Gradient Iterated Temporal-Difference Learning

Théo Vincent, Kevin Gerhardt, Yogesh Tripathi, Habib Maraqten, Adam White, Martha White, Jan Peters, Carlo D'Eramo

Keywords: temporal-difference learning, Bellman error, gradient temporal-difference learning.

Summary

Temporal-difference (TD) learning is highly effective at controlling and evaluating an agent's long-term outcomes. Most approaches in this paradigm implement a semi-gradient update to boost the learning speed, which consists of ignoring the gradient of the bootstrapped estimate. While popular, this type of update is prone to divergence, as Baird's counterexample illustrates. Gradient TD methods were introduced to overcome this issue, but have not been widely used, potentially due to issues with learning speed compared to semi-gradient methods. Recently, iterated TD learning was developed to increase the learning speed of TD methods. For that, it learns a sequence of action-value functions in parallel, where each function is optimized to represent the application of the Bellman operator over the previous function in the sequence. While promising, this algorithm can be unstable due to its semi-gradient nature, as each function tracks a moving target. In this work, we modify iterated TD learning by computing the gradients over those moving targets, aiming to build a powerful gradient TD method that competes with semi-gradient methods. Our evaluation reveals that this algorithm, called *Gradient Iterated Temporal-Difference* learning, has a competitive learning speed against semi-gradient methods across various benchmarks, including Atari games, a result that no prior work on gradient TD methods has demonstrated.

https://github.com/theovincent/Gi-DQN

Contribution(s)

1. We introduce *Gradient Iterated Temporal-Difference* learning, a new gradient temporal-difference learning algorithm, which is designed to learn a sequence of action-value functions in parallel, where each function is optimized to represent the application of the Bellman operator over the previous function in the sequence (Section 5).
   Context: Schmitt et al. (2022) first introduced the idea of learning a sequence of action-value functions where each function represents the application of the Bellman operator over the previous function. They suggest performing updates sequentially, starting with the first function in the sequence. However, this method is not scalable as it relies on multiple sequential gradient steps. Then, Vincent et al. (2025b) suggested performing the updates in parallel, but rely on semi-gradient updates, which can cause instability because each function is optimized with respect to a moving target. Finally, Vincent et al. (2026) developed a parameter-efficient version of this idea, which still relies on semi-gradient updates. Our method leverages the gradient TD method published by Ghiassian et al. (2020).
2. We derive and evaluate three instances of the introduced algorithm to showcase that the presented method can be used in many different settings and combined with many deep reinforcement learning algorithms (Section 7).
   Context: We present combinations of the proposed algorithm with DQN, SAC, and CQL, along with an advanced architecture (IMPOOLA, Trumpp et al. (2025)), a 3-step return, and a prioritized replay buffer.
3. We demonstrate that the learning speed of the presented gradient TD method is competitive against semi-gradient methods across multiple benchmarks, including Atari games. In particular, no prior work has demonstrated that gradient TD methods are competitive in the Atari benchmark (Section 7.2).
   Context: The proposed approach is competitive against semi-gradient methods such as DQN, SAC, and CQL, as well as their iterated variants, across 10 Atari games and 6 MuJoCo environments. Ghiassian et al. (2020); Elelimy et al. (2025) evaluate their proposed gradient TD method on MinAtar (Young & Tian, 2019). While Elsayed et al. (2024), Vasan et al. (2024), and Jiang et al. (2022) report experiments on Atari games, these methods use eligibility traces or emphatic weightings, which rely on semi-gradient updates.

Gradient Iterated Temporal-Difference Learning

Théo Vincent[1,2,†], **Kevin Gerhardt**[1,2], **Yogesh Tripathi**[1,2], **Habib Maraqten**[1,2], **Adam White**[3,4,5], **Martha White**[3,4,5], **Jan Peters**[1,2,6,7], **Carlo D'Eramo**[8]

[1]**DFKI, SAIROL**, [2]**TU Darmstadt**, [3]**Alberta Machine Intelligence Institute (Amii)**, [4]**Department of Computing Science, University of Alberta, Canada**, [5]**CIFAR AI Chair**, [6]**Hessian.ai**, [7]**Robotics Institute Germany**, [8]**University of Würzburg**

[†] correspondence to theovincentjourdat@gmail.com

Abstract

Temporal-difference (TD) learning is highly effective at controlling and evaluating an agent's long-term outcomes. Most approaches in this paradigm implement a semi-gradient update to boost the learning speed, which consists of ignoring the gradient of the bootstrapped estimate. While popular, this type of update is prone to divergence, as Baird's counterexample illustrates. Gradient TD methods were introduced to overcome this issue, but have not been widely used, potentially due to issues with learning speed compared to semi-gradient methods. Recently, iterated TD learning was developed to increase the learning speed of TD methods. For that, it learns a sequence of action-value functions in parallel, where each function is optimized to represent the application of the Bellman operator over the previous function in the sequence. While promising, this algorithm can be unstable due to its semi-gradient nature, as each function tracks a moving target. In this work, we modify iterated TD learning by computing the gradients over those moving targets, aiming to build a powerful gradient TD method that competes with semi-gradient methods. Our evaluation reveals that this algorithm, called *Gradient Iterated Temporal-Difference* learning, has a competitive learning speed against semi-gradient methods across various benchmarks, including Atari games, a result that no prior work on gradient TD methods has demonstrated.

1 Introduction

Temporal-difference (TD, Sutton (1988)) learning is a useful learning paradigm that eliminates the need for long rollouts in the environment, thereby increasing the frequency with which the learning agent receives feedback. This paradigm is successful in solving complex tasks where sample efficiency is key (Mnih et al., 2015; Degrave et al., 2022; Wurman et al., 2022). The core idea behind temporal-difference learning is to reduce the error between the estimated outcome at the current timestep and the target, where the target is the received reward plus the expected discounted outcome at the next timestep.

Applying stochastic gradient descent on this error is a straightforward approach. Unfortunately, this idea is limited to deterministic environments because two independent samples, each starting at the same timestep, are required to compute an unbiased estimate of the target. This issue is known as the double sampling problem (Baird, 1995). For this reason, semi-gradient methods, like TD(0), only use the gradient at the current timestep and ignore the gradient of the target (Watkins & Dayan, 1992). Interestingly, despite known divergence issues on simple problems due to the absence of this gradient term (Baird, 1995), this approach is pursued by all state-of-the-art TD learning methods (Hessel et al., 2018; Vieillard et al., 2020; Lee et al., 2025; Palenicek et al., 2026).

Gradient TD algorithms were developed to address this divergence issue (Sutton et al., 2008). This line of work led to provably convergent TD algorithms, even with nonlinear function approximation (Maei et al., 2009; Dai et al., 2018; Patterson et al., 2022b). While this strong property should steer the field towards gradient TD methods, semi-gradient methods remain the preferred option.

Recently, iterated TD (i-TD) learning was introduced by Vincent et al. (2025b) with the objective of improving the learning speed of TD methods by learning a sequence of action-value functions in

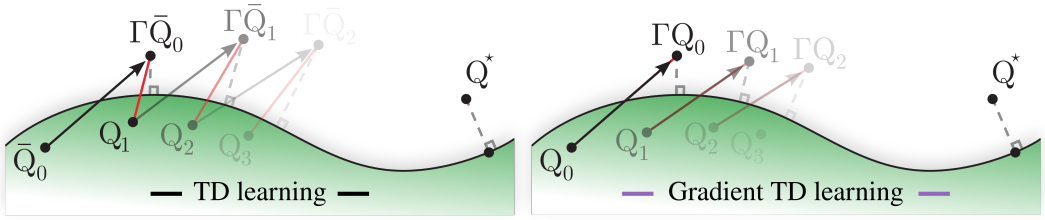

Figure 1: Representation of bootstrapping methods in the space of action-value functions. We represent the projection on the space of function approximation (in green) with dashed grey lines. **Left**: TD learning uses a target estimator $\bar{Q}_0$ to construct a regression target estimating the Bellman iteration $\Gamma\bar{Q}_0$, where Γ is the Bellman operator. This quantity is learned by the online estimator Q_1. At every T gradient steps, the target estimator is updated to the online estimator to learn the following Bellman iterations. **Right**: At every step k, Gradient TD learning minimizes the distance between Q_k and its regression target estimating the Bellman iteration ΓQ_k, leading to a new estimate Q_{k+1}.

parallel, where each function is optimized to represent the application of the Bellman operator over the previous function (see Figure 2, left). Yet, as each function tracks a moving target, the benefit of learning consecutive Bellman iterations in parallel can diminish when a function positioned at the beginning of the sequence changes faster than the following ones. This issue originates from the semi-gradient nature of this algorithm.

The objective is to develop a gradient TD method that is competitive in terms of learning speed[1] against semi-gradient methods by learning a sequence of action-value functions, where each function is optimized to represent the application of the Bellman operator over the previous one. To that end, we introduce *Gradient Iterated Temporal-Difference* (Gi-TD) learning, which incorporates the idea of recent gradient TD methods (Ghiassian et al., 2020) into the iterated TD approach (Vincent et al., 2025b). This results in an algorithm that optimizes the sequence of action-value functions as a whole, without semi-gradient updates.

Our empirical evaluations show that Gi-TD learning: (1) converges on counterexamples where semi-gradient methods (including i-TD learning) fail, (2) performs well at scale across multiple benchmarks, demonstrating for the first time that gradient TD based methods can provide competitive learning speed in the ALE benchmark (Bellemare et al., 2013), and (3) is particularly effective with high replay ratios, and in offline reinforcement learning.

2 The Foundations of Temporal-Difference Learning

We consider a Markov Decision Process (MDP) defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $\mathcal{R}$, a transition kernel $\mathcal{P}$ mapping state-action pairs to probability measures on the following state, and a discount factor γ. We aim to learn the optimal action-value function Q^*, which represents the maximum expected sum of discounted rewards a policy can obtain by interacting with the MDP. From there, an optimal policy π^* is obtained by choosing an action that maximizes the action-value function for each given state s, i.e. $\pi^*(s) \in \arg\max_a Q^*(s,a)$. Importantly, the optimal action-value function is the unique function that verifies the Bellman equation $Q = \Gamma Q$, where Γ denotes the Bellman operator, defined as, $\Gamma Q(s,a) = \mathbb{E}_{r\sim\mathcal{R}(s,a),s'\sim\mathcal{P}(s,a)}[r + \gamma \max_{a'} Q(s',a')]$, for a state-action pair (s,a).

In problems where the reward and transition dynamics are not directly accessible, the Bellman operator must be estimated from samples. Two main bootstrapping methods can be followed to learn the optimal action-value function. Both methods rely on the contraction property of the Bellman operator, whose fixed-point is the optimal action-value function. The first and most popular direction, called TD learning, iteratively applies the Bellman operator, starting from an initial function.

[1] We are interested in achieving high returns with as few samples as possible, a property we refer to as learning speed, and treat training time as a secondary metric, as TD learning is most useful for tasks that require sample efficiency.

In an ideal setting, the Banach fixed-point theorem guarantees that this iterative process converges to the fixed point of the Bellman operator. The other direction, called Gradient TD learning, aims to minimize the distance between a Q-estimator Q and its Bellman iteration ΓQ, thereby forcing the learned estimate to satisfy the Bellman equation.

Figure 1 (left) illustrates how TD learning operates in the space of action-value functions, when function approximation is used to cope with large state spaces. In the general case, the application of the Bellman operator to a Q-estimate ΓQ lands outside of the space of function approximators (represented in green). TD learning starts by instantiating a Q-estimator $\bar{Q}_0$, commonly referred to as the target estimator, used to build a regression target $r + \gamma \max_{a'} \bar{Q}_0(s', a')$, which is a stochastic estimation of its Bellman iteration $\Gamma \bar{Q}_0$. This regression target is learned by a second estimator Q_1, called the online estimator, using the mean squared error. We highlight that the bar over Q_0 is added to emphasize that Q_0 is not optimized, as its parameters remain frozen. After a predefined number of gradient steps T, a target update is performed, which consists of setting Q_1 as the new target estimator by freezing its parameters, and learning a new Q-estimator, noted Q_2, which takes the role of the online estimator to minimize the distance between $\Gamma \bar{Q}_1$ and Q_2 (Mnih et al., 2015).

The procedure repeats until the training stops. It benefits from the contraction property of the Bellman operator at each iteration, which brings the expected regression target closer to the optimal action-value function Q^*. This line of work is referred to as the semi-gradient method because the gradient with respect to the target estimator is ignored. The seminal algorithm Q-learning (Watkins & Dayan, 1992) follows this paradigm, with T set to 1. While it is known that semi-gradient methods can diverge (Baird, 1995), their convergence properties are actively studied (Asadi et al., 2023).

Figure 1 (right) depicts the training dynamics of Gradient TD learning. They are simpler than those of TD learning, since the objective function, i.e., the Bellman error $\|\Gamma Q - Q\|_2^2$, is optimized over all learnable parameters without stop-gradient operations. This is why convergence guarantees can be established for this approach. At each timestep k, the following Q-estimator Q_k is obtained by performing a stochastic gradient descent step on the objective function, evaluated at Q_{k-1}.

The difficult part of this method lies in building an unbiased estimate of the objective function's gradient. Indeed, the gradient of the Bellman error $(\Gamma Q_\theta(s,a) - Q_\theta(s,a))^2$ is equal to $2\,\partial_\theta \Gamma Q_\theta(s,a)(\Gamma Q_\theta(s,a) - Q_\theta(s,a)) - 2\,\partial_\theta Q_\theta(s,a)(\Gamma Q_\theta(s,a) - Q_\theta(s,a))$, for a state-action pair (s,a) and some parameters θ. Importantly, an unbiased estimate of the second term can be obtained from a single sample (s,a,r,s') by computing $-2\,\partial_\theta Q_\theta(s,a)(r + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a))$. Unfortunately, it is not possible to build an unbiased estimate of the first term with a single sample because it contains the product of expectations $\partial_\theta \Gamma Q_\theta(s,a) \cdot \Gamma Q_\theta(s,a)$.

This is why Sutton et al. (2009) learn the difference $\Gamma Q_\theta - Q_\theta$ with a different estimator, H_z, with some parameters z, to replace the need of two independent samples, and use the single sample that is available to estimate $\partial_\theta \Gamma Q_\theta(s,a)$. Adding a weight decay penalty on the z parameters leads to the algorithm called temporal-difference learning with regularized corrections (TDRC, Ghiassian et al. (2020)), where the gradients with respect to the learnable parameters θ and z, noted g_θ and g_z, are

$$g_\theta = \partial_\theta \left[r + \gamma \max_{a'} Q_\theta(s',a') \right] H_z(s,a) - \partial_\theta Q_\theta(s,a) \left[r + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a) \right] \quad (1)$$

$$g_z = \partial_z \left[H_z(s,a) - (r + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)) \right]^2 + \beta\, \partial_z \|z\|_2^2, \quad (2)$$

where β is the weight decay coefficient.

3 Iterated Temporal-Difference Learning

To increase the learning speed of temporal-difference learning methods, Vincent et al. (2025b) suggest to learn a sequence of $K+1$ action-value functions $(Q_k)_{k=0}^K$ where each function Q_k is optimized to represent ΓQ_{k-1}, which is the application of the Bellman operator Γ over the previous function in the sequence Q_{k-1}. For that, the algorithm aims at minimizing the sum of Bellman Errors (BEs), $\sum_{k=1}^K \|\Gamma Q_{k-1} - Q_k\|_2^2$.

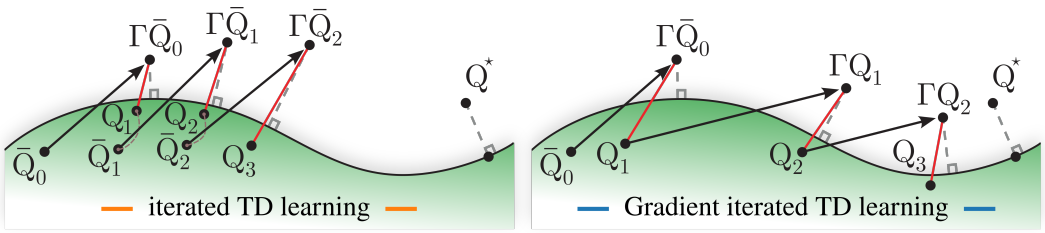

Figure 2: **Left**: In this figure, iterated TD learning learns 3 projected Bellman iterations in parallel, where each Bellman iteration $\Gamma\bar{Q}_{k-1}$, built from the target estimator $\bar{Q}_{k-1}$, is learned with an online estimator Q_k. **Right**: Gradient iterated TD learning minimizes the sum of Bellman errors $\|\Gamma\bar{Q}_0 - Q_1\|_2^2 + \|\Gamma Q_1 - Q_2\|_2^2 + \|\Gamma Q_2 - Q_3\|_2^2$. Each function Q_k not only learns to regress its target ΓQ_{k-1}, but also to make the target ΓQ_k for the following function Q_{k+1} easier to regress.

The motivation for this novel objective function lies in the fact that this term is included in the upper bound on error propagation for approximate value iteration, published in Farahmand (2011). Given a sequence of functions $Q_0, \dots, Q_K$, Farahmand (2011) derive an upper bound on the distance between the optimal action-value function Q^* and the action-value function Q^{π_K}, where π_K is the greedy policy over the last function Q_K (see Theorem 3.4 for a formal statement). This upper bound contains the weighted sum of BEs formed by the sequence of functions, i.e., $\sum_{k=1}^{K} \alpha_k \|\Gamma Q_{k-1} - Q_k\|_{2,\nu}^2$, where the weights α_k are approximated to 1 in the objective function of i-TD learning, and ν is the distribution of the collected data.

We present a schematic representation of i-TD learning in Figure 2 (left). It considers K online estimators $(Q_k)_{k=1}^K$, where each estimator Q_k is represented by parameters θ_k, and K target estimators $(\bar{Q}_k)_{k=0}^{K-1}$, where each estimator $\bar{Q}_k$ is represented by parameters $\bar{\theta}_k$. Similarly to TD learning, each online estimator Q_k learns from its respective target estimator $\bar{Q}_{k-1}$. Crucially, at every D training step, a chain structure is imposed by synchronizing each target estimator $\bar{Q}_k$ with its corresponding online estimator Q_k, i.e., $\bar{\theta}_k \leftarrow \theta_k$. D is generally chosen significantly smaller than T.

Since the number of Bellman iterations learned over the course of a training is generally too large to store all Q-estimators in memory, K is set to a reasonable value (≈ 5), and at every T training steps, each estimator parameter θ_k is updated to the following one θ_{k+1} so that the following Bellman iterations are considered. The same is done for each target parameter $\bar{\theta}_k, k \in \{0, K-1\}$, thereby leading to learning the following Bellman iterations. This step has a similar effect to the target update in TD learning. In fact, when $K = 1$, iterated TD learning reduces to TD learning.

The major limitation of i-TD learning is that each target estimator $\bar{Q}_k$ is constantly out of synchronization with its corresponding online estimator Q_k, which is the true moving target, as highlighted by the dashed brown curves in Figure 2. While this problem can be reduced by frequently updating each target estimator to its corresponding online estimator, it still persists even when this update is performed after each gradient step, i.e., $D = 1$. This is because the target estimators are frozen during semi-gradient updates. As a result, i-TD learning minimizes the sum of BEs only under certain conditions (see Proposition 4.1 in Vincent et al. (2025b)), which are not under the algorithm's control. i-TD learning can even lead to a diverging sum of BEs, as we show in Section 6.

4 Related Work

Stepping away from semi-gradient updates is not a new idea. Schweitzer & Seidmann (1985) propose to minimize the Bellman error $\|\Gamma Q - Q\|$ in the context of dynamic programming. In deterministic environments, where the double sampling problem does not occur, Baird (1995) suggest scaling the gradient of the bootstrapped estimate, to be more competitive against semi-gradient methods. In the general case, the Bellman error appears less favorable as a learning objective because it is not a function of the data, which leads Sutton & Barto (2018) (Section 11.6) to conclude that this quantity is not learnable. Instead, Sutton et al. (2008; 2009) focus on minimizing the projected Bellman error with linear function approximators.

Patterson et al. (2022b) generalizes this idea to nonlinear function approximation, resulting in the TDRC algorithm presented in Section 2. We denote $\mathcal{H}$ as the class of H approximators and $\mathcal{Q}$ the class of Q approximators. The authors demonstrate that the algorithm's learning objective becomes closer to the projected Bellman error when $\mathcal{H}$ becomes closer to $\mathcal{Q}$, and the same objective function becomes closer to the Bellman error when $\mathcal{H}$ increases in size. For completeness, Figure 9 (left) depicts the schematic representation of TDRC with the projected Bellman error as objective function, corresponding to the case when $\mathcal{H} = \mathcal{Q}$.

The idea of learning a sequence of action-value functions that represent consecutive Bellman iterations has been introduced by Schmitt et al. (2022). Vincent et al. (2024) extends the idea to nonlinear function approximation and infinite sequence length by learning a hypernetwork mapping the parameters of a Q-network to the parameters of the Q-network representing its projected Bellman iteration. Iterated TD learning (Vincent et al. (2025b), explained in Section 3) further develops the approach to scale to deep nonlinear function approximation and learn every action-value function in parallel without requiring sequential updates. In a follow-up work, Vincent et al. (2026) propose a parameter-efficient version where each function in the sequence is represented by a linear head, built on top of a shared feature extractor. While showing greater learning speed on challenging benchmarks, i-TD learning suffers from moving targets, making it prone to unexpected behavior.

Finally, the idea of learning the value improvement path (Dabney et al., 2021) is closely related to i-TD learning, differing in that it focuses on learning past Bellman iterations via auxiliary losses to improve representation learning, rather than actively learning the following Bellman iterations.

In the following, we propose to modify i-TD learning by computing the gradient of the stochastic targets, as in TDRC, yielding Gradient Iterated TD (Gi-TD) learning. The resulting algorithm is designed to directly minimize the sum of BEs, thereby combining the soundness of Gradient TD learning with the promise of greater learning speed coming from the idea motivating i-TD learning.

5 Gradient Iterated Temporal-Difference Learning

We design an algorithm, which we call *Gradient Iterated Temporal-Difference* (Gi-TD) learning, that learns a sequence of $K+1$ action-value functions in parallel. Each function Q_k is optimized to represent the Bellman iteration of the previous function in the sequence ΓQ_{k-1}. The objective function remains the same as that of i-TD learning, which is the sum of BEs $\sum_{k=1}^{K} \|\Gamma Q_{k-1} - Q_k\|_2^2$, but Gi-TD learning differs in the way it minimizes this sum. We note that the first function Q_0 is kept fixed, since it does not have a target to regress. Hence, we denote it as $\bar{Q}_0$.

In contrast to i-TD learning, Gi-TD learning computes the gradients of the stochastic targets rather than relying on semi-gradient updates. This means that every function Q_k is not only learned to approximate its target ΓQ_{k-1}, but also to make the target ΓQ_k for the following function Q_{k+1}, an easier target to regress towards. It ensures that Gi-TD learning *directly* minimizes the sum of BEs, as all parameters in the sum are optimized without ignoring the gradient of the targets.

Figure 2 (right) illustrates the mechanism behind the proposed algorithm for $K = 3$, where the sequence of functions $\bar{Q}_0, Q_1, Q_2, Q_3$ is learned to minimize the sum of BEs $\|\Gamma \bar{Q}_0 - Q_1\| + \|\Gamma Q_1 - Q_2\| + \|\Gamma Q_2 - Q_3\|$. Importantly, by directly minimizing the sum of BEs, the optimization procedure allows trade-offs between early and later Bellman errors. This means that future functions influence the learning dynamics of previous ones. This property is novel and avoids the greedy behavior of TD and i-TD learning, which only minimize the immediate Bellman error and disregard the following ones. Figures 1 (left) and 2 account for this property, representing Q_1 further away from $\Gamma \bar{Q}_0$ for Gi-TD learning than for TD and i-TD learning, with the following Bellman errors being smaller.

We now present the update rule for Gi-TD learning and derive a theoretical justification in Section A. We represent the sequence of action-value functions with $K+1$ Q-estimators $\bar{Q}_0, Q_1, \ldots, Q_K$, parameterized by $\bar{\theta}_0, \theta_1, \ldots, \theta_K$. We also use $K-1$ H-estimators $H_2, \ldots, H_K$, parameterized by $z_2, \ldots, z_K$, where each estimator H_k is trained to learn the difference $\Gamma Q_{k-1} - Q_k$.

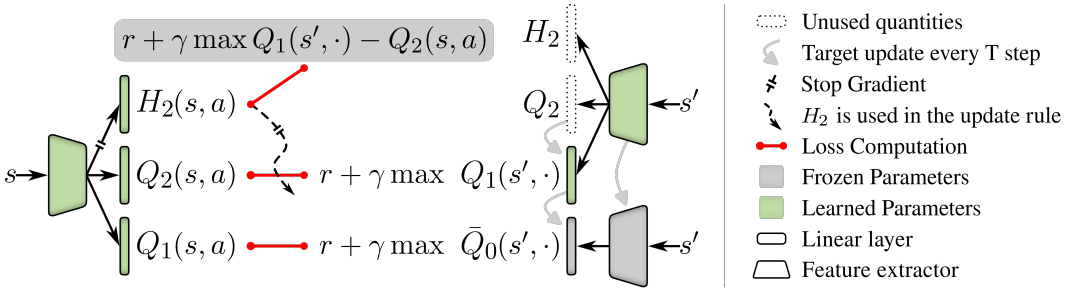

Figure 3: Training procedure of Gradient Iterated TD learning with $K = 2$. Q_1 not only learns the regression target built from $\bar{Q}_0$, but is also optimized so that the regression target built from itself is closer to Q_2. Q_2 learns the regression target built from Q_1. H_2 approximates the difference between the regression target built from Q_1, and Q_2. To save parameters, each Q and H-network is represented by a head, built on a shared feature extractor, reducing the number of networks to 2.

These H-estimators are used to compute each gradient term that includes a stochastic target $r + \gamma \max_{a'} Q_{\theta_k}(s', a')$, without requiring two independent samples, as explained in Section 2. Following Patterson et al. (2022b), a weight decay is added to the parameters of the H-estimators. This results in the gradient estimates Δ_{θ_k}, and Δ_{z_k} to perform stochastic gradient descent on the Q-estimator parameters, and H-estimator parameters. Given a sample (s, a, r, s'), we define

$$\Delta_{\theta_k} = \partial_{\theta_k} \left[r + \gamma \max_{a'} Q_{\theta_k}(s', a') \right] H_{z_{k+1}}(s, a) - \partial_{\theta_k} Q_{\theta_k}(s, a) \left[r + \gamma \max_{a'} Q_{\theta_{k-1}}(s', a') - Q_{\theta_k}(s, a) \right]$$

$$\Delta_{z_k} = \partial_{z_k} \left[H_{z_k}(s, a) - (r + \gamma \max_{a'} Q_{\theta_{k-1}}(s', a') - Q_{\theta_k}(s, a)) \right]^2 + \beta\, \partial_{z_k} \|z_k\|_2^2,$$

where β controls the importance given to the weight decay. We note that Δ_{θ_K} is only equal to $-\partial_{\theta_K} Q_{\theta_K}(s, a)(r + \gamma \max_{a'} Q_{\theta_{K-1}}(s', a') - Q_{\theta_K}(s, a))$, as θ_K is the last parameter of the sequence.

Similarly to i-TD learning, a target update is performed every T steps, allowing the following Bellman iterations to be considered. The implementation follows $\theta_k \leftarrow \theta_{k+1}$, and $z_k \leftarrow z_{k+1}$, for all possible values of k. An analogous update to soft-target updates (Haarnoja et al., 2018) is also possible, where θ_0 is updated to $\tau\theta_1 + (1 - \tau)\theta_0$ at every step, and τ regulates the speed at which the parameters θ_0 are updated to θ_1. When interacting with the environment with discrete action spaces, the action that maximizes the average prediction across the online Q-estimates is selected.

Figure 3 provides a summary of the training procedure with $K = 2$. In practice, parameters can be shared to reduce the memory footprint, with functions represented by different heads that are built on a shared feature extractor. This reduces the number of independent estimators to 2. We present the training procedures for TDRC and i-TD learning in Figure 11 to facilitate comparison. The pseudo-code for an instance of Gi-TD learning applied to Deep Q-network (DQN, Mnih et al. (2015)) is presented in Algorithm 1. Algorithm 2 presents an instance of Gi-TD learning applied to Soft Actor-Critic (SAC, Haarnoja et al. (2018)).

Finally, the representation capacity of the H-estimators can change the objective function (see Section 4). While Figure 2 (right) shows the sum of BEs as the objective function, we show in Figure 9 (right) the case where $\mathcal{H} = \mathcal{Q}$, for which the objective function is the sum of projected BEs.

6 Analysis in Controlled MDPs

We start by analyzing the behavior of Gi-TD learning on the Star, Hall, and Triangle Markov Processes (MPs) (Baird, 1995; Tsitsiklis & Van Roy, 1997) against TD, TDRC, and i-TD learning. The Star MP, also known as the Baird's counterexample, is presented on the left of Figure 4, and the Hall MP is presented on the right. The Triangle MP is presented in Figure 14. For each MP, the objective is to learn the value function from a dataset in which each state is visited with equal frequency. The value function is equal to zero since a reward of zero is given at every transition.

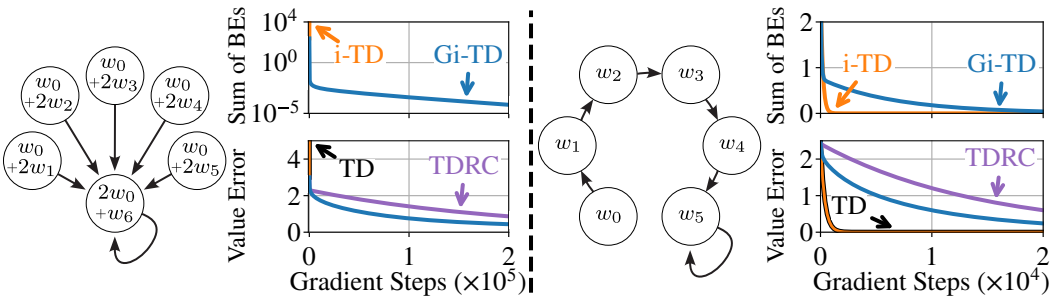

Figure 4: Evaluating the proposed approach in an **off-policy setting** with **linear function approximation**. We clarify that in the bottom plots, TD and i-TD learning overlap. **Left**: While i-TD and Gi-TD learning are both designed to decrease the sum of Bellman errors, only Gi-TD learning decreases this quantity when evaluated on Baird's counterexample. This leads to a low value error for Gi-TD learning, as opposed to i-TD learning, for which this error increases. **Right**: It is well-known that gradient TD methods have a slow learning speed when evaluated on the Hall problem (Baird, 1995). Notably, Gi-TD learning minimizes the value error faster than TDRC.

We leverage the low dimensionality of these MPs to study the behavior of the 4 algorithms in a fully deterministic setting. This means that only one evaluation per algorithm is needed. For that, we evaluate the expected version of each algorithm, in which updates are performed synchronously across all states. We remove the influence of the H-network by allowing the algorithm to access the true Bellman iteration, rather than a stochastic estimate. Therefore, TDRC becomes an instantiation of the residual gradient algorithm (Baird, 1995). We also remove the influence of target updates in i-TD and Gi-TD learning and instead set K to a sufficiently large value so that reward propagation is not limited. Finally, we set $T = 1$ for TD learning, and $D = 1$ for i-TD learning.

For each experiment, we plot the value error of each algorithm, which is the distance between the learned estimate and the true value function, as a function of the number of gradient steps. We also report the objective function of i-TD and Gi-TD learning, which is the sum of BEs $\sum_{k=1}^{K} \|\Gamma V_{k-1} - V_k\|_2^2$, where $(V_k)_{k=0}^{K}$ is the considered sequence of value functions.

6.1 Off-policy Learning with Linear Function Approximation

We first consider the Star and Hall MPs (Baird, 1995) with linear function approximation (see Figure 4), in an off-policy setting. We set the learning rate to 0.08, and $K = 300$.

For the Star MP, all weights are initialized to zero, except for w_6, which is set to 1, and $\gamma = 0.99$. As expected, TD and i-TD learning diverge due to their semi-gradient nature, whereas TDRC and Gi-TD learning converge, as they are part of the Gradient TD learning paradigm (see Figure 4, left). Importantly, while the sum of BEs decreases over time for Gi-TD learning, it increases for i-TD learning, indicating that i-TD learning does not directly minimize this quantity, due to its semi-gradient nature.

For the Hall MP, all weights are initialized to one, and $\gamma = 0.9$. Baird (1995) introduce this problem to demonstrate that semi-gradient methods can learn faster than gradient TD methods. Figure 4 (right) confirms this claim. Indeed, TD and i-TD learning minimize the value error faster than TDRC and Gi-TD learning. Notably, Gi-TD learning reduces the value error faster than TDRC. Interestingly, the learning speed difference between i-TD and Gi-TD learning can be explained by the difference in the sum of BEs, supporting the relevance of this objective function.

6.2 On-policy Learning with Nonlinear Function Approximation

We now focus on the Triangle MP (Tsitsiklis & Van Roy, 1997) in an on-policy setting, with nonlinear function approximation. We set the learning rate to 0.002, $K = 10$, $\gamma = 0.99$. All weights are initialized to 14 so that all value functions are equal before the training starts.

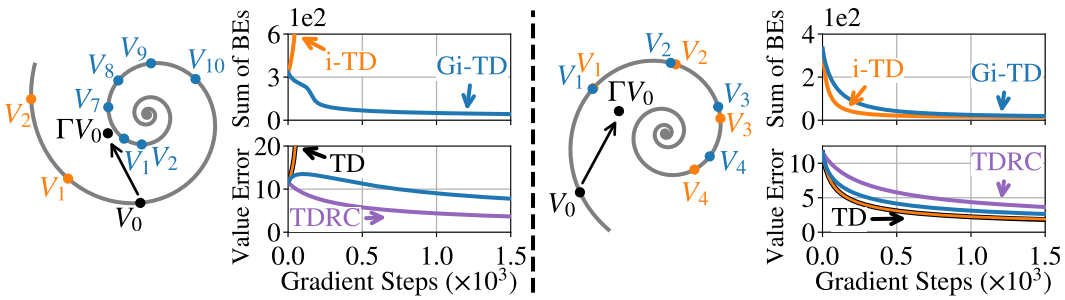

Figure 5: Evaluating the proposed approach in an **on-policy setting** with **nonlinear function approximation**, on the Triangle MP. We clarify that in the bottom plots, TD and i-TD learning overlap. **Left**: i-TD learning increases the sum of Bellman Errors (BEs) during training, while it is designed to minimize it. This translates to high value errors. In contrast, Gi-TD learning decreases this quantity during training, leading to a low final value error. **Right**: When changing the spiral direction, semi-gradient methods learn faster than gradient TD methods. Importantly, Gi-TD learning exhibits a faster learning speed than TDRC.

Studying this problem is helpful because the algorithm's behavior can be understood geometrically, as shown in Figures 1 and 2. Indeed, the Triangle MP has only 3 states, so the value functions lie in $\mathbb{R}^3$. The space of function approximation is a spiral spanning over the plane orthogonal to the vector $(1, 1, 1)$ and including the origin. In this experiment, the Bellman operator divides the norm of any vector (which represents a value function) by 2, and rotates the resulting vector by an angle of $-60°$ along the axis defined by the vector $(1, 1, 1)$. This means that the application of the Bellman operator on any value function belonging to the plane mentioned earlier, remains on this plane. The true value function $(V(s_1), V(s_2), V(s_3)) = (0, 0, 0)$ is also part of this plane. Therefore, all relevant quantities lie on this plane, and the analysis can be done in 2 dimensions.

Figure 5 (left) illustrates the sequence of value functions obtained at the end of training for i-TD and Gi-TD learning. We represent the initial value function V_0 on the plane, belonging to the space of representable value functions in gray, with the center of the spiral being the true value function $(0, 0, 0)$. As explained, ΓV_0 equals V_0 scaled by half and rotated by $-60°$. In this experiment, i-TD learning tries to bring V_1 (in orange) closer to ΓV_0, which makes it move further away from the center of the spiral. The same goes for V_2, which moves further away from the center to approximate ΓV_1. This continues for all terms in the sequence of value functions, which causes the value error and the sum of BEs to increase. This experiment is another example showing that i-TD learning can lead to increasing values of its objective function.

In contrast, Gi-TD learning exhibits a different behavior. While Gi-TD learning also brings V_1 (in blue) closer to ΓV_0, hence further away from the center, it makes ΓV_1 closer to V_2, which is initialized at the same location as V_0, and hence moves V_1 closer to the center. Overall, the sum of BEs decreases if all value functions are closer to the center of the spiral. This is why the functions move closer to the center as represented in Figure 5 (left). This behavior translates into lower value errors and a lower sum of BEs, demonstrating the relevance of the proposed approach.

Interestingly, changing the direction of the spiral drastically alters the behavior of the studied algorithms, since the Bellman operator now points to the direction of the center following the spiral. Indeed, while i-TD learning only moves V_1 (in orange) closer to ΓV_0, which brings V_1 closer to the center, Gi-TD learning also displaces V_1 (in blue) such that ΓV_1 is closer to V_2, which is initialized at the same location as V_0, hence moving further away from the center. This means that semi-gradient methods have an advantage, as they ignore the gradient term coming from the bootstrapped target. Due to this additional term, the value functions learned with Gi-TD learning lag behind those learned with i-TD learning in Figure 5 (right). Overall, all methods reduce the value error over time, with semi-gradient methods being faster than gradient TD methods. Notably, the learning speed of Gi-TD learning is higher than that of TDRC, as observed in the Hall MP experiment in Section 6.1.

7 Experiments

We evaluate the proposed method on various benchmarks against TD, TDRC, and i-TD learning to assess its behavior in more complex environments.

7.1 Setup

In this work, we focus on the agent's learning speed. This is why we place greater importance on the Area Under the Curve (AUC) of the performance curve than on final performance, as it favors algorithms that continuously improve performance over those that only achieve high returns at the end of training. The AUC also has the advantage of being less dependent on the training budget.

We use the Inter-Quantile Mean (IQM, Agarwal et al. (2021)) to remove outliers that may arise from lucky exploration, as the algorithms considered in this work compete at the level of optimization, given the collected data. This is done by discarding 25% of each tail of the distribution of scores when aggregating results. It also allows to discard runs with faulty resets, which can occur when using the Gymnasium library (Towers et al., 2026).

When aggregating results, we report baseline-normalized scores, with the TD learning end score as baseline, rather than the human-normalized score. Indeed, TD methods learn differently than humans (Delfosse et al., 2025), leading to human-normalized scores that differ significantly across environments (see Figure 15). Therefore, aggregating human-normalized scores with the IQM metric results in discarding experiments with the lowest and highest scores, which correspond to the easiest and hardest environments. This goes against our motivation to use the IQM metric. We still use the human-normalized score for selecting a set of 10 Atari games containing varying levels of difficulty (see Figure 15), as commonly done in RL (Graesser et al., 2022; Vincent et al., 2025a).

In the supplementary material, we report the list of hyperparameters in Tables 1, and 2, the detailed description of the aggregation protocol in Section D, and the individual learning curves for each experiment in Section F. Importantly, all shared hyperparameters are set to the same values as in the published version of the baseline for all algorithms. We set $K = 5$ for i-TD and Gi-TD learning, and $\beta = 1$ for TDRC and Gi-TD learning. We also set $D = 1$ for i-TD learning as Vincent et al. (2026) show that it performs well with this value.

For experiments with discrete action spaces, we represent all functions with linear heads, built on top of a shared feature extractor for TDRC, i-TD and Gi-TD learning. For experiments with continuous action spaces, we use independent networks for all algorithms except TDRC, which uses linear heads. We ablate those choices in Section 7.3. Finally, we report aggregated scores for 5 seeds per Atari game (Bellemare et al., 2013) and 10 seeds per MuJoCo environment (Todorov et al., 2012).

7.2 Benchmark Performance

Online Discrete Control. We combine each of the 4 considered TD learning methods with DQN equipped with the CNN architecture, yielding DQN, QRC, i-DQN, and Gi-DQN. We report the learning curves over 100 million frames aggregated over 10 Atari games in Figure 6 (top, left). Remarkably, Gi-DQN outperforms all considered algorithms, showing that gradient TD methods can be competitive against semi-gradient approaches. We also report the AUC for each method, normalized by DQN's AUC, along the x-axis of Figure 6 (bottom, left). The y-axis indicates which method is reported. Gi-DQN provides a 20% improvement over DQN, and increases QRC's performance by 50 points of percentage ($120 - 70 = 50$).

Online Continuous Control. We now investigate the ability of the 4 considered algorithms to learn informative critics when combined with SAC. Following Haarnoja et al. (2018), all networks are composed of 2 hidden layers of 256 neurons each. We let each algorithm interact for one million steps in 6 MuJoCo tasks and report the aggregated SAC-normalized score in Figure 6 (top, center). Here again, Gi-SAC remains competitive against the semi-gradient methods. It is not the case for SACRC, which performs worse than SAC. In Figure 6 (bottom, center), the comparison of AUCs indicates that Gi-SAC achieves a 7% improvement compared to SAC.

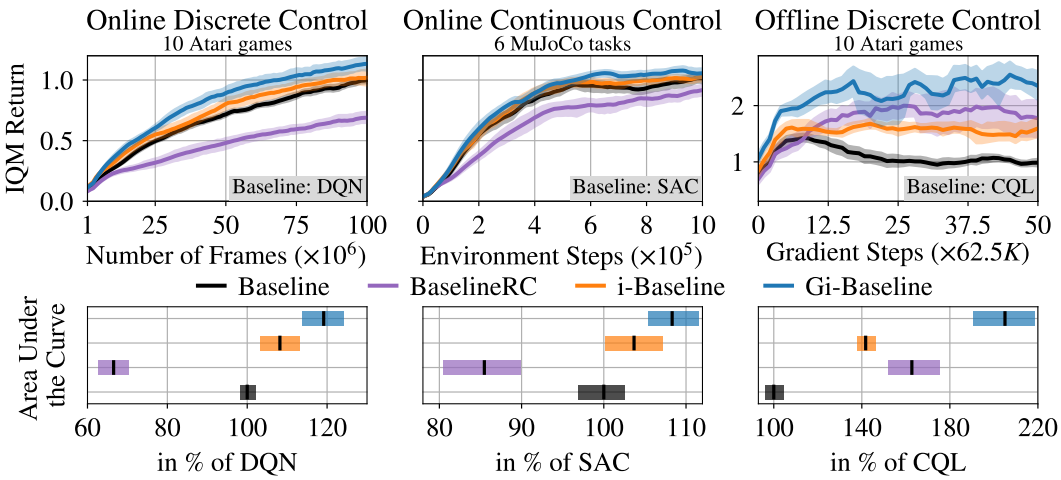

Figure 6: Evaluating the proposed approach on multiple benchmarks with **deep nonlinear function approximation**. When combined with DQN (left), SAC (center), and CQL (right), Gi-TD learning is competitive against semi-gradient methods. Importantly, the performance boost is the highest in the offline setting, where the choice of the optimization method is particularly important.

Offline Discrete Control. To remove the influence of exploration, we perform an evaluation in an offline scenario on 10 Atari games. We use the dataset released by Gulcehre et al. (2020), which is collected by a DQN agent trained online for 200 million interactions. This leads to a dataset of 50 replay buffers, each comprising 4 million frames. We combine the 4 considered methods with Conservative Q-learning (CQL, Kumar et al. (2020)) and let each algorithm have access to the first 10% of each offline replay buffer during training. Notably, Gi-CQL outperforms the other 3 algorithms by a large margin, as shown in Figure 6 (top, right), yielding an AUC that is twice that of CQL (see Figure 6, bottom, right). Interestingly, both gradient TD methods outperform the semi-gradient methods, highlighting the benefit of using theoretically sound objective functions.

Scaling Up the Baseline. We now investigate if Gi-DQN brings benefits over a stronger baseline. For that, we use the IMPALA architecture (Espeholt et al., 2018) and add a global average pooling for each channel of the last convolutional layer, as indicated in Trumpp et al. (2025); Sokar & Castro (2025). We also implement a prioritized experience replay buffer (PER, Schaul et al. (2016)) and use a 3-step return. Figure 7 (center) confirms that the resulting baseline yields better performance than the baseline presented in Figure 6 (left), as the previous baseline only reaches 75% of the new baseline's AUC on the same 10 Atari games and update-to-data ratio (Medium UTD $= 1/4$, see the grey dashed lines). In Figure 7 (center), we observe that Gi-DQN remains competitive against semi-gradient methods, improving the AUC by 5% over the baseline. We stress that achieving improvements over strong baselines, such as the one we consider in this experiment, is generally harder. Similar to the previous online experiments, TDRC exhibits lower learning speed.

Scaling Up the Replay Ratio. Theoretically sound methods should allow for higher data utilization without the risk of divergence. Therefore, we study the behavior of the 4 considered methods with a UTD ratio 16 times higher than the previous one, i.e., UTD $= 4$ instead of $1/4$. To reduce the computational budget, we maintain the same amount of total gradient steps, resulting in 6.25 million interactions ($100/16 = 6.25$). In Figure 7 (right), the baseline for normalizing the learning curves is still the end performance of the strong DQN version with a medium UTD ratio to ease comparison. Remarkably, Gi-DQN outperforms the other methods, yielding an average performance 130% higher than that of the strong DQN variant. In this setting, QRC also outperforms the semi-gradient methods, supporting the idea that theoretically sound methods generally perform better with more computing power. Finally, we evaluate the 4 methods in a low UTD regime where a gradient step is performed every 32 environment interactions, which is equal to the batch size, thereby evaluat-

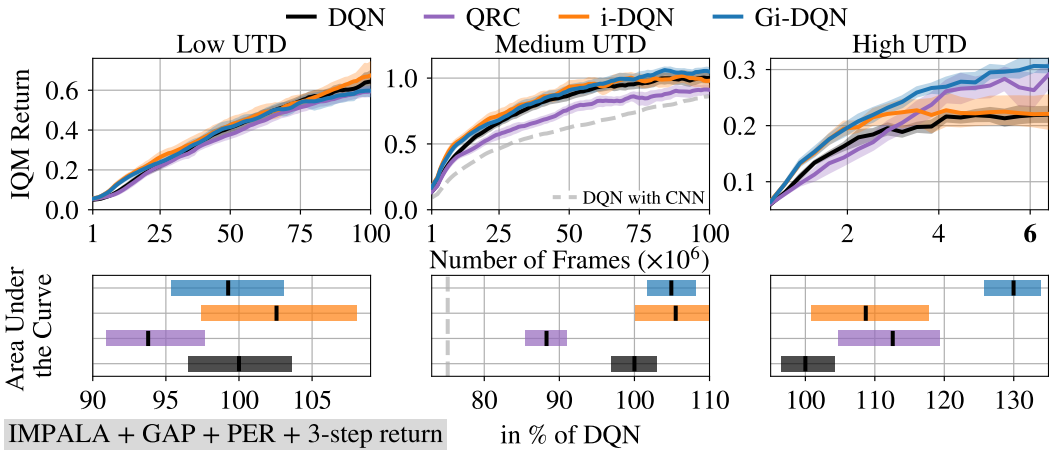

Figure 7: Evaluating the proposed approach in **combination with a strong baseline**, and with different **update-to-data (UTD) ratios**. Gi-TD learning remains competitive against semi-gradient methods when combined with a more powerful architecture (IMPALA + GAP), a prioritized experience replay buffer (PER), and a 3-step return. Notably, Gi-TD learning's learning speed is significantly greater than that of the other considered methods at high UTD ratios, a setting where the optimization method exerts a stronger influence on the learning behavior.

ing the lowest relevant UTD ratio. As expected, all methods perform similarly, confirming that the potential of gradient TD methods lies in high UTD regimes.

In Section E, we perform a similar study on the MuJoCo benchmark with varying UTD values. Figure 16 demonstrates a similar trend, confirming the superiority of Gi-TD at high UTD ratios.

7.3 Ablation Study

We ablate the design choices made in Section 7. Specifically, we study the dependency of Gi-TD learning on the weight decay coefficient β and the choice of the architecture. For that, we evaluate the combinations of Gi-TD learning with DQN, SAC, and CQL on 2 Atari games with 5 seeds, 2 MuJoCo tasks with 5 seeds, and 2 Gym environments with 10 seeds.

We run a grid search on 2 weight decay coefficients ($\beta = 1$, and 0.01), and 3 different architectures that are presented in Figure 10 (independent networks, shared feature extractor with linear heads, and with nonlinear heads), except for the experiments on Atari games, for which we do not evaluate independent architectures to reduce the computational budget. The CNN architecture is used for Gi-DQN, a 2-layer network with 256 neurons is used for Gi-SAC, and a 3-layer network with 50 neurons is used for Gi-CQL.

To reduce dependency on other hyperparameters, we add 2 axes to the grid and aggregate results across them. We select 2 values of UTD (0.25 and 1 for Gi-DQN, 1 and 4 for Gi-SAC) or 2 dataset sizes for Gi-CQL (10% and 100%), and 2 target update periods ($T = 1000$ and 8000 for Gi-DQN, $T = 10$ and 1000 for Gi-CQL) or 2 values of soft-target updates for Gi-SAC ($\tau = 0.005$ and 0.01).

Figure 8 (top) reports the aggregated performance over architecture designs, the UTDs or dataset sizes, and target update periods or soft-target update values. It demonstrates the relevance of regularizing the H-networks as advocated in Ghiassian et al. (2020), because setting the weight decay coefficient to a high value generally leads to better performance. Then, we report the performance of the different architectures in Figure 8 (bottom), selecting $\beta = 1$, and aggregating over the 2 other axes. The architecture with linear heads appears to perform best, except in the continuous-action setting, where independent networks yield the best results. This explains the choices made in Section 7. We believe that independent networks perform better for Gi-SAC because the architecture is

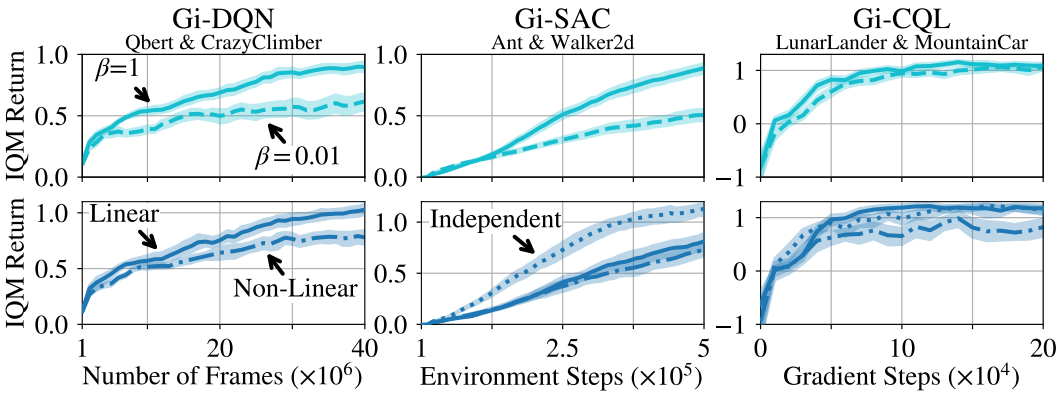

Figure 8: Ablation studies on the **weight decay coefficient** and the **architecture design choices**. **Top**: Setting $\beta = 1$, as advocated in Patterson et al. (2022b), yield better performance. **Bottom**: Defining a shared feature extractor with linear heads performs best for experiments with discrete actions. In the experiment with a continuous actions space (Gi-SAC), we believe that the neural networks are too shallow to benefit from parameter sharing.

too shallow (only 2 layers) to benefit from parameter sharing. We also perform the same study on TDRC and i-TD learning, which we report in Figure 17, and draw similar conclusions.

Finally, we fix $\beta = 1$, and use an architecture with linear heads to compare 2 values of K (5 and 50) for i-TD and Gi-TD learning. We aggregate the learning curves across the 2 other axes in Figure 18. Notably, the dependency of Gi-TD learning on this hyperparameter is lower than for i-TD learning, which struggles to learn for $K = 50$, a limitation pointed out in Vincent et al. (2026).

8 Limitations, Conclusion & Future Work

Due to its more elaborate loss, the presented method requires more floating-point operations per gradient step than the other methods, which entails a longer training time (see Figure 12). Nonetheless, we point out that when computing the algorithm's runtime, the environment steps are simulated, which does not reflect the real-world duration, as the simulator reduces the time required to collect data, which is generally the main bottleneck for real-world applications.

Another limitation is that the proposed objective function is not designed to minimize the entire bound on error propagation (see Theorem 3.4). It only focuses on the term that depends on the sequence of action-value functions observed during training. However, the proposed method still provides some benefit compared to i-TD learning, and the other terms are related to the algorithm's exploration strategy, which is beyond the scope of this work. Finally, studying the convergence properties of the proposed approach would provide more insights into the algorithm's behavior.

To conclude, we introduced *Gradient Iterated Temporal-Difference* (Gi-TD) learning, which is designed to learn a sequence of action-value functions in parallel, where each function is optimized to represent the application of the Bellman operator over the previous function in the sequence. Its objective function is the sum of Bellman errors formed by the sequence of action-value functions. Gi-TD learning minimizes this quantity by optimizing over every learnable parameter, including those used to compute stochastic targets. Therefore, it belongs to the gradient TD line of work. Our evaluations show competitive performance against semi-gradient methods across various benchmarks, including Atari games, a result that no prior work on gradient TD methods has demonstrated.

We believe that studying a version of Gi-TD learning with different coefficients for each Bellman iteration could further boost performance. Additionally, combining Gi-TD learning with gradient eligibility traces (Elelimy et al., 2025), distributional or robust losses (Bellemare et al., 2023; Qu et al., 2019; Patterson et al., 2022a) is a promising step towards sample-efficient algorithms.

Acknowledgments

This research was supported by the grant "Einrichtung eines Labors des Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) an der Technischen Universität Darmstadt". We gratefully acknowledge support from the hessian.AI Service Center (funded by the Federal Ministry of Education and Research, BMBF, grant no. 01IS22091) and the hessian.AI Innovation Lab (funded by the Hessian Ministry for Digital Strategy and Innovation, grant no. S-DIW04/0013/003).

Carbon Impact

As recommended by Lannelongue & Inouye (2023), we used GreenAlgorithms (Lannelongue et al., 2021) and ML CO_2 Impact (Lacoste et al., 2019) to compute the carbon emission related to the production of the electricity used for the computations of our experiments. We only consider the energy used to generate the figures presented in this work and ignore the energy used for preliminary studies and for building the computing infrastructure. The estimations vary between 2.23 and 2.29 tonnes of CO_2 equivalent. As a reminder, the Intergovernmental Panel on Climate Change advocates a carbon budget of 2 tonnes of CO_2 equivalent per year per person.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 2021.

Kavosh Asadi, Shoham Sabach, Yao Liu, Omer Gottesman, and Rasool Fakoor. Td convergence: An optimization perspective. *Advances in Neural Information Processing Systems*, 2023.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. *Machine learning*, 1995.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2013.

Marc G Bellemare, Will Dabney, and Mark Rowland. *Distributional reinforcement learning*. MIT Press, 2023.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. *JAX: composable transformations of Python+NumPy programs*, 2018.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, 2021.

Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song. Sbeed: Convergent reinforcement learning with nonlinear function approximation. In *International Conference on Machine Learning*, 2018.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 2022.

Quentin Delfosse, Jannis Blüml, Fabian Tatai, Théo Vincent, Bjarne Gregori, Elisabeth Dillies, Jan Peters, Constantin A Rothkopf, and Kristian Kersting. Deep reinforcement learning agents are not even close to human intelligence. In *Inductive Biases in Reinforcement Learning Workshop @RLC*, 2025.

Esraa Elelimy, Brett Daley, Andrew Patterson, Marlos C Machado, Adam White, and Martha White. Deep reinforcement learning with gradient eligibility traces. In *Reinforcement Learning Journal*, 2025.

Mohamed Elsayed, Gautham Vasan, and A Rupam Mahmood. Streaming deep reinforcement learning finally works. *arXiv preprint arXiv:2410.14606*, 2024.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 2018.

Amir-massoud Farahmand. Regularization in reinforcement learning. *University of Alberta*, 2011.

Sina Ghiassian, Andrew Patterson, Shivam Garg, Dhawal Gupta, Adam White, and Martha White. Gradient temporal-difference learning with regularized corrections. In *International Conference on Machine Learning*, 2020.

Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In *International Conference on Machine Learning*, 2022.

Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Thomas Paine, Sergio Gómez, Konrad Zolna, Rishabh Agarwal, Josh S Merel, Daniel J Mankowitz, Cosmin Paduraru, et al. Rl unplugged: A suite of benchmarks for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 2020.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, 2018.

Ray Jiang, Shangtong Zhang, Veronica Chelu, Adam White, and Hado van Hasselt. Learning expected emphatic traces for deep rl. In *Proceedings of the AAAI conference on artificial intelligence*, 2022.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

Loïc Lannelongue and Michael Inouye. Carbon footprint estimation for computational research. *Nature Reviews Methods Primers*, 2023.

Loïc Lannelongue, Jason Grealey, and Michael Inouye. Green algorithms: quantifying the carbon footprint of computation. *Advanced Science*, 2021.

Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. In *International Conference on Machine Learning*, 2025.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 2018.

Hamid Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. *Advances in neural information processing systems*, 2009.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 2015.

Daniel Palenicek, Florian Vogt, Joe Watson, Ingmar Posner, and Jan Peters. Xqc: Well-conditioned optimization accelerates deep reinforcement learning. *International Conference on Learning Representations*, 2026.

Andrew Patterson, Victor Liao, and Martha White. Robust losses for learning value functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022a.

Andrew Patterson, Adam White, and Martha White. A generalized projected bellman error for off-policy value estimation in reinforcement learning. *Journal of Machine Learning Research*, 2022b.

Chao Qu, Shie Mannor, and Huan Xu. Nonlinear distributional gradient temporal-difference learning. In *International Conference on Machine Learning*, 2019.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

Simon Schmitt, John Shawe-Taylor, and Hado Van Hasselt. Chaining value functions for off-policy learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 2020.

Paul J Schweitzer and Abraham Seidmann. Generalized polynomial approximations in markovian decision processes. *Journal of Mathematical Analysis and Applications*, 1985.

Ghada Sokar and Pablo Samuel Castro. Mind the gap! the challenges of scale in pixel-based deep reinforcement learning. *Advances in Neural Information Processing Systems*, 2025.

Richard Sutton and Andrew Barto. *Reinforcement learning: an introduction*. MIT press Cambridge, 2018.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 1988.

Richard S Sutton, Csaba Szepesvári, and Hamid Reza Maei. A convergent o (n) algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in neural information processing systems*, 2008.

Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning*, 2009.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.

Mark Towers, Ariel Kwiatkowski, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Kallinteris Andreas, Markus Krimmel, Arjun KG, Rodrigo De Lazcano Perez-Vicente, et al. Gymnasium: A standard interface for reinforcement learning environments. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2026.

Raphael Trumpp, Ansgar Schäfftlein, Mirco Theile, and Marco Caccamo. Impoola: The power of average pooling for image-based deep reinforcement learning. *Reinforcement Learning Journal*, 2025.

John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 1997.

Gautham Vasan, Mohamed Elsayed, Alireza Azimi, Jiamin He, Fahim Shariar, Colin Bellinger, Martha White, and A. Rupam Mahmood. Deep policy gradient methods without batch updates, target networks, or replay buffers. In *Advances in Neural Information Processing Systems*, 2024.

Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, 2020.

Théo Vincent, Alberto Maria Metelli, Boris Belousov, Jan Peters, Marcello Restelli, and Carlo D'Eramo. Parameterized projected bellman operator. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Théo Vincent, Tim Faust, Yogesh Tripathi, Jan Peters, and Carlo D'Eramo. Eau de q-network: Adaptive distillation of neural networks in deep reinforcement learning. *Reinforcement Learning Journal*, 2025a.

Théo Vincent, Daniel Palenicek, Boris Belousov, Jan Peters, and Carlo D'Eramo. Iterated q-network: Beyond one-step bellman updates in deep reinforcement learning. *Transactions on Machine Learning Research*, 2025b.

Théo Vincent, Yogesh Tripathi, Tim Faust, Yaniv Oren, Jan Peters, and Carlo D'Eramo. Bridging the performance gap between target-free and target-based reinforcement learning. *International Conference on Learning Representations*, 2026.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 2022.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Supplementary Materials

The following content was not necessarily subject to peer review.

Table of Contents

A Theoretical Results **17**

B Schematic Representations **19**

C Algorithmic Details **20**

D Experiment Setup **22**

E Additional Experiments **25**

E.1 Scaling Up the Replay Ratio in Continuous Actions Domains 25

E.2 Additional Ablation Studies 26

F Individual Learning Curves **29**

A Theoretical Results

Theorem 3.4 from Farahmand (2011). *Let $N \in \mathbb{N}^*$, and ρ, ν two probability measures on $\mathcal{S} \times \mathcal{A}$. For any sequence $(Q_k)_{k=0}^K \in \mathcal{Q}_\Theta^{K+1}$ where R_γ depends on the reward function and the discount factor, we have*

$$\|Q^* - Q^{\pi_K}\|_{1,\rho} \leq C_{K,\gamma,R_\gamma} + \inf_{r \in [0,1]} F(r; K, \rho, \nu, \gamma) \left(\sum_{k=1}^K \alpha_k^{2r} \|\Gamma^* Q_{k-1} - Q_k\|_{2,\nu}^2 \right)^{\frac{1}{2}}$$

where C_{K,γ,R_γ}, $F(r; K, \rho, \nu, \gamma)$, and $(\alpha_k)_{k=1}^K$ do not depend on $(Q_k)_{k=1}^K$. π_K is a greedy policy computed from Q_K.

Formal Derivation of Gi-TD learning's update rule. We present a theoretical justification for Gi-TD learning's update rule to complement the intuitive explanation given in Section 5. The goal is to reformulate the objective function of Gi-TD learning such that the double sampling problem is replaced by a min-max problem, as in Patterson et al. (2022b). Similar to Sutton et al. (2009), Gi-TD learning's update rule can then be derived using a saddle-point update and gradient correction.

The objective function of Gi-TD learning is the sum of Bellman Errors (BEs) formed by the sequence of considered action-value functions $(Q_{\theta_k})_{k=0}^K$. For each k, and state-action-reward-next-state (s, a, r, s'), we note $\delta_k(s, a, r, s') \coloneqq r + \gamma \max_{a'} Q_{\theta_{k-1}}(s', a') - Q_{\theta_k}(s, a)$. Therefore, Gi-TD learning's objective is to minimize the sum of BEs that can be written as

$$\sum_{k=1}^K \sum_{(s,a) \in \mathcal{S} \times \mathcal{A}} d(s, a)\, \mathbb{E}_{r \sim \mathcal{R}(s,a), s' \sim \mathcal{P}(s,a)} \left[\delta_k(s, a, r, s') \right]^2 ,$$

where $d(s, a)$ is the state-action distribution.

Unfortunately, estimating this quantity with a single sample yields a biased gradient, since the square of expectations requires two independent samples to be estimated without bias. This is why we reformulate the objective function using the fact that the square function is equal to its biconjugate, i.e., $x^2 = \max_{h \in \mathbb{R}} 2xh - h^2, \forall x \in \mathbb{R}$.

Starting from the original formulation, we replace the square function by its biconjugate to obtain the saddle-point formulation. We note $\delta_k^{\mathbb{E}}(s,a) \coloneqq \mathbb{E}_{r\sim\mathcal{R}(s,a),s'\sim\mathcal{P}(s,a)}\left[\delta_k(s,a,r,s')\right]$, and write:

$$\begin{aligned}
\sum_{k=1}^{K}\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} d(s,a)\delta_k^{\mathbb{E}}(s,a)^2 &= \sum_{k=1}^{K}\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} d(s,a)\max_{h\in\mathbb{R}} 2\,h\,\delta_k^{\mathbb{E}}(s,a) - h^2 \\
&\overset{(a)}{=} \sum_{k=1}^{K}\max_{H\in\mathcal{F}_{\text{all}}}\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} d(s,a)\left(2\,H(s,a)\,\delta_k^{\mathbb{E}}(s,a) - H(s,a)^2\right) \\
&\overset{(b)}{=} \max_{H\in\mathcal{F}_{\text{all}}^K}\sum_{k=1}^{K}\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} d(s,a)\left(2\,H_k(s,a)\,\delta_k^{\mathbb{E}}(s,a) - H_k(s,a)^2\right),
\end{aligned}$$

where $\mathcal{F}_{\text{all}}$ represents the space of all functions mapping state-action pairs to real values. We remark that Equations (a) and (b) are true because of the interchangeability of the summation and the max in this situation. Indeed, Equation (a) holds because a different value of $H(s,a)$ can be chosen for each state-action pair (s,a). Similarly, Equation (b) holds because a different value of $H_k(s,a)$ can be chosen for each state-action pair (s,a), and for each k.

We remark that the sum of BEs can be converted to the sum of projected BEs when $\mathcal{F}_{\text{all}} = \mathcal{Q}$, where $\mathcal{Q}$ is the space of function approximators, as explained in Patterson et al. (2022b).

From there, different update rules can be established to perform stochastic gradient descent on the min-max problem. We use K functions $(H_{z_k})_{k=1}^K$, parameterized by $(z_k)_{k=1}^K$, to represent the quantity optimized by the maximum operator. Overall, the update rule for all learnable parameters is

$$\begin{aligned}
\theta_k &\leftarrow \theta_k - \eta_\theta\,\Delta_{\theta_k}, \text{ for } k\in\{0,\dots,K\} \\
z_k &\leftarrow z_k - \eta_z\,\Delta_{z_k}, \text{ for } k\in\{1,\dots,K\},
\end{aligned}$$

where η_θ, and η_z are the learning rates shared across all values for k. In the following, we derive the gradient estimates Δ_{θ_k} and Δ_{z_k}.

Following GTD2 (Sutton et al., 2009), the gradient estimates for saddle-point updates are

$$\begin{aligned}
\Delta_{\theta_k} &= \partial_{\theta_k}\left[\sum_{j=1}^{K} 2\,H_{z_j}(s,a)\,\delta_j(s,a,r,s') - H_{z_j}(s,a)^2\right], \text{ where } \delta_j \text{ is a function of } \theta_{j-1} \text{ and } \theta_j, \\
&= \begin{cases}
2\,H_{z_1}(s,a)\,\partial_{\theta_0}\delta_1(s,a,r,s') & \text{if } k=0, \\
2\,H_{z_k}(s,a)\,\partial_{\theta_k}\delta_k(s,a,r,s') + 2\,H_{z_{k+1}}(s,a)\,\partial_{\theta_k}\delta_{k+1}(s,a,r,s') & \text{if } k\in\{1,\dots,K-1\}, \\
2\,H_{z_K}(s,a)\,\partial_{\theta_k}\delta_K(s,a,r,s') & \text{if } k=K,
\end{cases} \\
&= \begin{cases}
2\,H_{z_1}(s,a)\,\partial_{\theta_0}\left(r+\gamma\max_{a'}Q_{\theta_0}(s',a')\right) & \text{if } k=0, \\
-2\,H_{z_k}(s,a)\,\partial_{\theta_k}Q_{\theta_k}(s,a) + 2\,H_{z_{k+1}}(s,a)\,\partial_{\theta_k}\left(r+\gamma\max_{a'}Q_{\theta_k}(s',a')\right) & \text{if } k\in\{1,\dots,K-1\}, \\
-2\,H_{z_K}(s,a)\,\partial_{\theta_K}Q_{\theta_K}(s,a) & \text{if } k=K,
\end{cases} \\
\Delta_{z_k} &= -\partial_{z_k}\left[\sum_{j=1}^{K}\left(2\,H_{z_j}(s,a)\,\delta_j(s,a,r,s') - H_{z_j}(s,a)^2\right)\right] \\
&= -\left[2\,\delta_k(s,a,r,s')\,\partial_{z_k}H_{z_k}(s,a) - 2\,H_{z_k}(s,a)\,\partial_{z_k}H_{z_k}(s,a)\right] \\
&= 2\,\left(H_{z_k}(s,a) - \delta_k(s,a,r,s')\right)\partial_{z_k}H_{z_k}(s,a) \\
&= \partial_{z_k}\left[H_{z_k}(s,a) - \delta_k(s,a,r,s')\right]^2 \\
&= \partial_{z_k}\left[H_{z_k}(s,a) - \left(r+\gamma\max_{a'}Q_{\theta_{k-1}}(s',a') - Q_{\theta_k}(s,a)\right)\right]^2.
\end{aligned}$$

Finally, we reduce the influence of the learned estimates $(H_{z_k})_{k=1}^K$ on the update rule of the action-value function parameters. For that, following the TDC algorithm (Sutton et al., 2009), we incorporate a gradient correction term in Δ_{θ_k}. Additionally, we use a weight decay penalty regulated with the coefficient β in Δ_{z_k}, as advocated in Ghiassian et al. (2020).

We also freeze the first parameters θ_0. Indeed, K can only be set to a small value due to memory constraints. This value is generally far lower than the number of Bellman iterations required to ensure sufficient reward propagation. This means that target updates have to be implemented by setting $\theta_k \leftarrow \theta_{k+1}$, and $z_k \leftarrow z_{k+1}$ every T steps. By freezing the first action-value function in the sequence, we ensure it stays in place at the position that has been learned, and is not influenced by the following functions in the sequence that are still being optimized.

When incorporating gradient correction, weight decay penalty, and a frozen θ_0, we obtain the gradient estimates presented in Section 5:

$$\Delta_{\theta_k} = \begin{cases} -\left(r + \gamma \max_{a'} Q_{\theta_{k-1}}(s', a') - Q_{\theta_k}(s, a)\right) \partial_{\theta_k} Q_{\theta_k}(s, a) \\ \quad + H_{z_{k+1}}(s, a)\, \partial_{\theta_k}\left(r + \gamma \max_{a'} Q_{\theta_k}(s', a')\right) & \text{if } k \in \{1, \dots, K-1\}, \\ -\left(r + \gamma \max_{a'} Q_{\theta_{k-1}}(s', a') - Q_{\theta_k}(s, a)\right) \partial_{\theta_K} Q_{\theta_K}(s, a) & \text{if } k = K, \end{cases}$$

$$\Delta_{z_k} = \partial_{z_k} \left(H_{z_k}(s, a) - \left(r + \gamma \max_{a'} Q_{\theta_{k-1}}(s', a') - Q_{\theta_k}(s, a)\right)\right)^2 + \beta\, \partial_{z_k} \|z_k\|_2^2,$$

where a factor of 2 is ignored in all terms of Δ_{θ_k} for clarity.

Finally, we remark that H_{z_1} does not appear thanks to the gradient correction and the fact that θ_0 is frozen. Therefore, only $K - 1$ H-parameters are needed to compute the update rule.

B Schematic Representations

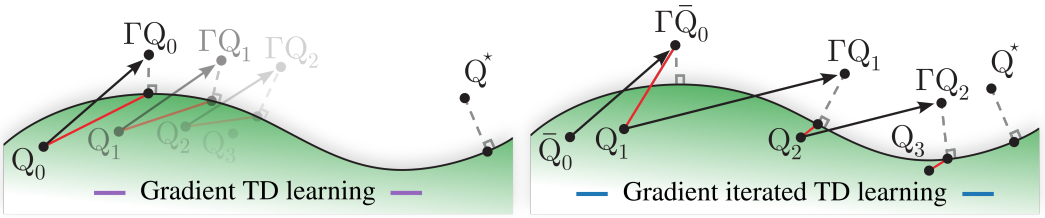

Figure 9: **Left**: As opposed to Figures 1 (right), which shows the case where Gradient TD learning minimizes the Bellman error, this figure depicts the case where Gradient TD learning minimizes the projected Bellman error. This case happens when the classes of function approximators for H-networks and Q-networks are equal. **Right**: Similarly, in this figure, gradient iterated TD learning minimizes the sum of projected Bellman errors, and not the sum of Bellman errors as shown in Figure 2 (right). Importantly, the first Bellman error is not projected as $\bar{Q}_0$ is kept frozen.

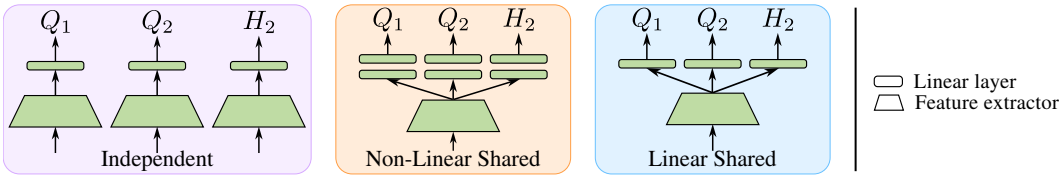

Figure 10: Different architecture choices. **Left**: Each action-value function or functions H uses a separate network. **Center**: All action-value functions and functions H use a shared feature extractor with heads that are composed of two linear layers with a ReLU function in between. **Right**: All action-value functions and functions H use a shared feature extractor with linear heads.

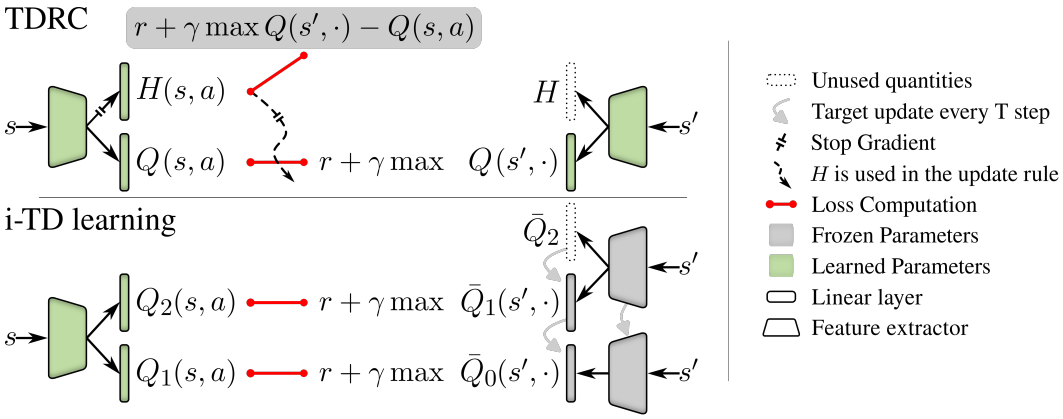

Figure 11: **Top**: Training procedure of **TDRC** with a shared architecture, and 2 heads. The 2 heads represent the action-value function Q, and the H-prediction H. The Q-head regresses a target built from itself. The H-head regresses the difference between the regression target and the Q-head. **Bottom**: Training procedure for **i-TD learning** for $K = 2$, with shared architecture. Each Q_k regresses a target built from Q_{k-1}. For building regression targets, parameters are kept frozen, depicted in grey on the right-hand side.

C Algorithmic Details

Our codebase relies on the JAX framework (Bradbury et al., 2018), the Adam optimizer (Kingma & Ba, 2015), and the NumPy library (Harris et al., 2020). It is available at:

- https://github.com/theovincent/Gi-DQN, for the experiments with DQN.
- https://github.com/theovincent/Gi-SAC, for the experiments with SAC.
- https://github.com/theovincent/Gi-CQL, for the experiments with CQL.

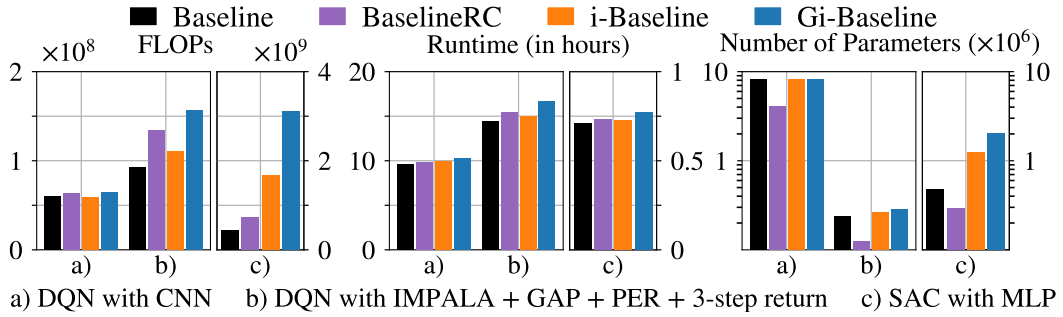

Figure 12: Comparison of computational requirements between TD, TDRC, i-TD, and Gi-TD learning. The comparison is made for the 2 different architectures used for the online Atari experiment, and for the Multi-Layer Perceptron (MLP) used for the MuJoCo experiment. All algorithms use a shared feature extractor on top of linear heads, except for i-SAC and Gi-SAC, which use independent networks. We do not report the metrics for the offline experiments as they are similar to those of the online experiments. **Left**: Number of Floating-Points Operations (FLOPs) required for each gradient step. As expected, gradient TD methods require more FLOPs as they compute the gradient of the target. **Center**: Training time of the different methods computed on an NVIDIA GeForce RTX 4090 with the game Breakout for the DQN experiments, and with the task Ant for the SAC experiments. Importantly, while a difference is noticeable in terms of FLOPs, the training times are similar. Gi-TD only requires a few minutes more than the other methods. **Right**: When sharing parameters, Gi-TD uses a similar amount of parameters compared to the other methods.

Algorithm 1 *Gradient iterated* Deep Q-Network (Gi-DQN). Modifications to DQN are in blue.

Initialize K Q-network parameters $(\theta_k)_{k=1}^K$, $K-1$ H-network parameters $(z_k)_{k=2}^K$, and an empty replay buffer $\mathcal{D}$. Initialize the frozen target Q-network parameters $\theta_0 \leftarrow \theta_1$.
Repeat
  Take action $a_t \sim \epsilon$-greedy$(\frac{1}{K}\sum_{k=1}^K Q_{\theta_k})$; Observe reward r_t, next state s_{t+1}.
  Update $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(s_t, a_t, r_t, s_{t+1})\}$.
  for UTD **updates**
    Sample a mini-batch $\mathcal{B} = \{(s, a, r, s')\}$ from $\mathcal{D}$.
    Set $\delta_k(d) = r + \gamma \max_{a'} Q_{\theta_{k-1}}(s', a') - Q_{\theta_k}(s, a)$, for $d \in \mathcal{B}$, and $k \in \{1, \dots, K\}$
    Compute the losses, for $d \in \mathcal{B}$: ▷ $\lceil \cdot \rceil$ indicates a stop gradient operation.

$$\mathcal{L}_Q(d) = \sum_{k=1}^{K-1} \Big[(r + \gamma \max_{a'} Q_{\theta_k}(s', a')) \cdot \lceil H_{z_{k+1}}(s, a) \rceil - Q_{\theta_k}(s, a) \cdot \lceil \delta_k(d) \rceil \Big] - Q_{\theta_K}(s, a) \cdot \lceil \delta_K(d) \rceil$$

$$\mathcal{L}_H(d) = \sum_{k=2}^{K} [\lceil \delta_k(d) \rceil - H_{z_k}(s, a)]^2 + \beta ||z_k||_2^2.$$

    Update θ_k with $\nabla_{\theta_k} \sum_{d \in \mathcal{B}} \mathcal{L}_Q(d)$, for $k \in \{1, \dots K-1\}$.
    Update z_k with $\nabla_{z_k} \sum_{d \in \mathcal{B}} \mathcal{L}_H(d)$, for $k \in \{2, \dots, K-1\}$.
  every T **steps**
    $\theta_k \leftarrow \theta_{k+1}$, for $k \in \{0, \dots, K-1\}$.
    $z_k \leftarrow z_{k+1}$, for $k \in \{2, \dots, K-1\}$.

Algorithm 2 *Gradient iterated* Soft Actor Critic (Gi-SAC). Modifications to SAC are in blue.

Initialize the policy network parameters ϕ, $2 \times K$ critic-network parameters $(\theta_k^1, \theta_k^2)_{k=1}^K$, $2\times$ $(K-1)$ H-network parameters $(z_k^1, z_k^2)_{k=2}^K$, an empty replay buffer $\mathcal{D}$, and α the entropy coefficient. Initialize the 2 frozen target critic-networks parameters $\theta_0^i \leftarrow \theta_1^i$ for $i = 1, 2$.
Repeat
  Take action $a_t \sim \pi_\phi(\cdot|s_t)$; Observe reward r_t, next state s_{t+1}.
  Update $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(s_t, a_t, r_t, s_{t+1})\}$.
  for UTD **updates**
    Sample a mini-batch $\mathcal{B} = \{(s, a, r, s')\}$ from $\mathcal{D}$.

$$\delta_k^i(d) = r + \gamma \min_{j \in \{1,2\}} Q_{\theta_{k-1}^j}(s', a') - \gamma\alpha \log \pi_\phi(a'|s') - Q_{\theta_k^i}(s, a),$$

    where $a' \sim \pi_\phi(\cdot|s_t)$, for $d \in \mathcal{B}$, $k \in \{1, \dots, K\}$, and $i \in \{1, 2\}$.
    Compute the critic losses, for $d \in \mathcal{B}$, and $i \in \{1, 2\}$: ▷ $\lceil \cdot \rceil$ indicates stop gradient.

$$\mathcal{L}_Q^i(d) = \sum_{k=1}^{K-1} \Big[\Big(r + \gamma \min_{j \in \{1,2\}} Q_{\theta_{k-1}^j}(s', a') - \gamma\alpha \log \pi_\phi(a'|s') \Big) \cdot \lceil H_{z_{k+1}^i}(s, a) \rceil - Q_{\theta_k^i}(s, a) \cdot \lceil \delta_k^i(d) \rceil \Big] - Q_{\theta_K^i}(s, a) \cdot \lceil \delta_K^i(d) \rceil, \text{ where } a' \sim \pi_\phi(\cdot|s_t)$$

$$\mathcal{L}_H^i(d) = \sum_{k=2}^{K} \Big[\lceil \delta_k^i(d) \rceil - H_{z_k^i}(s, a) \Big]^2 + \beta ||z_k^i||_2^2.$$

    Compute the actor loss, for $d \in \mathcal{B}$:

$$\mathcal{L}_\pi(d) = \min_{i \in \{1,2\}} \frac{1}{K} \sum_{k=1}^{K} Q_{\theta_k^i}(s, a) - \alpha \log \pi_\phi(a|s),$$

    where $a \sim \pi_\phi(\cdot|s)$, for $k \in \{1, \dots K-1\}$, and $i \in \{1, 2\}$.
    Update θ_k^i with $\nabla_{\theta_k} \sum_{d \in \mathcal{B}} \mathcal{L}_Q^i(d)$, for $k \in \{1, \dots K-1\}$, and $i \in \{1, 2\}$.
    Update z_k^i with $\nabla_{z_k} \sum_{d \in \mathcal{B}} \mathcal{L}_H^i(d)$, for $k \in \{1, \dots K-1\}$, and $i \in \{1, 2\}$.
    Update ϕ from $\nabla_\phi \sum_{d \in \mathcal{B}} \mathcal{L}_\pi(d)$.
    Update $\theta_0^i \leftarrow \tau\theta_1^i + (1-\tau)\theta_0^i$, for $i \in \{1, 2\}$.

```python
import jax
import jax.numpy as jnp


def loss_on_single_sample(params, target_params, sample):
    # Compute (Q_1, ..., Q_K), (H_2, ..., H_K) for state s,
    # with shape (K, n_actions), (K-1, n_actions).
    q_s, h_s = network.apply(params, sample.state)

    # Compute Q_0 for next state s', with shape (1, n_actions).
    q_first_next_s = target_network.apply(target_params, sample.next_state)

    # Compute (Q_1, ..., Q_K) for next state s', with shape (K, n_actions).
    q_following_next_s, _ = network.apply(params, sample.next_state)

    q_next_s = jnp.concatenate(q_first_next_s, q_following_next_s[:-1])
    targets = sample.reward + gamma * q_next_s.max(axis=1)
    td_errors = targets - q_s[:, sample.action]

    loss_targets = targets[1:] * jax.lax.stop_gradient(h_s)
    loss_online = -q_s[:, sample.action] * jax.lax.stop_gradient(td_errors)
    loss_h = jnp.square(h_s[:, sample.action] - jax.lax.stop_gradient(td_errors[1:]))

    return loss_targets.sum() + loss_online.sum() + loss_h.sum()
```

Figure 13: **Loss computation in JAX** of Gi-DQN for a single sample. Importantly, the loss for the Q-networks and H-networks can be computed in a single function using vector operations. The variable `network` is a neural network instantiated to output the values for all the action-value functions except the first one, $\bar{Q}_0$, and the H-values. The weights parameterizing this network are called `params`. The variable `target_network` outputs the values representing the first action-value function $\bar{Q}_0$. It uses the weights `target_params`.

D Experiment Setup

Triangle MP Setup. The Triangle MP is presented in Figure 14. It is composed of 3 states. The space of function approximation is a spiral lying on a plane in the space of value function. For a given parameter ω, the associated value function is defined by

$$\begin{pmatrix} V_\omega(1) \\ V_\omega(2) \\ V_\omega(3) \end{pmatrix} = e^{0.15\,\omega} \cdot \left[\cos(0.866\,\omega) \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} - \sin(0.866\,\omega) \frac{\epsilon}{\sqrt{3}} \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \right],$$

where ϵ determines the direction of rotation. ϵ is equal to -1 in Figure 5 (left), and 1 in Figure 5 (right).

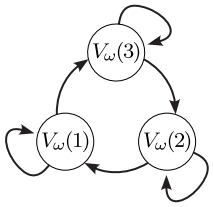

Figure 14: Triangle Markov Process, consisting of three states. The value each state i is approximated by $V_\omega(i)$, for some parameters ω.

Atari Setup. We build our codebase following Machado et al. (2018) standards and taking inspiration from Castro et al. (2018) and Vincent et al. (2025b) codebases[2]. Namely, we use the *game over* signal to terminate an episode instead of the life signal. The input given to the neural network is a concatenation of 4 frames in grayscale of dimension 84 by 84. To get a new frame, we sample 4 frames from the Gym environment (Brockman et al., 2016) configured with no frame-skip, and apply a max pooling operation on the 2 last grayscale frames. We use sticky actions to make the environment stochastic (with $p = 0.25$).

Atari Games Selection. Our evaluations are performed on 10 games, which are chosen to represent a large variety of human-normalized scores, reached by DQN agent trained on 200 million frames, as shown in Figure 15. The human-normalized scores are computed from human and random scores that are reported in Schrittwieser et al. (2020). For the of-

[2] https://github.com/google/dopamine, and https://github.com/slimRL/slimDQN

fline experiment, we used the datasets provided by Gulcehre et al. (2020). The codebase for the offline experiments is adapted from the code released by Vincent et al. (2025a)[3].

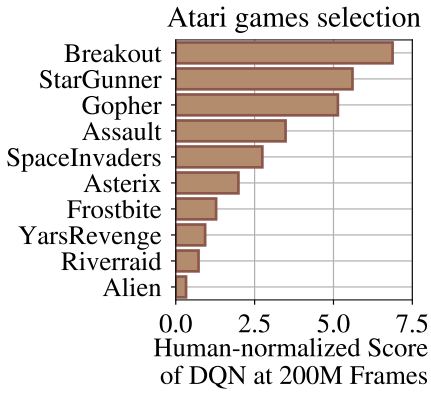

Figure 15: Human-normalized score of 10 Atari games obtained by a DQN agent after collecting for 200 million frames.

MuJoCo Setup. Our codebase is a fork from the codebase released by Vincent et al. (2025a)[4]. It relies on the Gym library (Towers et al., 2026).

Performance Aggregation. In most figures in this work, we aggregate results over seeds and environments to account for the stochasticity of the training process and the variation in the training dynamics over the different environments. For that, we first extract the undiscounted returns collected during training for each seed. Then, we smooth the signal for readability by replacing each value with the average over a window of size 5 that covers the neighboring collected returns. We note the resulting metric for an algorithm a at timestep t, for a seed i, and an environment j, $s^a_{i,j}(t)$.

We stress that the collected returns are not those obtained during a separate evaluation phase, as focusing on the returns obtained during training is closer to the initial motivation behind online learning (Machado et al., 2018).

We leverage the Inter-Quantile Mean (Agarwal et al., 2021) to aggregate the collected returns. Before computing the IQM over all runs, we normalize each collected return $s^a_{i,j}(t)$ by the average end performance of the baseline in the same environment. Therefore, the learning curve report for each timestep t, $\text{IQM}\left(\left\{\frac{s^a_{i,j}(t)}{\frac{1}{\text{n\_seeds}}\sum_i s^b_{i,j}(L)}, \text{for all } i \text{ and } j\right\}\right)$, where L is the last timestep of the training. The computation of the 95% stratified bootstrap confidence intervals follows the same methodology.

We also report the normalized IQM Area Under the Curve (AUC). For that, we compute the normalized scores as described before, sum them across all timesteps, and compute the IQM. We then divide the obtained values by the baseline IQM AUC to facilitate comparison.

[3] https://github.com/slimRL/slimCQL
[4] https://github.com/slimRL/slimSAC

Table 1: Summary of hyperparameters used for the Atari experiments. We note $\text{Conv}_{a,b}^{d}C$ a 2D convolutional layer with C filters of size $a \times b$ and of stride d, and FC E a fully connected layer with E neurons.

Environment	
Discount factor γ	0.99
Horizon	27 000
Full action space	No
Reward clipping	clip$(-1, 1)$
Shared hyperparameters	
Batch size	32
Torso architecture CNN	$\text{Conv}_{8,8}^{4}32$ $-\text{Conv}_{4,4}^{2}64$ $-\text{Conv}_{3,3}^{1}64$
Torso architecture IMPALA	$\text{Conv}_{8,8}^{4}16$ $-\text{Conv}_{4,4}^{2}32$ $-\text{Conv}_{3,3}^{1}32$
Head architecture	FC 512 $-$FC $n_{\mathcal{A}}$
Activations	ReLU
Online Training	
Target update period T	8 000
Initial number of samples	20 000
Maximum replay buffer capacity	1 000 000
Number of training steps per epoch	250 000
Starting ϵ	1
Ending ϵ	10^{-2}
ϵ linear decay duration	250 000
Learning rate	6.25×10^{-5}
Adam ϵ	1.5×10^{-4}
Offline Training	
Target update period T	2 000
CQL α	0.1
Learning rate	5×10^{-5}
Adam ϵ	3.125×10^{-4}
Number of training steps per epoch	62 500
Replay buffer size	5 000 000

Table 2: Summary of all hyperparameters used for the MuJoCo experiments. We note FC E a fully connected layer with E neurons.

Environment	
Discount factor γ	0.99
Horizon	1 000
Shared Hyperparameters	
Initial number of samples	5 000
Maximum replay buffer capacity	1 000 000
Batch size	256
Learning rate	10^{-3}
Policy delay	1
Actor and critic architecture	FC 256 $-$FC 256
Soft-target update τ	5×10^{-3}
Adam ϵ	10^{-8}

Table 3: Summary of hyperparameters used for the **offline** ablations on LunarLander **(LL)** and MountainCar **(MC)**. We note FC E a fully connected layer with E neurons.

Environments	
Discount factor γ	0.99
Horizon	500 (LL) 200 (MC)
Offline Training	
CQL α	0.1
Learning rate	5×10^{-4}
Adam ϵ	1×10^{-8}
Number of training steps per epoch	10 000
Batch size	32
Torso architecture	FC 50 -FC 50
Head architecture	-FC 50 -FC $n_{\mathcal{A}}$

Sample collection with DQN	
Number of initial samples	1 000
Number of training steps per epoch	10 000
Maximum replay buffer capacity	10 000
Target update period	100 (LL) 200 (MC)
Batch Size	64 (LL), 32 (MC)
UTD	1
Starting ϵ	1
Ending ϵ	10^{-2}
Features	FC 200 (LL) FC 50 (MC)
Torso architecture	-FC 200 (LL) -FC 50 (MC) -FC $n_{\mathcal{A}}$
Learning rate	3×10^{3} (LL) 3×10^{4} (MC)

E Additional Experiments

E.1 Scaling Up the Replay Ratio in Continuous Actions Domains

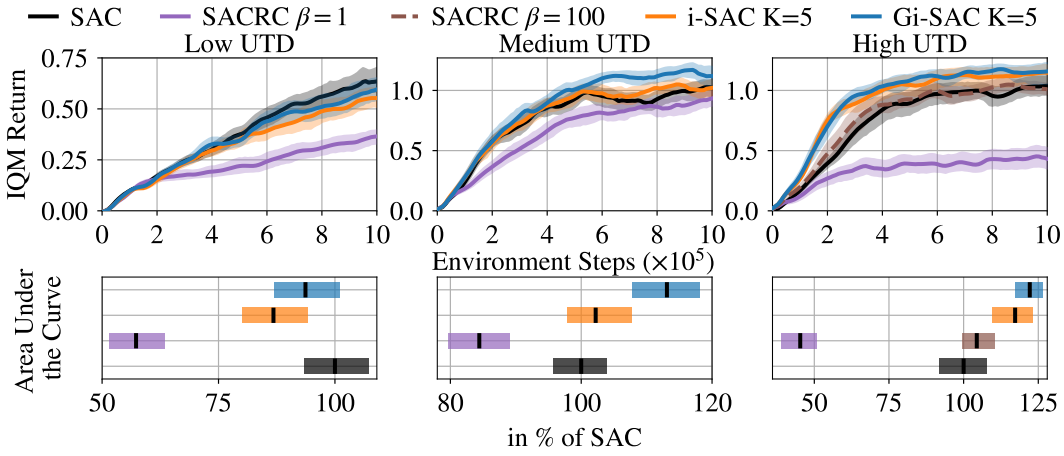

Figure 16: Evaluating the proposed approach on the **MuJoCo** benchmark with different **update-to-data (UTD) ratios** (UTD $\in \{0.1, 1, 10\}$). All algorithms are normalized with respect to SAC for medium UTD to ease comparison. We note that all algorithms diverge for the Humanoids and HumanoidStandUp tasks on the high UTD setting. This is why we aggregate the results over the 4 remaining MuJoCo tasks (Hopper, Ant, HalfCheetah, and Walker2d). We believe that this divergence is due to the representational capacity of the neural network being too small to cope with high UTD ratios. Importantly, Gi-SAC is competitive against semi-gradient methods, especially for high UTD ratios.

E.2 Additional Ablation Studies

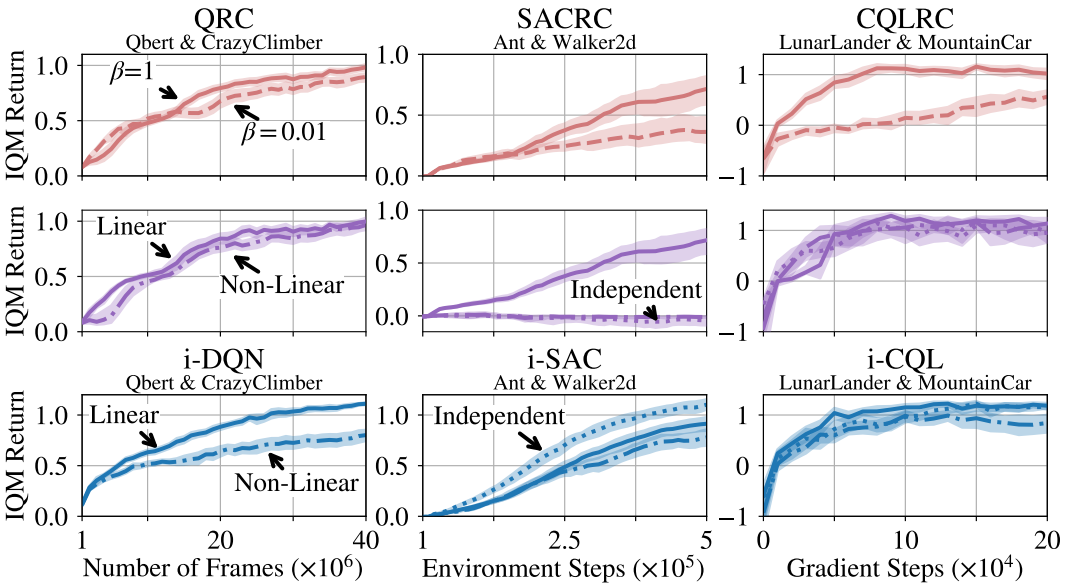

Figure 17: **Top row**: **Ablation study** on the **weight decay coefficient** β for TDRC. TDRC performs better with a high value of β indicating that weight decay is an important component of the algorithm, as suggested in Ghiassian et al. (2020). **Center row**: **Ablation study** on the choice of the **architecture** for TDRC. Using linear heads on top of a shared feature extractor performs better than the 2 other considered architectures. **Bottom row**: **Ablation study** on the choice of the **architecture** for i-TD learning. Using linear heads on top of a shared feature extractor performs better than the 2 other considered architectures, except for the experiment on continuous action-spaces, where the independent architecture performs better.
Overall, we stress that each learning curve is aggregated over 2 values of UTD (0.25 and 1 for DQN experiments, 1 and 4 for SAC experiments) or 2 dataset sizes for CQL experiments (10% and 100%), and 2 target update periods ($T = 1000$ and 8000 for DQN experiments, $T = 10$ and 1000 for CQL experiments) or 2 values of soft-target updates for SAC experiments ($\tau = 0.005$ and 0.01). The learning curves for ablation on the weight decay coefficient are also aggregated over the 3 archicture designs, except for SACRC, for which we only use the linear shared architecture as the other architecture perform poorly. β is set to 1 for the ablation on the architecture designs.

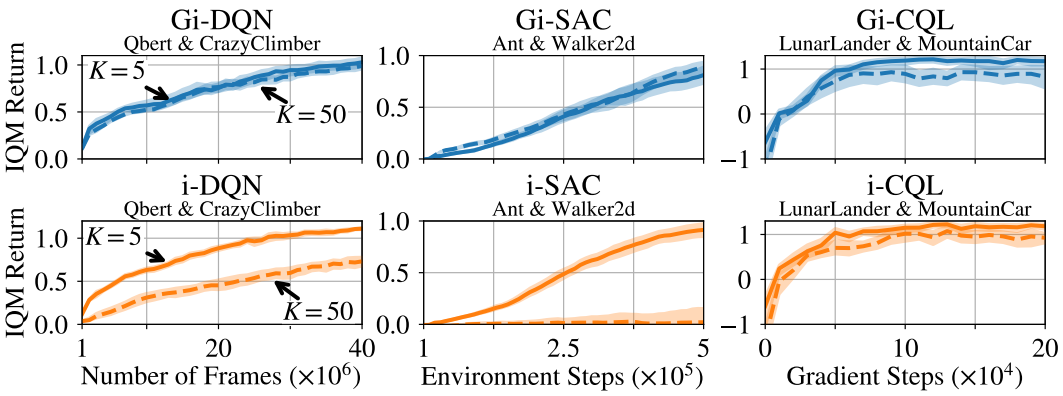

Figure 18: **Ablation study** on the **number of Bellman iterations** K learned in parallel for i-TD (**bottom**) and Gi-TD (**top**) learning. Remarkably, while a performance drop is noticeable when increasing K to 50 for i-TD learning, Gi-TD performs similarly for the 2 considered values of K. This indicates that Gi-TD learning is less sensitive to this hyperparameter than i-TD learning.

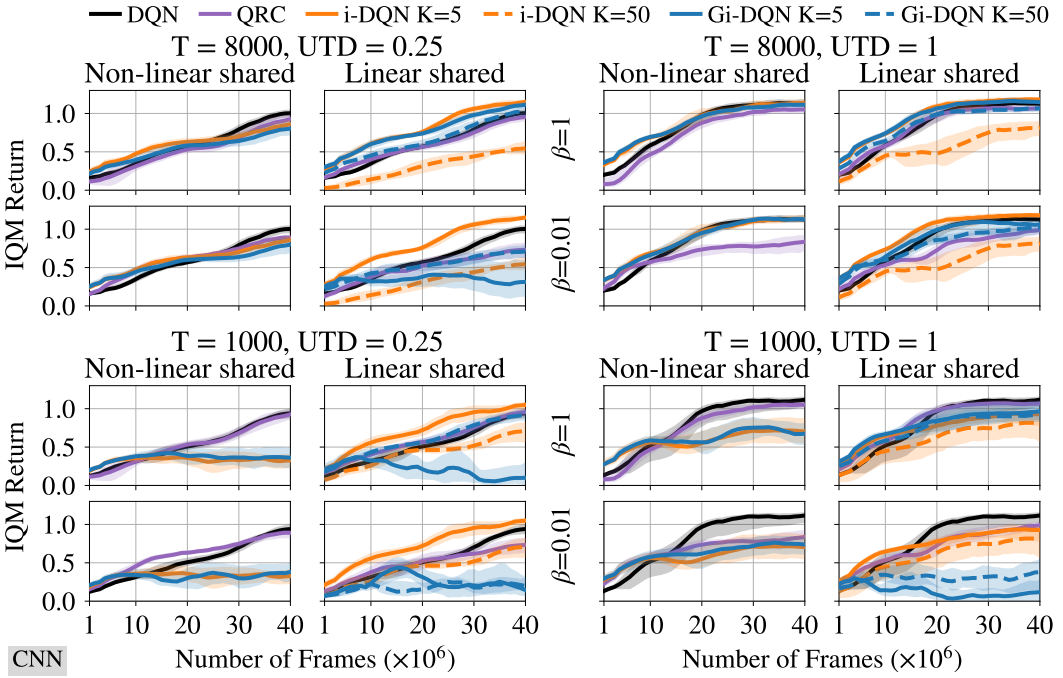

Figure 19: **Ablation study** over the **weight decay coefficient** β, and the choice of the **architecture** on 2 **Atari** games (Qbert, and CrazyClimber). We also vary the target update period T, and update-to-data (UTD) ratio to better grasp the dependency of the algorithms on the studied hyperparameters. DQN with T= 8000, and UTD= 0.25 is used to normalize the performance curves.

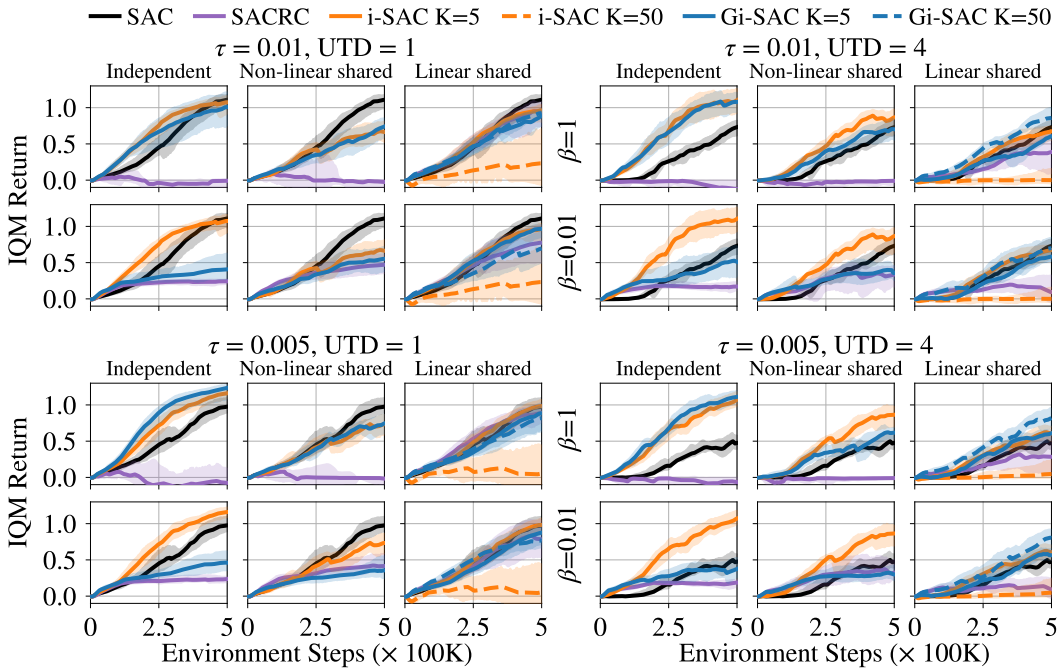

Figure 20: **Ablation study** over the **weight decay coefficient** β, and the choice of the **architecture** on 2 **MuJoCo** tasks (Ant, and Walker2d). We also vary the soft-update coefficient τ, and update-to-data (UTD) ratio to better grasp the dependency of the algorithms on the studied hyperparameters. SAC with $\tau = 0.005$, and UTD$= 1$ is used to normalize the performance curves.

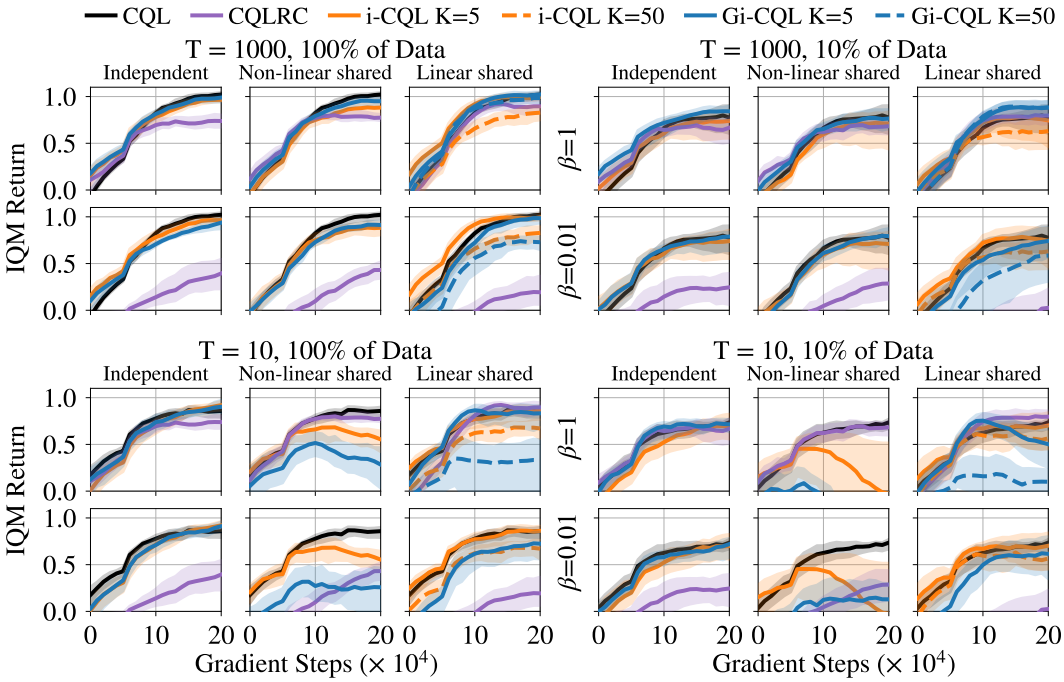

Figure 21: **Ablation study** over the **weight decay coefficient** β, and the choice of the **architecture** on 2 **Gym** environment (LunarLander, and MountainCar). We also vary the target update period T, and the percentage of dataset given to the learning algorithm to better grasp the dependency of the algorithms on the studied hyperparameters. The CQL version with T$= 1000$ that has access to 100% of the dataset is used to normalize the performance curves.

F Individual Learning Curves

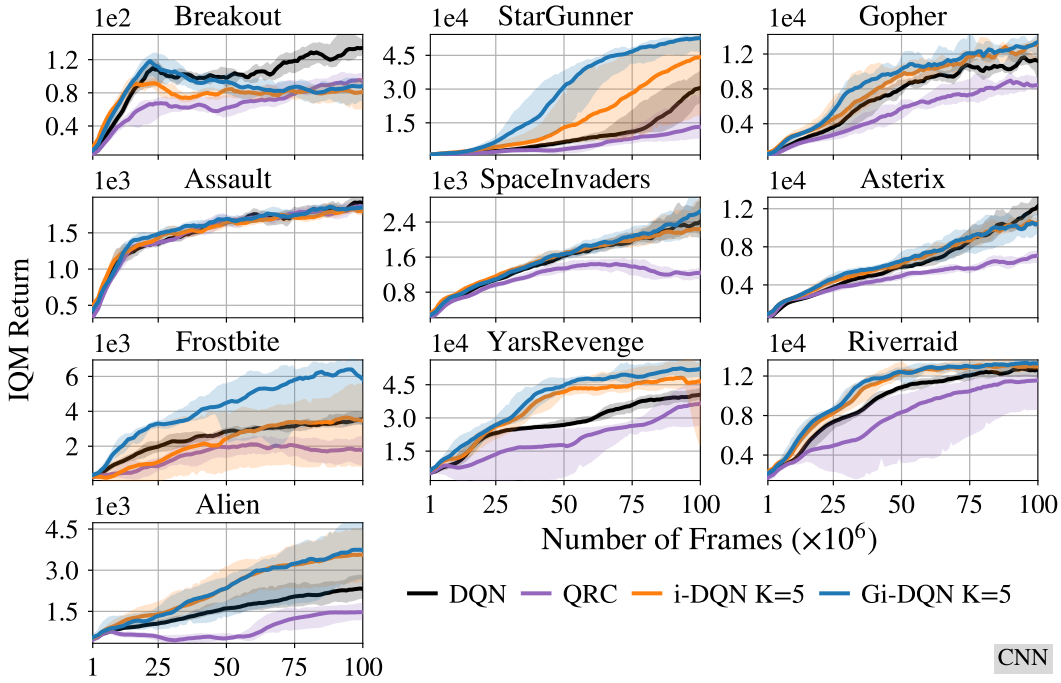

Figure 22: Per **Atari** game performance for DQN, QRC, i-DQN, and Gi-DQN, in an **online** learning setting, with the CNN architecture. Except for Breakout, Gi-DQN generally performs better than or is on par with the 3 other algorithms.

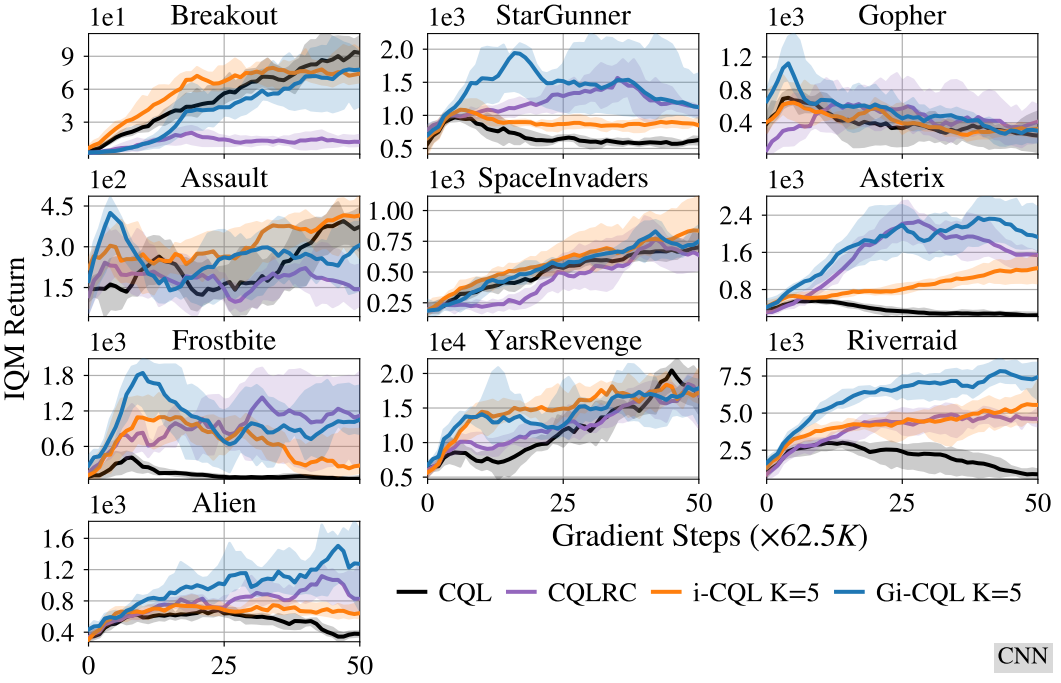

Figure 23: Per **Atari** game performance for CQL, CQLRC, i-CQL, and Gi-CQL, in an **offline** learning setting, with the CNN architecture. Except for Breakout and Assault, Gi-CQL generally performs better than or is on par with the 3 other algorithms.

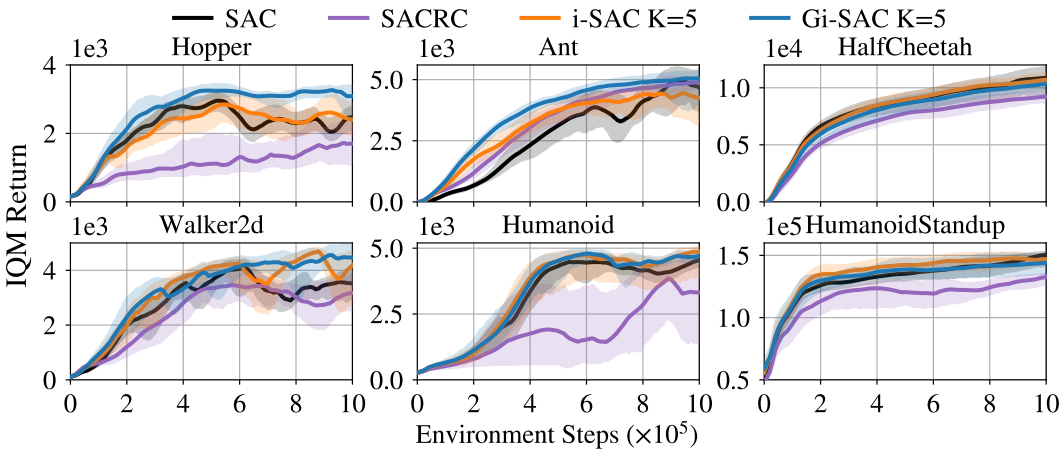

Figure 24: Per **MuJoCo** task performance for SAC, SACRC, i-SAC, and Gi-SAC, in an **online** learning setting, with a 2-layer MLP architecture. Gi-SAC learns more reliably across tasks than SACRC, while remaining at least competitive with i-SAC, if not outperforming it.

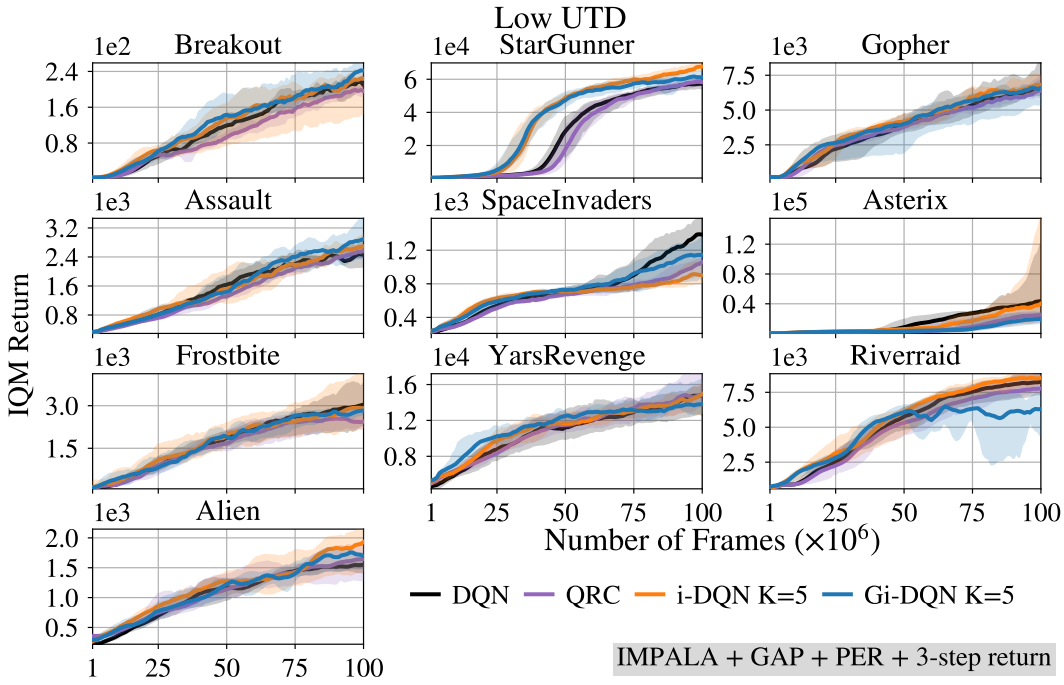

Figure 25: Per **Atari** game performance for DQN, QRC, i-DQN, and Gi-DQN, in an **online** learning setting, with the CNN architecture, and a UTD ratio of $\frac{1}{32}$. All algorithms use the IMPALA architecture with global average pooling (GAP), a prioritized experience replay buffer (PER), and a 3-step return. Due to the low UTD value, all algorithms perform similarly.

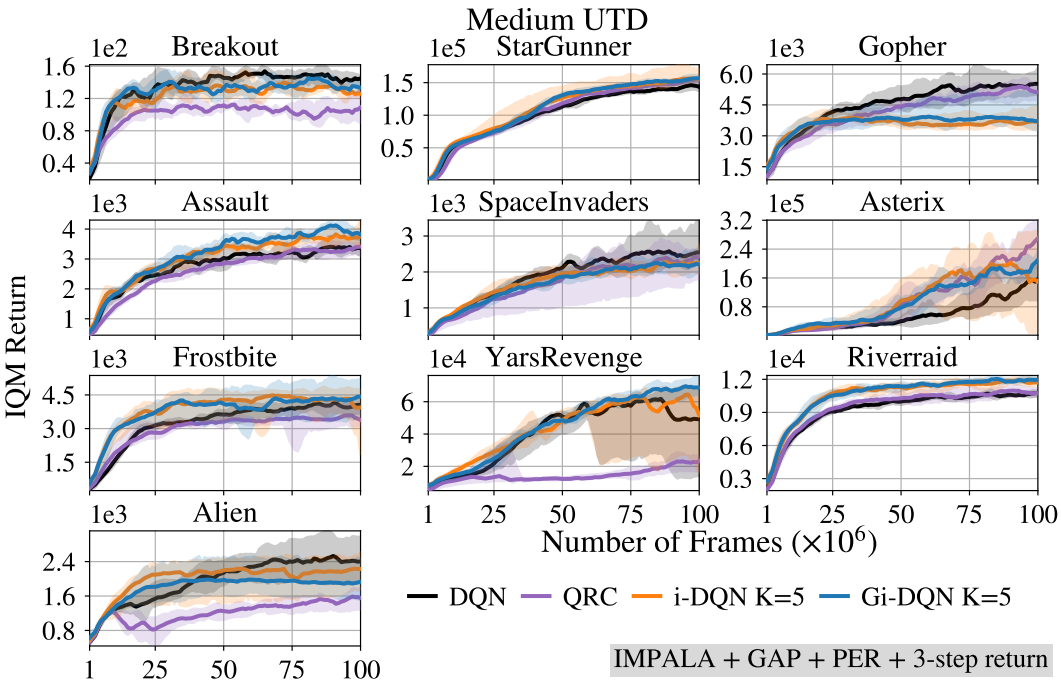

Figure 26: Per **Atari** game performance for DQN, QRC, i-DQN, and Gi-DQN, in an **online** learning setting, with the CNN architecture, and a UTD ratio of $\frac{1}{4}$. Gi-DQN remains competitive against semi-gradient methods even when the baseline algorithm is equipped with the IMPALA architecture with global average pooling (GAP), a prioritized experience replay buffer (PER), and a 3-step return.

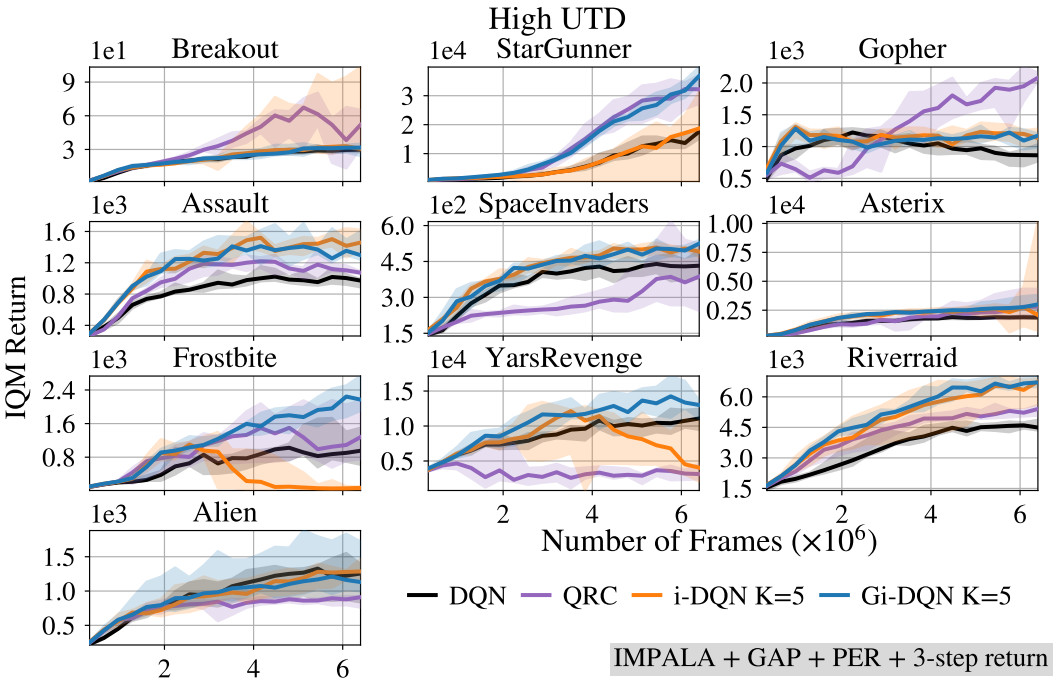

Figure 27: Per **Atari** game performance for DQN, QRC, i-DQN, and Gi-DQN, in an **online** learning setting, with the CNN architecture, and a UTD ratio of 4. All algorithms use the IMPALA architecture with global average pooling (GAP), a prioritized experience replay buffer (PER), and a 3-step return. Except for Gopher, Gi-DQN provides a significant boost in learning speed compared to the other algorithms.

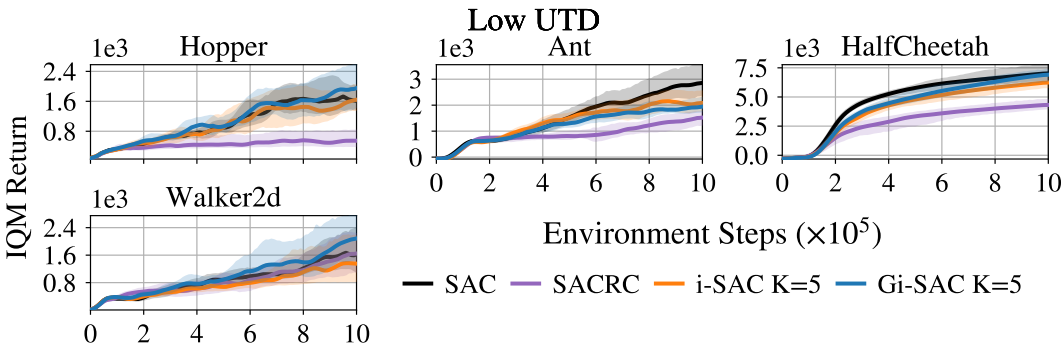

Figure 28: Per **MuJoCo** task performance for SAC, SACRC, i-SAC, and Gi-SAC, in an **online** learning setting, with a 2-layer MLP, and a UTD ratio of 0.1. SAC, i-SAC, and Gi-SAC reach similar performance.

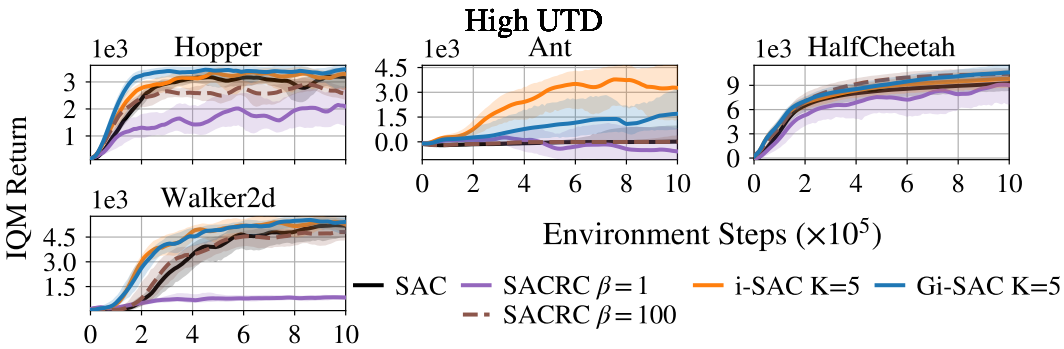

Figure 29: Per **MuJoCo** task performance for SAC, SACRC, i-SAC, and Gi-SAC, in an **online** learning setting, with a 2-layer MLP, and a UTD ratio of 10. For SACRC, an additional experiment with a higher value of β is conducted, since the algorithm underperforms with $\beta = 1$. Except for Ant, Gi-SAC performs competitively against i-SAC.