Bisimulation Prioritized Experience Replay: Enhancing Online Reinforcement Learning through Bisimulation Behavioral-Based Priorities

by

Oscar Guarnizo

Student ID: 2575296

Supervisor: Mirco Giacobbe, Ph.D.

Co-Supervisor: Leandro Stella, Ph.D.

School of Computer Science

College of Engineering and Physical Sciences

University of Birmingham

2023-24

Abstract

Prioritized Experience Replay has been an effective traditional solution for value-based reinforcement learning algorithms to efficiently address non-stationary and correlated data issues. However, standard prioritization often overlooks the nuanced, task-specific behaviors of states, leading to a "task-agnostic" sampling problem. This work introduces a novel non-uniform sampling approach, named Bisimulation Prioritized Experience Replay (BPER), by incorporating a surrogate on-policy bisimulation metric into the experience replay prioritization process. This metric allows us to measure behavioral similarities and diversify the training data, aiming to enhance learning by focusing on behaviorally relevant transitions. Specifically, our method utilizes a Matching under Independent Couplings (MICo) metric, a more general surrogate metric learned through state abstractions. The proposed method balances conventional TD-error-based and bisimulation-based prioritization by reweighting priorities with an introduced hyperparameter, and two possible strategies to assigning priorities. The method demonstrates superior performance in a 31-state Grid World and shows promising results in classical pixel-based environments. The 31-state Grid World empirically validates the proof of concept by efficiently achieving to 1) emphasize behavioral relevant transition, thereby avoiding task-agnostic sampling, 2) alleviate the outdated priorities by having a better tendency to constant fixed priorities, and 3) mitigate the insufficient sample space coverage, increasing the data diversity.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my principal advisor, Mirco Giacobbe, Ph.D., and my co-advisor, Leonardo Stella, Ph.D., for their continuous support of my research, as well as their patience, responsibility, motivation, and enthusiasm.

Equally important, I would like to thank Dr. Pablo Samuel Castro, the author of the MICo paper, for his invaluable advice and guidance, which helped me to better understand the topic. I truly appreciate your prompt responses to my questions and your enthusiasm in explaining your work to me. I hope to collaborate with you on future projects.

I would also like to thank my friends and colleagues who offered ideas and comments about my project. In particular, I want to highlight the observations of Fernando Zhapa, Anthony Ramos, Jose Seraquive, and Joseph Gonzalez, which were instrumental in shaping this project.

A special thank you goes to my dear friend Yu Wen (黃郁雯), who accompanied me every day in the library throughout the entire journey of this thesis. Thank you for feeding me and cheering me up with your weird situations (like losing a laptop). Without a doubt, you made this thesis a more enjoyable and pleasant experience, and I am deeply grateful for your company.

Last but not least, I would like to express my heartfelt thanks to my family, who always motivated and supported me throughout this master's journey. I am particularly grateful to my parents, Elizabeth and Vicente, for their constant calls and for taking the time to talk to me during both challenging and happy moments. I also thank my sisters, Evelyn and Ericka, for their frequent video calls filled with stories that lifted my spirits and kept me moving forward. Without their unwavering support, I would not have successfully completed this stage of my life.

Abbreviations

RL	Reinforcement Learning
BPER	Bisimulation Prioritized Experience Replay
BPERcn	Bisimulation Prioritized Experience Replay with strategy current-vs-next
BPERaa	Bisimulation Prioritized Experience Replay with strategy all-vs-all
PER	Prioritized Experience Replay
TD-error	Temporal Difference Error
MICo	Matching under Independent Couplings
ELP	Expected Learning Progress
DQN	Deep Q-Network
SAC	Soft Actor-Critic

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Incorporating deep learning techniques into Reinforcement Learning (RL) frameworks has been challenging due disparity in data assumptions between deep learning and RL algorithms [40]. Traditional deep learning relies on the independence of data samples for effective neural network training, whereas RL is characterized by a temporal sequential process that results in **highly correlated states**. Moreover, the data distribution in RL is **non-stationary**; it evolves as the algorithm acquires new behaviors. This dynamism leads to instability when deep learning techniques are applied to RL algorithms.

Experience Replay (ER) has been implemented in online RL algorithms, such as DQN [40], DDPG [33], SAC [21] to address both data correlation and non-stationary distributions issues. Essentially, an ER functions as a database where experience tuples[2] are stored, allowing training to occur on minibatches sampled from this experience buffer. This simple mechanism facilitates breaking temporal data correlations, leading to approximate independent and identically distributed (iid) data distributions.

While ER benefits online RL, significant iterations may still be required for convergence. Schaul et al. [44] note that a DQN algorithm revisits the same experience tuple an average of eight times, not all of which lead to significant improvements. In consequence, they proposed a Prioritized Experience Replay (PER), assigning probabilities to each experience based on the Temporal Difference (TD) error [47]. The TD-error priority works as an indicator of the **Expected Learning Progress (ELP)** [44][1]; encouraging more frequently replay experiences which lead to higher improvements. However, this prioritization can face several issues, such as task-agnostic sampling, outdated priorities, and insufficient state space coverage.

Prioritizing purely on TD-error can overlook the task-specific behaviors of states, lead-

[2] An experience tuple consists of (state s_t, action a_t, reward R_t, next state s_{t+1})

[1] According to Schaul et al. [44], the **Expected Learning Progress** (ELP) is the idealised criterion to assign a priority value with the amount the RL agent can learn from a transition in its current state.

ing to what we term as **task-agnostic sampling** problem, similar to representation learning findings in [53]. From this perspective, PER fails to recognize that certain states in a Markov Decision Process (MDP), despite being structural dissimilar, can exhibit similar long-term behaviors (w.r.t. the RL downstream task), resulting in similar expected returns in the long run. In other words, it fails to recognize that some states can be behavioral similar. The **outdated priorities** limitation, in the other hand, arises from practical implementation issues. According to Theorem 1 in Pan Y. et al. [43], priorities should be updated over all experiences in the replay buffer using updated training parameters at each time step in order to obtain a faster convergence rate. However, this approach is impractical due to the large replay buffer capacity, which imposes a high computational cost when updating all priorities during each learning iteration. PER [44] methods update only the priorities of experiences from a sampled mini-batch, leaving others unchanged, resulting in inaccurate sampling distributions. Indeed, PER is quite restrictive because priorities are calculated only from experience tuples already visited and stored in the experience buffer, representing just a small subset of the entire state space. This exploration issue, called **insufficient sample space coverage**, has been highlighted by other sources [15, 43]. Lack of sample space coverage can lead to data imbalance issues, when during learning the experiences are unevenly distributed in the state space [12], producing high correlated transitions due to high levels of correlation between the data-generating distribution (the experience replay) and the current evaluated policy [15], tending to an on-policy updating behavior.

Bisimulation metrics [16, 17, 18, 9] provides a means of quantifying the behavioral similarity between states by considering the immediate rewards along with how states transition under a given MDP. Leveraging this behavioral concept aims to prioritize more informative tuples in the experience replay by identifying state pairs with significant behavioral differences, as they often correspond to more *'surprising'* transitions, leading to more effective improvements. Using bisimulation metrics also alleviates outdated priorities by providing more realistic long-term behavior definitions. Although bisimulation-based priorities will be still updated in the sampled mini-batch, the behavioral-based bisimulation metrics theoretically rely on a dynamic programming operator (commonly used in RL, such as the value iteration algorithm [47]), which is proven to converge to a fixed point where the distance metric remains constant [9, 10]. Unlike TD-error, this provides a more realistic and stable definition of priority in the long term, alleviating the outdated priorities. Additionally, prioritizing behavioral dissimilar states encourages data diversity and exploration, tackling the insufficient sample space coverage problem. Specifically, learning state abstractions with a bisimulation metric organizes latent codes in a latent space based on behavior, inducing large state space coverage of behavioral dissimilar states.

1.1 Contributions

In this work, we propose incorporate a bisimulation metric as priority in a prioritized experience replay by learning state abstractions. We explored the on-policy bisimulation metric proposed by Castro [9], which specifically emphasizes the dynamics of the Markov

Chain induced by the current policy. Specifically, we used a surrogate on-policy bisimulation metric called Matching under Independent Couplings (MICO) [10] metric, which works more efficiently under both discrete and stochastic environments and is obtained as part of the learning of state abstractions. According to [53] and [10], the calculation of a bisimulation metric (or surrogate) in an online manner as part of the learning of state abstractions is a more effective way to obtain this metric. Our Bisimulation Prioritized Experience Replay (BPER) aims to 1) emphasize behaviorally relevant transitions, thereby avoiding task-agnostic experience sampling, 2) alleviate the outdated priorities by having a better tendency to constant fixed priorities, and 3) mitigate the insufficient sample space coverage by encouraging the sampling of behavioral dissimilar states, increasing the data diversity.

1.2 Document Structure

This thesis is organized into several chapters that should generally be read sequentially. We recognize that there are two essential areas of background knowledge required to fully understand this work: the foundations of RL and Bisimulation. We strongly encourage readers to first read the background (Section 2) to gain a comprehensive understanding of the methodology. However, certain sections may be skipped depending on the reader's familiarity and specific interests:

- Readers already familiar with reinforcement learning may choose to skip Section 2.1.
- For a quick understanding of the methodology, we suggest reviewing the Introduction (Section 1) and the Bisimulation Prioritized Experience Replay (Section 4.2), while skipping the Motivating Example (Section 4.1), which relies heavily on the background material.
- For those interested in potential extensions of this work, we recommend focusing on the Introduction (Section 1) and then proceeding directly to the Conclusion and Future Work (Section 7) to grasp the current state of the research.

1.3 Legal, Social, Ethical and Professional Issues

- **Legal.** The proposed method utilizes data that is openly accessible from reinforcement learning environments available in Gymnasium. For the development of the custom Grid World, we used and modified the open-source code provided by the Farama-Foundation/gym-examples. To replicate the MICo learning approach, we retrieved the openly available code from Google Research. All datasets in this repository are released under the CC BY 4.0 International license, which can be reviewed here, while the source files are licensed under the Apache 2.0 license. For the reproduction of the DQN algorithm, we employed the state-of-the-art implementation provided by the TorchRL repository, licensed under the MIT License. All necessary acknowledgments and citations have been duly provided.

- **Social.** The present work, in its current research state, does not produce any social bias. However, if applied to real-world scenarios, such as autonomous vehicles or healthcare decision systems, there is a need to ensure that the algorithms do not reinforce existing biases or result in unintended consequences that could adversely affect certain social groups.
- **Ethical.** The present work, in its current research state, does not produce any ethical bias. However, if applied to real-world scenarios, such as surveillance systems, there is a need to continually assess whether these approaches unintentionally disadvantage certain outcomes or populations.
- **Professional Issues.** The present work follows guidelines suggested in professional and academic sectors, such as the British Computer Society (BCS) Code of Conduct, the ACM and IEEE. This includes maintaining integrity, competence, and responsibility in the development and implementation of RL methods.

CHAPTER 2

Background

2.1 Reinforcement Learning

Reinforcement Learning (RL) [47] is one of the three main paradigms in machine learning, alongside supervised and unsupervised learning. What distinguishes RL from other approaches is that the learning involves a trial-and-error process in which an **agent** interacts with an **environment** to maximize long-term rewards. The agent iteratively observes the current state of the environment, takes an action, and receives a **reward** signal (positive or negative), along with the next state (see Figure 2.1). The primary objective of the RL agent is to maximize the cumulative reward, or **return**. In this section, we will formally define these concepts.

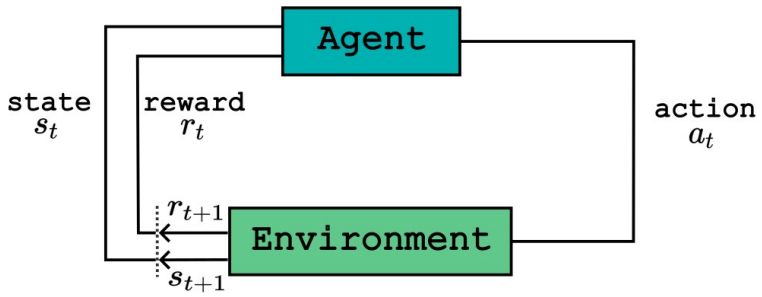


Figure 2.1: Reinforcement Learning Loop. The agent iteratively interacts with the environment by taking action a_t based on state s_t and receiving reward r_t and next state s_{t+1}.

2.1.1 Markov Decision Process

Markov Decision Processes (MDP) provide the formal framework upon which most RL algorithms are built. A **finite** MDP is an state transition system defined as a 5-tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where

- $\mathcal{S}$ is a finite set of states,
- $\mathcal{A}$ is a finite set of actions,
- $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is a transition kernel, where $\mathcal{P}(\mathcal{S})$ is the set of probability distributions on S, and $P(s'|s, a)$ is the probability of transitioning from state s to state s',
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function,
- and $\gamma \in [0, 1)$ is a discount factor.

For reference, a MDP can be visualized as a graph, as shown in Figure 2.2, where the interaction between all components is illustrated. A MDP intuitively could be understood as the environment, while the agent decision-making mechanism is defined by a policy, which induces a Markov Chain on top of a MDP.

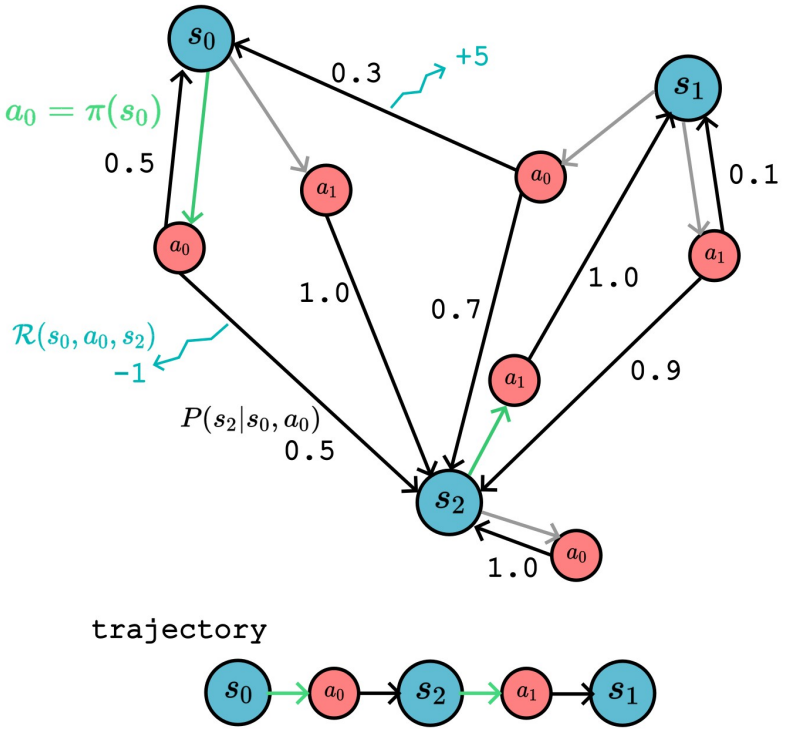


Figure 2.2: Markov Decision Process. MDP elements: states, actions, rewards, and a policy determining decisions. The bottom part shows a trajectory generated by the current policy.

A **policy** $\pi \in \mathcal{P}(\mathcal{A})^{\mathcal{S}}$ describes the agent actions in a given state. A stochastic policy is a mapping from states to distributions over actions

$$a_t \sim \pi(\cdot|s_t) \tag{2.1}$$

, while a deterministic policy is a direct mapping from states to actions

$$a_t = \pi(s_t) \tag{2.2}$$

2.1.2 Trajectories, Return, and Value Functions

The iterative interaction of an agent in the environment (see Figure 2.2) is depicted in a **trajectory**[1] (also called rollout or **episode**), which is a sequence of states and actions.

$$\tau = (s_0, a_0, s_1, a_1, \dots)$$

where a **transition** is what happen between state s_t and state s_{t+1}, in a deterministic transition as

$$s_{t+1} = f(s_t, a_t)$$

, or a stochastic transition

$$s_{t+1} \sim P(\cdot|s_t, a_t)$$

, where the action come from the policy.

The **reward** function $\mathcal{R}$ in the current work depends only on the current state and action taken, such that

$$r_t = \mathcal{R}(s_t, a_t)$$

The **return** $\mathcal{R}(\tau)$ is defined as the cumulative reward over a trajectory. It can be represented as a **finite-horizon undiscounted return**, which is the sum of rewards obtained within a fixed window of steps:

$$\mathcal{R}(\tau) = \sum_{t=0}^{T} r_t$$

, or as the **infinite-horizon discounted return**, which is the sum of all rewards ever obtained by the agent, but discounted by how far in the future they're obtained. This formulation of return includes a discount factor $\gamma \in (0, 1)$, which provide better theoretical convergence guarantees:

$$\mathcal{R}(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

In RL, any agent aims to obtain an optimal policy π^* that maximizes the expected return when the agent acts according to it.

[1] The trajectory, return and value functions explanations were based on OpenAI Spinning Up

$$\pi^* = \underset{\pi}{\text{argmax}} \underset{\tau \sim \pi}{\mathbb{E}} [\mathcal{R}(\tau)] \tag{2.3}$$

To achieve this goal, in many cases such as with DQN (as we will discuss later), it is necessary to have a notion of the value of a state or state-action pair. The **value** is defined as the expected return when starting from that state or state-action pair and then following a particular policy indefinitely. Below are some key concepts related to **value functions** retrieved from [3]:

- The **On-policy Value Function** gives the expected return if you start in state s and always act according to policy π:

$$V^{\pi}(s) = \underset{\tau \sim \pi}{\mathbb{E}} [\mathcal{R}(\tau | s_0 = s)] \tag{2.4}$$

- The **On-Policy Action-Value Function**, $Q^{\pi}(s, a)$, which gives the expected return if you start in state s, take an arbitrary action a (which may not have come from the policy), and then forever after act according to policy π:

$$Q^{\pi}(s, a) = \underset{\tau \sim \pi}{\mathbb{E}} [R(\tau) \, | s_0 = s, a_0 = a] \tag{2.5}$$

- The **Optimal Value Function**, $V^*(s)$, which gives the expected return if you start in state s and always act according to the optimal policy in the environment:

$$V^*(s) = \max_{\pi} \underset{\tau \sim \pi}{\mathbb{E}} [\mathcal{R}(\tau) \, | s_0 = s] \tag{2.6}$$

- The **Optimal Action-Value Function**, $Q^*(s, a)$, which gives the expected return if you start in state s, take an arbitrary action a, and then forever after act according to the optimal policy in the environment:

$$Q^*(s, a) = \max_{\pi} \underset{\tau \sim \pi}{\mathbb{E}} [\mathcal{R}(\tau) \, | s_0 = s, a_0 = a] \tag{2.7}$$

2.1.3 Deep Q-Learning (DQN)

The Deep Q-Learning (DQN) [40, 41] algorithm introduced a novel approach to train Q-learning algorithms using neural networks in high-dimensional, partially observable state spaces (e.g., raw pixels). It effectively addresses two important challenges in training deep neural networks for RL, such as highly-correlated states and non-stationary distribution issues. To address both issues, DQN introduced a experience replay mechanism, which facilitates breaking temporal data correlations, leading to approximate independent and identically distributed (iid) data distributions.

Experience Replay (ER) functions as a dataset with a **fixed capacity** N, storing tuples of experiences from the agent's interactions with the environment at different time steps during training, represented as

$$e_t = (s_t, a_t, r_t, s_{t+1}), \quad e_t \in \mathcal{E}$$

During training, mini-batches are randomly sampled from the experience replay to train a neural network Q_θ, which is tasked with approximating the optimal state-action value function Q^*. In fact, DQN learning aims to estimate this optimal state-action value function, such that the optimal policy[2] is

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{2.8}$$

To estimate the Q-value function, the DQN algorithm relies on a loss function derived from the Bellman equations [47], which state that the value of a state is the reward you expect to receive from that state, plus the value of the subsequent state you transition to.

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} \left[r_t + \gamma \max_{a'} Q^*(s',a') \right].$$

As we aim to approximate $Q(s,a;\theta) \approx Q^*(s,a)$, a Q-network is trained to reduce the mean-square error in the Bellman equation by minimizing a sequence of loss function $L_i(\theta_i)$ that changes at each iteration i

$$\begin{aligned} L_i(\theta_i) &= \mathop{\mathbb{E}}_{e_t \sim \mathcal{E}} \left[(y_i - Q(s_t,a_t;\theta_i))^2 \right] \\ &= \mathop{\mathbb{E}}_{(s_t,a_t,r_t,s_{t+1}) \sim \mathcal{E}} \left[(r_t + \gamma \max_{a'} Q(s_{t+1},a';\theta_i^-) - Q(s_t,a_t;\theta_i))^2 \right] \end{aligned}$$

where the optimal targets $r_t + \gamma \max_{a'} Q^*(s',a')$ are substituted with **approximated target values** for iteration i

$$y_i = r_t + \gamma \max_{a'} Q(s_{t+1},a';\theta_i^-) \tag{2.9}$$

, and the gradients respect to the parameters θ_i are

$$\nabla_{\theta_i} L(\theta_i) = \mathop{\mathbb{E}}_{(s_t,a_t,r_t,s_{t+1}) \sim \mathcal{E}} [\delta_t \nabla_{\theta_i} Q(s_t,a_t;\theta_i)] \tag{2.10}$$

where the **temporal difference error** (or td-error) for the e_i experience tuple corresponds to

$$\delta_i = r_t + \gamma \max_{a'} Q(s_{t+1},a';\theta_i^-) - Q(s_t,a_t;\theta_i) \tag{2.11}$$

Several important considerations should be taken into account when using this loss function in practice.

- The parameters θ_i^- corresponds to a separated copy of the q-network parameters that is updated with θ_i every C gradient descent steps to stabilize the training.

[2] Notice that by definition $Q^*(s,a)$ in Equation 2.7 represents the expected return for starting in state s, taking an arbitrary action a, and then following the optimal policy forever after. Therefore, by selecting the action with the maximum Q-value, we obtain the optimal policy that maximizes the expected return, same as the main RL goal in Equation 2.3.

- Although the algorithm approximates the greedy policy in Equation 2.8, it learns this strategy by using a behavioral **ϵ-greedy policy**, which encourages exploration by taking a random action with probability ϵ during training

$$\pi^{\epsilon}(a|s) = \begin{cases} a \sim \text{Uniform}(\mathcal{A}) & \text{with probability } \epsilon \\ \arg\max_a Q(s,a;\theta) & \text{with probability } 1-\epsilon \end{cases} \tag{2.12}$$

  Typically, this epsilon value is annealed over iterations to maintain a small value as learning approaches convergence.

- When learning from pixels, multiple frames n are typically **stacked together** to represent a state in a preprocessing step. Additionally, the same action can be repeated over the n stacked frames using a method called **skip frames** to reduce the training load. Thus, a state will be represented by $s_t = \phi(\{x_{t-n}, x_{t-(n-1)}, \cdots, x_t\})$, where the function ϕ stacks a window of n frames (commonly $n = 4$) while repeating the same action. For simplicity, we will refer to states in the algorithm and not the preprocessing function, but it is important to remind the reader that stacking and skip frames are used in a preprocessing step.

Algorithm 1 presents the pseudo-code for the DQN algorithm, modified slightly from the version in Mnih et al. [40, 41] to run for a fixed total budget of T (the total number of steps during the entire training), rather than running it per episode. This facilitates easier and fair comparisons with other extensions of the algorithm.

Algorithm 1 Deep Q-learning with Experience Replay (Mnih et al. [40, 41])

Input: minibatch k, step-size η, replay period K and size N, budget T (total steps).
Initialize action-value function Q with random weights θ
Initialize target action-value function Q^- with weights $\theta^- = \theta$
Initialize replay memory $\mathcal{D} = \emptyset$ with capacity N
for $t = 1$ to T **do**
    Observe s_t
    Choose action $a_t \sim \pi^{\epsilon}_{\theta}(s_t)$
    Execute action a_t and observe r_t and s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample minibatch $B \sim \text{Uniform}(\mathcal{D})$ of transitions e_j
    Set $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q^-(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Compute TD-error $\delta_j = y_j - Q(s_j, a_j; \theta)$, and TD-loss $\mathcal{L}_{\text{TD}} = \delta_j^2$
    Perform a gradient descent step on $\mathcal{L}_{\text{TD}}$
    Every C optimizing steps update $\theta^- \leftarrow \theta$
end for

2.1.4 Prioritized Experience Replay

Schaul et al. [44] explored the limitations of the ER and discovered that a DQN algorithm revisits the same experience tuple an average of eight times, with not all revisits leading to significant improvements. As a result, they proposed Prioritized Experience Replay (PER), which non-uniformly samples experiences from the replay buffer based on a priority measure. While they left room for exploring other potential priority measures, they hypothesized that TD-error could serve as an indicator of expected learning progress. According to Schaul et al. [44], the **Expected Learning Progress** (ELP) is the idealised criterion to assign a priority value with the amount the RL agent can learn from a transition in its current state. By using TD-error as a priority measure, they aimed to more frequently replay experiences that are likely to result in those greater improvements.

Specifically, the sampling probability $P(i)$ of an experience tuple e_i is defined as

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{2.13}$$

where α controls the degree of prioritization, with $\alpha = 0$ the uniform case, and p_i is the priority of an experience tuple e_i. Then, a **proportional prioritization**[3] assigns the priority as

$$p_i = |\delta_i| + \epsilon \tag{2.14}$$

where ϵ is used to revised experiences even when td-error equal zero. This strategy uses a 'sum-tree' data structure to sample efficiently from a large experience buffer without depending on the buffer capacity N.

The prioritized method, however, introduces a bias in the estimation of the Q-values. As the algorithm prioritize certain experiences over others, they will be sampled more frequently than what normally occur; skewing the distribution of experiences. This skewness does not correspond to the actual distribution of experiences encountered in the environment (the true distribution), leading to biased Q-value estimates. To address this issue, Schaul et al. [44] introduced a **weighted Importance Sampling** [39] to adjust the contribution of each sampled experience in the update step, compensating for the fact that some experiences were over-sampled (and therefore should be down-weighted) while others were under-sampled (and therefore should be up-weighted). The weight per experience is defined as

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{2.15}$$

, where N is the buffer capacity, and β is an hyperparameter to adjust the importance sampling, which in practice is annealed from an initial value β_0 to 1. Additionally, this weight is normalized by the maximal weight $\frac{1}{\max_k w_k}$ to stabilize the learning by reducing very large updates (only scaling downwards). Then, the weighted gradient is defined as

$$\nabla_{\theta_i} L(\theta_i) = \mathop{\mathbb{E}}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{E}} [w_i \delta_t \nabla_{\theta_i} Q(s_t, a_t; \theta_i)] \tag{2.16}$$

[3] An alternative **ranked-based prioritization**, defined as $p_i = \frac{1}{rank(i)}$, was also evaluated, where the $rank(i)$ represents the index of the experience in the buffer sorted according the TD-error $|\delta_i|$. This strategy employs a piece-wise linear function with k segments to enhance sampling efficiency

Algorithm 2 presents the pseudo-code for the DQN algorithm including the PER extension. It is important to note that, for practical reasons, priorities are updated only for the transitions in the current sampled mini-batch, as it would be computationally inefficient to update the entire replay buffer, especially when it is large. Additionally, new transitions added to the experience replay are assigned with a maximum priority to ensure that all experiences are sampled at least once.

Algorithm 2 DQN with Prioritized Experience Replay (PER) (Schaul et al. [44])

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T (total steps).
Initialize action-value function Q with random weights θ
Initialize target action-value function Q^- with weights $\theta^- = \theta$
Initialize replay memory $\mathcal{D} = \emptyset$ with capacity N, $p_1 = 1$ (inital priority)
for $t = 1$ to T **do**
    Observe s_t
    Choose action $a_t \sim \pi^\epsilon_\theta(s_t)$
    Execute action a_t and observe r_t and s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$ with maximal priority $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \mod K$ **then**
        Sample minibatch B of transitions e_j with probability $P(j) = \frac{p_j^\alpha}{\sum_i p_i^\alpha}$
        Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} \,/\, \max_i w_i$
        Set $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q^-(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Compute TD-error $\delta_j = y_j - Q(s_j, a_j; \theta)$, and TD-loss $\mathcal{L}_{\text{TD}} = \delta_j^2$
        Perform a gradient descent step on $\mathcal{L}_{\text{TD}}$, weighting the updates by w_j
        Update transition priority $p_j \leftarrow |\delta_j|$
        Every C optimizing steps update $\theta^- \leftarrow \theta$
    end if
end for

2.2 Bisimulation

Bisimulation, a concept of behavioral similarity, is central to this work. In the following sections, we will outline the key definitions of bisimulation that form the basis of our approach. For convenience, we reused the notation from [10] that is used interchangeable from now on. The next-state distribution for a given state-action pair (s, a) is defined as,

$$P_s^a = P(\cdot|s, a) \in \mathcal{P}(\mathcal{S})$$

, and the associated immediate reward

$$r_s^a = r_t = \mathcal{R}(s, a)$$

Note not get confused with $P_s^a \neq P(s'|s,a)$, the former represents a distribution and the second one a probability value.

2.2.1 Bisimulation in a Markov Decision Process

Initially introduced in the field of concurrency theory, **bisimulation** [32, 1, 7] is an equivalence relation between states of a transition system (e.g. MDP) that preserves the branching structure of the system, and which thus can simulate each other in a stepwise manner. Two states are bisimilar if they can simulate each other's behavior, thereby a bisimulation serves as a form of state abstraction that groups states $x, y \in \mathcal{S}$ that are 'behaviorally equivalent'.

Definition 1 (Bisimulation relation, Givan et al. [19])**.** Given an MDP M, an equivalence relation E between states is a **bisimulation relation** if, for all states $x, y \in \mathcal{S}$ that are equivalent under E the following conditions hold:

$$r_x^a = r_y^a \tag{2.17}$$

$$P_x^a(C) = P_y^a(C) \qquad \forall a \in \mathcal{A}, \quad \forall C \in \mathcal{S}_E, \tag{2.18}$$

where $\mathcal{S}_E$ is the partition of $\mathcal{S}$ under the relation E, and $P_x^a(C) = \sum_{x' \in C} P_x^a(x')$.

Two states $x, y \in S$ are **bisimilar** if there exists a bisimulation relation E such that $(x, y) \in E$; consequently, their optimal value functions are equal, $V^*(x) = V^*(y)$[4].

The properties in Equations 2.17 and 2.18 are referred to as **Reward Equivalence** and **Transition Equivalence**, respectively. There can be various equivalence relations that satisfy these conditions, but in most cases, we are interested in the largest equivalence relation, denoted by $\sim$ [5].

Intuitively, a bisimulation of a transition system allows us to replace a complex system with a simpler system that has fewer states while still capturing the same behavior as the original.

2.2.2 On-policy bisimulation

The strong behavioral guarantees of the aforementioned bisimulation relation can potentially hinder their applicability for RL algorithms. This is because they strictly require **exact action matching** between states, which can be highly challenging to achieve in complex environments (as it will be illustrated in detail later with an example in Section 4.1). For example, there may be actions that do not lead to positive outcomes, but the equivalence conditions will still require proper matching between them. In other cases, even when states exhibit similar optimal values, $V^*(x) = V^*(y)$, they may not satisfy the equivalence properties.

[4] Note that the converse does not hold; that is, even if $V^*(x) = V^*(y)$, it does not necessarily imply that the states x and y are bisimilar according to this definition

[5] The smallest bisimulation relation is the identity relation, which is a trivial solution where each equivalence class contains only a single state.

Under these circumstances, Castro's work [9] proposes an **on-policy bisimulation relation**, which, instead of looking to all actions matching, *focus only on the current policy actions*. This approach aligns better with what RL algorithm does, where commonly is of interest to keep a behavioral current policy which is updated iteratively as the learning progress; allowing to focus over time only on the transitions of interest.

Definition 2 (On-policy Bisimulation Relation, Castro [9])**.** Given an MDP M, an equivalence relation E^π between states is a π-**bisimulation relation** if, for all states $x, y \in \mathcal{S}$ that are equivalent under E^π the following conditions hold:

$$r_x^\pi = r_y^\pi \tag{2.19}$$

$$\mathcal{P}_x^\pi(C) = \mathcal{P}_y^\pi(C) \quad \forall C \in \mathcal{S}_{E^\pi}, \tag{2.20}$$

where $\mathcal{S}_{E^\pi}$ is the partition of $\mathcal{S}$ under the relation E^π, and

$$\begin{aligned} r_x^\pi &:= \sum_a \pi(a \mid x) r_x^a \\ \forall C \in \mathcal{S}_{E^\pi}, \mathcal{P}_x^\pi(C) &:= \sum_a \pi(a \mid x) \sum_{x' \in C} P_x^a(x') \end{aligned} \tag{2.21}$$

Two states $x, y \in S$ are π-**bisimilar** if there exists a π-bisimulation relation E^π such that $(x, y) \in E^\pi$; consequently, their on-policy value functions are equal, $V^\pi(x) = V^\pi(y)$. Denoting the largest bisimulation relation as $\sim_\pi$.

Notice that both reward and transition equivalence now only account for one action a, which is the action sampled from the policy π. This action could come from an stochastic or deterministic policy.

2.2.3 On-policy Bisimulation metric

The direct use of bisimulation relations is generally problematic because these relations are highly sensitive to infinitesimal variations in the reward function and environmental dynamics (transitions), which often arise from data-driven estimations. As a result, it is highly unlikely to exactly satisfy reward and transition equivalence in practice. **Bisimulation metrics** [16, 17, 18, 9] have been proposed to address this issue and provide a smoother notion of similarity than that offered by strict equivalence relations. These metrics are defined within a pseudometric space[6] on $\mathcal{S}$

$$\mathcal{M}(\mathcal{S}) = \{d \in [0, \infty)^{\mathcal{S} \times \mathcal{S}} : d \text{ symmetric and satisfies the triangle inequality}\} \tag{2.22}$$

, where a distance function $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ quantifies the '**behavioral similarity**' between two states. Accordingly, it is said that a pseudometric $d \in \mathcal{M}$ induces an equivalence relation $E_d := \{(x, y) | d(x, y) = 0\}$. In other terms, any two states with distance 0 will be collapsed onto the same equivalence class.

[6] Notice a pseudometric has a weaker "identity of indiscernibles" axiom $x = y \implies d(x, y) = 0$, which permits cases where two states, x and y, may be behavioral distinct yet still have a distance of 0. Please refer to Appendix A.1 for a details explanation about metrics.

In this context, Castro's work [9] proposed an **on-policy bisimulation metric**, which induces the on-policy bisimulation equivalence relation mention before, specifically the largest one $\sim_\pi$.

Definition 3 (On-policy Bisimulation metric, Castro [9] - Theorem 2)**.** A π-bisimulation metric $d^\pi_\sim$ is the unique fixed-point of the operator $T^\pi_K : \mathcal{M}(\mathcal{S}) \to \mathcal{M}(\mathcal{S})$, where

$$T^\pi_K(d)(x, y) = |r^\pi_x - r^\pi_y| + \gamma \mathcal{W}_d(P^\pi_x, P^\pi_y) \tag{2.23}$$

, where $\mathcal{W}_d$ corresponds to the Kantorovich distance (also known as Wasserstein distance) over the set of distributions $\mathcal{P}(\mathcal{S})$ with based distance d, where the minimal transport cost is taken over all the couplings (see Section 2.2.4).

Notice a π-bisimulation metric allow us to relate the behavioral distances with the value function under the current policy π, using the following theoretical guarantee.

Definition 4 (Theorem 3, Castro [9])**.** Given any two states $x, y \in \mathcal{S}$ in an MDP, $|V^\pi(x) - V^\pi(y)| \leq d^\pi_\sim(x, y)$.

The operator $T^\pi_K(d)$ is a contraction mapping, which works as standard operators in dynamic programming for reinforcement learning (e.g. value iteration [46, 47]), which in an iterative recurrent process will eventually converge to a fixed point $d^\pi_\sim$ up to an accuracy δ, where $T^\pi_K(d)$ maps effectively $\mathcal{M}(S)$ into itself, and the operator corresponds to the bisimulation metric, that is $T^\pi_K(d) = d^\sim : S \times S \to \mathbb{R}$. Then, let an initial estimate d_0, we have iterative recurrent updates as

$$d_0 \to T^\pi_1(d_0) = d_1 \to T^\pi_2(d_1) = d_2 \cdots \to d^\pi_\sim$$

2.2.4 Couplings and Kantorovich

In probability theory, **Coupling** [34] is a method that allows us to related two (possible unrelated) random variables X, Y by constructing a new joint random variable (W, Z), such that the marginal distributions corresponds to X, Y, that is

$$P(W = w) = \sum_z P(W = w, Z = z) = P(X) \tag{2.24}$$

$$P(Z = z) = \sum_w P(W = w, Z = z) = P(Y) \tag{2.25}$$

This coupling allow us to compare X and Y through the behavior of the joint distribution (W, Z). Notice that there can be different couplings from joint distributions that satisfy the marginal distribution conditions; but in most cases we are interested in analyzing joint distributions where W and Z are dependent.

The **Kantorovich (or Wasserstein) distance** [51] is defined as the minimum cost required to transform one probability distribution into another, often conceptualized as transporting mass from one distribution to another. This distance is based on the concept

of coupling, where the optimal coupling represents the most efficient way to pair elements from the two distributions in order to move mass between them. In other words, the minimal cost is determined by finding the optimal coupling that minimizes the expected transport cost. In the on-policy bisimulation metric[7] (see Equation 2.23), it is

$$W_d(P_x^\pi, P_y^\pi) = \inf_{\psi \in \Psi(P_x^\pi, P_y^\pi)} \mathbb{E}_{(x',y')\sim\psi}\left[d(x', y')\right] \tag{2.26}$$

where

- $\Psi(P_x^\pi, P_y^\pi)$ is the set of all possible couplings of the distributions P_x^π and P_y^π,
- $\psi(x', y')$ is a joint distribution that defines a coupling of P_x^π and P_y^π, and
- $d(x', y')$ is the distance between x' and y'.

2.2.5 Matching under Independent Couplings (MICO) metric

The on-policy bisimulation metric still faces challenges when computed at large scales, mainly because it requires the exact computation of the Kantorovich distance, which involves solving an optimal transport problem. This process can be computationally expensive, particularly in high-dimensional spaces.

Although the on-policy bisimulation can be computed via fixed-point iteration on the operator T_K^π, the **iterative cost is significant**. According to Castro et al. [10], the overall practical cost of computing a bisimulation metric is $\tilde{O}(|\mathcal{S}|^5|\mathcal{A}\log(\epsilon)/\log(\gamma)$, where ϵ is the tolerance and γ is the discount factor rate. Most of this complexity is dominated by the calculation of the Kantorovich distances W_d, which account for approximately $|\mathcal{S}|^2|\mathcal{A}|$ per calculation. This makes it impractical to compute the metric for high-dimensional state spaces. Additionally, computing the Kantorovich distances **requires knowledge of the transition distributions**, which are typically not readily available. As a result, other works have employed approximators for the operator under deterministic conditions [9] or Gaussian assumptions [53] to estimate these distributions.

To address the aforementioned issues, Castro et al. [10] introduced the **Matching under Independent Coupling (MICo) metric**, which employs the independent coupling to replace the Kantorovich distance. While this surrogate metric introduces a looser bound on convergence guarantees, it offers a more tractable alternative that still provides valuable insights into state similarity, even though it does not capture the exact Kantorovich distance. Furthermore, it is broadly applicable to both deterministic and stochastic environments.

Definition 5 (MICo update operator, Castro et al. [10])**.** Given $\pi \in \mathcal{P}(\mathcal{A})^\mathcal{S}$, the MICO update operator $T_M^\pi : \mathbb{R}^{\mathcal{S}\times\mathcal{S}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{S}}$

$$T_M^\pi(U)(x, y) = |r_x^\pi - r_y^\pi| + \gamma\mathbb{E}_{x'\sim P_x^\pi, y'\sim P_y^\pi}\left[U(x', y')\right] \tag{2.27}$$

for all $U : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, with $r_x^\pi = \sum_a \pi(a \mid x) r_x^a$ and $P_x^\pi = \sum_a \pi(a \mid x) P_x^a(\cdot)$ for all $x \in \mathcal{S}$,

[7] Notice bisimulation metric uses the 1-Wassertein metric not the general case d-Wassertein metric.

In an independent coupling, both x' and y' are sampled independently from $P_x^\pi(x')$ and $P_y^\pi(y')$, respectively, with no attempt to optimize the joint distribution ψ to minimize the expected distance U, and having $\psi(x', y') = P_x^\pi \cdot P_y^\pi$.

Similar to the on-policy bisimulation metric, the MICo operator T_M^π is a contraction mapping and has a unique fixed point $U^\pi \in \mathbb{R}^{\mathcal{S}\times\mathcal{S}}$. An iterative recursive application of the operator will converge to that fixed point. Additionally, the MICo operator provides similar theoretical guarantees as other bisimulation metrics.

$$|V^\pi(x) - V^\pi(y)| \leq U^\pi(x, y) \tag{2.28}$$

Diffuse Metric

The MICo metric is not a standard metric; it is a **diffuse metric**.

Definition 6 (Diffuse metric, Castro et al. [10])**.** Given a set X, a function $d : X \times X \to \mathbb{R}$ is a diffuse metric if the following axioms hold: (i) $d(x, y) \geq 0$ for any $x, y \in X$; (ii) $d(x, y) = d(y, x)$ for any $x, y \in X$; (iii) $d(x, y) \leq d(x, z) + d(y, z)$ $\forall x, y, z \in X$.

According to condition (i), in addition to being a pseudo-metric (like the on-policy bisimulation metric, which allows zero values for distinct states), a diffuse metric permits **self-distances greater than zero**.

Formally, the second term in the MICo operator (Equation 4.2.1) coincides with the Lukaszyk-Kaminski distance [37]:

$$d_{LK}(d)(\nu, \mu) = \mathbb{E}_{X\sim\nu, Y\sim\mu}[d(X, Y)], \tag{2.29}$$

which is itself a diffuse metric. Consequently, the MICo metric is also a diffuse metric, with a fixed point given by $U^\pi = |r_x^\pi - r_y^\pi| + d_{LK}(U^\pi)(P_x^\pi, P_y^\pi)$.

Although it might initially seem counterintuitive for self-distances to be greater than zero, this approach offers a practical alternative, especially in scenarios where the policies are stochastic. In such cases, trajectories can overlap significantly, complicating the distinction between them.

The Lukaszyk-Kaminski distance d_{LK} addresses this by measuring the expected distance between random variables from two distributions, focusing on the "spread" rather than exact point-to-point matches. This allows the MICO metric to capture expected similarities and overlaps between next-states distributions, making it robust against variability and noise. This is crucial in reinforcement learning, where state distributions often overlap without exact matching.

It is important to note that the notion of state self-distance serves as an indicator of the dispersion within the distribution [10], having in general $U^\pi(x, x) > 0$ and $U^\pi(x, x) \neq U^\pi(y, y)$ for distinct states $x, y \in \mathcal{S}$. Nonetheless, $U^\pi(x, x) = 0$ iff the policy π is deterministic when evaluated at x.

CHAPTER 3

Literature Review

3.1 Non-Uniform Sampling Experience Replays

Prioritized Experience Replay (PER) [44] is by far the most relevant non-uniform sampling method, which samples visited experiences proportional to the absolute TD errors, thereby efficiently reducing convergence time. Following these promising results, numerous attempts have been made to further improve and refine non-uniform sampling to address various challenges, such as sparse reward assignment [6, 13], experience retention [14], the bias-variance trade-off [15, 23, 46, 47], trajectory sampling [13, 35], among others. The wide variety of approaches can make it complex and unclear to differentiate between distinct and complementary methods. To clarify these distinctions, we categorize the methods into three groups that aim to capture the primary nuances of these methodologies: Priority Substitution, Priority Reweighting, and Auxiliary Mechanisms.

3.1.1 Priority Substitution

Substitution-based non-uniform sampling methods replace the priority definition with an alternative proxy for expected learning progress, while still preserving the core concept of assigning a priority to each element in the experience replay.

Andrychowicz et al. [6] proposed a hindsight method using universal policies, known as Hindsight Experience Replay (HER). This method accepts both a current state and a goal state, encouraging the experience replay to sample experiences in hindsight; in other words, using a different goal on the fly than the one the agent was originally trying to achieve in the episode. HER effectively handles sparse and binary rewards without requiring additional reward manipulation. Subsequent efforts by de Bruim et al. [14] explored a concept of experience selection, which address both retention and sampling by using different proxies to define priorities based on the immediate and long-term utility.

They provided promising implementation guidelines to proxies that can be highly sensible to a task at hand.

Alternatively, Dopamine Rainbow [23] empirically evaluates six alternatives of DQN algorithms, demonstrating the significant importance of prioritized replay and multi-step targets in enhancing DQN performance. Unlike single-step targets in temporal difference (TD) error, a multi-step target considers n steps with intermediate actions determined by the behavioral policy. This approach effectively substitutes single-step td-errors priorities for multi-step td-error priorities to train DQN algorithms. The multi-step bootstrap target [46, 47] efficiently balances the bias-variance trade-off, enabling the rapid propagation of newly observed rewards to previously visited states. Fedus et al. [15] extends this work by rigorously studying how different elements of ER impact the DQN algorithm. The findings reveal the imporatance of multi-step targets for leveraging replay buffers with large capacities, despite the significant degree of off-policyness they may introduce.

Recent works have explored the use of trajectories for both experience replay and priority assignment. Dai et al. [13] proposed a two-stage method to enhance Hindsight Experience Replay (HER) by incorporating diversity-based trajectory sampling. First, trajectories are non-uniformly sampled from an experience replay buffer using priorities based on determinantal point processes (DPPs). Second, a k-DPP is applied to each trajectory to sample transitions with diverse goal states from the previously selected trajectories during training. Without relying on semantic knowledge of the goal space or tuning a curriculum, this method achieves efficient training and performance in robotic manipulation tasks. Subsequently, Liu et al. [35] further explored the concept of Trajectory Replay (TR), which stores complete trajectories instead of individual transition tuples. In these methods, transitions are sampled in a backward manner, building on the findings of Lee et al. [31], by considering the last steps of the trajectories as the initial candidates for sampling in the current batch. Once all transitions from a trajectory are sampled, a new trajectory is included in the batch. Priorities are then assigned to entire trajectories rather than individual transitions, a method known as Prioritized Trajectory Replay (PTR). The method outperforms Atari baselines by achieving a fast and stable learning that only requires 10% of samples.

3.1.2 Priority Reweighting

Reweighting-based non-uniform sampling methods modify the priority definition by reweighting the TD-error in some manner, while still preserving the core concept of assigning a priority to each element in the experience replay.

Kumar et al. [26] propose a corrective feedback method, reweighing samples by adjusting the TD-error target value with an estimated target value error, mitigating error accumulation. Their method, called DisCor, outperforms in various RL scenarios, such us Atari bechmark, maniputation tasks, and multi-task RL, and can easily be integrated in other valued-based algorithms. Consequently, Liu et al. [36] provided theoretical explanation for PER and DisCor methods, and proposed an optimal prioritization strategy based on regret minimization, indicating transitions with higher hindsight TD error should be prioritized. Their method ReMERT exploits temporal state ordering, showing outperforming results

in RL benchmarks such as MuJoCo, Atari, and Meta-world. Concurrently, SUNRISE [30] reweighs target Q-values based on uncertainty estimates from a Q-ensemble, improving signal-to-noise in Q-updates and stabilizing learning. Their method was integrated in off-policy RL algorithms, such us SAC and Rainbow DQN, consistently, outperforming the state-of-the-art results. Lastly, Sinha et al. [45] introduced a method that reweights priorities by the likelihood-free density ratio between on-policy and off-policy experiences. This method focuses on maintaining a diverse set of experiences in the replay buffer by ensuring that experiences are sampled from low-density regions of the state space. While their method is only applied to actor-critic methods, it shows promising results in 12 relevant Atari games, and 6 Deepmind control suite tasks.

3.1.3 Auxiliary Mechanism

Auxiliary mechanisms can be integrated with non-uniform sampling methods to modify the sampling procedure and reduce dependence on priority, without necessarily preserving to the core concept of assigning a priority to each element in the experience replay

Zha et al. [52] introduced an additional replay policy that learns to filter out irrelevant experiences and apply priorities only to the relevant ones. Their method was evaluated on various continuous control tasks using the DDPG algorithm, where it consistently demonstrated improved sample efficiency. Later, Liu et al. [36] proposed a method called ReMERN, which uses an error network to assign priorities. By training a task-specific neural network, this method ensures greater robustness across different environments, addressing various sources of randomness compared to the ReMERT alternative. Later, Pan et al. [43] proposed a model-based Stochastic Gradient Langevin Dynamics (SGLD) sampling method, which generates hypothetical experiences using the current policy and combines them with uniformly sampled experiences from an ER. The proposed SGLD method efficiently addresses issues related to outdated priorities and insufficient sample space coverage and outperforms PER in classical environments. Finally, Zhao and Tresp [54] introduced a Curiosity-driven Experience Prioritization mechanism, utilizing an auxiliary curiosity module to encourage the over-sampling of trajectories with rare achieved goal states, leading to a more balanced exploration and exploitation. This method can be integrated with any off-policy RL algorithm and was tested on six robot manipulation tasks, where it outperformed in terms of sample efficiency and final performance.

Our work aligns with methods that reweigh priorities but also includes an auxiliary mechanism to calculate bisimulation metrics. Specifically, our method learns to approximated a bisimulation metric by learning state abstractions with an encoder neural network. This network is updated with an objective loss function, which encourages to keep behaviorally similar states closer together and behaviorally dissimilar states farther apart in a latent space. The bisimulation metric defines a behavioral similarity distance, which will be used to reweigh priorities by trading off between single step TD-error and bisimulation distance, encouraging more diverse sampling.

3.2 Learning State Abstractions

RL is a well-studied topic in machine learning, with multiple success and diverse applications. Nonetheless, there are still certain challenges in practice that have been widely studied. Particularly, the **sample-inefficient problem**, which refers to the large amount of data (experiences) required to learn a feasible policy, is of interest for this current work.

According to works from Lake et al. [28], Kaiser et al. [25], and Laskin et al. [29], it has been empirically demonstrated that learning from high-dimensional observations such as raw pixel is sample-inefficient. In fact, more concise observations based on state-based features makes the learning of policies less computational intensive and sample-efficient than learning from pixels [48]. According to Laskin et al. [29], it is feasible extract the relevant state information from pixel data, as long as this information is present in the raw data, making viable the learning of policies. Then, the only dilemma is to contextualize an adequate a method to extract (or learn) these representations.

Under these circumstances, in RL, a state abstraction is defined as a technique used to simplify the representation of the state space, making it more manageable and easier for the learning algorithm to process and understand. Specifically, a **state abstraction** [2] is a function, $\phi : S \to S_\phi$, that maps each true environmental state $s \in S$ into an abstract state $s_\phi \in S_\phi$, which lives in a given lower dimensional state space. This state abstraction is regularly an auxiliary task learned during training simultaneously with the main RL objective. In this work, we are interesting in learning a surrogate bisimulation metric, called MICO [10], as part of the learning of state abstractions; thereby we identified three different approaches to learn these representations, termed as reconstruction-based, contrastive-based, and behavioral-based abstractions.

3.2.1 Reconstructions Abstractions

Reconstruction-based abstractions use reconstruction objectives to reduce the dimensionality of the states, requiring additional hand-engineering to learn explicit temporal transition models in order to account for the dynamic properties of the RL environment.

The most representative approach by far is World Models [20], which is a model-based RL method based on three models: a visual, a memory and a controller. In this approach, a variational autoencoder (visual) is trained together with a LSTM model that represents the memory, making possible to account for the environment transitions (temporal information). The low-dimensional temporal abstract representation in the visual model enables a direct training in the reduced state space without requiring to spent expensive training cycles in the actual environment. Along the same lines, Hafner et al. [22] proposed a model-based agent, known as Deep Planning Network (PlaNet) that focus on planning in a learned latent space. Specifically, their proposal uses a multi-step variational inference objective (termed as Latent Overshooting), which aims to attempt two objectives (1) encoding a latent state and reconstruction of the original image, and (2) infer approximate posterior distributions of the latent states, which are consequently used to predict the next states and rewards. By learning this lower-dimensional latent space that captures the underlying state dynamic of the environment, the method achieves to solve

several continuous control tasks using considerable less episodes compare to model-free algorithms.

While reconstruction-based methods is a viable solution to learn state abstractions, they may encode parts of the observations that are not relevant to a given task. After all, the inherit unsupervised learning that governs a variational auto-encoder cannot, by definition, know what will be useful for the RL task at hand.

3.2.2 Contrastive Abstractions

Contrastive-based abstractions focus on learning more useful semantic representation by distinguishing between similar (positive) and dissimilar (negative) pairs of data points, typically by maximizing the distance between negatives and minimizing the distance between positives in a latent space.

In this context, van den Oord et al. [42] proposed Contrastive Predictive Coding (CPC), an unsupervised learning approach for representation learning, which uses autoregressive models (e.g. LSTM) for predicting future data points in a latent space. To do that, the method uses Noise-Contrastive Estimation (NCE) to contrast true future samples and a set of randomly sampled negative examples. Their method provides promising performance on several domains, suggesting a possible universal method for different modalities. Consequently, in the RL domain, Srinivas et al. [29] eliminates the dependency on autoregresive pipelines by incorporating implicit temporal information through augment temporal sequential frames. Their method, CURL, includes a simple instance auxiliary contrastive learning model that maximizes agreement between augmented versions of the same state, where each state is a stack of temporally sequential frames. The positive and negative pairs are chosen, performing instance discrimination on those frame stacks, which allow to learn both spatial and temporal discriminative features. Along the same lines, Anand et al. [5] proposed a contrastive approach based on maximizing mutual information between local, global, and temporal features. To do that, an InfoNCE technique is used where the positive samples corresponds to temporarily close states (e.g. the state in the next time step), and negative samples corresponds to distant temporal states. This approach successfully learns more granular semantic representations, such as agent location and objects, while ignoring irrelevant information, such as background textures.

Contrastive-based abstractions learn efficient semantic state representations without relying on reconstruction losses. However, they still have certain limitations because they do not incorporate information about the downstream task that guides the RL system. According to Zhang et al. [53], these methods are task-agnostic.

3.2.3 Behavioral Abstractions

Behavioral-based abstractions incorporate the downstream task knowledge, by considering the behavioral equivalence between states. Specifically, bisimulation metrics serve as a form of state abstraction that groups states s_i and s_j that are 'behaviorally equivalent'. Roughly speaking, we can understand that two states are behavioural equivalent if they can get similar long-term expected accumulated reward (the RL objective).

Deep bisimulation for control (DBC) [53] leverages this metric to learn task-aware invariant representations by assuming that the model that governs the transitions in the implicit MDP follow a Gaussian distribution. In this approach, an encoder is trained to minimize the mean square error between the on-policy bisimulation metric and L1 distance in the latent space. By distributing the states according to the bisimulation metric in the latent space, this method achieves to account for behavioral similarity, avoiding distractors in images (e.g: clouds for an automatic driving system). Additionally, as the bisimulation metric considers explicitly transitions and rewards, it achieves to address the temporal dynamics of the system when learning representations. Consequently, Castro et al. [10] successfully eliminates the Gaussian assumptions by proposing a surrogate on-policy bisimulation metric, which work more generally in deterministic and stochastic environment, and with different value-based algorithms. Their method, Matching under Independent Coupling (MICo), replace the computational expensive calculation of the Kantorovich term for the independent coupling, providing a higher theoretical bound, but still capturing behavioral similarity. As a result, the incorporation of MICo learning on other algorithms allows to outperform them in Atari Learning Environments and DM-Control benchmarks. Finally, Castro et al. [11] proposes an alternative perspective of MICo metric by using positive definite kernels. They introduced a state similarity kernel in the state space, which induces the reduced MICo distance [10], yielding similar strong results, and providing a more complete theoretical analysis.

CHAPTER 4

Methodology

This section outlines the methodology used in the present work by walking through a motivating example that revisits the bisimulation definitions from Section 2.2 to illustrate why our method is feasible and effective. Following this, we define our proposed method and provide a detailed explanation of the algorithm used in practice.

4.1 A Motivating Example: Grid World

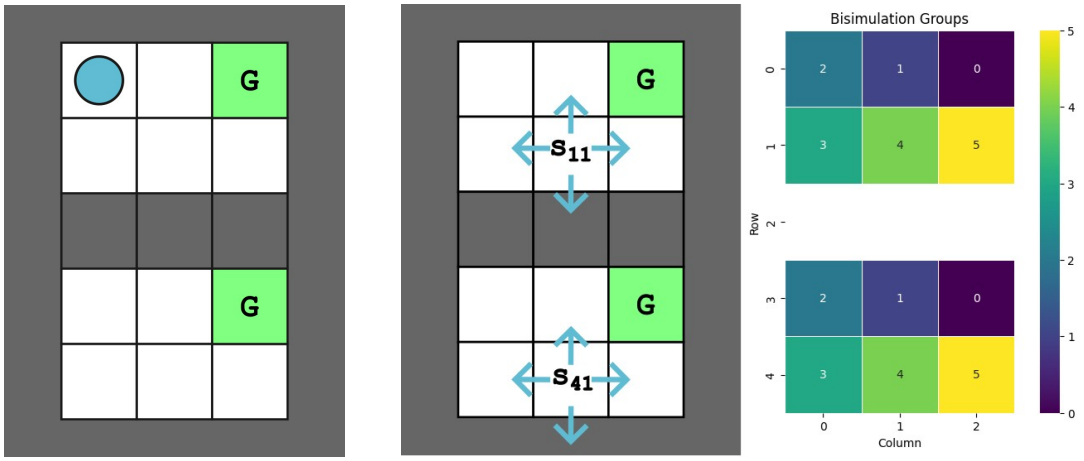
The Grid World, a simple toy environment (see Figure 4.1a), is used to explain our method. It has the following properties:

- The state space is discrete an given by all the positions of the agent in the grid, $s_{i,j} \in \mathcal{S} : \{0, n\} \times \{0, m\}$.
- The action space is discrete given by the four possible directions: down (0), right (1), up (2) and left (4), $a \in \mathcal{A} : \{0, 3\}$.
- The transitions are deterministic and restricted to adjacent cells, such that the agent will move to any adjacent cell with a probability of $P(s, a) = 1$ (if the agent encounters a wall, it remains in the same state).
- The rewards are binary and sparse, with the immediate reward always being -1 unless the agent reaches the goal (G), obtaining a reward 100.
- And the discount factor γ is 0.99

4.1.1 Bisimulation in a Grid World

The two isolated rooms in the environment (see Figure 4.1b) clearly showcase similar behaviors due to their symmetry, making them an coherent starting point for analyzing behavioral similarities. If we examine the properties of bisimulation (see Definition 1), we can notice that both reward and transition probability equivalences are easily satisfied when states from each room are paired symmetrically, corresponding to the largest bisimulation $\sim$.

It is important to note that verifying reward and transition equivalences requires considering all actions, states, and equivalence classes C, which can be quite complex. However, in the case of the Grid World, it is straightforward to check that these properties hold because the transitions are deterministic. As a result, the sum over the equivalence relation $P_s^a(C)$ are always 4 (except for the terminal state), and the rewards are the same consistently throughout the environment, except when the agent reaches the goal (when it becomes 100). We invite the reader to verify these statements.



(a) Grid World **(b)** Bisimulation

Figure 4.1: Bisimulation in Grid World. A basic Grid World illustrating the process of obtaining the largest bisimulation $\sim$, showing the agent (sky blue circle), goals (green G square), transitions, actions, and reward structures, demonstrating how bisimulation captures behavioral similar states into groups of equivalence classes.

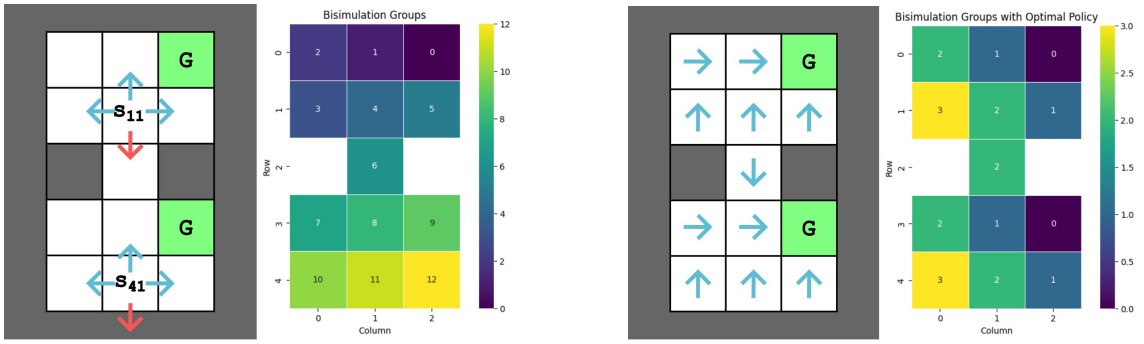
However, RL generally exhibits more complex behaviors rather than just symmetrical ones. We progressively analyze more complex behaviors later. For now, we start by opening a small passage between the two rooms. When a passage is introduced, the equivalence classes collapse to the identity relation, a trivial solution (see Figure 4.2a). This occurs because, as mentioned earlier, bisimulation is a very strong theoretical assumption that requires exact matching of rewards and transitions, which is often challenging to achieve in practice.

4.1.2 On-policy Bisimulation in a Grid World

On-policy bisimulation (Definition 2) addresses the bisimulation collapse issue by considering only the transitions and rewards specified by a given policy[1]. In the deterministic case of our Grid World, Equation 2.21 simplifies to:

$$\begin{aligned} \textbf{Given } a = \pi(x), \\ r_x^\pi = r_x^a, \\ \forall C \in \mathcal{S}_{B^\pi}, \quad \mathcal{P}_x^\pi(C) = \sum_{x' \in C} P_x^a(x'). \end{aligned} \tag{4.1}$$

Thus, it is easy to verify that the properties are satisfied; we only need to check one reward and one transition per state to ensure the reward and transition equivalence properties hold, allowing us to obtain the largest on-policy bisimulation relation, denoted as $\sim^\pi$. Figure 4.2b illustrated how the on-policy bisimulation effectively reduces the initial MDP of 13 states to an MDP of 4 states.



(a) Bisimulation Collapse **(b)** On-policy Bisimulation

Figure 4.2: Bisimulation Collapse and On-Policy Bisimulation. (a) Bisimulation collapse due to the opening of a passage, violating transition equivalence between state pairs (e.g., s_{11}, s_{41}). (b) On-policy bisimulation solution, ensuring reward and transition equivalence only for a given policy (denoted by sky blue arrows).

While these groups clearly capture the behavioral similarity of states in the Markov chain induced by the given policy, they are still not practical for RL, where we commonly estimate values from data, and the calculation of equivalence relations can be highly sensitive to infinitesimal variations.

4.1.3 On-policy Bisimulation Metric in a Grid World

On-policy bisimulation metric (see Definition 3) provide a smoother equivalence notion with the use of behavioral distances from every state to all others. Specifically, the deter-

[1]Note that this method requires knowledge of the policy being used; we employed the optimal policy π^* obtained using a value iteration algorithm [47]. In practice, an online policy also can be considered.

ministic nature of the Grid World allow us to reduce the on-policy bisimulation operator to the operator bellow according to Castro [9].

$$T_k^\pi(d)(x,y) = |r_x^\pi - r_y^\pi| + \gamma d(x',y') \tag{4.2}$$

where $x' = \mathcal{N}(x, \pi(x))$ corresponds to the next state given the state x and the action taken by the current policy $\pi(x)$; in other words, $\mathcal{N}$ is a function that deterministically selects the next state with $P(\mathcal{N}(s, \pi(S)) = 1$.

Given that the on-policy operator $T_k^\pi(d)$ is a contraction mapping that converges to a fixed point $d_\sim^\pi$, we can leverage the recurrence relation over d in Equation 4.2 to obtain the **exact on-policy bisimulation metric** through an iterative application of the operator, starting from an initial estimate d_0.

$$d_0 \to T_1^\pi(d_0) = d_1 \to T_2^\pi(d_1) = d_2 \cdots \to d_\sim^\pi$$

Specifically, we can initialize d_0 in a tabular form as a matrix of full zeros, where each cell corresponds to a pair of states in the environment (see Figure 4.3). The updates are then made using a dynamic programming approach by repeatedly calculating:

$$d_n(x,y) \leftarrow |r_x^a - r_y^a| + \gamma d_{n-1}(x',y'), \tag{4.3}$$

where x' and y' are the next states from taking action a in states x and y, respectively.

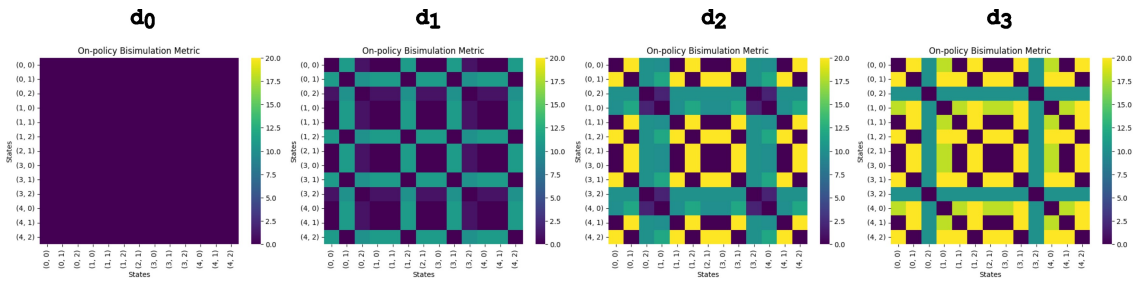


Figure 4.3: Recursive On-Policy Bisimulation Operator. Calculation of exact on-policy bisimulation distances through recursive updates of an initial estimate d_0 that converges to the metric $d_3 = d_\sim^\pi$.

By applying a Multidimensional Scaling algorithm to the final fixed point distances $d_\sim^\pi$, an approximation of the states in a 2D plane is obtained (see Figure 4.4), effectively clustering states with similar behaviors under a soft notion of distance.

The current work explores these behaviors with respect to the experiences stored in the experience replay. Note that an experience tuple is defined as $e_t = (s_t, a_t, r_t, s_{t+1})$. It is evident that experiences with a higher on-policy bisimulation distance between s_t and s_{t+1} are likely to be more informative (or 'surprising') than transitions between behaviorally similar states (see Figure 4.5a). By prioritizing transitions with greater behavioral differences, our method encourages more diversity in the sampling process. This is the main idea behind our approach.

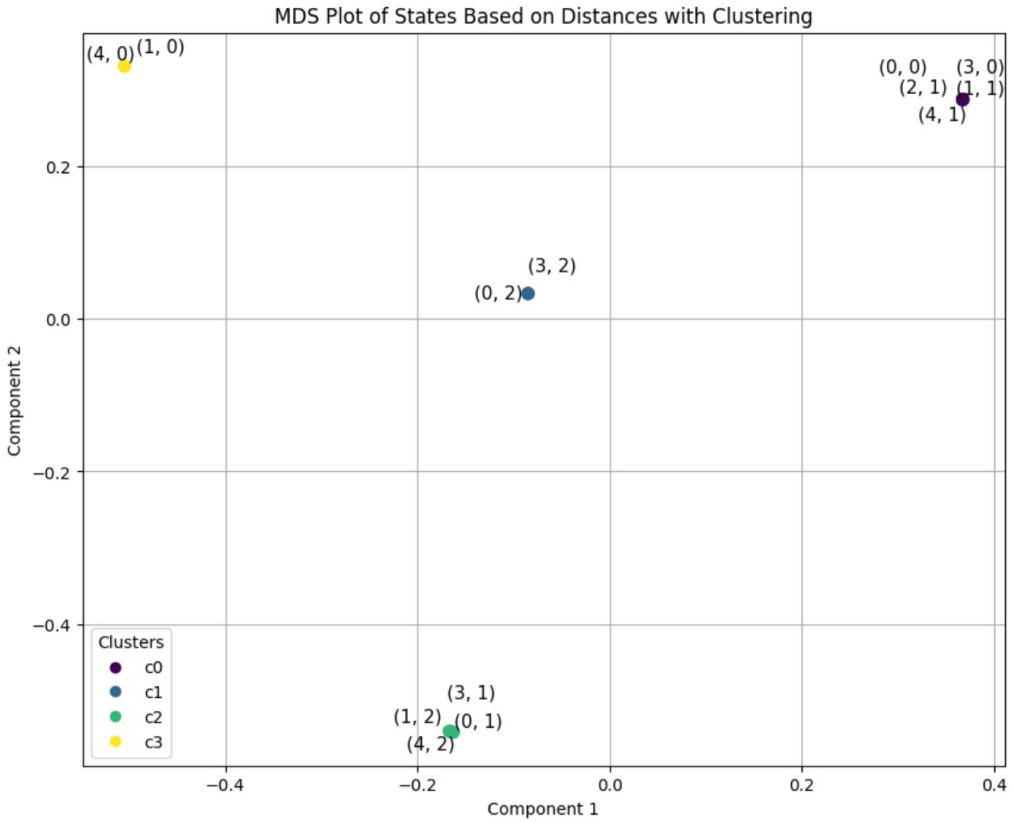
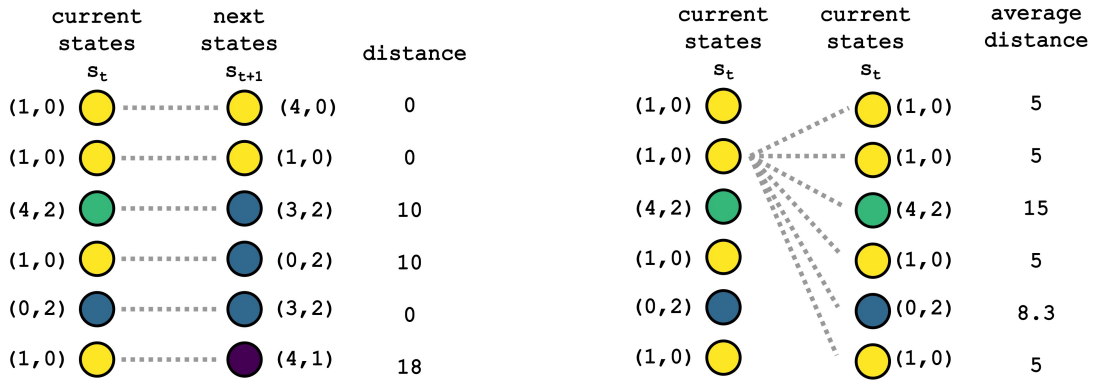


Figure 4.4: Multidimensional Scaling of On-Policy Bisimulation Distances. The MDS algorithm is applied to on-policy bisimulation distances, revealing clusters of states separated according to the computed distances.



(a) Current-vs-Next Strategy (BPERcn)

(b) All-vs-All Strategy (BPERaa)

Figure 4.5: Bisimulation Prioritization Strategies. Transitions corresponding to a mini-batch with 6 experiences, where (a) depicts the Current-vs-Next Strategy, calculated between current and next states, and (b) depicts the All-vs-All Strategy, calculated as the average distance between each current state and all other current states in the mini-batch.

4.2 Bisimulation Prioritized Experience Replay

Complex environments with high-dimensional representations (e.g., pixels) make the estimation of on-policy bisimulation metrics in a tabular form intractable. To address this challenge, we employ an indirect approach to approximate the bisimulation distances within the RL loop. Specifically, works by Zhang et al. [53] and Castro et al. [10] have demonstrated that it is more effective and general to obtain a bisimulation metric as part of the online learning of state abstractions. In the following sections, we will describe the specific method for learning these abstractions and how this approach enables us to approximate the metric and utilize it for prioritization on the fly.

4.2.1 Learning State Abstractions

The present work employs a surrogate on-policy bisimulation metric known as the MICO metric [10] (see Definition 5), where the Kantorovich distance $\mathcal{W}_d$ calculation is substituted with the independent coupling. For reference, we restate the operator equation here:

$$T_M^\pi(U)(x,y) = |r_x^\pi - r_y^\pi| + \gamma \mathbb{E}_{x' \sim P_x^\pi, y' \sim P_y^\pi} \left[U(x', y') \right]$$

This MICo operator is used to learn a parameterized state abstraction $\phi_\omega(x) : \mathcal{S} \to \mathcal{S}_\phi$ (parameterized by ω), which maps a true environmental state (e.g., pixels) to a lower-dimensional latent representation. The goal is to position these representations in such a way that a chosen parameterized distance $U_\omega(x,y)$ coincides with the MICo distance in the latent space (see Figure 4.6). Since the MICo distance is a diffuse metric (see Definition 6), the parameterized distance must ensure positive self-distances. To achieve this, the following parameterized distance function was proposed in [10] for any $x, y \in \mathcal{S}$:

$$U^\pi(x,y) \approx U_\omega(x,y) := \frac{\|\phi_\omega(x)\|_2^2 + \|\phi_\omega(y)\|_2^2}{2} + \beta\theta\left(\phi_\omega(x), \phi_\omega(y)\right) \tag{4.4}$$

where the first term ensures positive self-distances[2], $\phi_\omega(x), \phi_\omega(y)$ corresponds to the abstract representations, $\theta\left(\phi_\omega(x), \phi_\omega(y)\right)$ is the angle between vectors $\phi_\omega(x)$ and $\phi_\omega(y)$, and β is an hyperparameter.

Consequently, the recursive nature of the operator $T_M^\pi(U)(x,y)$ is used to define a loss function, which works similarly to the Bellman recurrence process in DQN. In this approach, a target is defined, and we aim to approximate this target with a online estimate. Specifically, in our case, the loss function measures the difference between a learning target MICo metric[3] $T_{\bar{\omega}}^U$ and the online MICo metric U_ω, as

$$\mathcal{L}_{\text{MICo}}(\omega) = \mathbb{E}_{\langle x, r_x, x' \rangle, \langle y, r_y, y' \rangle} \left[\left(T_{\bar{\omega}}^U \left(r_x, x', r_y, y' \right) - U_\omega(x,y) \right)^2 \right] \tag{4.5}$$

$$T_{\bar{\omega}}^U \left(r_x, x', r_y, y' \right) = |r_x - r_y| + \gamma U_{\bar{\omega}} \left(x', y' \right) \tag{4.6}$$

[2] In theory, the second term can be any other notion of distance, such as the euclidean distance. However, empirical results from MICo[10] have shown that the angular distance provides greater numerical stability.

[3] $U_{\bar{\omega}}$ is an unbiased estimator of the independent coupling.

where $\bar{\omega}$ is a separate copy of the network parameters, synchronized with ω at infrequent intervals, and the pairs of transitions $\langle x, r_x, x' \rangle$ and $\langle y, r_y, y' \rangle$ are sampled from the experience replay[4] [5].

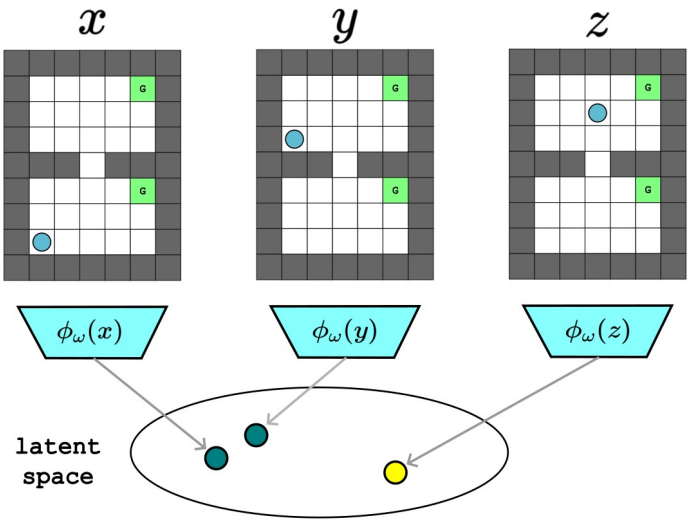


Figure 4.6: MICo Latent Space. An illustration of the latent space discovered by MICo Learning, where state representations are grouped based on behavioral similarity. Behaviorally similar states are clustered together (states x, y), while dissimilar states are positioned farther apart (states y, z).

The MICO learning can be integrated into any value-based agent by learning an estimate $Q_{\xi,\omega}(x, \cdot) = \psi_\xi(\phi_\omega(x))$, where $\phi_\omega(x)$ corresponds to the representation of state x, and ψ_ξ corresponds to the value approximator. This work specifically focuses on the DQN algorithm (see Section 2.1.3), where the $Q_{\xi,\omega}$ corresponds to the Q-values. In this scenario, the MICO loss $\mathcal{L}_{\text{TD}}$ is combined with the temporal-difference loss $\mathcal{L}_{\text{MICo}}$ as

$$\mathcal{L}_\alpha(\xi, \omega) = (1 - \alpha)\mathcal{L}_{\text{TD}}(\xi, \omega) + \alpha\mathcal{L}_{\text{MICo}}(\omega) \tag{4.7}$$

, where $\alpha \in (0, 1)$. Figure 4.7 illustrates the network architecture used for learning, showing that the MICo loss is applied after the convolution layers and used to calculate the total loss $\mathcal{L}_\alpha$. Note that the same parameters ω are shared for both losses, without the need for additional parameters.

[4] The actions are not considered because we assume a fixed policy under the on-policy assumption.

[5] When distances are large, they can oversaturate the MICo loss, causing instability in gradient updates. Therefore, in practice, a Huber loss is employed to focus on small differences between states.

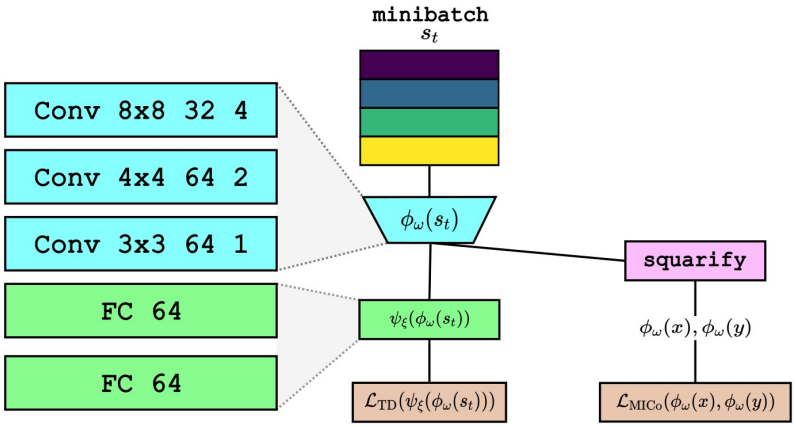


Figure 4.7: MICo Learning. An illustration of the MICo Learning framework using a standard Q-Network with a slight modification: an additional output from the encoder ϕ_ω (sky-blue layers) is extracted immediately after the convolutional layers to obtain state representations, apply an squarify method, and compute the MICo loss. The TD-loss is computed as in the standard DQN algorithm..

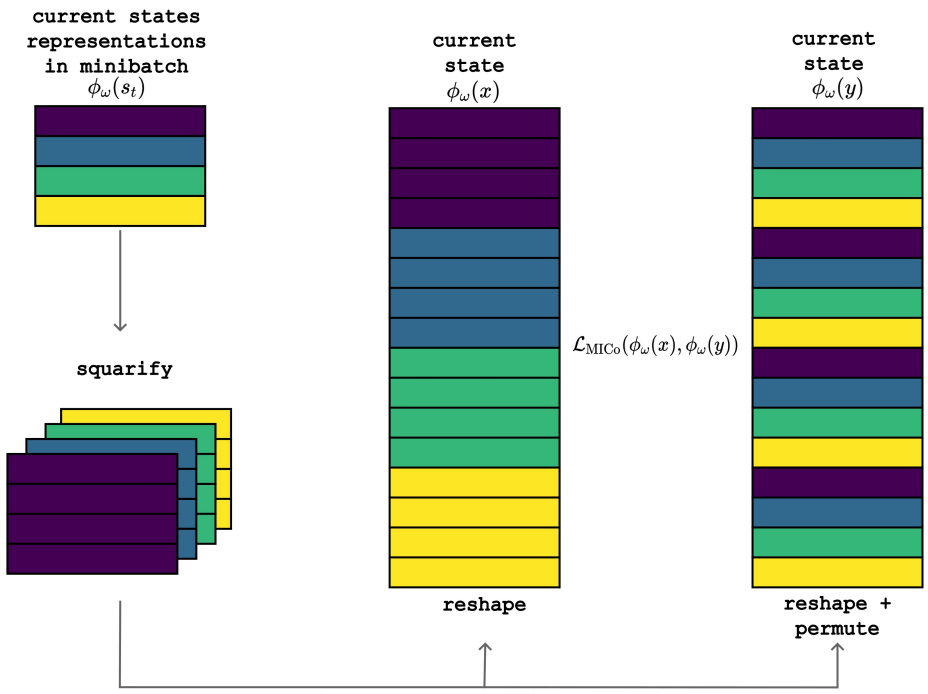


Figure 4.8: Illustration of the squarify method. This method replicates the samples along an additional dimension, reshapes the resulting square tensor, and then duplicates and permutes it; resulting in an all-vs-all comparison between each representation in the current mini-batch.

Although the MICo loss $\mathcal{L}_{\text{MICo}}$ requires pairs of transitions $\langle x, r_x, x' \rangle$ and $\langle y, r_y, y' \rangle$, in

practice, transitions are not sampled as pairs; we only have access to a mini-batch of unpaired transitions, necessitating a method to pair them. To address this issue, a **squarify** method [9] was proposed to construct pairs of transitions by pairing each transition with all others within the current mini-batch and then calculating the MICo loss. Figure 4.8 illustrates how this squarify process works.

4.2.2 Priority Strategies

In the following, we propose our method, **Bisimulation Prioritized Experience Replay (BPER)**, along with **two strategies** for assigning priorities on the fly. The MICo metric is approximated online as part of the state abstraction learning process, so that it is enough to use U_ω to calculate a MICo distance between two states and update the priorities accordingly.

Another possible option for calculating the distance online is to use the target distance $T^U_{\bar{\omega}}$, which uses the corresponding rewards difference term. However, since the metric will eventually converge to a fixed point, it is sufficient and more convenient to use U_ω, as this approach does not require incorporating rewards into its calculation.

Strategy 1 (Current vs Next: BPERcn). Consider a current minibatch $B \subset \mathcal{E}$, $|B| = k$, containing transition experiences $e_i = (s_i, r_i, a_i, s_{i+1})$. Each transition can be associated with a distance $U_\omega(s_i, s_{i+1})$, which effectively quantifies how behaviorally different the **current state** s_i is compared to the **next state** s_{i+1}. Consequently, the re-weighted priority is calculated as follows:

$$p_i = (1 - \mu)|\delta_i| + \mu U_\omega(s_i, s_{i+1}) + \epsilon, \tag{4.8}$$

where δ_i corresponds to the TD-error of the experience, $\mu \in [0, 1)$ is the priority weight that controls the balance between the TD-error and the bisimulation distance, and ϵ is a small positive constant to ensure that experiences are revisited even when the weighted priority is equal zero.

The distance $U_\omega(s_i, s_{i+1})$ serves as an indicator of a 'surprising' transition, suggesting that transitions with larger distances are more informative and likely to contribute to greater expected learning progress. While using TD-error as an standalone method could potentially overemphasize some transitions, leading to a loss of diversity [44],[43],[15], our method consistently encourages diversity in the experiences by reweighting the TD-error, regulated by a parameter $\eta \in [0, 1)$.

Similar to a proportional prioritization in PER [44] (see Section 2.1.4), the sampling probability for an experience e_i is given by $P(t) = \frac{p_t^\alpha}{\sum_k p_k^\alpha}$, where α controls the degree of prioritization. Additionally, the weighted importance sampling for unbiased updates is computed as $w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$, followed by a normalization step $\frac{1}{\max_k w_k}$. In practice, a sum tree data structure is used to optimize the sampling procedure.

Strategy 1 provides a theoretical valid method to characterize the priority of a transition based on a behavioral distance between the current and next state. However, in practice, the trajectories may overlap significantly, resulting in **overlapping states distributions** with highly similar behaviors. Additionally, since the bisimulation is approximated online, there are sources of variability in the approximation until reaching the convergence point. These issues can trigger trajectories with many similar distances between current and next states, with only a few behavioral different states appearing after long rollouts; resulting in scarce high-priority transitions, and consequently affecting the effectiveness of prioritization technique. To address these issues, the following empirical solution is proposed.

Strategy 2 (All vs All: BPERaa). Consider a minibatch $B \subset \mathcal{E}$, $|B| = k$, with transition experiences $e_i = (s_i, r_i, a_i, s_{i+1})$. Each transition e_i can be associated with a relative bisimulation distance

$$U^B_\omega(s_i) = \sum_{i=1}^{k} U_\omega(s_i, s_k), \quad \forall e_k = (s_k, r_k, a_k, s_{k+1}) \in B \tag{4.9}$$

Consequently, the re-weighted priority will be calculated as follows

$$p_i = (1-\mu)|\delta_i| + \mu U^B_\omega(s_i) + \epsilon \tag{4.10}$$

where δ_i corresponds to the TD-error of the experience, $\mu \in [0, 1)$ is the priority weight that controls the balance between the TD-error and the bisimulation distance, and ϵ is a small positive constant to ensure that experiences are revisited even when the weighted priority is equal zero.

Although the current states from different transitions are not directly connected by a valid transition, the bisimulation metric allows us to measure behavioral dissimilarity between all states in the environment. This enables the calculation of a behavioral distance relative to the current minibatch by computing the mean distance between each current state and all other current states in the minibatch (see Figure 4.5b). This relative distance serves as a smoother indicator of how behaviorally dissimilar one initial state is compared to the others, indicating 'surprising' or more informative transitions. In practice, we use the squarify method, along with reshaping and averaging operations, to compute these distances on the fly. Similar as before, the α-priority, weighted importance sampling, and sum tree data structure are used for the sampling procedure.

Algorithm 3 presents the pseudo-code for the entire prioritizing process. Similar to the PER method, the priorities are only updated for the current sampled mini-batch to maintain computational efficiency. Notice the algorithm depicts the procedure for Strategy 1 (BPERcn). Strategy 2 (BPERaa) requires a minor replacement in line 19, where $U_\omega(s_i, s_{i+1})$ should be replaced with $U^B_\omega(s_i)$. An additional algorithm illustrating only MICo learning with DQN, without BPER, is provided in Appendix A.2 for reference.

Algorithm 3 DQN with Bisimulation Prioritized Experience Replay (BPERcn)

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T (total steps), priority weigh μ.
Initialize action-value function Q with random weights ξ, ω
Initialize target action-value function Q^{-} with weights $\{\bar{\xi}, \bar{\omega}\} \leftarrow \{\xi, \omega\}$
Initialize replay memory $\mathcal{D} = \emptyset$ with capacity N, $p_1 = 1$ (inital priority)
for $t = 1$ to T **do**
    Observe s_t
    Choose action $a_t \sim \pi^{\epsilon}_{\theta}(s_t)$
    Execute action a_t and observe r_t and s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$ with maximal priority $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \mod K$ **then**
        Sample minibatch B of transitions e_j with probability $P(j) = \frac{p_j^{\alpha}}{\sum_i p_i^{\alpha}}$
        Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} \,/\, \max_i w_i$
        Set $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q^{-}(s_{j+1}, a'; \{\bar{\xi}, \bar{\omega}\}) & \text{otherwise} \end{cases}$
        Compute TD-error $\delta_j = y_j - Q(s_j, a_j; \{\xi, \omega\})$, and loss $\mathcal{L}_{\text{TD}} = \delta_j^2$
        Squarify minibatch B to get transition pairs $\langle x, r_x, x' \rangle, \langle y, r_y, y' \rangle$
        Compute MICo loss $\mathcal{L}_{\text{MICo}} = \left(T^{U}_{\bar{\omega}}\left(r_x, x', r_y, y'\right) - U_{\omega}(x, y)\right)^2$
        Compute total loss $\mathcal{L}_{\alpha}(\xi, \omega) = (1-\alpha)\mathcal{L}_{\text{TD}}(\xi, \omega) + \alpha\mathcal{L}_{\text{MICo}}(\omega)$
        Perform a gradient descent step on $\mathcal{L}_{\alpha}(\xi, \omega)$, weighting the updates by w_j
        Update transition priority $p_j \leftarrow (1-\mu)|\delta_j| + \mu U_{\omega}(s_j, s_{j+1}) + \epsilon$
        Every C optimizing steps update $\{\bar{\xi}, \bar{\omega}\} \leftarrow \{\xi, \omega\}$
    end if
end for

CHAPTER 5

Experimental Setup

The proposed method was tested in two setups: 1) 31-state Grid Worlds illustrated in Figure 4.6; similar to those environment in the works of Pan et al. [43] and Castro [9], where calculating the bisimulation metrics is relatively straightforward; and 2) the Classic Control benchmark suite[1], and the Lunar Lander from the Box2D suite[2]. In these setups, states are **partially observable derived from pixels**, rather than full descriptive states.

The experiments aims to validate our proof of concept rather than test state-of-the-art results. Although the Grid Word was trained using pixels-based states, the underlying 31-state MDP enabled us to measure evaluation metrics without high computational cost. Specifically, we used the Grid World to evaluate our algorithm's performance on three key problems:

- **Task-agnostic Sampling.** We examined how well the algorithm prioritizes behaviorally dissimilar states by showing distributions of exact current-next on-policy bisimulation distances and distribution of priorities.

  Specifically, the on-policy bisimulation recurrent operator T_K^π for deterministic environments (see Equation 4.2) was employed to compute exact on-policy bisimulation distances, providing a quantitative evaluation. In the experience replay, we associated each transition with the exact on-policy bisimulation distance between the current and next state. This exact distance effectively serves as a behavioral value indicator per transition, and allow us to assess the degree of behavioral similar or dissimilar transitions in an experience replay by plotting their distributions over time. The reader must keep in mind that these exact distances differ from the approximated distances used in the BPERcn strategy; while the BPERcn provides

[1] Classic Control benchmark: https://www.gymlibrary.dev/environments/classic_control/
[2] Box2D suit benchmark: https://gymnasium.farama.org/environments/box2d/

approximate measures, the operator T_K^π offers 'theoretical' exact distances[3].

Additionally, a visual inspection was conducted to examine the transition frames for each quartile of the priority values in the experience replay.

- **Outdated Priorities.** we replicated the sampling distribution experiments from Pan et al.'s work [43]. The purpose of this experiment is to assess how closely the sampling priorities distribution updated per minibatch aligns with the ideal distribution, achieved when all possible transitions under the current policy are updated at each time step[4].

  Given the sampling distribution of priorities $p_i(\cdot)$ from the experience buffer and the corresponding ideal distribution $p_i^*(\cdot)$, (1) the on-policy weighting is given by $\sum_{j=1}^{31} w^\pi(s_j)|p_i(s_j) - p_i^*(s_j)|, i \in \{1,2\}$, and (2) the uniform weighting is $\frac{1}{31}\sum_{j=1}^{31}|p_i(s_j) - p_i^*(s_j)|, i \in \{1,2\}$, where p_1 corresponds to the PER method and p_2 corresponds to the BPER method. Please refer to Appendix A.3 for a more detailed explanation of this procedure.

- **State Space Coverage.** We visually and quantitatively assessed the level of exploration per algorithm. We plotted the distribution of visited states in our Grid Worlds, similar to [43], and calculated the corresponding entropy of the state visitation distribution as $H(\mathcal{S}) = -\sum_{i=1}^{|\mathcal{S}|} p(s_i)\log p(s_i)$, where $p(s_i)$ is the probability of visiting state s_i, calculated as the ratio of visits to state s_i over the total number of visits to all states.

Additionally, our algorithm was evaluated against standard ER and Prioritized ER in classical environments, such as: Cartpole, Mountain Car, Acrobot, and LunarLander, using pixel-based states. For measure the performance, we use the following evaluation metrics:

- **Episode Reward.** The episode reward consist in the accumulated reward in an entire trajectory. It serves as an indicator of performance in RL algorithms, where large values indicate that the agent is learning and acting efficiently.

- **Episode Reward Gain.** Due to overlapping results, an episode reward metric was calculate to compare the methods PER, BPERaa, and BPERcn against two baselines: DQN and DQN + MICo.

  Episode Reward Gain is calculated per episode as follows: $\mathcal{R}^G(\tau) = \bar{\mathcal{R}}_{\text{method}}(\tau) - \bar{\mathcal{R}}_{\text{baseline}}(\tau)$, where τ can correspond to trajectories of different lengths[5], and $\bar{\mathcal{R}}$ corresponds to the mean episode reward over independent executions. A positive

[3]The calculation of the exact on-policy bisimulation distances is only feasible in Grid World, where the true states are fully known. In practice, such information is typically not accessible.

[4]While the ideal distribution represents the best-case scenario, it is impractical to update all possible state priorities under the current policy at each step in practice.

[5]To handle these differences, we use back and forward filling for NaN values in the episodic reward trajectory results.

Episode Reward Gain indicates an improvement over the baseline, while a negative value indicates underperformance compared to the baseline.

- **Mean Batch Priority.** It is impractical to calculate prioritization distributions for classical environments due to high-dimensional **stacked** states. However, it is still possible to gain some insight into the priority values by studying the average priority in each minibatch sampled during training.

The RL algorithm used for all our experiments was DQN [40], with extensions added in a plug-in fashion while maintaining consistent hyperparameters across different environments and algorithms to ensure fair comparisons. For PER and MICo state-of-the-art hyper-parameters were used, extracted from the original papers, and only a sweep over the newly introduced hyperparameter ν was conducted for the 31-state Grid World. A detailed explanation of the hyperparameters used is provided in Appendix A.4. The algorithms were implemented using the TorchRL library[6]. Specifically, the baseline DQN algorithm was taken from the SOTA implementations available at the TorchRL GitHub repository, and the corresponding extensions (PER, MICo, MICo + BPER) were developed and added. The MICO extension was reproduced on TorchRL based on the original repository: Google Research MICO. The experiments were executed on an RTX 4060 GPU with 8GB of memory. The algorithm is accessible through the university GitLab and a personal GitHub.

[6]TorchRL: https://pytorch.org/rl/stable/index.html

CHAPTER 6

Results and Discussion

This section presents a series of experiments conducted in different RL environments. A 31-state simple GridWorld was used as a proof of concept to evaluate three key problems in non-uniform sampling methods: task-agnostic sampling, outdated priorities, and state space coverage. After that, a general evaluation was performed on classical environments. This section integrates results with inline discussion to enhance clarity and understanding; however, an additional summary discussion is provided at the end to highlight the most relevant findings.

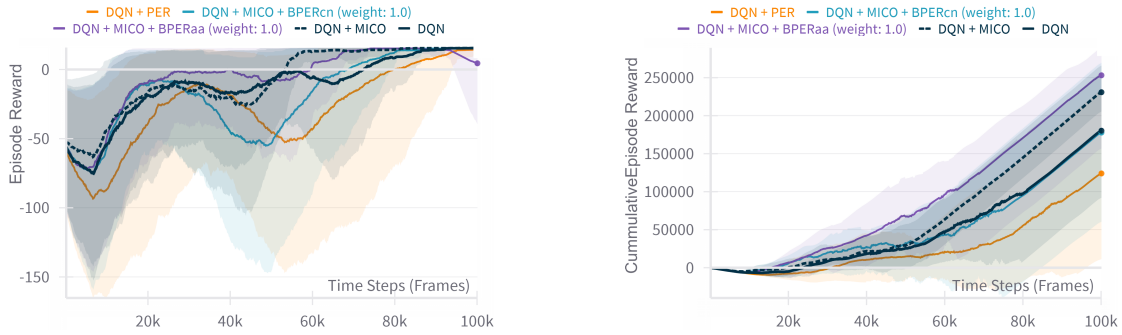
6.1 Episode Reward in Grid World

In the following experiments, the full priority weighting (weight: 1.0) version was used for both BPERcn and BPERaa variants, such that prioritization is based solely on bisimulation distances, without considering TD-error[1]. We make that decision deliberately to contrast the methods under a strong bisimulation prioritization. The algorithm was executed for 5 independent runs, with the mean, minimum, and maximum values (represented by the lower and upper margins of the shaded regions) shown per time step in Figure 6.1. The methods DQN and DQN + MICo served as baseline methods for comparison.

Figure 6.1a shows the episode reward per time step during training, calculated using a 100-episode moving average window. While all methods demonstrate improvement over time and converge after 100k steps, both BPERcn and BPERaa show promising improvements early in the training process, outperforming the baselines and the PER extension. However, as training progresses, BPERaa maintains more consistent and stable improvements, suggesting better management of priorities in experience replay, whereas BPERcn experiences a decrease in performance after 30k steps. The PER alternative underperforms

[1] Additionally, a sweep over various priority weights (0.1, 0.25, 0.5, 0.75, and 1.0) was conducted, and the results are provided in Appendix A.6.

compared to all other methods, highlighting the effectiveness of prioritizing behaviorally dissimilar states.



(a) Episode Reward **(b)** Cumulative Reward

Figure 6.1: Episode and Cumulative Reward in Grid World. (a) Episode reward performance over time, averaged with a moving window of 100 episodes across 5 independent executions. (b) Cumulative Reward performance over time, averaged over 5 independent executions, with shaded regions representing the variability for each method.

The cumulative reward, as shown in Figure 6.1b, indicates the total reward collected over time, which provides insight into the overall reward collection of each method. These results suggest that BPERaa produces a policy that consistently collects higher rewards, outperforming the direct baseline DQN + MICo, and demonstrating that BPERaa positively impacts DQN + MICo's performance. In contrast, the BPERcn strategy shows improvements in the early stages of learning but starts to decline around the 50k time step, reducing reward acquisition compared to DQN + MICo and eventually performing similarly to the baseline DQN. Overall, the PER method performs the worst, struggling to collect rewards and hindering the learning of the baseline DQN.

(a) Baseline DQN **(b)** Baseline DQN + MICO

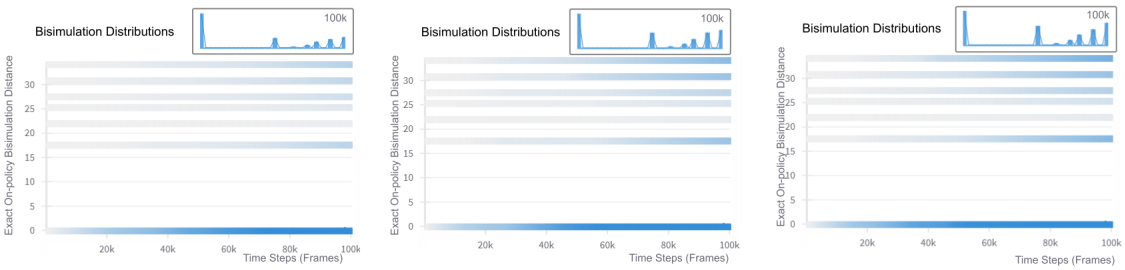
Figure 6.2: Episode Reward Gain in Grid World. Episode reward gain performance over time, averaged with a moving window of 100 episodes across 5 independent executions. The values was calculated against two baselines (a) DQN, and (b) DQN + MICO.

The episode reward gain, shown in Figure 6.2, provides an alternative perspective for evaluating improvements relative to the baselines, DQN and DQN + MICo, respectively. This evaluation metric is calculated using a moving average window of 100 episodes. The results confirm the promising improvements of the BPERaa strategy over the other methods relative to both baselines. Both strategies, BPERaa and BPERcn, exhibit similar improvements in the early stages of training. However, the performance of BPERcn declines around the 30k step under both baselines. Interestingly, Figure 6.2b reveals a decreasing trend around the 50k step for the BPERaa method. Although the performance eventually recovers to match that of DQN + MICo, the decline suggests some instability. Under both baselines, the PER method significantly underperforms, hindering the effective learning of the policy.

6.2 Task-agnostic Sampling

This section aims to empirically demonstrate that the proposed method efficiently prioritizes samples based on their behavioral relevance.

Figure 6.3 illustrates the distribution of exact current-next on-policy bisimulation distances in the experience replay over time, with the final distribution (at 100k step) highlighted at the top of the figure. In both BPERcn and BPERaa strategies, the behavioral dissimilar transitions evolve to larger values compared to PER over time, with frequency peaks on distances around 15, 20, and 30. It clearly demonstrates how our method prioritizes more frequently those larger behavioral dissimilar transitions. As the BPERcn encourages highly dissimilar transition between the current and next states approximated during training, the results from Figure 6.3b are expected. However, a more interesting result is that, despite of BPERaa prioritizes transitions based on relative priorities within the current mini-batch, this second strategy is still able to prioritize dissimilar experiences as efficiently as the strategy BPERcn (see Figure 6.3c).



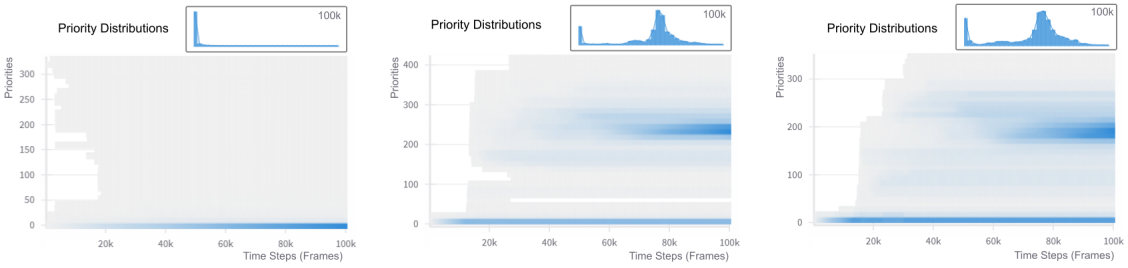
(a) DQN + PER **(b)** DQN (MICO) + BPERcn **(c)** DQN (MICO) + BPERaa

Figure 6.3: Exact On-policy Bisimulation Distributions. The distribution of exact current-next on-policy bisimulation distances in the experience replay over time for three methods (a) DQN + PER, (b) DQN + MICO + PER, and (c) DQN + MICO + BPERaa, with the final distribution (at 100k step) highlighted at the top of each subfigure.

Moreover, despite the methods aiming to prioritize highly dissimilar transitions, Figure

6.3 still shows a significant number of transitions with exact bisimulation distances close to zero. This outcome could be influenced by the use of a large buffer size of 100 000, which was deliberately set to a high capacity to observe behavior under large-scale conditions; more common in state-of-the-art methods.

Figure 6.4 illustrates the evolution of priority distributions in the experience replay over time, with the final distribution (at 100k steps) highlighted at the top. While these results do not directly associate each priority value with the exact bisimulation distances mentioned earlier, they provide insight into how many behaviorally dissimilar transitions in the experience replay are being prioritized over time. The results show an accumulation of low priority values over time in the PER method, with relatively few higher priority values. Although those priority values are low close to zero, their combined effect can occupy a large portion of the probability area, resulting in transitions with low TD-error being sampled more frequently. This phenomenon hinders DQN learning and may explain the poor episode rewards observed in previous experiments. In contrast, both proposed strategies, BPERcn and BPERaa, consistently generate higher priority values over time, encouraging more behaviorally dissimilar transitions, compensating the accumulation of low priority values. Notably, BPERaa exhibits slightly greater variability in Figure 6.4c. Despite of BPERcn and BPERaa still exhibiting an accumulation of low priority values, their frequency is lower compared to the PER method.



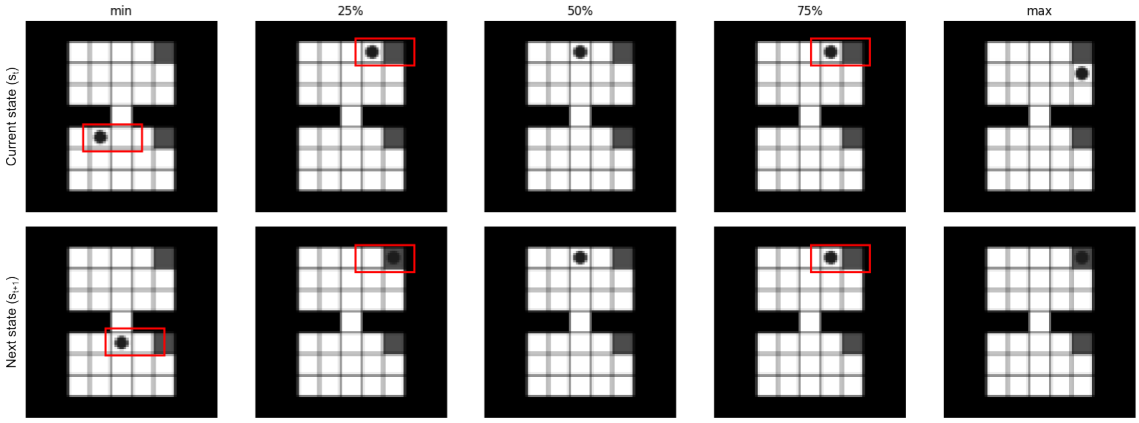
(a) DQN + PER **(b)** DQN (MICO) + BPERcn **(c)** DQN (MICO) + BPERaa

Figure 6.4: Priority Distributions. The distribution of priorities in the experience replay over time for three methods (a) DQN + PER, (b) DQN + MICO + PER, and (c) DQN + MICO + BPERaa, with the final distribution (at 100k step) highlighted at the top of each subfigure.

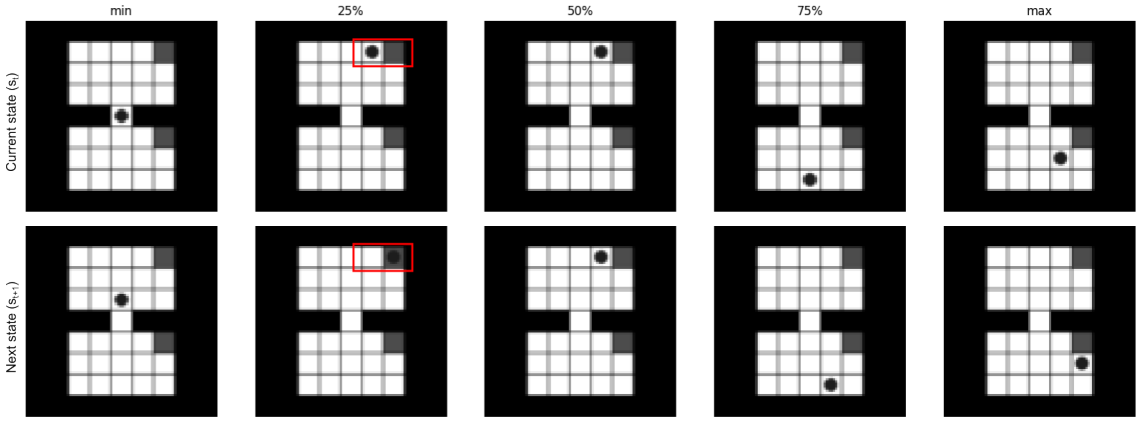
6.2.1 Visual Inspection

Figure 6.5 provides a qualitative visual inspection of the frames corresponding to the main quartiles of priority values in the experience replay collected at the 90k time step. The lower quartiles (min and 25%) represent frames with low sampling probabilities, while the higher quartiles (75% and max) represent frames with higher sampling probabilities. The top row shows the current states, and the bottom row shows the next states for each transition.

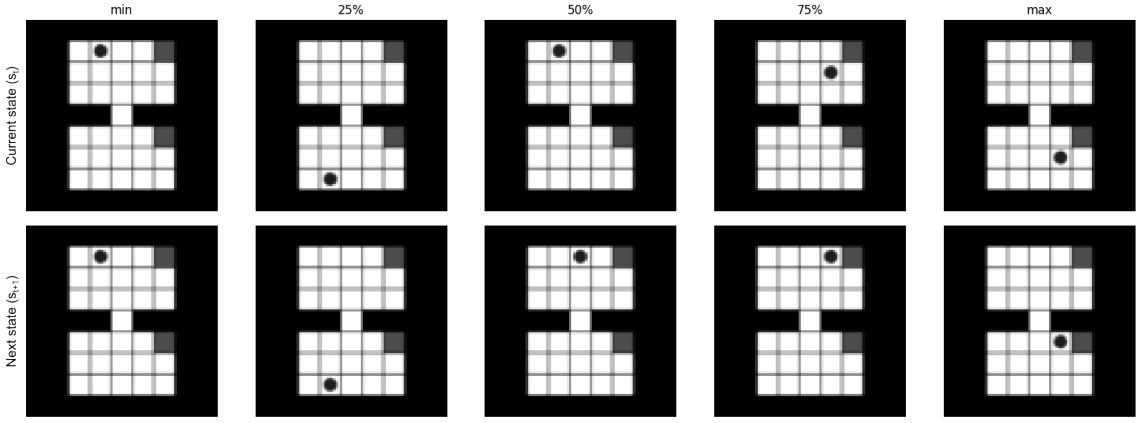
Figure 6.5a illustrates the difficulties of the PER method when attempting to prioritize state transitions using the TD-error. In the 25% quartile transition, the algorithm assigns



(a) DQN + PER



(b) DQN (MICO) + BPERcn



(c) DQN (MICO) + BPERaa

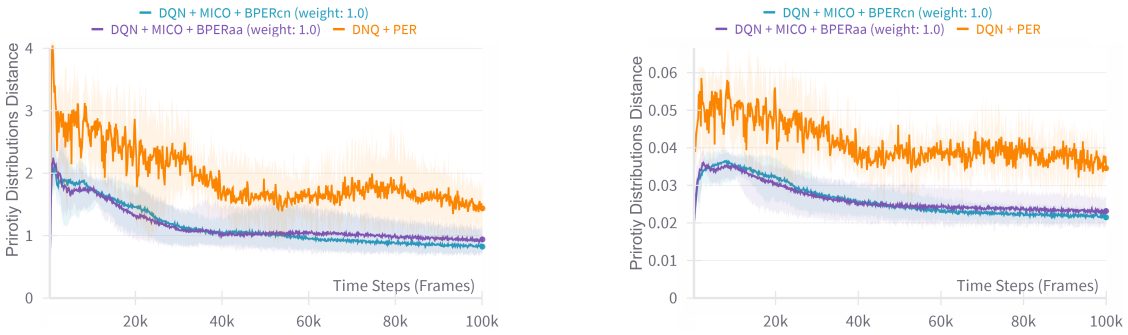
Figure 6.5: Visual Inspection. Frames from the experience replay at the 90k time step, corresponding to the main quartiles (min, 25%, 50%, 75%, and max) of priority values. Red figures highlight different problematic transitions. The top row displays the current states, while the bottom row shows the next states for each transition.

a low priority to a transition that reaches the goal, while assigning a high priority in the 75% quartile to a transition where the agent is stuck in the same position without reaching the goal, clearly resulting in low expected learning progress. In contrast, the BPERcn and BPERaa strategies, show in Figures 6.5b, 6.5c, demonstrate consistency in the lower quartiles by assigning lower priorities to transitions where the agent gets stuck or moves backward relative to the goal, while maintaining higher priorities for transitions that move towards the goal in the higher quartiles. However, the BPERcn method shows small variances in the 25% quartile, where a lower priority is assigned to a transition that leads towards the goal, suggesting **occasional issues with assigning high priorities**. Refer to reference Appendix A.5 for a visual inspection at the 50k time step.

6.3 Outdated Priorities

This section aims to empirically demonstrate that the proposed method alleviates the problem of outdated priorities. Recall that outdated priorities occur because priority updates are performed only on the current minibatch, rather than across the whole possible transition under the current policy.

Figures 6.6a and 6.6b show the distances between the sampling p_i and ideal distributions p_i^* averaged over 5 independent executions over time, using two weighting schemes, the on-policy and uniform weighting, respectively. The PER method exhibits greater distances over time than the BPERcn and BPERaa strategies. Additionally, BPERcn and BPERaa show less variability among executions. Although our method does not provides distances close to zero (such as Pan et al.'s work [43]) indicating a perfect match between distributions, our method reduce and alleviate the difference between the distributions without relying on a model-based technique.



(a) On-policy Weighting

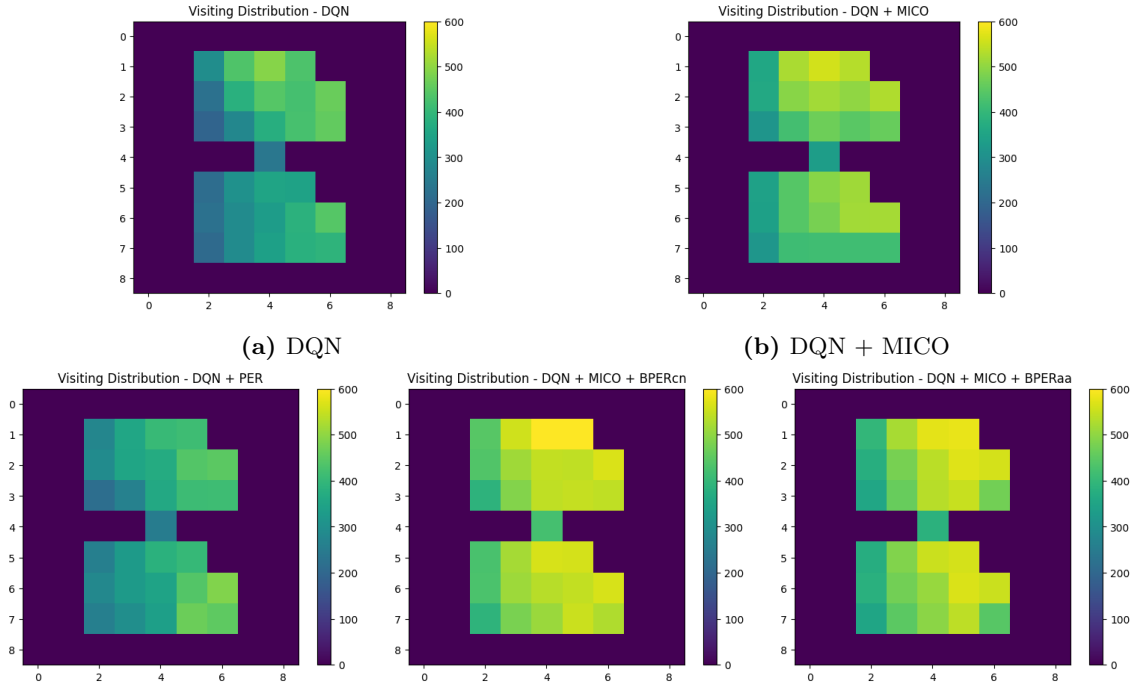
(b) Uniform Weighting

Figure 6.6: Sampling Distributions Distances. Distance between the sampling p_i and ideal distributions p_i^* averaged over 5 independent executions over time, using two weighting schemes, (a) the on-policy and (b) uniform weighting.

6.4 State Space Coverage

This section aims to empirically evaluate that the proposed method addresses the state space coverage problem. Recall the state space coverage problem arises from insufficient exploration, where the experience replay only captures a limited number of possible states, leading to suboptimal learning outcomes.

Figure 6.7 shows the state visitation distribution at the 90k time step in the 31-state Grid World, where cell values corresponds to the visit count per state. The results indicate that the strategies BPERcn and BPERaa visit more states more frequently compared to other methods, with BPERcn performing slightly better than BPERaa. Additionally, the DQN + MICO baseline outperforms DQN, and DQN + PER methods. While some of the increased exploration in BPERcn and BPERaa may be attributed to the MICo learning, the bisimulation prioritized technique further enhances exploration (relative to DQN + MICo) by significantly increasing the state visit counts, reaching values around 600.



(a) DQN **(b)** DQN + MICO

(c) DQN + PER **(d)** DQN + MICO + BPERcn **(e)** DQN + MICO + BPERaa

Figure 6.7: Visitation Distributions. Illustrations showing the frequency of state visits in the GridWorld, with most yellowish highlighting the most frequently visited states.

Figure 6.8 illustrates the visitation entropy calculated averaged over 5 independent runs over time. The results clearly showcases that BPERaa outperforms the other methods with a large and consistent entropy, while the BPERcn although initially performs efficiently its performance decays around the time step 30k. PER method is the worse among all the methods. The high entropy in BPERaa method indicates that visitation distributions

are more dispersed, consequently increasing the exploration consistently among different runs.

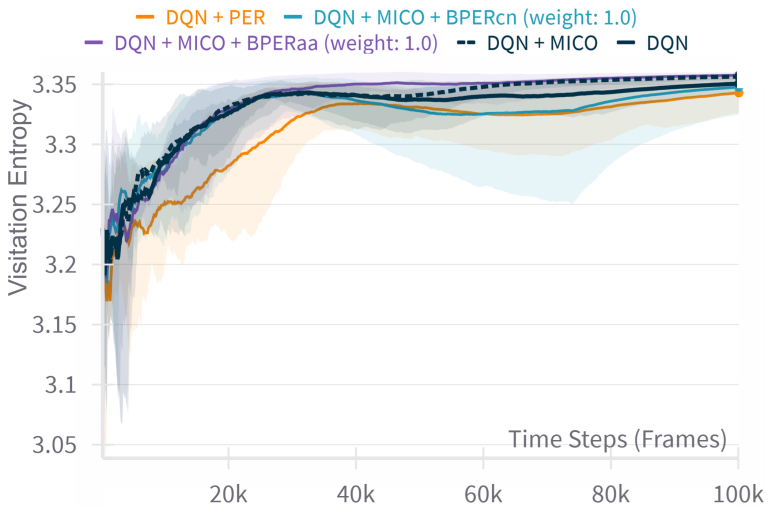


Figure 6.8: Visitation Entropy. Visitation entropy performance averaged over 5 independent runs over time, with shaded regions representing the variability for each method.

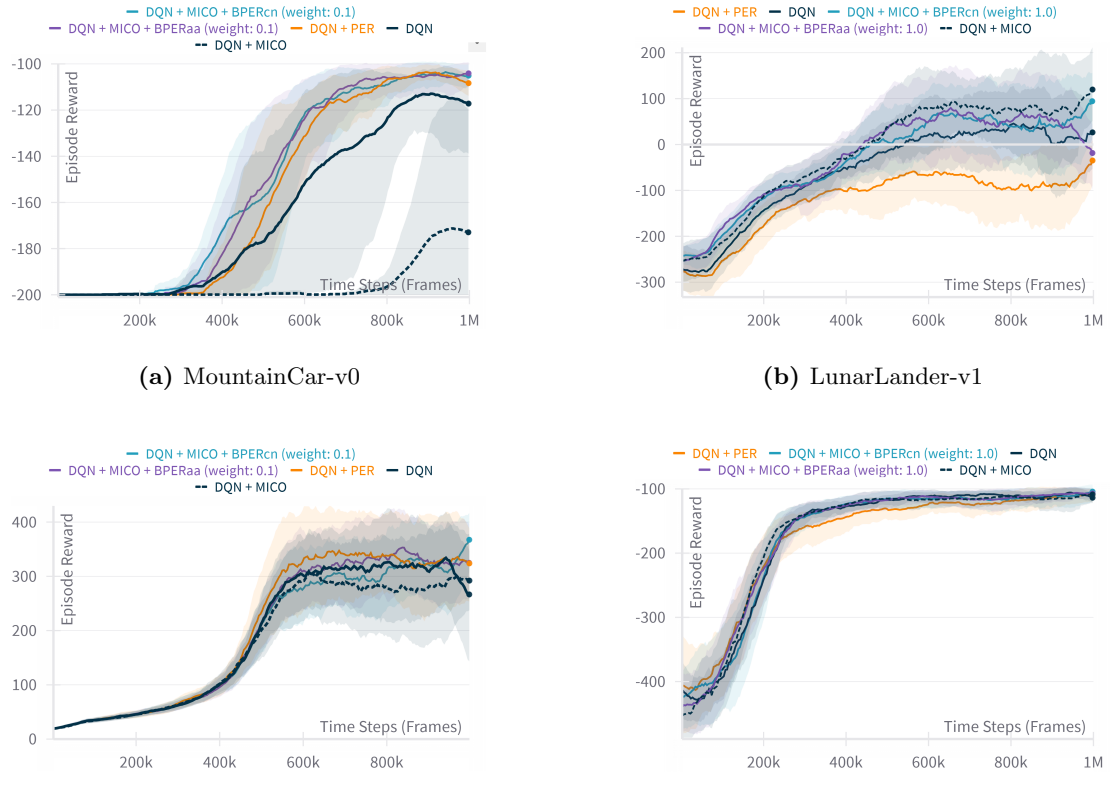
6.5 Classical Environments

Figure 6.9 illustrates the episode reward calculated over a moving average window of 100 episodes in four different environments: MountainCar-v0, LunarLander-v1, CartPole-v1, and Acrobot-v1, averaged over 5 independent executions. The hyperparameter priority weight μ was set using the best values found[2]. Please refer to Appendix A.7 for experiments with prioritization weights 1.0 for Mountain Car and Cart Pole. Additionally, refer to Appendix A.8 for validation episode reward results.

Figure 6.9a explores the episode reward in the Mountain Car environment, where the bisimulation techniques, BPERcn and BPERaa, outperforms the other methods, with considerable improvements over DQN + MICO baseline. It is important to note that the priority weight is set to 0.1, meaning that most of the prioritization is influenced by the TD-error. This suggests that the bisimulation distance may not be a useful indicator of expected learning progress in this scenario, where the action space and rewards are limited.

Figure 6.9b shows the episode reward in the Lunar Lander environment. BPERcn and BPERaa show initially the best performance over all the methods. However, over

[2]An exhaustive grid search was not conducted for the priority weight μ. Instead, the values were determined using a trial-and-error approach, where it was observed that when bisimulation is less generally useful due to the environment simplicity, it is better to set smaller values.



(a) MountainCar-v0

(b) LunarLander-v1

(c) CartPole-v1

(d) Acrobot-v1

Figure 6.9: Episode Reward in Classical Environments. Episode reward performance over time, averaged with a moving window of 100 episodes across 5 independent executions, with shaded regions representing the variability for each method. The values are calculated in four different environments: (a) MountainCar-v0, (b) LunarLander-v1, (c) CartPole-v1, and (d) Acrobot-v1.

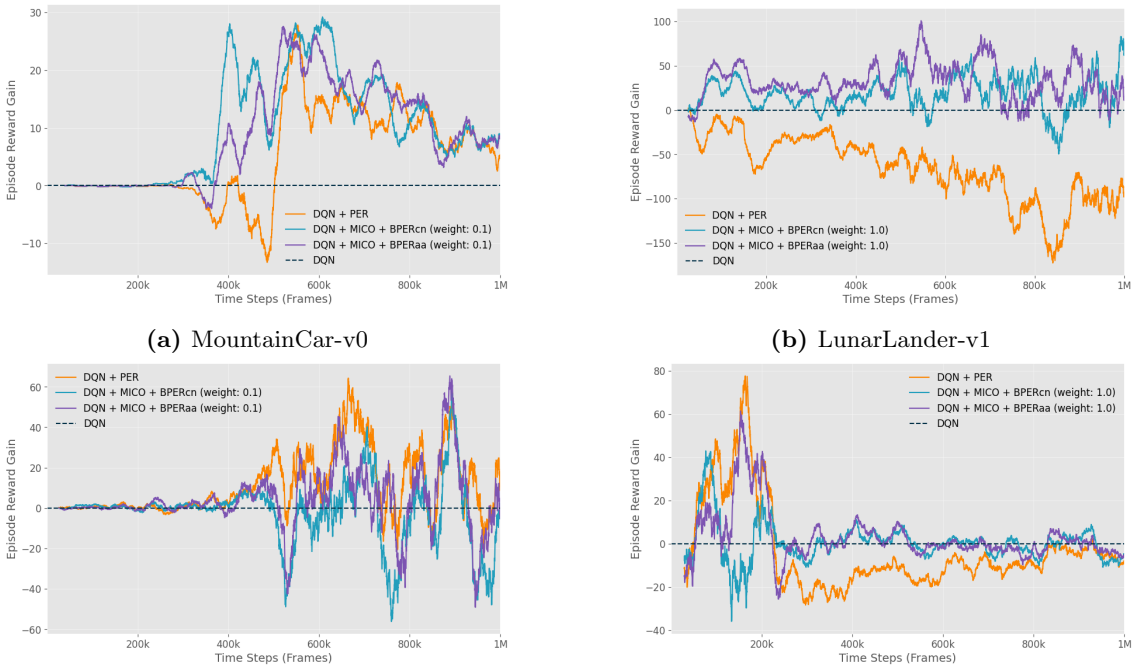
time, their performance decreases affecting even the learning of the DQN + MICO baseline. Although both strategies outperforms the DQN and DQN + PER methods, the observed benefits appear to derive more from the MICo learning than from the proposed prioritization strategies [Disclaimer[3]].

Figure 6.9c shows the episode reward in the CartPole v1 environment. In this scenario, the PER variant is more consistent and outperforms the other methods in most cases. Although the BPERcn and BPERaa alternatives achieve improvements over the direct baseline DQN + MICo, they do not surpass the performance of the DQN baseline.

[3]During our observations, BPERaa and BPERcn typically outperforms the direct baseline DQN + MICo, even when DQN + MICo does not perform better than DQN. However, in this specific instance, a different behavior was observed. Upon revisiting the code and hyperparameter settings, we found a typo where the bisimulation priorities were incorrectly normalized using a logarithmic scale only for this experiment, which could explain the observed issue. These results are provided for reference, but future work should aim to redo this experiment to ensure accuracy.

Particularly, DQN + MICo performs worse than DQN, suggesting that in this environment, MICo learning does not provide a clear advantage. This may be due to the difficulty in distinguishing between behaviorally similar or dissimilar states, given the simple reward function (a reward of 1 for each time step) and the limited action space (two actions: left and right). Consequently, it is expected that the BPERcn and BPERaa strategies underperform compared to the other methods.

Figure 6.9d shows the episode reward in the Acrobot-v1 environment. In this scenario, BPERcn and BPERaa methods initially demonstrate better performance than the other methods, and they eventually converge to a similar behavior as DQN and DQN + MICo. In contrast, the PER method shows the worst performance over time; even worse than the DQN baseline.



(a) MountainCar-v0 **(b)** LunarLander-v1

(c) CartPole-v1 **(d)** Acrobot-v1

Figure 6.10: Episode Reward Gain in Classical Environments. Episode reward gain performance calculated against the DQN baseline over time, averaged with a moving window of 100 episodes across 5 independent executions. The values are calculated in four different environments: (a) MountainCar-v0, (b) LunarLander-v1, (c) CartPole-v1, and (d) Acrobot-v1.

Figure 6.10 illustrates the episode reward gain relative to the DQN baseline for the three prioritization extensions: PER, BPERcn, and BPERaa; measured using a moving average window of 100 episodes. Mountain Car, Lunar Lander, and Acrobot-v1 results show that the bisimulation strategies, BPERcn, and BPERaa, obtain mostly positive episode reward gains over time compare to PER alternatives, confirming the findings mentioned earlier. Notice, in Figure 6.10d, the slight improvements of the bisimulation strategies over the baseline DQN are more clearly observed, which were not evident in the

episode reward plots. However, the CartPole-v1 results reveal a highly variable episode reward gain for the BPERcn and BPERaa methods, with predominantly positive values for the PER method over time. Please refer to Appendix A.9 for the evaluation relative to the DQN + MICo baseline.

Table 6.1 summarizes the episode reward gains for each method and baseline, presenting the mean and standard deviation over all time steps for each environment. The BPERaa strategy shows the best performance on both the DQN and DQN + MICo baselines in the Acrobot and LunarLander environments, while BPERcn performs best on Mountain Car, and the PER method is most effective on CartPole-v1. It is important to note that the results are highly variable, especially in more complex environments like LunarLander, where the standard deviations reach 185.015 and 191.315 for the DQN and DQN + MICo baselines, respectively.

	Baseline	MountainCar-v0	Cartpole-v1	Acrobot-v1	LunarLander-v1
PER	**DQN**	5.344 ± 21.016	$\mathbf{10.975 \pm 133.140}$	-3.279 ± 79.455	-65.620 ± 175.613
	MICO	39.088 ± 39.151	$\mathbf{24.23 \pm 135.122}$	-6.878 ± 74.916	-107.075 ± 183.781
BPERcn	**DQN**	$\mathbf{10.145 \pm 21.471}$	-2.100 ± 134.322	0.250 ± 78.057	18.800 ± 191.309
	MICO	$\mathbf{43.889 \pm 38.863}$	11.648 ± 129.970	-3.350 ± 73.122	-22.654 ± 194.957
BPERaa	**DQN**	9.021 ± 19.756	3.790 ± 134.959	$\mathbf{2.915 \pm 75.618}$	$\mathbf{32.125 \pm 185.015}$
	MICO	42.765 ± 39.060	17.539 ± 134.593	$\mathbf{-0.685 \pm 70.853}$	$\mathbf{-9.330 \pm 191.315}$

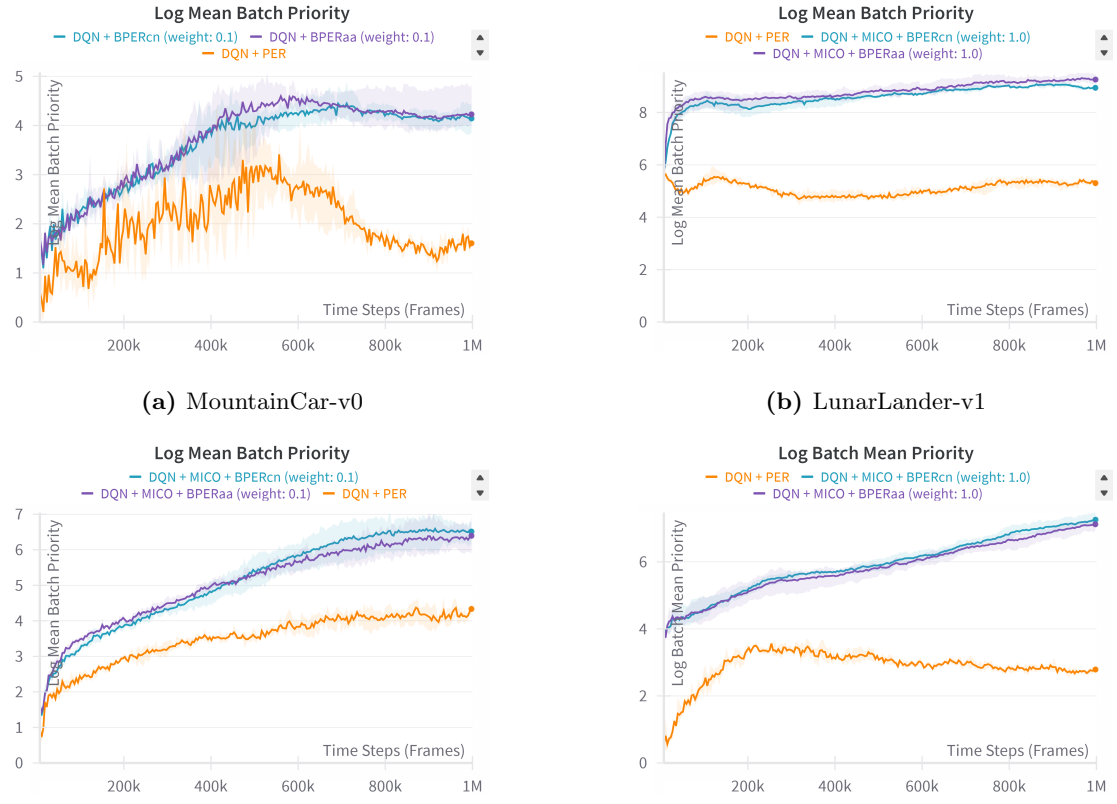
Table 6.1: Episode Reward Gain Comparison of Different DQN Variants. The table summarizes the episode reward gains for each method and baseline, presenting the mean and standard deviation over all time steps for each environment. The best values per method and baseline are highlighted in bold.

6.5.1 Mean Batch Priority

Figure 6.11 illustrates the log mean batch priority per method over time, averaged over 5 independent executions. A logarithmic scale was used to compare the methods because bisimulation distances and TD-errors can vary by several orders of magnitude, and both can reach large values over time. In all environments, the BPERcn and BPERaa methods consistently exhibit higher average priority values compared to the PER method.

These results can be interpreted in two ways: (1) A high average priority value might indicate efficient prioritization, where the sampled minibatches contain experiences with high priority values, effectively increasing the average priority over time; (2) A high average priority value could also suggest a saturated scenario, where only a few experiences are highly prioritized, potentially reducing diversity in the data. According to the previous results for episode reward in Figure 6.9, both interpretations seem relevant depending on the environment. For example, in the LunarLander environment, although there is an initial increase in episode reward using BPER strategies, performance eventually declines to levels similar to DQN + MICo, suggesting that the algorithm is likely falling into the second scenario. Conversely, results from Mountain Car, Acrobot, and CartPole indicate

sustained performance improvements over the DQN + MICo baseline, aligning with the first scenario. Notice that this improvement over DQN + MICo is observed even with a priority weight of 1.0 (refer to Appendix **??**), where, although MICo learning does not enhance DQN performance, the BPERaa and BPERcn methods still outperform DQN + MICo.



(a) MountainCar-v0

(b) LunarLander-v1

(c) CartPole-v1

(d) Acrobot-v1

Figure 6.11: Log Mean Batch Priority. Priority calculated for each mini-batch sampled, averaged across 5 independent executions. The results are presented on a logarithmic scale for improved visualization.

6.6 Summary Discussion

The propose method yield superior performance on the 31-state GridWorld and promising results in classical environments. On the one hand, the GridWorld experiments help to empirically confirm our proof of concept by demonstrating the effectiveness and benefits of our proposed bisimulation strategies, BPERcn and BPERaa, in addressing three key problems: task-agnostic sampling, outdated priorities, and state space coverage.

- The **episode reward** and **task-agnostic sampling** results suggests that, although

the TD-error serves as an indicator of potential improvement in the Q-values, it is insufficient as a proxy for expected learning progress in our GridWorld. In contrast, a notion of behavioral similarity proves to be a better indicator of expected learning progress.

- The **outdated priorities** results suggests that our proposal effectively alleviates the problem of outdated priorities by reducing the distance between the ideal and sampling probability distributions. Although the priorities are updated only in the current minibatch, bisimulation distances provide a more stable priority over time compared to TD-errors.

- The **state space coverage** results indicates that bisimulation strategies can encourage more exploration in the environment without explicitly instructing the policy to explore more, as is done with ϵ-policies and other exploration mechanism [4, 27].

- Additionally, in most experiments, the **BPERaa strategy outperforms BPERcn** by demonstrating a higher episode reward, greater variability in priority distributions, higher entropy for exploration, and fewer issues with priority assignment upon visual inspection.

On the other hand, the performance in classical environments is only slightly better and does not show consistent improvements compare to the DQN. However, what remains consistent across experiments is that the bisimulation strategies outperform the direct baseline DQN + MICo, even if they do not surpass DQN or DQN + PER. This result suggest the following promising directions:

- The cases where our method does not outperform DQN or DQN + PER (e.g., CartPole and Mountain Car with a priority weight of 1.0) appear to be related to **scenarios where DQN + MICo is not efficient**. When the MICo learning itself is ineffective, it is expected that our approach would also yield suboptimal results.

- The effectiveness of MICo, and, consequently, **our method is connected to the structure of the evaluated environments**. The MICO operator, as defined in Definition 5, utilizes both the reward difference and the independent coupling calculated over transitions. Consequently, the effectiveness of MICO learning is inherently dependent on the structure of the environment, particularly the reward structure and the transitions determined by the actions taken. In simpler classical environments, the effective state space is smaller with few transitions and the same rewards per time step, making it more challenging to identify behaviorally similar states for MICo learning. In fact, results in simple Atari environments (e.g., Pong) from MICO reported by Castro et al. [10] do not show evident improvements.

- The **mean batch priority is a viable indicator** for analyzing the level of prioritization in each environment, showing that our proposal consistently encourage large priority values within the current mini batch. However, the results have different

interpretations, which must be interpreted alongside the episode reward results to determine whether a high mean batch priority is having a positive impact or not.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

This work present a non-uniform sampling technique to sample transitions from an experience replay, named Bisimulation Prioritized Experience Replay (BPER) along with two strategies: current-vs-next (BPERcn) and all-vs-all (BPERaa). The priorities are assigned based on a surrogate on-policy bisimulation, called MICo metric, that is used to encourage sampling transitions with more behavioral dissimilar states (promoting data diversity). The MICo metric is estimated as part of the learning of state abstractions. The proposed technique demonstrates strong performance in a GridWorld environment used to validate the effectiveness of the method, and promising results in classical environments.

In the GridWorld experiments, the results indicate that the BPERcn and BPERaa strategies effectively address three key sampling challenges: task-agnostic sampling, outdated priorities, and state space coverage. Both visual and quantitative analyses show that our approach outperforms all other evaluated methods (DQN, DQN + PER, and DQN + MICO) in each of these areas, demonstrating the viability of our method.

In slightly more complex environment, however, the proposed approach is partially effective, consistently outperforming the direct baseline DQN + MICo, but outperforming DQN and DQN + PER only in special cases.

The method offers a potentially valuable approach for future applications, but its successful implementation will require careful consideration of certain factors. These include the structure of the environment, such as the reward function, the action space, and the transitions. Further research is needed to explore these aspects in depth and to refine the method for broader applicability.

7.2 Future Work

We present the following future directions for the current work:

- **Complex Environment to Evaluate.** The impact of MICO learning (inherently influenced by the structure of the environment as defineds in Definition 5) becomes more evident in complex environments with a greater variety of actions and rewards. In this project, we tested the method only on relatively simple environments with limited actions and rewards, which made it complicated to fully discern its effectiveness. We recommend to explore more complex environments in future works.

- **Heavily Skewed Sampling Distributions.** The bisimulation values are significantly higher compared to the TD-errors (see Figure 6.11). Although high priority values in a minibatch can be associated with positive effects, consistently assigning excessively large values is not ideal, as it may lead to uneven and heavily skewed probability distributions that disproportionately weight certain experiences over others. A potential solution to mitigate this issue could involve using normalization techniques[1] or clipping techniques before assigning priorities. Additionally, alternative sampling methods, such as temperature-based sampling (commonly used in large language models [24]) or rank-based sampling [44], could be explored.

- **Lack of Theoretical Guarantees.** This work provides an empirical evaluation of the proposed method; however, it does not include a theoretical proof that guarantees the algorithm's convergence. Future work should explore developing a formal proof to establish such guarantees. We suggest following approaches similar to those in Pan et al. [43], which provided a formal proof for the convergence of PER.

- **Prioritization Hyper-parameter Adjustments.** In the evaluated experiments, the best alpha α and beta β hyperparameters, which control the prioritization level (α-probability) and the weighted importance sampling for the PER [44] method, were used. However, although these hyperparameters are optimal for the PER method, they may not necessarily be the best for our approach. We recommend to conduct a hyperparameter sweep to find the adequate setting for our method.

- **Benchmarks and Algorithms.** Similar to MICO learning [10], our method can be integrated into any value-based agent, adapted to different algorithms that employs an experience replay as part of the learning, such as Double DQN [50], DDPG [33], and SAC [21]. Additionally, in this work, we have only tested our approach in classical environments without considering other relevant benchmarks, such as Atari [8, 38] or MuJoCo [49] bechmarks. Future work should explore the performance of our method with these state-of-the-art algorithms and benchmarks.

[1] We conducted experiments using a logarithmic and min-max scaling not yielding satisfactory results.

Bibliography

[1] A. Abate, M. Giacobbe, and Y. Schnitzer. Bisimulation learning. *arXiv preprint arXiv:2405.15723*, 2024. 13

[2] D. Abel. A theory of abstraction in reinforcement learning. *arXiv preprint arXiv:2203.00397*, 2022. 21

[3] J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018. 8

[4] S. Amin, M. Gomrokchi, H. Satija, H. Van Hoof, and D. Precup. A survey of exploration methods in reinforcement learning. *arXiv preprint arXiv:2109.00157*, 2021. 50

[5] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm. Unsupervised state representation learning in atari. *Advances in neural information processing systems*, 32, 2019. 22

[6] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017. 18

[7] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT press, 2008. 13

[8] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. 53

[9] P. S. Castro. Scalable methods for computing state similarity in deterministic markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10069–10076, 2020. 2, 14, 15, 16, 27, 32, 35

[10] P. S. Castro, T. Kastner, P. Panangaden, and M. Rowland. Mico: Improved representations via sampling-based state similarity for markov decision processes. *Advances*

in Neural Information Processing Systems, 34:30113–30126, 2021. 2, 3, 12, 16, 17, 21, 23, 29, 50, 53, 59

[11] P. S. Castro, T. Kastner, P. Panangaden, and M. Rowland. A kernel perspective on behavioural metrics for markov decision processes. *arXiv preprint arXiv:2310.19804*, 2023. 23

[12] Z. Chen, H. Li, and R. Wang. Attention loss adjusted prioritized experience replay. *arXiv preprint arXiv:2309.06684*, 2023. 2

[13] T. Dai, H. Liu, K. Arulkumaran, G. Ren, and A. A. Bharath. Diversity-based trajectory and goal selection with hindsight experience replay. In *PRICAI 2021: Trends in Artificial Intelligence: 18th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2021, Hanoi, Vietnam, November 8–12, 2021, Proceedings, Part III 18*, pages 32–45. Springer, 2021. 18, 19

[14] T. De Bruin, J. Kober, K. Tuyls, and R. Babuška. Experience selection in deep reinforcement learning for control. *Journal of Machine Learning Research*, 19(9):1–56, 2018. 18

[15] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pages 3061–3071. PMLR, 2020. 2, 18, 19, 32

[16] N. Ferns, P. Panangaden, and D. Precup. Metrics for finite markov decision processes. In *UAI*, volume 4, pages 162–169, 2004. 2, 14

[17] N. Ferns, P. Panangaden, and D. Precup. Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714, 2011. 2, 14

[18] N. Ferns and D. Precup. Bisimulation metrics are optimal value functions. In *UAI*, pages 210–219, 2014. 2, 14

[19] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003. 13

[20] D. Ha and J. Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018. 21

[21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. 1, 53

[22] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019. 21

[23] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 18, 19

[24] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd edition, 2024. Online manuscript released August 20, 2024. 53

[25] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019. 21

[26] A. Kumar, A. Gupta, and S. Levine. Discor: Corrective feedback in reinforcement learning via distribution correction. *Advances in Neural Information Processing Systems*, 33:18560–18572, 2020. 19

[27] P. Ladosz, L. Weng, M. Kim, and H. Oh. Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85:1–22, 2022. 50

[28] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253, 2017. 21

[29] M. Laskin, A. Srinivas, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International conference on machine learning*, pages 5639–5650. PMLR, 2020. 21, 22

[30] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *International Conference on Machine Learning*, pages 6131–6141. PMLR, 2021. 20

[31] S. Y. Lee, C. Sungik, and S.-Y. Chung. Sample-efficient deep reinforcement learning via episodic backward update. *Advances in neural information processing systems*, 32, 2019. 19

[32] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. *AI&M*, 1(2):3, 2006. 13

[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 1, 53

[34] T. Lindvall. *Lectures on the coupling method*. Courier Corporation, 2002. 15

[35] J. Liu, Y. Ma, J. Hao, Y. Hu, Y. Zheng, T. Lv, and C. Fan. Prioritized trajectory replay: A replay memory for data-driven reinforcement learning. *arXiv preprint arXiv:2306.15503*, 2023. 18, 19

[36] X.-H. Liu, Z. Xue, J. Pang, S. Jiang, F. Xu, and Y. Yu. Regret minimization experience replay in off-policy reinforcement learning. *Advances in Neural Information Processing Systems*, 34:17604–17615, 2021. 19, 20

[37] S. Łukaszyk. A new concept of probability metric and its applications in approximation of scattered data sets. *Computational mechanics*, 33:299–304, 2004. 17

[38] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. 53

[39] A. R. Mahmood, H. P. Van Hasselt, and R. S. Sutton. Weighted importance sampling for off-policy learning with linear function approximation. *Advances in neural information processing systems*, 27, 2014. 11

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 1, 8, 10, 37

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 8, 10

[42] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 22

[43] Y. Pan, J. Mei, A.-m. Farahmand, M. White, H. Yao, M. Rohani, and J. Luo. Understanding and mitigating the limitations of prioritized experience replay. In *Uncertainty in Artificial Intelligence*, pages 1561–1571. PMLR, 2022. 2, 20, 32, 35, 36, 43, 53, 61

[44] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. 1, 2, 11, 12, 18, 32, 53

[45] S. Sinha, J. Song, A. Garg, and S. Ermon. Experience replay with likelihood-free importance weights. In *Learning for Dynamics and Control Conference*, pages 110–123. PMLR, 2022. 20

[46] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988. 15, 18, 19

[47] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 1, 2, 5, 9, 15, 18, 19, 26

[48] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. 21

[49] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. 53

[50] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016. 53

[51] C. Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009. 15

[52] D. Zha, K.-H. Lai, K. Zhou, and X. Hu. Experience replay optimization. *arXiv preprint arXiv:1906.08387*, 2019. 20

[53] A. Zhang, R. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020. 2, 3, 16, 22, 23, 29

[54] R. Zhao and V. Tresp. Curiosity-driven experience prioritization via density estimation. *arXiv preprint arXiv:1902.08039*, 2019. 20

APPENDIX A

Appendices

A.1 Metrics

The following information was retrieved from the Supplementary Material provide by Castro et al. [10].

A **metric** d on a set X is a function $d : X \times X \to [0, \infty)$ respecting the following axioms for any $x, y, z \in X$:

1. **Identity of indiscernibles:** $d(x, y) = 0 \iff x = y$;
2. **Symmetry:** $d(x, y) = d(y, x)$;
3. **Triangle inequality:** $d(x, y) \leq d(x, z) + d(z, y)$.

A **pseudometric** is similar, but the "identity of indiscernibles" axiom is weakened:

1. $x = y \implies d(x, y) = 0$;
2. $d(x, y) = d(y, x)$;
3. $d(x, y) \leq d(x, z) + d(z, y)$.

Note that the weakened first condition *does* allow one to have $d(x, y) = 0$ when $x \neq y$.

A **(pseudo)metric space** (X, d) is defined as a set X together with a (pseudo)metric d defined on X.

A **diffuse metric** is similar to as pseudometri, but the 'identity of indiscernibles' axiom is weakened even more:

1. $d(x, y) \geq 0$ for any $x, y \in X$;

2. $d(x, y) = d(y, x)$;

3. $d(x, y) \leq d(x, z) + d(z, y)$.

Note that the weakened first condition *does* allow one to have self-distance greater than zero $d(x, x) > 0$.

A.2 Algorithm: DQN with Matching under Independent Couplings (MICo)

Algorithm 4 DQN with Matching under Independent Couplings (MICo)

Input: minibatch k, step-size η, replay period K and size N, budget T (total steps).
Initialize action-value function Q with random weights ξ, ω
Initialize target action-value function Q^- with weights $\{\bar{\xi}, \bar{\omega}\} \leftarrow \{\xi, \omega\}$
Initialize replay memory $\mathcal{D} = \emptyset$ with capacity N
for $t = 1$ to T **do**
    Observe s_t
    Choose action $a_t \sim \pi^{\epsilon}_{\theta}(s_t)$
    Execute action a_t and observe r_t and s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    if $t \equiv 0 \mod K$ **then**
        Sample minibatch $B \sim \text{Uniform}(\mathcal{D})$ of transitions e_j
        Set $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q^-(s_{j+1}, a'; \{\bar{\xi}, \bar{\omega}\}) & \text{otherwise} \end{cases}$
        Compute TD-error $\delta_j = y_j - Q(s_j, a_j; \{\xi, \omega\})$, and loss $\mathcal{L}_{\text{TD}} = \delta_j^2$
        Squarify minibatch B to get transition pairs $\langle x, r_x, x' \rangle, \langle y, r_y, y' \rangle$
        Compute MICo loss $\mathcal{L}_{\text{MICo}} = \left(T^U_{\bar{\omega}}(r_x, x', r_y, y') - U_{\omega}(x, y)\right)^2$
        Compute total loss $\mathcal{L}_{\alpha}(\xi, \omega) = (1 - \alpha)\mathcal{L}_{\text{TD}}(\xi, \omega) + \alpha\mathcal{L}_{\text{MICo}}(\omega)$
        Perform a gradient descent step on $\mathcal{L}_{\alpha}(\xi, \omega)$
        Every C optimizing steps update $\{\bar{\xi}, \bar{\omega}\} \leftarrow \{\xi, \omega\}$
    end if
end for

A.3 Distance between Priority Sampling Distributions

The following explanation is based on the Supplementary Materials from Pan et al. [43]. The distance between the ideal priority distribution $p_i^*(\cdot)$ and the sampling priority distribution $p_i^*(\cdot)$ is estimated by the following three steps:

1. In order to compute the ideal sampling distribution, we use the implicit discrete state space (not raw pixels) from the Grid World, and

   - In the DQN + PER experiment, we calculate the absolute TD error of each state (coordinates x, y) by using the true environment model and **the current learned $Q_{\{\xi,\omega\}}$ network**.
   - In the DQN + MICO + PER experiment, we calculate the mico distance of each state (coordinates x, y) by using the true environement model and **the current learned ϕ_ω encoder**

   We then normalize these priorities to get probability distribution p_i^* with $i = 1$ for DQN + PER and $i = 2$ for DQN + MICO + PER. Note that this distribution is considered as the desired one since we have access to all states across the state space with priorities computed by current network at each time step.

2. The sampling distribution is estimated by randomly sampling 3k states and count the number of states and normalize these counts to get p_i, with $i = 1$ for DQN + PER and $i = 2$ for DQN + MICO + PER.

3. Finally, the distances of p_1, p_2 to p_1^*, p_2^* by two weighting schemes:

   (a) **on-policy weighting**:

   $$\sum_{j=1}^{2500} d^\pi(s_j)|p_i(s_j) - p_i^*(s_j)|, \quad i \in \{1, 2\},$$

   where d^π is approximated by uniformly sampling 3k states from a recent buffer and normalizing their visitation counts on the GridWorld;

   (b) **uniform weighting**:

   $$\frac{1}{2500} \sum_{j=1}^{2500} |p_i(s_j) - p_i^*(s_j)|, \quad i \in \{1, 2\}.$$

Two weighting schemes were employed for two different purposes: **on-policy weighting** focuses on the asymptotic convergence behavior, thereby reducing the weight of states with relatively high TD errors that are rarely visited as the policy approaches optimality. In contrast, **uniform weighting** is more appropriate during the initial learning phase, where all states are considered equally important, encouraging the agents to thoroughly explore the entire state space.

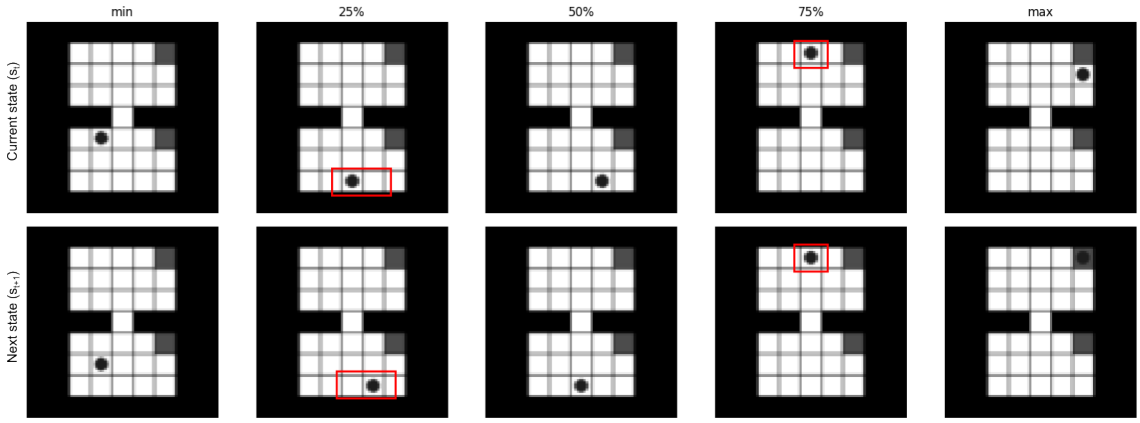
A.4 Hyperparameters Setting

	Hyperparameter	DQN	DQN PER	DQN MICo	DQN BPER
Collector	Total Budget	100,000	100,000	100,000	100,000
	Replay Period	128	128	128	128
	Eps Start	0.1	0.1	0.1	0.1
	Eps End	0.005	0.005	0.005	0.005
	Eps Annealing Frames	520,000	520,000	520,000	520,000
	Init Random Frames	200	200	200	200
	Frame Stack	1	1	1	1
	Frame Skip	1	1	1	1
Policy	CNN Net (Cells)	[32, 64, 64]	[32, 64, 64]	[32, 64, 64]	[32, 64, 64]
	Kernel Sizes	[8, 4, 3]	[8, 4, 3]	[8, 4, 3]	[8, 4, 3]
	Strides	[4, 2, 1]	[4, 2, 1]	[4, 2, 1]	[4, 2, 1]
	MLP Net (Cells)	[64, 64]	[64, 64]	[64, 64]	[64, 64]
	Activation	ReLU	ReLU	ReLU	ReLU
Buffer	Buffer Size	100,000	100,000	100,000	100,000
	Batch Size	256	256	256	256
	Alpha	–	0.6	–	0.6
	Beta	–	0.4	–	0.4
	Priority Weight	–	–	1.0	1.0
Optim	Learning Rate	0.0015	0.0015	0.0015	0.0015
	Max Grad Norm	10	10	10	10
	Weight Decay	0.00001	0.00001	0.00001	0.00001
	Eps	1.5e-4	1.5e-4	1.5e-4	1.5e-4
Loss	Gamma	0.99	0.99	0.99	0.99
	MICO Weight	–	–	0.01	0.01
	MICO Beta	–	–	0.1	0.1
	MICO Gamma	–	–	0.99	0.99
	Hard Update Freq	50	50	50	50
	Num Updates	1	1	1	1

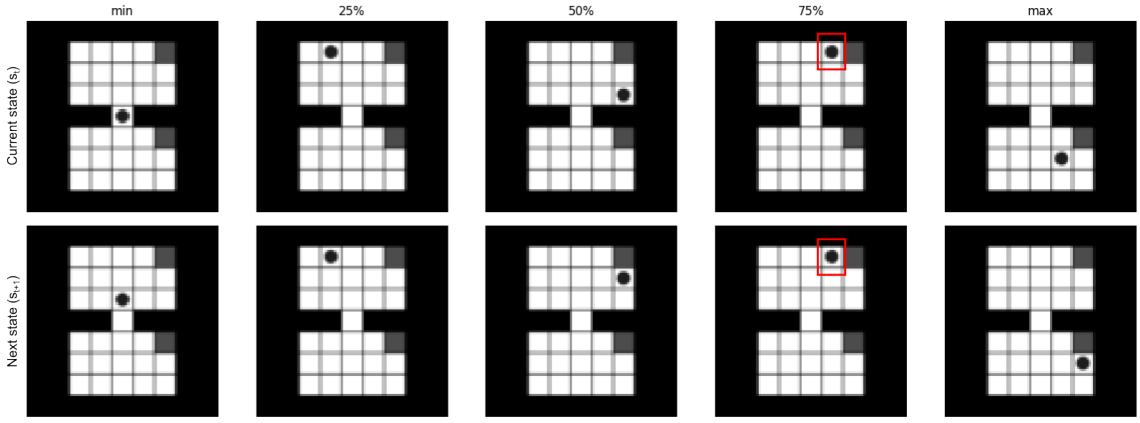
Table A.1: Hyperparameter Configurations for Grid World. This configuration was consistently used across different independent runs. The settings for DQN BPER are applicable to both BPERcn and BPERaa.

	Hyperparameter	DQN	DQN PER	DQN MICo	DQN BPER
Collector	Total Budget	1,000,064	1,000,064	1,000,064	1,000,064
	Replay Period	128	128	128	128
	Eps Start	0.5	0.5	0.5	0.5
	Eps End	0.005	0.005	0.005	0.005
	Eps Annealing Frames	520,000	520,000	520,000	520,000
	Init Random Frames	20,000	20,000	20,000	20,000
	Frame Stack	4	4	4	4
	Frame Skip	4	4	4	4
Policy	CNN Net (Cells)	[32, 64, 64]	[32, 64, 64]	[32, 64, 64]	[32, 64, 64]
	Kernel Sizes	[8, 4, 3]	[8, 4, 3]	[8, 4, 3]	[8, 4, 3]
	Strides	[4, 2, 1]	[4, 2, 1]	[4, 2, 1]	[4, 2, 1]
	MLP Net (Cells)	[64, 64]	[64, 64]	[64, 64]	[64, 64]
	Activation	ReLU	ReLU	ReLU	ReLU
Buffer	Buffer Size	100,000	100,000	100,000	100,000
	Batch Size	256	256	256	256
	Alpha	–	0.6	–	0.6
	Beta	–	0.4	–	0.4
	Priority Weight	–	–	1.0	1.0
Optim	Learning Rate	0.0015	0.0015	0.0015	0.0015
	Max Grad Norm	10	10	10	10
	Weight Decay	0.00001	0.00001	0.00001	0.00001
	Eps	1.5e-4	1.5e-4	1.5e-4	1.5e-4
Loss	Gamma	0.99	0.99	0.99	0.99
	MICO Weight	–	–	0.01	0.01
	MICO Beta	–	–	0.1	0.1
	MICO Gamma	–	–	0.99	0.99
	Hard Update Freq	50	50	50	50
	Num Updates	1	1	1	1

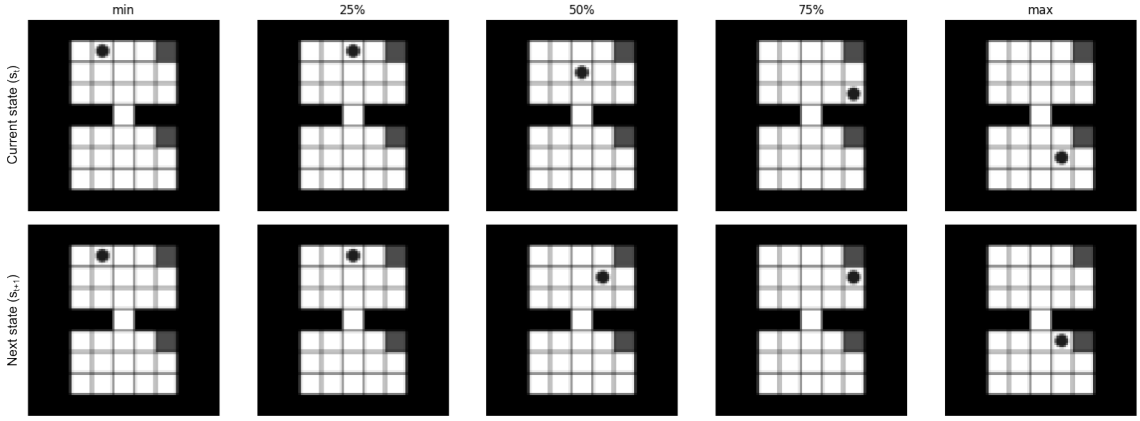
Table A.2: Hyperparameter Configurations for Other Environments. This configuration was consistently applied across the MountainCar, CartPole, Acrobot, and LunarLander environments for each independent run. The settings for DQN BPER are applicable to both BPERcn and BPERaa. The priority weight configuration was adjusted as described in the results.



(a) DQN + PER



(b) DQN (MICO) + BPERcn

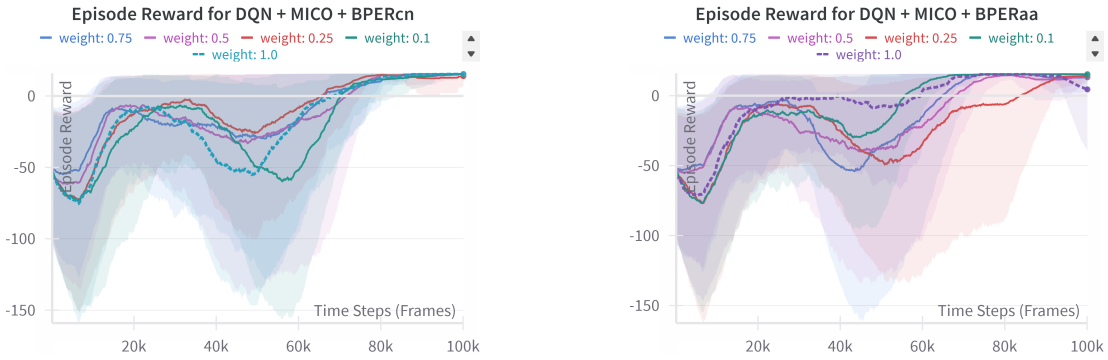


(c) DQN (MICO) + BPERaa

Figure A.1: Visual Inspection at the 50k time step. Frames corresponding to the main quartiles (min, 25%, 50%, 75% and max) of priority values in the experience replay collected at the 50k time step, highlighting different problematic transitions with red colour. The top row shows the current states, and the bottom row shows the next states for each transition.

A.5 Visual Inspection 50k Time Step

A.6 Priority Weight Sweep Results

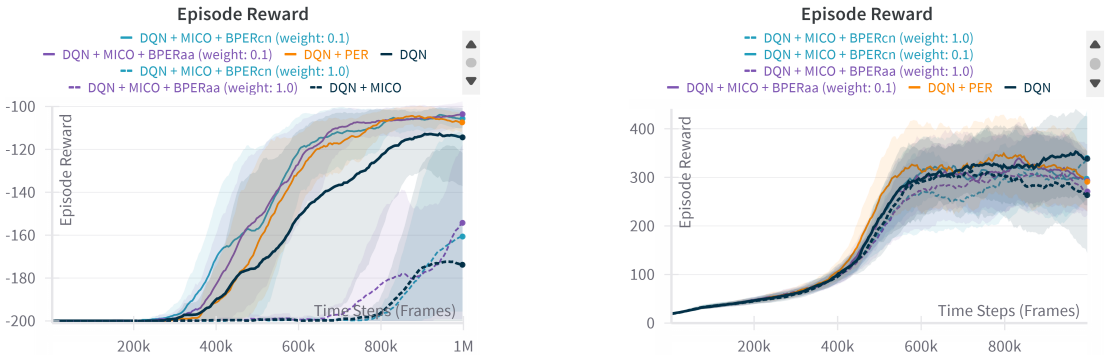


(a) Sweep BPERcn

(b) Sweep BPERcn

Figure A.2: Priority Weight Sweep in Grid World. Episode reward sweep over priority weights with values 0.1, 0.25, 0.5, 0.75, and 1.0, averaged with a moving window of 100 episodes across 5 independent executions over time. Shaded regions represent the variability for each method. The values are calculated for (a) the current-next strategy (BPERcn) and (b) the all-vs-all strategy (BPERaa).

A.7 Mountain Car and CartPole with Priority Weight 1.0

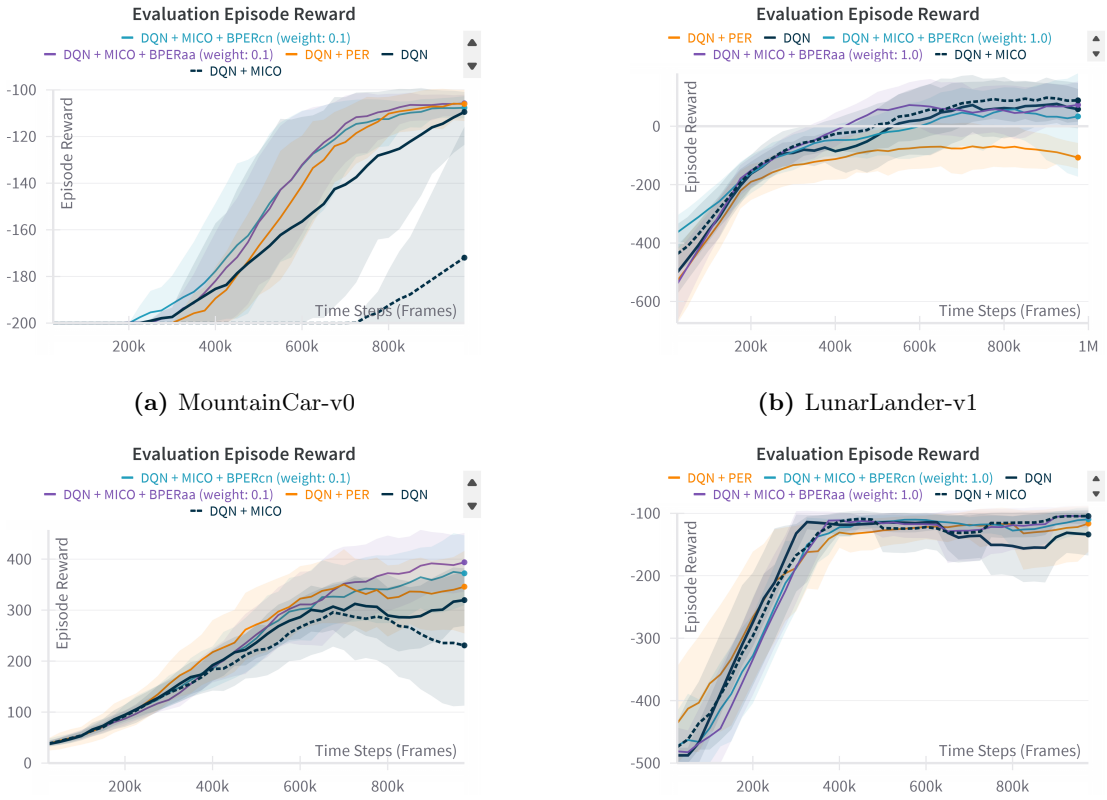


(a) Mountain Car

(b) CartPole

Figure A.3: Episode Reward in Classical Environments using Priority Weight 1.0 Episode reward performance over time, averaged with a moving window of 100 episodes across 5 independent executions, with shaded regions representing the variability for each method. The values are calculated in four different environments: (a) MountainCar-v0, (b) LunarLander-v1, (c) CartPole-v1, and (d) Acrobot-v1.

A.8 Validation Episode Reward



(a) MountainCar-v0

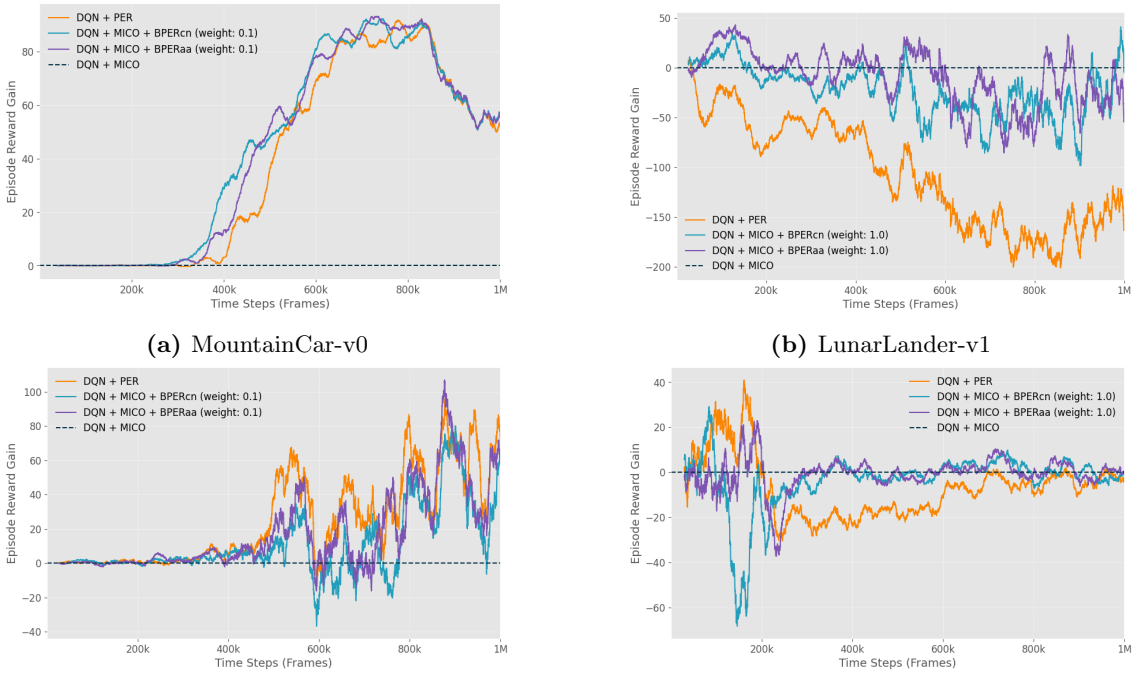
(b) LunarLander-v1

(c) CartPole-v1

(d) Acrobot-v1

Figure A.4: Validation Episode Reward in Classical Environments. Validation episode reward performance over time, averaged with a moving window of 100 episodes across 5 independent executions, with shaded regions representing the variability for each method. The values are calculated in four different environments: (a) MountainCar-v0, (b) LunarLander-v1, (c) CartPole-v1, and (d) Acrobot-v1.

A.9 Episode Reward Gain baseline DQN + MICO



(a) MountainCar-v0

(b) LunarLander-v1

(c) CartPole-v1

(d) Acrobot-v1

Figure A.5: Episode Reward Gain in Classical Environments against DQN MICO baseline. Episode reward gain performance calculated against the DQN + MICO baseline over time, averaged with a moving window of 100 episodes across 5 independent executions. The values are calculated in four different environments: (a) MountainCar-v0, (b) LunarLander-v1, (c) CartPole-v1, and (d) Acrobot-v1.