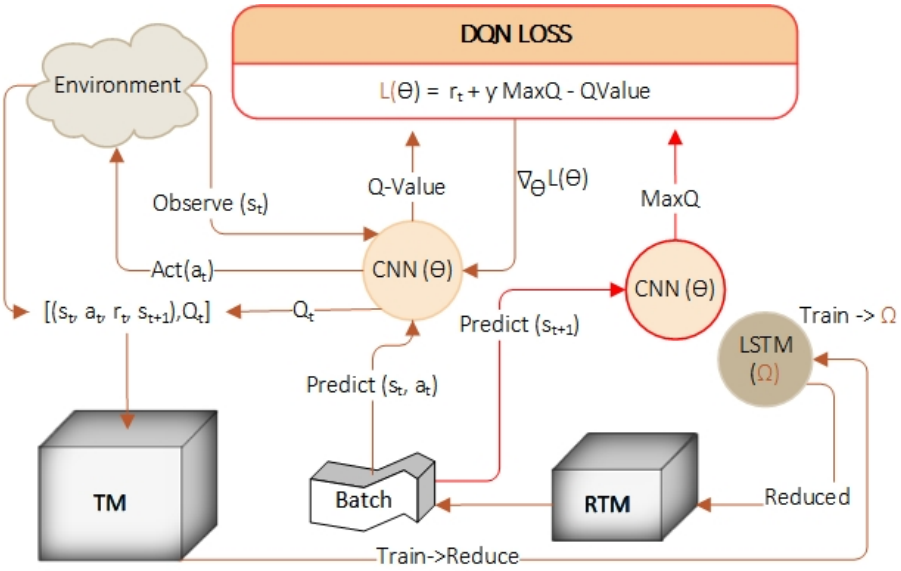


Figure 37 – Version 1: COMPER-F

To evaluate COMPER, COMPER-*F*, COMPER-*M*, and DQN, we adopted the same experimental protocol from Section 5.3. We defined a limit of 100,000 frames for each training run and logged the results at every 100 iterations and at the end of each episode. We divided the number of episodes obtained in each run to split the results into three checkpoints, calculated the average scores (with standard deviations) for the last episodes preceding each checkpoint, and used a decay rate of ϵ, from 1.0 to 0.01 in 90,000 frames. The weights Ω in $QT(\tau_t, \Omega)$ were updated through backpropagation over sets of similar transitions every 100 agent steps, using a similarity threshold $\delta = 10^{-4}$. In turn, the update of $Q(s_t, a_t, \Theta)$ was performed every four agent steps. Moreover, we started weight updating after the first 100 agent steps. For COMPER, we represented the states of the environment with a single frame in a matrix with dimensions of $84 \times 84 \times 1$, which is the best format for COMPER . For DQN, we stacked the frames in a matrix of $84 \times 84 \times 4$. In any case, we used only the luminance values for every two frames composed by the color-averaging method of ALE. We implemented our DQN agent as defined in Mnih et al. (2015) and only adopted new hyperparameters values for the total number of frames, the ϵ value decay, the results log frequency, the learning start, and the target function update frequency to accomplish our experiments. Moreover, we started weight updating after the first 100 steps in COMPER and 1,000 steps in DQN. Finally, we adjusted the hyperparameters in three training runs in Space Invaders, using the best values to execute the experiments on all other games.

In Table 29, we pick up some previous results from Section 5.3.1 as a reference performance for baseline. In Table 30, we present our results and highlight the differences between the three versions of COMPER. Both tables contain the average scores (standard

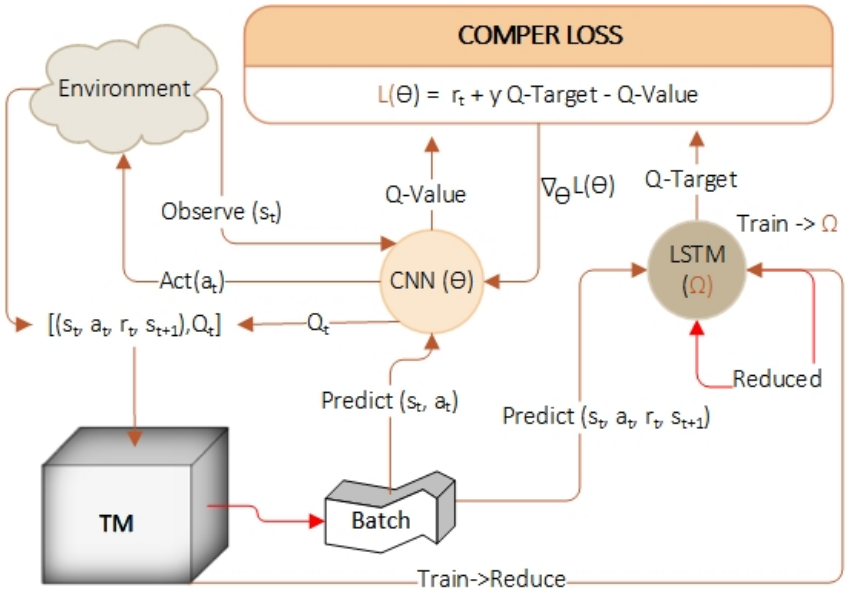


Figure 38 – Version 2: COMPER-M

deviation in parentheses and best results in bold) calculated for the last 3* or 10 training episodes over five training trials with a limit of 100,000 frames. For some games, the scores were averaged only over the last three episodes because of the fewer total episodes obtained than the other games.

We report our ablation results of training COMPER, COMPER-F, COMPER-M, and DQN on five games, considering they represent different challenges regarding exploration, randomness, and sparse rewarding. It is interesting to note that removing any of the two main components of COMPER will visibly worsen its performance in all games. The unique exception is in Asterix, for which the three versions of the method achieved similar results.

COMPER presents the best results for Freeway. From Table 30, one can note that in both COMPER-F and COMPER-M, the agent learned the policy in only one of the five trials, as illustrated in Figure 39, making it initially difficult to attribute specific credit for one or the other component of the original COMPER. In any case, they achieved better results than DQN. We hypothesize that the best performance depends on all COMPER's components working together when it achieves the overall best and most stable results. Freeway presents very sparse rewards, but on the other hand, the agent can learn how to cross the road once it learns a simple policy: get moving forward and go ahead. However, to discover that, it needs to reach the other side of the road at least once (probably more) because only then will it receive a reward for each successful trial, which is not easy. After realizing this in one episode of a training trial, the agent starts to accumulate points successively, episode by episode, as we can see in the logs and reflected in Figure 39.

Table 29 – Baseline – Average scores of the last 3* and 10 training episodes

	COMPER	DQN
Freeway*	**22.33** (0.0)	0.0 (0.0)
Asteroids	**827.8** (142.93)	237.80 (47.57)
Battle Zone*	**4,733.33** (2,080.6)	1.666.66 (1,135.29)
Video Pinbal*	**7,353.26** (7,366.18)	6,936.13 (2,992.37)
Seaquest	**279.6** (24.21)	132.0 (10.2)
Asterix	**258.0** (130.64)	182.00 (22.05)
Beam Rider*	509.06 (89.83)	**563.20** (89.26)
Space Invader	182.2 (68.27)	**199.80** (49.78)

It is interesting to note that the original COMPER could learn and exploit it more times.

In Seaquest, the agent controls a submarine that needs to identify the position of each obstacle or opponent to destroy or avoid it and rescue the divers, which it must collect to bring to the surface, in a very complex dynamic with differences in the value of penalties or rewards. Moreover, an air supply represents when it can be submerged and run out until it emerges and can now convert all the remaining time into a corresponding score. One can note that learning these tasks requires complex exploration and long-term action policies learning. Also, in this case, removing any of the two components worsens the agent's performance, producing less consistent learning between trials, as seen in Figure 40, and removing the Q-target function (in COMPER-F) worsen the agent's performance than removing the reduced memory (in COMPER-M) regarding the average score. However, in COMPER-M, there is a considerable standard deviation, such as in the other game, and we will discuss this following.

In Asterix, the agent must collect rewarding elements that rapidly and randomly appear and disappear in a scenario where collision and penalizing objects move horizontally in different rows. It seems to impose learning from more exploration and depending on the number of frames the agent observes. COMPER-M presents the worst result, especially considering its standard deviation, while COMPER-F presents the best result than COMPER. However, if we check the averaged scores in different training trials in Figure 41, we can note that COMPER-F produced a very high score in only one training trial but scored worse in the others. Again in Beam Rider, COMPER presents

Table 30 – Ablative Versions – Average scores of the last 3* and 10 training episodes

	COMPER	COMPER-F	COMPER-M	DQN
Freeway*	**20.7** (0.8)	4.06 (8.13)	4.06 (8.13)	0.00 (0.00)
Seaquest	**388.8** (23.95)	72.0 (56.36)	140.8 (115.31)	123.2 (17.75)
Asterix	254.0 (85.35)	**286.0** (68.44)	239.0 (124.23)	166.0 (51.28)
Beam Rider*	**547.46** (49.54)	434.66 (80.74)	366.66 (140.06)	559.2 (99.79)
Space Invaders	**177.2** (51.37)	175.1 (33.56)	171.3 (81.24)	204.9 (33.63)

the best results compared with its other versions. The agent opponents appear in the upper portion of the video frame (projected in perspective), moving in four directions and the diagonal. They shrink or increase in size as they move closer or further away from the agent. More distant and smaller enemies are more challenging to hit. Also, some opponents reward more than others in similar but more complex dynamics that Space Invaders. COMPER-M presents the worst result and larger standard deviation, compared with COMPER-F, but with COMPER presenting the best result. However, checking the averaged scores Figure 42, we can note that COMPER-F produced a very high score in only one training trial but scored worse in the others, similar to Asterix. We observe the same behavior in Space Invaders, such as in Asterix and Beam Rider.

From our analysis of the averaged score and standard deviation and looking at the graphics that show the distribution of the results over the training trials, one can note that in all the games (except for Freeway, for which it is not possible to discriminate), the behavior of COMPER-F is to produce, in some of the trials, scores outside the curve, which are higher than the scores of the other trials, leading to most significant standard deviation when comparing results between trials, as illustrated in Figure 43. In turn, COMPER-M leads to a more stable behavior, with closer results among the five attempts. When looking at COMPER, which is the complete method, one could note that its results are better than its variation and DQN and, simultaneously, stable between trials and along the checkpoints. Therefore, we could state that removing any of these components impairs the performance of COMPER. However, it still obtained better results than a DQN agent we used as a baseline. We also could hypothesize that (i) using the RTM contributes more to exploiting opportunities from good experiences but in more rare occurrences, based on the results of COMPER-F and (ii) using recurrence over Q-value updates from similar transitions to learn its dynamics and predict the Q-target value contributes to bootstrapping from those good experiences based on the results of

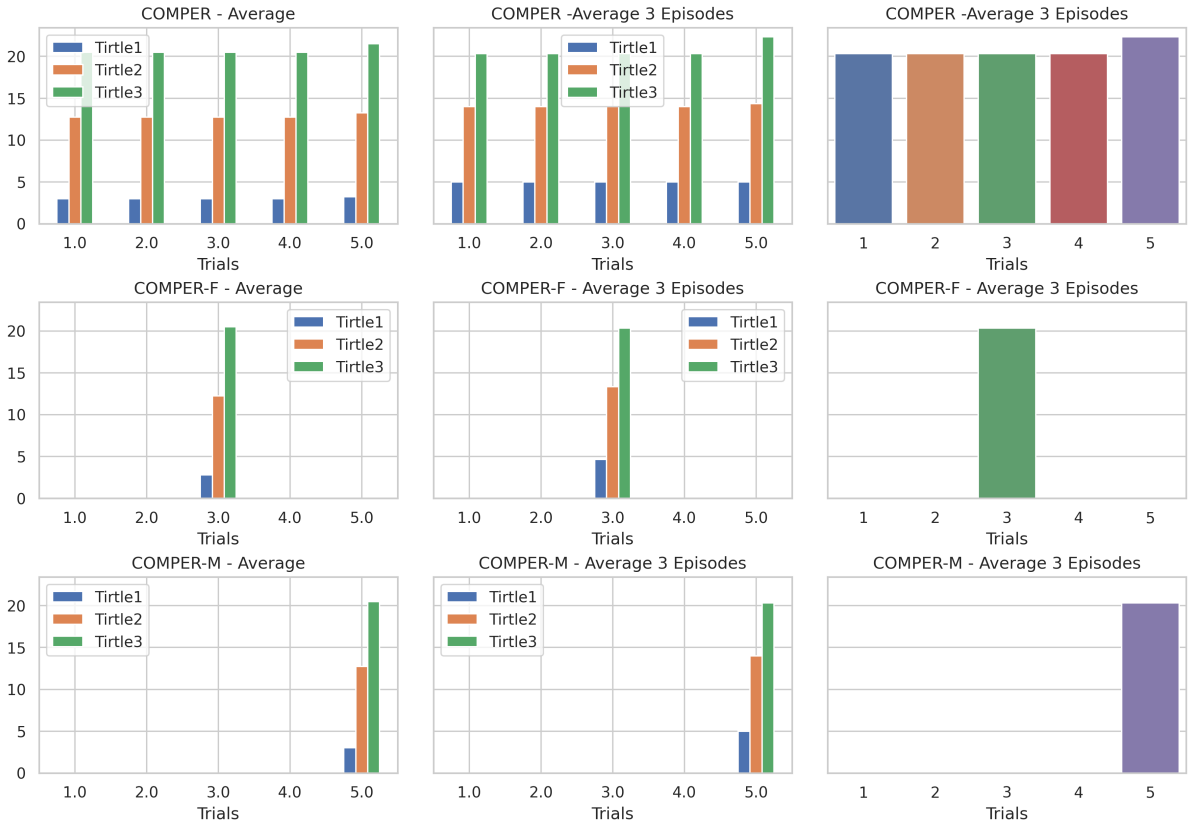


Figure 39 – Freeway - Average reward values

COMPER-M. Moreover, we could hypothesize that the above statement (i) is closely related to improving the chances of sampling more informative experiences and experiences more closely to the current policy, and statement (ii) is related to improving the Q-value function updating due to better target value estimating. In any case, we can state that the performance of COMPER is due to the joint work of all its components.

Table 31 presents the averaged scores for each checkpoint's last 3* and 5 episodes as defined in Section 5.3.1. Figures 43a, 43b, 44a, and 44b presents the average score and standard deviation for five trials at each checkpoint (CKPT).

Table 31 – Average scores of the last 3* and 5 last episodes of each tercile

	First Checkpoint			Second Checkpoint			Third Checkpoint		
	COMPER	COMPER-F	COMPER-M	COMPER	COMPER-F	COMPER-M	COMPER	COMPER-F	COMPER-M
Freeway	5.0 (0.0)	1.00 (2.23)	0.93 (2.08)	14.6 (0.14)	2.80 (6.26)	2.66 (5.96)	20.73 (0.89)	4.06 (9.09)	4.06 (9.09)
Seaquest	184.0 (22.62)	92.0 (89.97)	88.8 (45.81)	416.0 (33.58)	82.4 (106.15)	123.2 (139.23)	355.2 (10.35)	59.2 (48.11)	100.0 (74.99)
Asterix	266.0 (40.92)	248.0 (51.18)	258.0 (35.63)	294.0 (54.12)	262.0 (69.78)	292.0 (74.96)	254.0 (101.63)	296.0 (124.41)	264.0 (211.49)
Beam Rider	393.06 (108.47)	396.00 (49.73)	442.13 (100.90)	493.06 (55.85)	473.06 (112.09)	340.26 (120.23)	547.46 (55.38)	434.66 (90.27)	366.66 (156.59)
Space Invaders	127.80 (32.25)	162.4 (69.70)	160.6 (52.91)	209.48 (88.32)	152.8 (35.71)	150.2 (26.67)	170.40 (49.76)	177.8 (87.82)	143.0 (86.98)

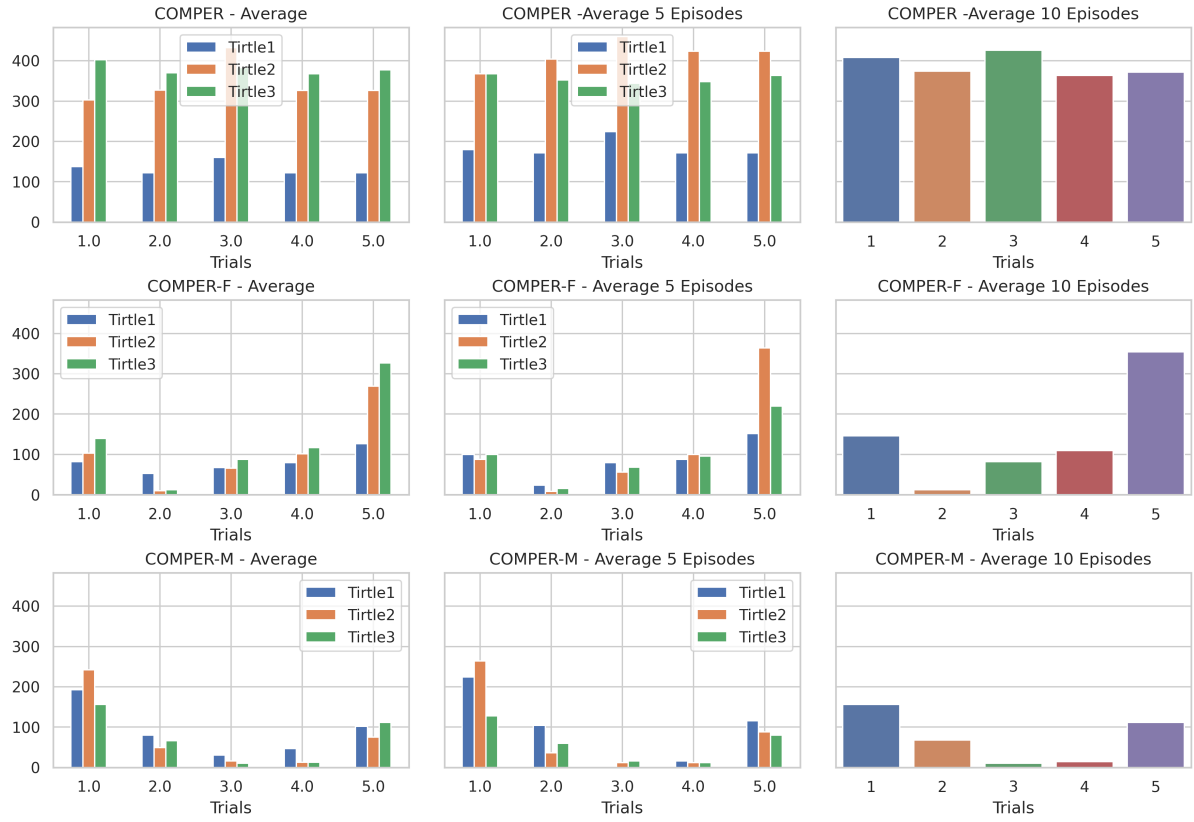


Figure 40 – Seaquest - Average reward values

7.2 Considerations

The propositions of COMPER approach relevant issues in the literature. Avoiding storing many potentially similar experiences makes it take longer until the memory reaches its limit may reduce catastrophic forgetting. Not explicitly limiting the memory size but making it grow slowly to hold a representative of each observed transition mitigate the bias of a significantly smaller memory and still increase the sampling diversity. Furthermore, by learning the dynamics of updating Q-value estimates and predicting the target value from similar future transitions, the method improves the update of the Q-value function. Our objectives were to assess the behavior of each method's main components and how much each one contributes to overall performance. The best result is when the components work together, but each separately can still contribute, in different ways, to achieving good results on average. These results indicate the possibility of employing the method propositions and implementing its components, together or not, in other reinforcement learning methods or to investigate specific problems and methods that could benefit from the possibilities of mitigating catastrophic forgetting or using a smaller but less biased replay buffer.

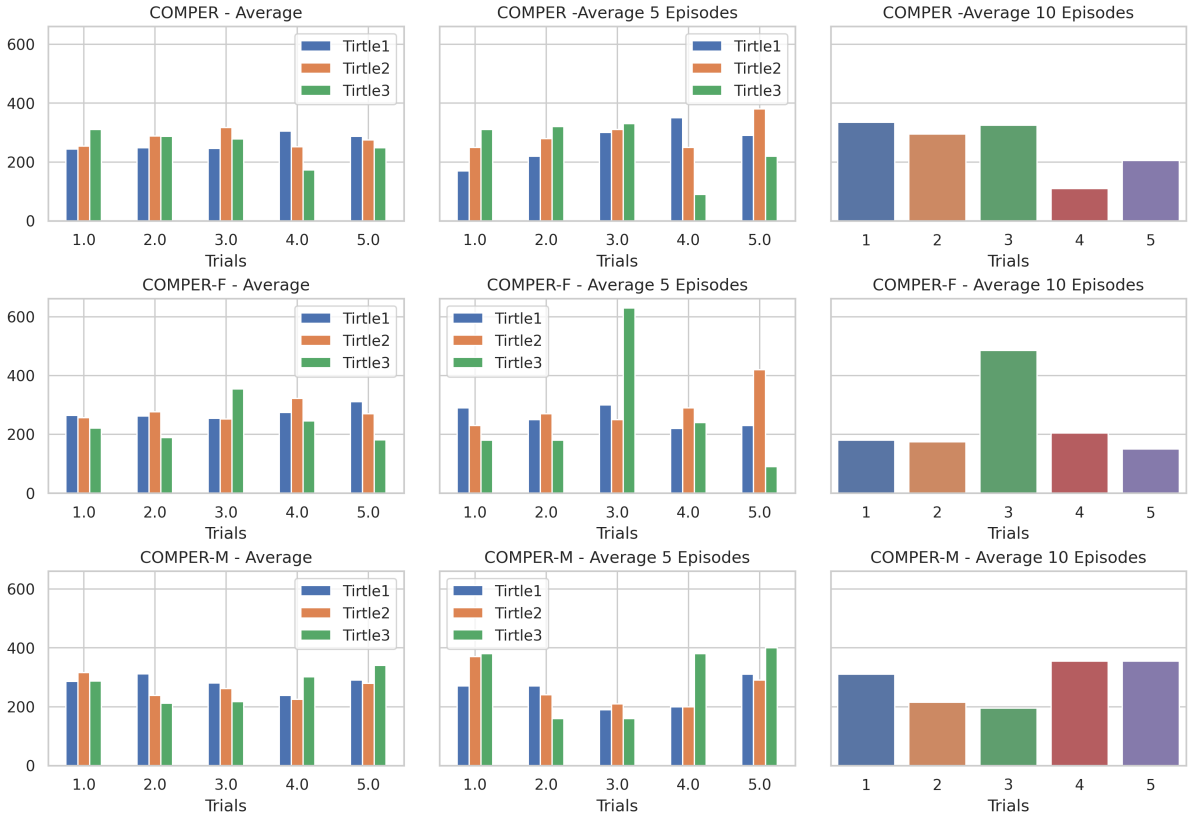


Figure 41 – Asterix - Average reward values

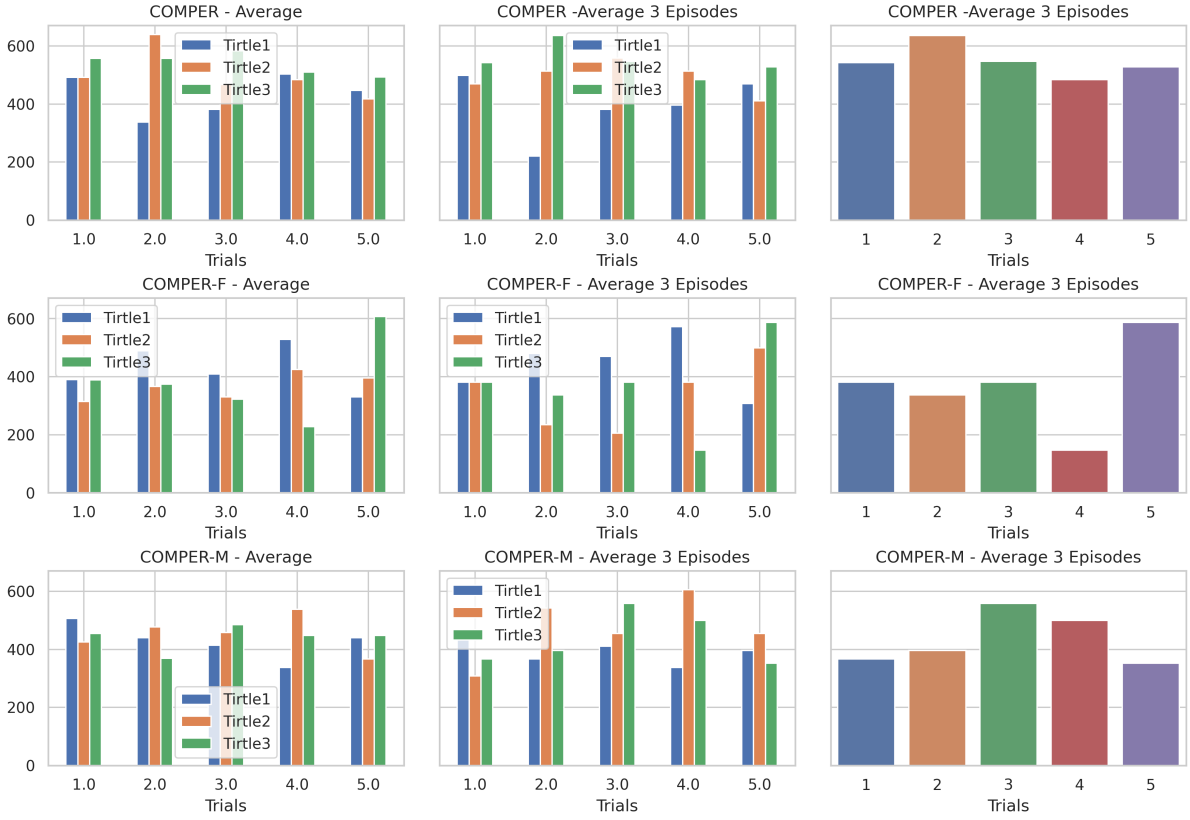
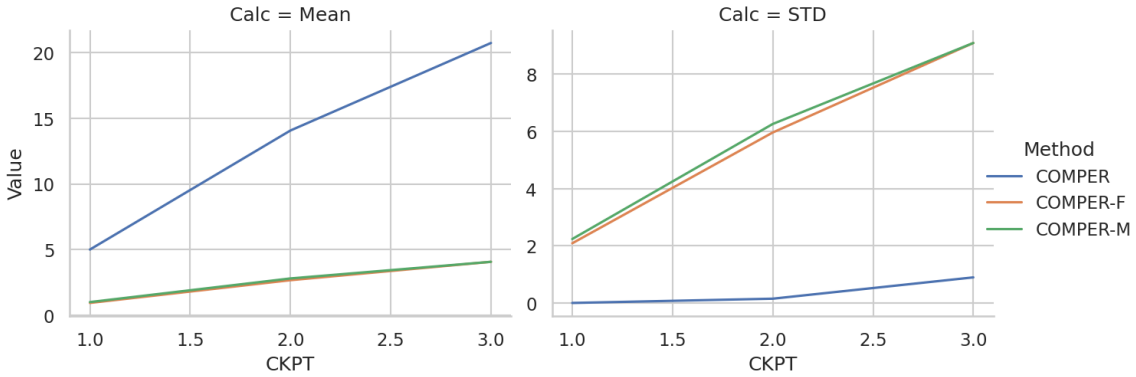
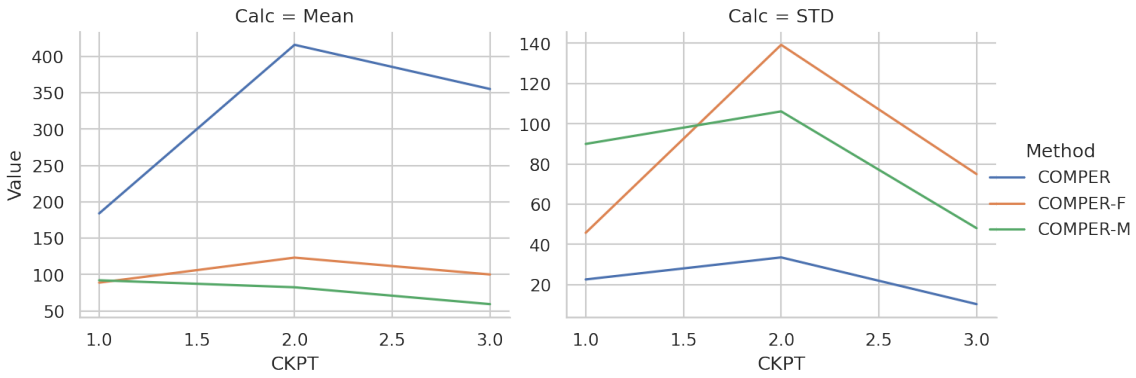


Figure 42 – Beam Rider - Average reward values

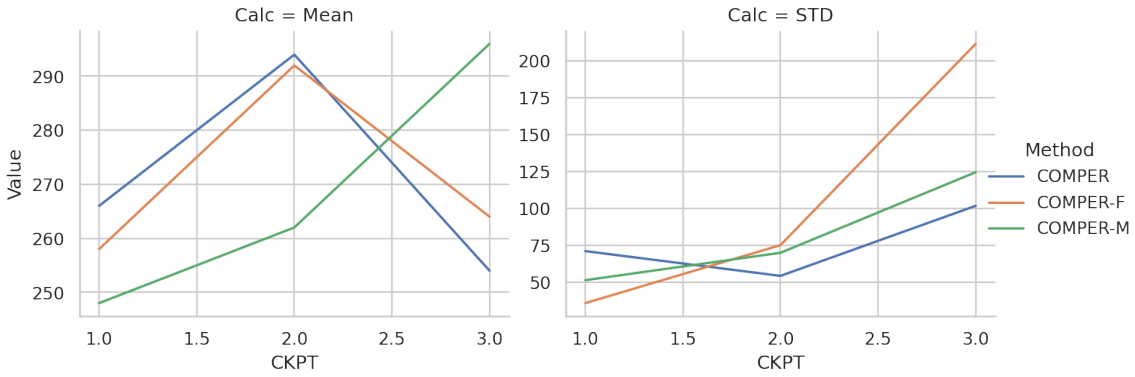


(a) Freeway

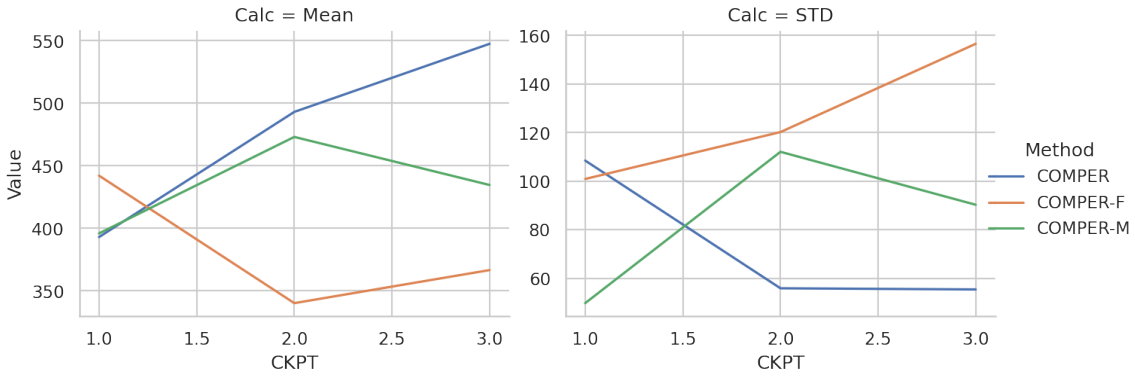


(b) Seaquest

Figure 43 – Average scores and standard deviation for five trials at each checkpoint



(a) Asterix



(b) Beam Rider

Figure 44 – Average scores and standard deviation for five trials at each checkpoint

8 CONCLUSIONS

Motivated by the idea that there would be some relationship, pattern, or information contained in the agents' experiences that would allow them to project one step ahead, as if in a predictive way and not only based on a mathematical expectation, which experiences could contribute more effectively to the learning process, not at the given moment of its sampling, but soon in the future; inspired by the impression that every human being has that they will become capable of dealing better with possible future situations, even if they have not yet experienced them, due to their experience with similar situations; and hypothesizing that this ability seems to be related not only to patterns of events but also to patterns of successive events over time, this work formalized a complex thesis about learning from similar experiences organized in sets and sampled from a compact transition memory. Its theoretical contributions, heavily based on the literature and the theory of reinforcement learning, brought relevant discussions on assumptions we tend to assume as truth (in any case) regarding the memories of agents' experiences, the value function approximation models, and many other relevant aspects. Moreover, this research investigated each of its hypotheses by formally proposing a new method of experience replay and a new target function for temporal difference learning. It empirically corroborated each hypothesis by proposing and evaluating two new reinforcement learning methods with compact experience replay. Their results proved the hypothesis, allowed us to understand their behaviors and algorithm decisions, and demonstrated how this research proposition makes off-policy and model-free reinforcement methods with experience replay more data-efficient.

From Section 1.3, we recall our hypotheses to approach the problems discussed in Section 1.2. They are based on the idea that sampling from a compact memory and learning to predict the Q-target values from similar transition sets could make experience replay more data-efficient and produce more effective update steps for temporal difference learning. In summary, we hypothesized that we could think about a compact memory of experiences, which is capable, at the same time, of (i) being fresher but not so biased, (ii) incrementally growing but not linearly at the number of interactions of the agent and (iii) not so large that faster makes lesser likely to resemble rare and expensive experiences and make possible to learning its dynamics to improve the estimations of the long-run reward used to update the agents' value functions.

Therefore, as part of our general propositions, in Section 4.1.1, we formally defined (i) how to model the memory of the transition as diverse sets of similar transitions, (ii) how to define these sets, (iii) how to identify and store and index similar transitions, and how (iv) to compact that lager transitions memory to a reduced transitions memory and, in the same time (v) learn a model to predict Q-target values.

The results in Section 5.3.1 and 6.2.1 demonstrate how applying our propositions improved the agents' performances. In addition, Section 5.3.1 empirically verified the behavior of memories and presented a detailed analysis of the number of similar transitions throughout the training episodes and the number and sizes of the similar transitions sets. We could verify three main behaviors: (i) new sets were created in different moments; (ii) some of these sets are removed from the memory and do not occur again, while others remain updated until the last training episode; and (iii) when using single frame, the number of similar transitions is larger than when using stacked frames. This information is relevant because the number of similarities implies the number of times the Q-value for similar transitions was updated instead of reinserting a new tuple in the memory, which helps maintain its size small and under control. As the number of observed frames increases throughout the successive episodes, the size of the transitions memory varies but keeps an upper limit because of the reduction step applied when similar transitions are removed to train the Q-target network and update the compact memory. The small size of the compact memory is verified directly by the total number of unique transitions at the end of each run. And, even sampling from such a small number of transitions, our method manages to achieve better results. All these results directly corroborate Hypotheses 1, 2, and 3 regarding (i) the occurrence of similar transitions, (ii) the possibility of producing a compact and dynamically sized memory representing the universe of all experiences already observed, (iii) the data efficiency achieved by sampling from such compact memory, and that (iv) exist underlying behaviors on sets of similar transitions regarding the estimated Q-value whose dynamics can be learned and used to produce better estimations for the target function.

Specifically, Section 5.3.2 discussed the importance of the similarity threshold, and their results demonstrate how the similarity computations are really important to the results, directly reinforcing Hypothesis 1. Chapter 7 deepened if and how each principal component – the compact transitions memory and the new Q-target function, effectively improves data efficiency. The results demonstrate that each component can contribute individually. Still, they must operate jointly to achieve the best results, and this directly reinforces Hypothesis 2 and corroborates Hypothesis 3, mainly when it states that using a smaller set but with a great representation of the agent's universe of experiences can contribute to reducing the convergence time in reinforcement learning methods with temporal difference learning by providing more informative transition samples for calculating the temporal difference error and, consequently, reducing the training time required for agent learning.

Our propositions introduce a new hyperparameter, which consists of the threshold value for the similarity between the agent's experiences. We demonstrated that variations in the values of this hyperparameter can improve the efficiency of the method but that

in none of the cases was the set of its results worse than the set of previous results or the results obtained by the DQN method and used for comparison. The capacity for generalization, i.e., how much the proposed methods depend on a precise value for the similarity threshold, was a question that motivated our experiments to evaluate the results under different values for this threshold. The results demonstrate that COMPER is completely generalizable, requiring little or no adjustments for this hyperparameter. Moreover, there are no game-specific configurations. We carried out all the experiments under the same experimental protocol and using the same hyperparameter values without algorithm modifications for all the games. In addition, the games are emulated over the Arcade Learning Environment (ALE) (MACHADO et al., 2018), a specific RL platform designed to evaluate RL agents widely used in the literature.

COMPER does not approach efficiency in experience replay at the sampling level, like in prioritization or importance sampling schemas. One can observe that COMPER uniformly samples experiences from a compact memory in a space order many times smaller than the experience buffers used in the literature. Therefore, we presented propositions at the level of transition memory modeling, directly approaching the problem of data efficiency on how to store agents' experiences, which is an original approach in the literature. Therefore, we compared the COMPER with DQN because the general algorithmic structure around Q-learning and Temporal Difference Learning is basically the same. Moreover, we demonstrate how the components (the reduced transitions memory and the target function approximated by recurrence over similar transition sets) can be used individually and, even so, contribute to data efficiency. Specifically, suppose one removes the new components in COMPER. In that case, it will bring the method to the same formulation of DQN, which motivated us to investigate the specific contributions of each COMPER's components individually. In front of this, we believe that DQN was the best choice for making an objective and direct comparison with COMPER, making evident their contributions. Comparing COMPER with prioritization or importance sampling methods does not make sense because they are entirely different approaches. For example, one could use prioritized experience replay on the COMPER sampling process instead of uniform sampling without needing any modification in any of the methods because they are different and approach different algorithmic decisions in the Q-learning-based formulations. At this point, our contributions are in demonstrating that each component of COMPER brings unique improvements to the method and that each could be used separately. However, we also demonstrate that combining all of them combines these improvements and makes the experience replay even more data efficient. The same reasoning applies to the choice of DDPG as the reference method for comparing it with COMCACT on the Gymnasium environments to deal with continuous-valuated action spaces.

One could ask about the possibility of experiences representing outdated policy occurring in the sets of similar transitions in $\mathcal{TM}$ and how it could affect the quality of $\mathcal{RTM}$ and the effectiveness of the Q-target function predictions. It is important to remember that the sets of similar transitions are removed from $\mathcal{TM}$ every time they are used to produce the $\mathcal{RTM}$. Therefore, we do not accumulate outdated transitions. Moreover, transitions representing experiences divergent from the current policy tend to be more scarce while the agent converges to an optimal policy, and the ones even close to the current policy tend to occur more and more throughout the process. From that, we could hypothesize that rare and expensive experiences could benefit from the small size of $\mathcal{RTM}$ by getting more chances to be sampled and that they tend to be closer to the current policy than divergent. However, such an investigation is not trivial and could be carried out in another extensive research work. It is essential to highlight that when we refer to such an experience, the rare and expansive ones, we refer to it as informative, not good or bad experiences, because it is complex to affirm what is a good or a bad experience in the general context.

This work presents many theoretical propositions we could explore more deeply; however, the deadlines and limited computational resources make extending experiments challenging. Nevertheless, we are confident that we have explored the fundamental aspects of our propositions, proved our hypotheses, and demonstrated our method's original, unique contributions.

Future work could bring many contributions. For example, experiments with more frames or iterations for agent training, in addition to the 100,000 and 50,000 used here to conduct the experiments, would allow for evaluation of whether COMPER and COMCACT would reach state-of-the-art in terms of accumulated reward (compared to the literature), measuring their efficacy while still data efficient. Other relevant contributions would come from studying the effects of this work's propositions in other domains and different problems to measure their application or adaptation capacity more generically. In particular, a future investigation on the classic problem of catastrophic forgetting in multi-agent methods seems promising because one of the causes is the sizing of the buffer of experiences. Evaluating the effects of other formulations for similarity computation, such as L1 instead of L2, or performing other operations, such as considering maximum inner product search $argmax_i \langle x, x_i \rangle$, could bring insights into how numerical precision impacts the behavior and the results of both methods. Moreover, given the particular behavior of the Q-target function and its effectiveness in directly predicting the Q-value for the temporal difference error computation, it would be interesting to analyze the effects of variations on the discount factor value represented by the hyperparameter γ. We randomly selected a transition to represent the set of similar transitions, and this strategy may not be the best if we think the transition chosen could not be so similar (on

average) to all the others in the set. Using other strategies, such as selecting a centroid, could improve the representativity of the resulting $\mathcal{RTM}$.

Regarding some questions for future investigations, some intriguing examples are: (i) Could it be possible to turn the similarity hyperparameter from a static into a learned or dynamically adjusted value? (ii) What could be the effects of a prioritization mechanism based on the estimates of the long-term reward associated with the transitions in the compact memory? (iii) What are the effects of using the mechanisms of compact experience replay in a multi-agent scenario? (iv) Could the proposed methods mitigate the problems regarding catastrophic forgetting?

Finally, the status of the contributions to the literature is as follows: (i) Chapters 2 and 3 are part of the survey published by a journal with JCR[1] >=87, 5% (equivalent to the A1 classification in Brazil) and 99% percentile in Scopus, (ii) Chapters 4 and 5 were presented at an A4 conference, (iii) Chapter 6 and 7 are in two papers submitted to high-ranking journals soon.

[1] Journal Citation Reports

REFERENCES

AN, C.; ZHOU, J. Adaptive dynamic programming for data-based optimal state regulation with experience replay. Neurocomputing, v. 554, p. 126616, 2023. ISSN 0925-2312.

ANDRYCHOWICZ, M. et al. Hindsight experience replay. In: GUYON, I. et al. (Ed.). Advances in Neural Information Processing Systems (NIPS 2017). [S.l.: s.n.], 2017. v. 30.

ANZALDO, A.; ANDRADE, Á. G. Experience replay-based power control for sum-rate maximization in multi-cell networks. IEEE Wireless Communications Letters, IEEE, v. 11, n. 11, p. 2350–2354, 2022.

BELLEMARE, M. G.; DABNEY, W.; MUNOS, R. A distributional perspective on reinforcement learning. In: PMLR. International Conference on Machine learning. [S.l.], 2017. p. 449–458.

BELLEMARE, M. G. et al. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, v. 47, p. 253–279, jun 2013.

BROCKMAN, G. et al. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

CASTRO, P. S. et al. Dopamine: A research framework for deep reinforcement learning. arXiv preprint arXiv:1812.06110, 2018.

CHEN, X. et al. 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. [S.l.: s.n.], 2022. p. 1316–1325.

CHEN, X. et al. 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. [S.l.]: Association for Computing Machinery, Inc, 2022. p. 1316 – 1325. ISBN 978-145038732-3.

CROWDER, D. C.; ABREU, J.; KIRSCH, R. F. 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, IEEE, v. 29, p. 1016–1025, 2021.

CUI, J. et al. Multi-input autonomous driving based on deep reinforcement learning with double bias experience replay. IEEE Sensors Journal, v. 23, n. 11, p. 11253–11261, 2023.

DABNEY, W. et al. Implicit quantile networks for distributional reinforcement learning. In: PMLR. Proceedings of the 35th International Conference on Machine Learning. [S.l.], 2018. p. 1096–1105.

DEGRIS, T.; WHITE, M.; SUTTON, R. S. Off-Policy Actor-Critic. In: 29th International Conference on Machine Learning. [S.l.: s.n.], 2012.

DONG, L.; LI, N.; GONG, G. Curiosity-tuned experience replay for wargaming decision modeling without reward-engineering. Simulation Modelling Practice and Theory, v. 129, p. 102842, 2023. ISSN 1569-190X.

DOUZE, M. et al. The Faiss library. 2024.

DU, Y. et al. Lucid dreaming for experience replay: refreshing past states with the current policy. Neural Computing and Applications, Springer, v. 34, n. 3, p. 1687–1712, 2022.

ESPEHOLT, L. et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In: PMLR. 35th International Conference on Machine Learning. [S.l.], 2018. p. 1407–1416.

FÄHRMANN, D. et al. Double deep q-learning with prioritized experience replay for anomaly detection in smart environments. IEEE Access, IEEE, v. 10, p. 60836–60848, 2022.

FEDUS, W. et al. Revisiting fundamentals of experience replay. In: PMLR. International Conference on Machine Learning. [S.l.], 2020. p. 3061–3071.

FORTUNATO, M. et al. Noisy networks for exploration. In: 6th International Conference on Learning Representations. [S.l.: s.n.], 2018.

FUJIMOTO, S.; HOOF, H. van; MEGER, D. Addressing function approximation error in actor-critic methods. In: DY, J.; KRAUSE, A. (Ed.). Proceedings of the 35th International Conference on Machine Learning. [S.l.]: PMLR, 2018. v. 80, p. 1587–1596.

FUJIMOTO, S.; MEGER, D.; PRECUP, D. An equivalence between loss functions and non-uniform sampling in experience replay. In: LAROCHELLE, H. et al. (Ed.). Advances in Neural Information Processing Systems. [S.l.]: Curran Associates, Inc., 2020. v. 33, p. 14219–14230.

GAO, J. et al. Prioritized experience replay method based on experience reward. In: IEEE. 2021 International Conference on Machine Learning and Intelligent Systems Engineering (MLISE). [S.l.], 2021. p. 214–219.

GU, S. et al. Continuous deep q-learning with model-based acceleration. In: PMLR. Proceedings of The 33rd International Conference on Machine Learning. [S.l.], 2016. p. 2829–2838.

HAARNOJA, T. et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: PMLR. Proceedings of the 35th International Conference on Machine Learning. [S.l.], 2018. p. 1861–1870.

HASSELT, H. P. van; HESSEL, M.; ASLANIDES, J. When to use parametric models in reinforcement learning? In: WALLACH, H. et al. (Ed.). Advances in Neural Information Processing Systems. [S.l.]: Curran Associates, Inc., 2019. v. 32.

HASSELT, H. V.; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. [S.l.: s.n.], 2016. v. 30, n. 1.

HASSELT, H. van. Double Q-learning. In: Advances in Neural Information Processing Systems 23 (NIPS 2010). [S.l.: s.n.], 2010. p. 2613–2621.

HAUSKNECHT, M.; STONE, P. Deep recurrent q-learning for partially observable mdps. In: AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15). [S.l.: s.n.], 2015.

HESSEL, M. et al. Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. [S.l.: s.n.], 2018. v. 32, n. 1.

HORGAN, D. et al. Distributed prioritized experience replay. In: International Conference on Learning Representations. [S.l.: s.n.], 2018.

HU, Z. et al. Asynchronous curriculum experience replay: A deep reinforcement learning approach for uav autonomous motion control in unknown dynamic environments. IEEE Transactions on Vehicular Technology, v. 72, n. 11, p. 13985–14001, 2023.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: BACH, F.; BLEI, D. (Ed.). Proceedings of the 32nd International Conference on Machine Learning. [S.l.]: PMLR, 2015. v. 37, p. 448–456.

JIANG, W.-C.; HWANG, K.-S.; LIN, J.-L. An experience replay method based on tree structure for reinforcement learning. IEEE Transactions on Emerging Topics in Computing, IEEE, v. 9, n. 2, p. 972–982, 2021. ISSN 2168-6750.

KAISER, L. et al. Model-based reinforcement learning for atari. arXiv preprint arXiv:1903.00374, 2019.

KAISER Łukasz et al. Model-based reinforcement learning for atari. In: International Conference on Learning Representations. [S.l.: s.n.], 2020.

KANG, C. et al. Deep deterministic policy gradient based on double network prioritized experience replay. IEEE Access, IEEE, v. 9, p. 60296–60308, 2021.

KAPTUROWSKI, S. et al. Recurrent experience replay in distributed reinforcement learning. In: International Conference on Learning Representations. [S.l.: s.n.], 2019.

KIM, M. et al. Motion planning of robot manipulators for a smoother path using a twin delayed deep deterministic policy gradient with hindsight experience replay. Applied Sciences, MDPI AG, v. 10, n. 2, p. 575, 2020. ISSN 2076-3417.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. In: BENGIO, Y.; LECUN, Y. (Ed.). 3rd International Conference on Learning Representations. [S.l.: s.n.], 2015.

KINGMA, D. P.; BA, J. Adam: A Method for Stochastic Optimization. 2017. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

KONG, S.-H.; NAHRENDRA, I. M. A.; PAEK, D.-H. Enhanced off-policy reinforcement learning with focused experience replay. IEEE Access, IEEE, v. 9, p. 93152–93164, 2021. ISSN 2169-3536.

KOROGLU, Y.; SEN, A. Fast witness generation for readable gui test scenarios via generalized experience replay. IEEE Access, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), v. 10, p. 116224–116240, 2022. ISSN 2169-3536.

KUMAR, A.; GUPTA, A.; LEVINE, S. Discor: Corrective feedback in reinforcement learning via distribution correction. In: LAROCHELLE, H. et al. (Ed.). Advances in Neural Information Processing Systems. [S.l.]: Curran Associates, Inc., 2020. v. 33, p. 18560–18572.

LANKA, S.; WU, T. ARCHER: Aggressive Rewards to Counter bias in Hindsight Experience Replay. 2018.

LI, C. et al. Sler: Self-generated long-term experience replay for continual reinforcement learning. Applied Intelligence, Kluwer Academic Publishers, v. 51, n. 1, p. 185–201, jan 2021. ISSN 0924-669X.

LI, M.; HUANG, T.; ZHU, W. Clustering experience replay for the effective exploitation in reinforcement learning. Pattern Recognition, Elsevier, v. 131, nov 2022. ISSN 0031-3203.

LI, X. et al. Progression cognition reinforcement learning with prioritized experience for multi-vehicle pursuit. IEEE Transactions on Intelligent Transportation Systems, p. 1–14, 2024.

LI, Y.; AGHVAMI, A. H.; DONG, D. Path planning for cellular-connected UAV: A DRL solution with quantum-inspired experience replay. IEEE Transactions on Wireless Communications, IEEE, v. 21, n. 10, p. 7897–7912, 2022.

LI, Y.; JI, J. Parallel curriculum experience replay in distributed reinforcement learning. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. [S.l.]: International Foundation for Autonomous Agents and Multiagent Systems, 2021. p. 782–789. ISBN 9781450383073.

LIANG, Y. et al. State of the art control of atari games using shallow reinforcement learning. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS '16). [S.l.: s.n.], 2016. p. 485–493.

LILLICRAP, T. P. et al. Continuous control with deep reinforcement learning. In: International Conference on Representation Learning. [S.l.: s.n.], 2016.

LIN, L.-J. Self-improving reactive agents based on reinforcement learning, planning, and teaching. Machine Learning, v. 8, n. 3-4, p. 293–321, May 1992. ISSN 1573-0565.

LIU, R.; ZOU, J. The effects of memory replay in reinforcement learning. In: 2018 56th Annual Allerton Conference on Communication, Control, and Computing. [S.l.: s.n.], 2018. p. 478–485.

LIU, X. et al. Value distribution ddpg with dual-prioritized experience replay for coordinated control of coal-fired power generation systems. IEEE Transactions on Industrial Informatics, v. 20, n. 6, p. 8181–8194, 2024.

LIU, X. et al. Prioritized experience replay based on multi-armed bandit. Expert Systems with Applications, Elsevier, v. 189, p. 116023, 2022.

LUO, Y. et al. Relay hindsight experience replay: Self-guided continual reinforcement learning for sequential object manipulation tasks with sparse rewards. Neurocomputing, v. 557, p. 126620, 2023. ISSN 0925-2312.

MA, J. et al. Fresher experience plays a more important role in prioritized experience replay. Applied Sciences, MDPI AG, v. 12, n. 23, p. 12489, 2022. ISSN 2076-3417.

MACHADO, M. C. et al. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. Journal of Artificial Intelligence Research, v. 61, p. 523–562, 2018.

MANELA, B.; BIESS, A. Bias-reduced hindsight experience replay with virtual goal prioritization. Neurocomputing, Elsevier, v. 451, p. 305–315, 2021.

MANELA, B.; BIESS, A. Curriculum learning with hindsight experience replay for sequential object manipulation tasks. Neural Networks, Elsevier, v. 145, p. 260–270, 2022.

MNIH, V. et al. Asynchronous methods for deep reinforcement learning. In: BALCAN, M. F.; WEINBERGER, K. Q. (Ed.). Proceedings of The 33rd International Conference on Machine Learning. [S.l.]: PMLR, 2016. (Proceedings of Machine Learning Research, v. 48), p. 1928–1937.

MNIH, V. et al. Playing atari with deep reinforcement learning. In: Deep Learning Workshop NIPS 2013. [S.l.: s.n.], 2013.

MNIH, V. et al. Human-level control through deep reinforcement learning. Nature, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

MORENO-VERA, F. Performing deep recurrent double q-learning for atari games. In: IEEE. 2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI). [S.l.], 2019. p. 1–4.

NEVES, D. E.; ISHITANI, L.; PATROCÍNIO JR, Z. K. G. When less may be more: Exploring similarity to improve experience replay. In: SPRINGER. Intelligent Systems: 11th Brazilian Conference, BRACIS 2022, Part II. [S.l.], 2022. p. 96–110.

NICHOLAUS, I. T.; KANG, D.-K. Robust experience replay sampling for multi-agent reinforcement learning. Pattern Recognition Letters, Elsevier, v. 155, p. 135–142, 2022.

NOVATI, G.; KOUMOUTSAKOS, P. Remember and forget for experience replay. In: CHAUDHURI, K.; SALAKHUTDINOV, R. (Ed.). Proceedings of the 36th International Conference on Machine Learning. [S.l.]: PMLR, 2019. v. 97, p. 4851–4860.

OSEI, R. S.; LOPEZ, D. Experience replay optimisation via ATSC and TSC for performance stability in Deep RL. Applied Sciences, MDPI AG, v. 13, n. 4, p. 2034, 2023. ISSN 2076-3417.

PANDA, D. K. et al. Prioritized experience replay based deep distributional reinforcement learning for battery operation in microgrids. Journal of Cleaner Production, v. 434, p. 139947, 2024. ISSN 0959-6526.

PRIANTO, E. et al. Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay. Sensors, MDPI, v. 20, n. 20, p. 5911, 2020.

REMMAN, S. B.; LEKKAS, A. M. Robotic lever manipulation using hindsight experience replay and shapley additive explanations. In: IEEE. 2021 European Control Conference (ECC). [S.l.], 2021. p. 586–593.

ROLNICK, D. et al. Experience replay for continual learning. Advances in Neural Information Processing Systems, v. 32, 2019.

ROSENBAUER, L. et al. XCSF with experience replay for automatic test case prioritization. In: IEEE. 2020 IEEE Symposium Series on Computational Intelligence (SSCI). [S.l.], 2020. p. 1307–1314.

SCHAUL, T. et al. Prioritized experience replay. In: Proceedings of the International Conference on Representation Learning. [S.l.: s.n.], 2016.

SCHMITT, S.; HESSEL, M.; SIMONYAN, K. Off-policy actor-critic with shared experience replay. In: III, H. D.; SINGH, A. (Ed.). Proceedings of the 37th International Conference on Machine Learning. [S.l.]: PMLR, 2020. (Proceedings of Machine Learning Research, v. 119), p. 8545–8554.

SHI, H. et al. Task offloading and trajectory scheduling for uav-enabled mec networks: An madrl algorithm with prioritized experience replay. Ad Hoc Networks, v. 154, p. 103371, 2024. ISSN 1570-8705.

SILVER, D. et al. Deterministic policy gradient algorithms. In: XING, E. P.; JEBARA, T. (Ed.). Proceedings of the 31st International Conference on Machine Learning. Bejing, China: PMLR, 2014. v. 32, p. 387–395.

SINHA, S. et al. Experience replay with likelihood-free importance weights. In: FIROOZI, R. et al. (Ed.). Proceedings of The 4th Annual Learning for Dynamics and Control Conference. [S.l.]: PMLR, 2022. v. 168, p. 110–123.

SOVRANO, F.; RAYMOND, A.; PROROK, A. Explanation-aware experience replay in rule-dense environments. IEEE Robotics and Automation Letters, v. 7, n. 2, p. 898–905, 2022.

SUN, P.; ZHOU, W.; LI, H. Attentive experience replay. Proceedings of the AAAI Conference on Artificial Intelligence, v. 34, n. 04, p. 5900–5907, Apr. 2020.

SUTTON, R. S. Reinforcement learning architectures. In: Proceedings ISKIT'92 International Symposium on Neural Information Processing. [S.l.: s.n.], 1992.

SUTTON, R. S.; BARTO, A. G. Reinforcement Learning: An Introduction. Second. Cambridge: The MIT Press, 2018.

SZEPESVáRI, C. Algorithms for Reinforcement Learning. [S.l.]: Morgan & Claypool Publishers, 2010.

TAO, X.; HAFID, A. S. Deepsensing: A novel mobile crowdsensing framework with double deep q-network and prioritized experience replay. IEEE Internet of Things Journal, IEEE, v. 7, n. 12, p. 11547–11558, 2020. ISSN 2327-4662.

TASSA, Y. et al. Deepmind control suite. arXiv preprint arXiv:1801.00690, abs/1801.00690, 2018.

TODOROV, E.; EREZ, T.; TASSA, Y. Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. [S.l.: s.n.], 2012. p. 5026–5033.

TOWERS, M. et al. Gymnasium. 2023. An API standard for single-agent reinforcement learning environments, with popular reference environments and related utilities (formerly Gym).

UHLENBECK, G. E.; ORNSTEIN, L. S. On the theory of the brownian motion. Phys. Rev., American Physical Society, v. 36, p. 823–841, Sep 1930.

VECCHIETTI, L. F.; SEO, M.; HAR, D. Sampling rate decay in hindsight experience replay for robot control. IEEE Transactions on Cybernetics, IEEE, v. 52, n. 3, p. 1515–1526, 2022. ISSN 2168-2267.

WANG, B.; ZHAO, D.; CHENG, J. Adaptive cruise control via adaptive dynamic programming with experience replay. Soft Computing, Springer Verlag, v. 23, n. 12, p. 4131 – 4144, 2019. ISSN 14327643.

WANG, C.; ROSS, K. W. Boosting soft actor-critic: Emphasizing recent experience without forgetting the past. arXiv preprint arXiv:1906.04009, abs/1906.04009, 2019.

WANG, Y. et al. Deep reinforcement learning based on balanced stratified prioritized experience replay for customer credit scoring in peer-to-peer lending. Artificial Intelligence Review, v. 57, n. 4, p. 93, Mar 2024. ISSN 1573-7462.

WANG, Z. et al. Dueling network architectures for deep reinforcement learning. In: BALCAN, M. F.; WEINBERGER, K. Q. (Ed.). Proceedings of The 33rd International Conference on Machine Learning. [S.l.]: PMLR, 2016. v. 48, p. 1995–2003.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. Machine Learning, v. 8, n. 3, p. 279–292, May 1992. ISSN 1573-0565.

Wei, Q. et al. Deep reinforcement learning with quantum-inspired experience replay. IEEE Transactions on Cybernetics, p. 1–13, 2021.

WEI, Q. et al. Deep reinforcement learning with quantum-inspired experience replay. IEEE Transactions on Cybernetics, IEEE, v. 52, n. 9, p. 9326–9338, 2022. ISSN 2168-2267.

WERBOS, P. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, v. 78, n. 10, p. 1550–1560, 1990.

WU, D.-F. et al. State aware-based prioritized experience replay for handover decision in 5g ultradense networks. Wireless Communications and Mobile Computing, John Wiley and Sons Ltd., v. 2022, 2022. ISSN 1530-8669.

XU, C.; MA, J.; TAO, H. Batch process control based on reinforcement learning with segmented prioritized experience replay. Measurement Science and Technology, IOP Publishing, v. 35, n. 5, p. 056202, feb 2024.

YANG, J.; PENG, G. DDPG with meta-learning-based experience replay separation for robot trajectory planning. In: 2021 7th International Conference on Control, Automation and Robotics (ICCAR). [S.l.: s.n.], 2021. p. 46–51.

YANG, R.; WANG, D.; QIAO, J. Policy gradient adaptive critic design with dynamic prioritized experience replay for wastewater treatment process control. IEEE Transactions on Industrial Informatics, IEEE, v. 18, n. 5, p. 3150–3158, 2022. ISSN 1551-3203.

YANG, X.; HE, H. Adaptive critic learning and experience replay for decentralized event-triggered control of nonlinear interconnected systems. IEEE Transactions on Systems, Man, and Cybernetics: Systems, v. 50, n. 11, p. 4043–4055, 2020.

YARATS, D.; KOSTRIKOV, I.; FERGUS, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In: International Conference on Learning Representations. [S.l.: s.n.], 2021.

YU, L. et al. Hybrid attention-oriented experience replay for deep reinforcement learning and its application to a multi-robot cooperative hunting problem. Neurocomputing, Elsevier Science Publishers B. V., NLD, v. 523, n. C, p. 44–57, feb 2023. ISSN 0925-2312.

ZHA, D. et al. Experience replay optimization. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. [S.l.]: International Joint Conferences on Artificial Intelligence Organization, 2019. p. 4243–4249.

ZHANG, H. et al. Self-adaptive priority correction for prioritized experience replay. Applied Sciences, MDPI AG, v. 10, n. 19, p. 6925, 2020. ISSN 2076-3417.

ZHANG, S.; SUTTON, R. S. A deeper look at experience replay. In: 31st Conference on Neural Information Processing Systems (NIPS 2017). [S.l.: s.n.], 2017.

ZHANG, Y.; QIU, H. DDQN with prioritized experience replay-based optimized geographical routing protocol of considering link stability and energy prediction for uanet. Sensors, MDPI AG, v. 22, n. 13, p. 5020, 2022. ISSN 1424-8220.

ZHANG, Y. et al. A cooperative ev charging scheduling strategy based on double deep q-network and prioritized experience replay. Engineering Applications of Artificial Intelligence, v. 118, p. 105642, 2023. ISSN 0952-1976.

ZHAO, X.; DU, J.; WANG, Z. Hcs-r-her: Hierarchical reinforcement learning based on cross subtasks rainbow hindsight experience replay. Journal of Computational Science, v. 72, p. 102113, 2023. ISSN 1877-7503.

ZHOU, C. et al. Attention-based advantage actor-critic algorithm with prioritized experience replay for complex 2-d robotic motion planning. Journal of Intelligent Manufacturing, Springer-Verlag, Berlin, Heidelberg, v. 34, n. 1, p. 151–180, aug 2022. ISSN 0956-5515.

APPENDIX A – COMPER – EXPERIMENTAL SETUP

Experimental methodology and evaluation protocol were discussed in Section 5.3. However, there are specific sets of hyperparameters for different purposes. Some of them are independent of the evaluated methods while others address aspects related to each of them, as shown in the following.

A.1 Algorithm for Store, Index, and Compute Similarities for Transitions.

To implement the processes of storing, indexing, and retrieving similar transitions and compute the similarities, we used a library for efficient similarity search and clustering of dense vectors developed primarily at Meta's Fundamental AI Research group and called Faiss (DOUZE et al., 2024). It contains diverse algorithms to search in sets of vectors of any size, up to ones that possibly do not fit in RAM. Vectors similar to a query vector have the lowest L2 distance or the highest dot product with the query vector. It also supports cosine similarity since this is a dot product on normalized vectors. Specifically, we use the algorithm IndexFlatL2 from Faiss. Given a set of vectors $x_1, ..., x_n$ with dimension d, Faiss builds an indexing data structure in RAM. Given a new vector x in dimension d, it efficiently compute $i = argmin_i \parallel x - x + 1 \parallel$, where $\parallel \cdot \parallel$ is the Euclidian Distance (L2). Computing the $argmin$ is the search operation on the index. It can (i) return from the first nearest neighbor vector until the $k - th$ nearest neighbor, (ii) search for a batch of input vectors at a time rather than one (For many index types, this is faster than searching one vector after another), (iii) trade precision for speed, i.e., give an incorrect result 10% of the time with a method that's 10x faster or uses 10x less memory, and many other capacities and indexing and search strategies. From the beginning to the final version of this research work, Faiss remained accessible in an open-access repository as open-source, free-of-charge software under the MIT License.

A.2 Specific parameters of COMPER

As discussed in Section 4.1, COMPER uses different models and architectures of deep neural networks to approximate their value functions, two transition memories, and a reduction process from one to another, requiring specific sets of parameters, which are presented in Tables 32, 33, and 34.

A.3 Specific parameters of DQN

In Table 35, we present the parameter used on DQN. Those related to the memory of experiences and the target network update frequency (**in bold**) were adjusted due to

Table 32 – Parameters used in CNN training by COMPER.

Hyperparameter	Value	Description
Step-size (α)	0.00025	Step-size used by RMSProp.
Gradient momentum	0.95	Gradient momentum used by RMSProp.
Squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
Min squared gradient	0.01	Constant added to the denominator of the RMSProp update.

Table 33 – Parameters used in LSTM training.

Hyperparameter	Value	Description
Step-size (α)	0.00025	Step-size used by RMSProp.
Gradient momentum	0	Gradient momentum used by RMSProp.
Decay	0.9	Discounting factor for the history and coming gradient used by RMSProp.
Epsilon	10^{-10}	Constant added to the denominator of the RMSProp update.

the smaller number of frames used in the experiments. All others have the same values defined by Machado et al. (2018).

A.4 Deep Neural Network Architectures used in COMPER

The convolutional neural network (CNN) used in COMPER to approximate the Q-Value function has the same general architecture as proposed by Mnih et al. (2015) and used in Machado et al. (2018). The difference is in the input layer because when COMPER is using a single frame it represents the frames as a matrix of $84 \times 84 \times 1$ and when using stacked frames its shape is $84 \times 84 \times 4$. This representation is produced by a preprocessing step that reduces the original dimensionality of the game frames and extracts the luminance channel. Another difference is in the output layer. As recommended by Machado et al. (2018) and discussed in Section 5.3, the COMPER agent has access to the complete set of actions on ALE, independently from the game, so the output layer is a fully-connected layer with a single output for each of the 18 valid actions. The hidden layers are three convolutional and one fully-connected. The first convolutional layer applies 32 filters of 8×8 with stride 4 on the input frame image representation, followed by a ReLU nonlinearity. The second layer convolves 64 filters of 4×4 with stride 2 and also applies a ReLU nonlinearity. The third hidden layer convolves 64 filters of 3×3 with stride 1, again followed by a ReLU nonlinearity. The final hidden layer consists of 512 units. The RMSProp optimizer uses the parameters shown in Table 32 and this architecture is detailed in Table 36.

Table 34 – General parameters used by COMPER.

Hyperparameter	Value	Description
Transition memory size	100,000	The transition memory size is defined to stores up to 100,000 sets of last similar transitions. However, it is reduced every when these sets are sampled.
Replay start size	100	Number of frames over which a random policy is executed to first populate the transition memory.
History length (single frame)	1	Number of most recent frames the agent observed that are given as input to the CNN when using single frame.
History length (stacked frames)	4	Number of most recent frames the agent observed that are given as input to the CNN when using stacked frames.
Train Frequency (TF)	4	Number of iterations between the CNN model updates.
Transitions batch size (K)	32	The number of transitions sampled from the reduced transitions memory to update the CNN.
Similar transitions batch size	1,000	The number of sets of similar transitions taken from the transition memories to produce the reduced transition memory.
Similarity Threshold δ	0.0	Maximum distance to consider two transitions as similar
Update Frequency (UTF)	100	The frequency with which the target network (LSTM) is trained and the reduced transition memory is updated by the QLSTM algorithm.
Color averaging	True	The observation received is the average between the previous and the current frame.
Number of different colors	8	NTSC is the color palette in which each screen is encoded but only the luminance channel is used

The LSTM network that predicts values for the target function has the input layer with shapes of 1×14114 or 1×56450, which represents a transition τ for single frame and stacked frames, respectively. The batch size is 16. The output is a fully-connected linear layer with only a single output for the predicted Q-Value to the input transition. The hidden layers are three LSTM and one fully connected. The first LSTM layer contains 64 internal units, applies a hyperbolic tangent activation function and sequence return. The second layer contains 32 internal units, and also applies a hyperbolic tangent activation function and sequence return. The third LSTM layer also contains 32 internal units and applies a hyperbolic tangent activation function, but has no sequence return. The last hidden layer contains 18 fully connected units and applies a ReLU nonlinearity. The RMSProp optimizer uses the parameters shown in Table 33 and this architecture is detailed in Table 37.

Table 35 – Parameters used by DQN

Hyperparameter	**Value**	**Description**
Step-size (α)	0.00025	Step-size used by RMSProp.
Gradient momentum	0.95	Gradient momentum used by RMSProp.
Squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
Min squared gradient	0.01	Constant added to the denominator of the RMSProp update.
Replay memory size	100,000	The samples used in the algorithm's updates are drawn from the last 100 thousand recent frames.
Replay start size	1,000	Number of frames over which a random policy is executed to first populate the replay memory.
History length (stacked frames)	4	Number of most recent frames the agent observed that are given as input to the CNN.
Update frequency	4	Number of actions the agent selects between successive updates.
Update frequency of target network	1,000	Number of actions the agent selects between successive updates of the target CNN parameters.
Frame pooling	True	The observation received consists of the maximum pixel value between the previous and the current frame.
Number of different colors	8	NTSC is the color palette in which each screen is encoded but only the luminance channel is used.

Table 36 – CNN architecture used in COMPER with single frame. Note that * indicates the values for single frame and ** the values for stacked frames.

Layer (type)	**Definition**	**Output Shape**	**Parameters**
Conv1 (Conv2D)	Filters = (32, (8, 8)) Strides= (4, 4) Input = (84, 84, 1)* or Input = (84, 84, 4)**	(None, 20, 20, 32)	2,080* or 8,192**
Conv2 (Conv2D)	Filters = (64, (4, 4)) Strides= (2, 2)	(None, 9, 9, 64)	32,832
Conv3 (Conv2D)	Filters = (64, (3, 3)) Strides= (1, 1)	(None, 7, 7, 64)	36,928
Flatten	Flatten	(None, 3136)	0
Dense1 (Dense)	512 Units	(None, 512)	1,606,144
Dense2 (Dense)	18 Units	(None, 18)	9,234

Table 37 – LSTM architecture used in COMPER. Note that * indicates the values for single frame and ** the values for stacked frames.

Layer (type)	**Definition**	**Output Shape**	**Parameters**
Lstm1 (LSTM)	Units = 64 Return Sequences = True Input = (1, 14114)* or Input = (1, 56450)**	(None, 1, 64)	3,629,824* or 14,467,840**
Lstm2 (LSTM)	Units = 32 Return Sequences = True	(None, 1, 32)	12,416
Lstm3 (LSTM)	Units = 32	(None, 32)	8,320
Dense1 (Dense)	Units = 18	(None, 18)	594
Dense2 (Dense)	Units = 1	(None, 1)	19

APPENDIX B – COMCACT – EXPERIMENTAL SETUP

We use a deep neural network to approximate the actor function. The first layer is an input layer with a shape equivalent to the number of states in the training batch. This is followed by a batch normalization layer with the same shape as the input layer. The next layer is a dense hidden layer that comprises 400 units and utilizes the *Rectified Linear Unit* (ReLU) activation function. This is followed by another batch normalization layer and another dense hidden layer with 300 units and ReLU activation. Another batch normalization layer also follows this. Lastly, we have an output-dense layer with the number of units equal to the number of actions. This layer uses the hyperbolic tangent (*tanh*) activation function and a uniform-initialized kernel with values in the interval [-0.003,0.003]. We use another deep neural network to approximate the critic function. The input and hidden layers are equal to the first two in the actor and intercalated with batch normalization layers. After the second hidden layer, we insert the action layers, which are composed of an input layer with a shape equivalent to the number of actions, which we call action input. The last layer contains only 1 unit and applies a uniform-initialized kernel with values in the interval [-0.0003,0.0003]

Regarding the architecture of the *Long Short-Term Memory* (LSTM) that learns to predict the Q-target value from the sets of similar transitions, its first layer is an input layer with the shape defined as a tuple (1, transitions size). The second is an LSTM layer comprising 64 units, using the *tanh* activation function and returning the entire output sequence. The next layer is also an LSTM with 64 units using *tanh* but returns only the last output in the output sequence. The next is a dense layer with 32 units and utilizes the ReLU activation function. Lastly, we have an output-dense layer with only 1 unit, using the *tanh* activation function and a uniform-initialized kernel with values in the interval [-0.0003, 0.0003].

We used Adam (KINGMA; BA, 2017) as the optimizer of the actor and critic networks with a learning rate equal to 10^{-3} and 20^{-3}, respectively. For optimization in the LSTM, we used the root mean square propagation (RMSprop) algorithm with a learning rate equal to 10^{-3}. We defined the discount factor $\gamma = 0.99$ for updating the critic from its corresponding critic target and updating it from the Q-target function. For the soft target updates, we used $\tau = 0.005$. We trained the actor and critics using batches of 64 transitions uniformly sampled from $\mathcal{RTM}$. For the Q-target function, we used 50,000 transitions removed from $\mathcal{TM}$.

APPENDIX C – 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 38 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 38 – 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).

C.1 ACM-DL

Table 39 shows the results of applying each query on the ACM-DL Full-Text Collection and the ACM-DL Guide to Computing Literature. Figures 45 and 46 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 39 – 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

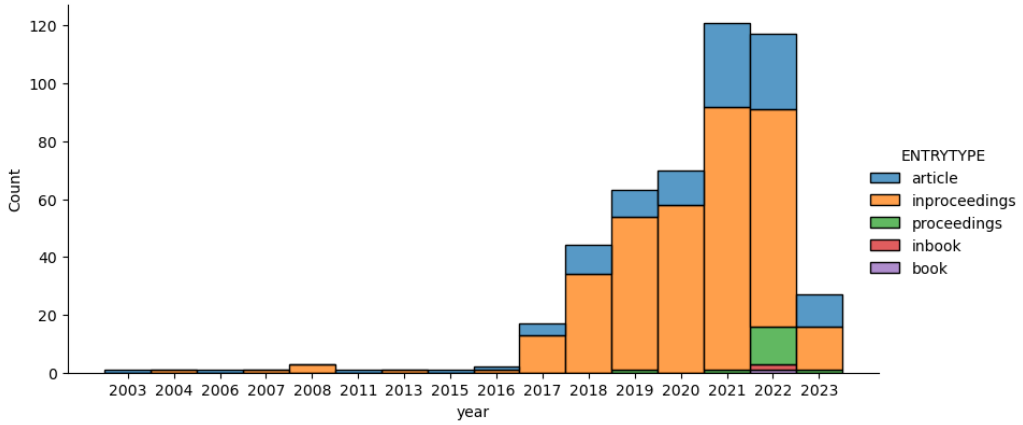


Figure 45 – Q1- Types of publications over the years on the ACM-DL Full-Text Collection.

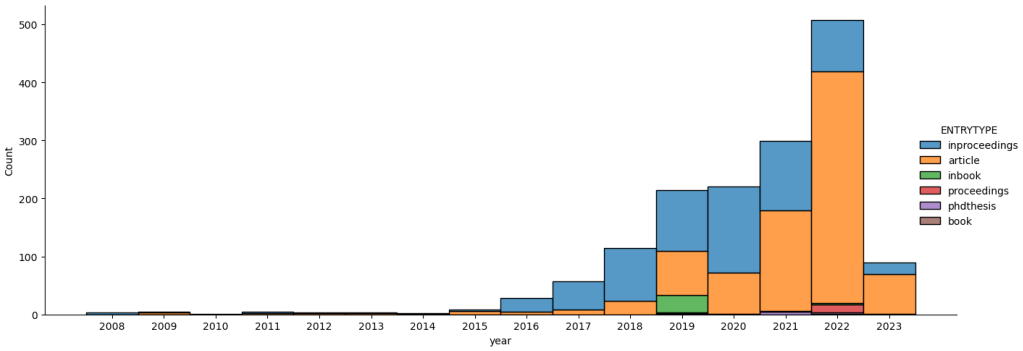


Figure 46 – Q1- Types of publications over the years on the ACM-DL Guide to Computing Literature.

C.2 CAPES

Table 40 shows the results of applying each query on CAPES. Figure 47 illustrates the distribution of results from Q1 (the least restrictive query) over the years on CAPES.

C.3 IEEE Xplore

Table 41 shows the results of applying each query on IEEE Xplore. Figure 48 illustrates the distribution of results from Q1 (the least restrictive query) over the years on IEEE Xplore.

C.4 Science Direct

Table 42 shows the results of applying each query on Science Direct. Figure 49 illustrates the distribution of results from Q1 (the least restrictive query) over the years

Table 40 – 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	*

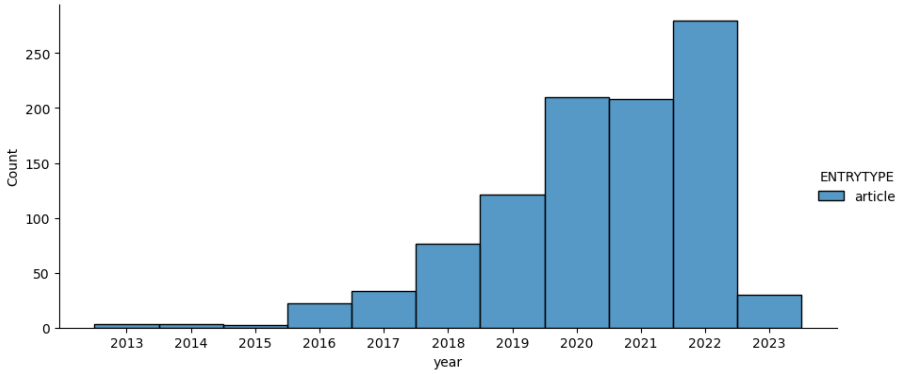


Figure 47 – Q1- Types of publications over the years on CAPES.

on Science Direct.

C.5 Scopus

Table 43 shows the results of applying each query on Scopus. Figure 50 illustrates the distribution of results from Q1 (the least restrictive query) over the years on Scopus.

Table 41 – 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

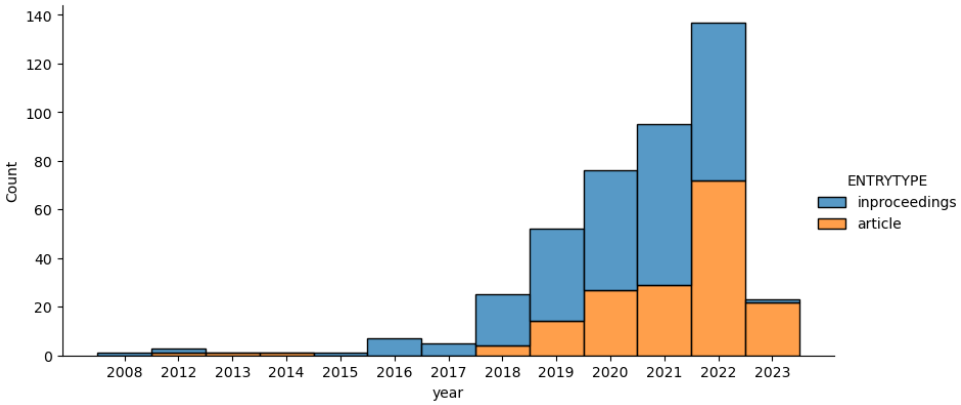


Figure 48 – Q1- Types of publications over the years on IEEE Xplore.

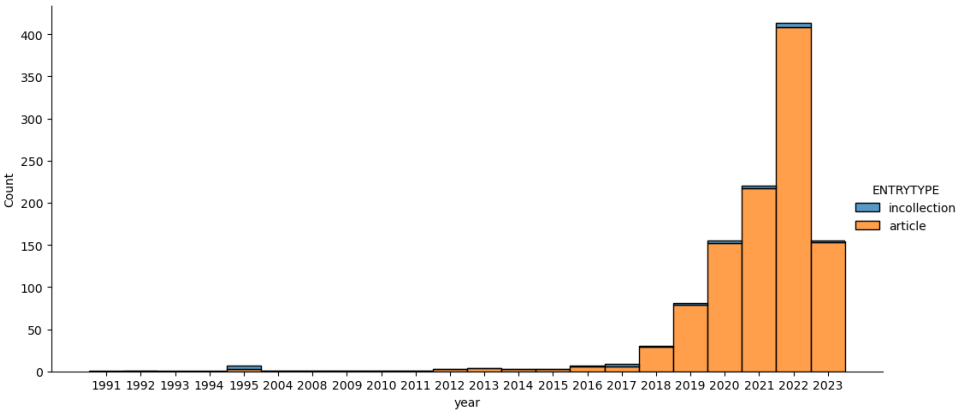


Figure 49 – Q1- Types of publications over the years on Direct Science.

Table 42 – 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 43 – 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

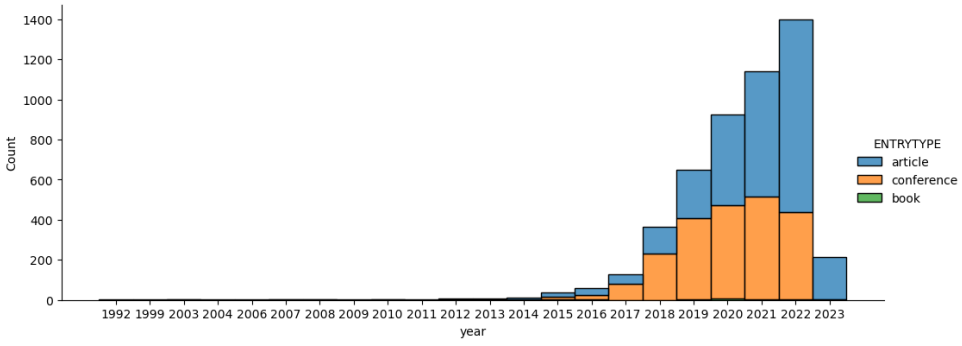


Figure 50 – Q1- Types of publications over the years on Scopus.