POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Dynamic Control Frequency in Reinforcement Learning through Action Persistence

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Luca Al Daire

Advisor: Prof. Marcello Restelli

Co-advisors: Dott. Lorenzo Bisi, Dott. Alberto Maria Metelli, Dott. Luca Sabbioni

Academic year: 2020-2021

1. Introduction

In recent years, Reinforcement Learning (RL, [24]) methods have proven to be successful in a wide variety of applications, where sequential decision-making problems are typically modelled as a Markov Decision Process (MDP, [19]), a formalism that addresses the agent-environment interactions through *discrete-time* transitions. *Continuous-time* control problems, instead, are usually addressed by means of time discretization, which induces a specific control frequency f, or, equivalently, a time step $\delta = \frac{1}{f}$ [17]. This represents an environment hyperparameter, which may have dramatic effects on the process of learning the optimal policy [14]. Indeed, higher frequencies allow for greater control opportunities, but they have significant drawbacks. The most relevant one is related to the toned down effect of the selected actions. In the limit for time discretization $\delta \to 0$, the advantage of each action collapses to zero, preventing the agent from finding the best action [25] and leading to higher sample complexity. Moreover, a random uniform policy played at high frequency may not be adequate for exploration, as it tends to visit only a local neighborhood of the initial state [17]. This is problematic, especially in goal-based or sparse rewards environments, where the most informative states may never be visited. On the other hand, large time discretizations benefit from a higher probability of reaching far states, but they also deeply modify the transition process, hence a possibly large subspace of states may not be reachable.

One of the solutions to achieve the advantages related to exploration and sample complexity, while keeping the control opportunity loss bounded, consists in *action persistence* [11, 14]. When the dynamics are very rapid, action repetition is equivalent to acting at lower frequencies. Thus, the agent can achieve, in some environments, a more effective exploration, better capture the consequences of each action, and, as a final consequence, learn the optimal policy faster.

In this work, we propose a value-based approach in which the agent does not only choose the *action*, but also its *persistence*, with the goal of making the most effective use of samples collected at different persistences. The information collected at *one* persistence is then

used to improve the action value function estimates of *all* the considered possible persistences, by decomposing the observed history in many sub-transitions of reduced length to update lower persistence values, and by using a suitable *bootstrapping* procedure of the missing information for higher persistences. This procedure is formalized with the introduction of the *All-persistence Bellman Operator*, which enjoys a contraction property analogous to that of the traditional optimal Bellman operator. This new operator is then embedded into the classic Q-learning algorithm, obtaining *Persistent Q-learning* (Per$Q$-learning). This novel algorithm, allowing for an effective use of the transitions sampled at different persistences, let us experience a faster convergence and fosters a better exploration of the state space. Furthermore, in order to deal with more complex domains, we consider the Deep RL scenario, extending the Deep Q-Network (DQN) algorithm to its persistent version *Persistent Deep Q-Network* (PerDQN). Finally, we evaluate the proposed algorithms, in comparison on both illustrative and complex domains, highlighting strengths and weaknesses.

2. Related Works

The first attempts to extend classical RL algorithms with the introduction of action persistence go back to 2003 [20]. In this paper, multi-step actions (MSAs), i.e., a sequence of repeated actions, were introduced, reducing the number of decisions needed to reach the goal and making the time scale coarser. Thus, MSAs extend Q-Learning by allowing each action to be repeated and the next decision is taken only at the end of the sequence. Action persistence has acquired practical relevance since the introduction of Deep RL [16], because of the frame skipping [2] for Atari games. Frame skipping consists in letting the environment evolve for multiple steps before observing the new state and performing a new action. Several works [5, 14] had shown the importance of persistence for helping exploration and policy learning. Among these works, [7] introduced an $\epsilon z-$greedy exploration, with a random exploratory variable deciding the duration of each action. As explained in [14], changing frequency deeply modifies the underlying MDP, as a special instance of a configurable MDP [13], where environmental parameters can be tuned to improve the performance. Indeed, in [9] the authors proposed an algorithm to automatically tune the control frequency, along with other learning hyperparameters. Mann, Mannor, and Precup [12] illustrate that approximate value iteration techniques can converge faster with action persistence (seen as *options* with longer duration).

Action repetition has many advantages, but it reduces the control opportunities. Consequently, researchers have been trying to include the possibility to *dynamically* change the control frequency during learning. In the context of Deep RL, [11] introduced the idea of enlarging the action space, duplicating actions and paring them with a specific repetition value. In the Augmented DQN, the last layer of the network is duplicated to output the Q-values for actions at two different repetition rates. The main drawback is that repetition rates are hyperparameters, hence there is no automatic adaptation of frequency.

Two recent algorithms proposed different approaches. In [21], two networks are employed: the first is a classic action policy, used to learn primitive actions over the environment, while the second one, called *skip network* is used to learn how many times the action persistence in a specific state, regardless of the chosen action. In this way it is not possible to tune the persistence for each action, but only to regulate it according to its averaged effect on the performance for the specific state.

One way to differentiate actions is introduced in a similar fashion with TempoRL [4]. Here, two different networks are employed: while the base one is a normal DQN, the skip network depends on both state and action and approximates Q-values for different possible frequencies. While the first network is used to choose the action to perform in a state, the state-action pair is then fed to the second network to choose skip value.

In the framework of policy-gradient methods, we can find examples of persistence in [28], with the introduction of secondary binary policy, with the main purpose of choosing whether to repeat the previous action or to change it according to the principal agent. A completely different approach is presented by [17]. The authors claim that when the $\delta \to 0$ policy-based methods tend to degrade (in a similar way as in [25] for Q-

learning). Persistence is a good solution, but in systems with unexpected events, it may lead to a loss of control. Their algorithm introduces the notion of *safe region*, as the agent keeps repeating an action until the distance of the states visited overcomes a certain threshold. This state locality can guarantee reactivity, especially in some environments where the blind repetition of an action can be dangerous.

3. Background

In this chapter we will introduce the main concepts about Reinforcement Learning. We define a Markov Decision Process, a mathematical framework widely adopted to describe the interaction between an agent and an environment.

3.1. Markov Decision Processes

A discrete-time Markov Decision Process (MDP,[19]) is defined as a tuple $\mathcal{M} \coloneqq \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the finite action space, $P : \mathcal{S} \times \mathcal{A} \to \mathscr{P}(\mathcal{S})$ is the Markovian transition kernel, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. A Markovian stationary policy $\pi : \mathcal{S} \to \mathscr{P}(\mathcal{A})$ maps states to probability measures over $\mathcal{A}$. We denote with Π the set of Markovian stationary policies. The *action-value function*, or Q-function, of a policy $\pi \in \Pi$ is the expected discounted sum of the rewards obtained by performing action a in state s and following policy π thereafter:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\Big[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = a\Big],$$

where $r_{t+1} = r(s_t, a_t)$, $a_t \sim \pi(\cdot|s_t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$ for all $t \in \mathbb{N}$.
The optimal Q-function is given by: $Q^{\star}(s, a) = \sup_{\pi \in \Pi} Q^{\pi}(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. An *optimal policy* $\pi^{\star} \in \Pi$ is any policy greedy w.r.t. $Q^{\star}$, i.e., $\pi^{\star}(\cdot|s) \in \mathscr{P}\left(\arg\max_{a \in \mathcal{A}} Q^{\star}(s, a)\right)$.

3.2. Q-learning

The *Bellman Optimal Operator* $T^{\star} : \mathscr{B}(\mathcal{S} \times \mathcal{A}) \to \mathscr{B}(\mathcal{S} \times \mathcal{A})$ is defined for every $f \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$ and $(s, a) \in \mathcal{S} \times \mathcal{A}$ as [3]:

$$(T^{\star} f)(s,a) = r(s,a) + \gamma \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) \max_{a' \in \mathcal{A}} f(s', a').$$

$T^{\star}$ is a γ-contraction in L_{∞}-norm and its unique fixed point is the optimal Q-function ($T^{\star}Q^{\star} = Q^{\star}$). When the P and r are known, the (action) value-iteration algorithm [19] allows to retrieve $Q^{\star}$ by means of the iterative application of $T^{\star}$. When the environment is unknown, Q-learning [27] collects samples with a *behavioral* policy (e.g., ϵ-greedy) and then uses them to update a Q-function estimate based on the updated rule:

$$Q(s_t,a_t) \leftarrow (1-\alpha) Q(s_t,a_t) + \alpha\left(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a')\right),$$

where $\alpha > 0$ is the learning rate.

3.3. Deep Q-Network

Classic Q-learning suffers from curse of dimensionality: in high dimensional state spaces, e.g. with images, the number of state-action pairs grows exponentially and the explicit computation of the action value function becomes unfeasible. Among the most famous approaches, Deep Q-Network (DQN, [15]) employs a function approximator $Q_{\boldsymbol{\theta}}(s, a)$ parameterized by a deep neural network with weights $\boldsymbol{\theta}$ to estimate $Q^{\star}$. Interactions with the environment are stored in the *replay buffer* $\mathcal{D} = \{(s_t, a_t, r_{t+1}, s_{t+1})\}_{t=1}^{n}$. To improve stability, a *target network*, whose parameters $\boldsymbol{\theta}^{-}$ are kept fixed for a certain number of steps, is employed. The Q-Network is trained to minimize the mean squared temporal difference error $r + \gamma \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}^{-}}(s', a') - Q_{\boldsymbol{\theta}}(s, a)$ on a batch of tuples sampled from the replay buffer.

3.4. Action Persistence

The execution of actions with a persistence $k \in \mathbb{N}$ can be modeled by means of the k-persistent MDP [14], characterized by the k-persistent transition model P_k and reward function r_k. To formally define them, the *persistent transition model* is introduced:

$$P^{\delta}(\cdot,\cdot|s,a) = \int_{\mathcal{S}} P(\mathrm{d}s'|s,a) \delta_{(s',a)}(\cdot,\cdot),$$

which replicates in the next state s' the previous action a. Thus, we have:

$$P_k(\cdot|s,a) = \Big((P^{\delta})^{k-1} P\Big)(\cdot|s,a),$$

$$r_k(s,a) = \sum_{i=0}^{k-1} \gamma^i \Big((P^{\delta})^i r\Big)(s,a).$$

This framework eases the analysis of *fixed* persistences, but it does not allow the action repetition for a *variable* number of steps.

4. All-Persistence Bellman Update

We introduce our approach to make effective use of the samples collected at *any* persistence. In our framework, the agent chooses a primitive action a together with its persistence k with the introduction of the *persistence option*.

Definition 4.1. *Let $\mathcal{A}$ be the space of* primitive *actions of an MDP $\mathcal{M}$ and $\mathcal{K} \coloneqq \{1, \dots, K_{\max}\}$, where $K_{\max} \geq 1$, be the set of persistences. A* persistence option *$o \coloneqq (a, k)$ is the decision of playing primitive action $a \in \mathcal{A}$ with persistence $k \in \mathcal{K}$. We denote with $\mathcal{O}^{(k)} \coloneqq \{(a, k) : a \in \mathcal{A}\}$ the set of options with fixed persistence $k \in \mathcal{K}$ and $\mathcal{O} \coloneqq \bigcup_{k \in \mathcal{K}} \mathcal{O}^{(k)} = \mathcal{A} \times \mathcal{K}$.*

At any *acting step* t, the agent observes $s_t \in \mathcal{S}$, selects a persistence option $o_t = (a_t, k_t) \in \mathcal{O}$ and repeats the primitive action a_t for k_t times, observing the sequence of the states encountered and of the rewards collected. The next acting step is then $t + k_t$. During the execution of the persistence option, the agent is not allowed to change the primitive action. We now extend the policy and state-action value function definitions to consider this particular form of options. A *Markovian stationary policy over persistence options* $\psi : \mathcal{S} \to \mathscr{P}(\mathcal{O})$ is a mapping between states and probability measures over persistence options. We denote with Ψ the set of the policies of this nature. The state-option value function $Q^{\psi} : \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ following a policy over options $\psi \in \Psi$ is defined as:

$$Q^{\psi}(s,a,k) \coloneqq \mathbb{E}_{\psi}\Big[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = a, k_0 = k\Big].$$

In this context, the optimal action-value function is defined as:

$$Q^{\star}_{\mathcal{K}}(s,a,k) = \sup_{\psi \in \Psi} Q^{\psi}(s,a,k).$$

4.1. All-Persistence Bellman Operator

We start by defining a $\overline{\kappa}$-persistence transition as $(s, s', a, r_1, r_2, \dots, r_{\overline{\kappa}})$ where $s \in \mathcal{S}$ is the state where we choose action $a \in \mathcal{A}$, $s' \in \mathcal{S}$ is visited after our agent repeated action a for $\overline{\kappa}$ time steps and $(r_1, r_2, \dots, r_{\overline{\kappa}})$ are all the collected rewards. When we collect a $\overline{\kappa}$-persistence transition it is important to underline that we are trying to use that transition to learn $Q^{\star}_{\mathcal{K}}(\cdot, \cdot, k)$ $\forall k \in \mathcal{K}$, both for k values less than the actually collected $\overline{\kappa}$ transition and for k values greater than $\overline{\kappa}$. Suppose that $\overline{\kappa} \geq k$, then, we can exploit any sub-transition of k steps from the $\overline{\kappa}$-persistence transition to update the value $Q^{\star}_{\mathcal{K}}(\cdot, \cdot, k)$. Thus, we extend the Bellman optimal operator to persistence options, as follows $T^{\star} : \mathscr{B}(\mathcal{S} \times \mathcal{O}) \to \mathscr{B}(\mathcal{S} \times \mathcal{O})$ with $f \in \mathscr{B}(\mathcal{S} \times \mathcal{O})$:

$$(T^{\star} f)(s,a,k) = r_k(s,a) + \gamma^k \int_{\mathcal{S}} P_k(\mathrm{d}s' | s,a) \max_{(a',k') \in \mathcal{O}} f(s',a',k').$$

If, instead, $\overline{\kappa} < k$, in order to update the value $Q^{\star}_{\mathcal{K}}(\cdot, \cdot, k)$, we partially exploit the $\overline{\kappa}$-persistent transition, but then, we need to *bootstrap* from a lower persistence Q-value, to compensate the remaining $k - \overline{\kappa}$ steps.
To this end, we introduce the *bootstrapping* operator $T^{\overline{\kappa}} : \mathscr{B}(\mathcal{S} \times \mathcal{O}) \to \mathscr{B}(\mathcal{S} \times \mathcal{O})$ with $f \in \mathscr{B}(\mathcal{S} \times \mathcal{O})$:

$$\big(T^{\overline{\kappa}} f\big)(s,a,k) = r_{\overline{\kappa}}(s,a) + \gamma^{\overline{\kappa}} \int_{\mathcal{S}} P_{\overline{\kappa}}(\mathrm{d}s' | s,a) f(s',a,k-\overline{\kappa}).$$

By combining these two operators, we obtain the *All-Persistence Bellman operator* $\mathcal{H}_{\overline{\kappa}} : \mathscr{B}(\mathcal{S} \times \mathcal{O}) \to \mathscr{B}(\mathcal{S} \times \mathcal{O})$ defined for every $f \in \mathscr{B}(\mathcal{S} \times \mathcal{O})$ as:

$$(\mathcal{H}^{\overline{\kappa}} f)(s,a,k) = \big((\mathbb{1}_{k \leq \overline{\kappa}} T^{\star} + \mathbb{1}_{k > \overline{\kappa}} T^{\overline{\kappa}}) f\big)(s,a,k).$$

Thus, given a persistence $\overline{\kappa} \in \mathcal{K}$, $\mathcal{H}^{\overline{\kappa}}$ allows updating all the Q-values with $k \leq \overline{\kappa}$ by means of $T^{\star}$, and all the ones with $k > \overline{\kappa}$ by means of $T^{\overline{\kappa}}$. The following result demonstrates the soundness of the proposed operator.

Theorem 4.1. *The all-persistence Bellman operator $\mathcal{H}^{\overline{\kappa}}$ fulfills the following properties:*

(i) *$\mathcal{H}^{\overline{\kappa}}$ is a γ-contraction in L_{∞} norm;*
(ii) *$Q^{\star}_{\mathcal{K}}$ is its unique fixed point;*
(iii) *$Q^{\star}_{\mathcal{K}}$ is monotonic in k, i.e., for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ if $k \leq k'$ then $Q^{\star}_{\mathcal{K}}(s, a, k) \geq Q^{\star}_{\mathcal{K}}(s, a, k')$.*

Thus, operator $\mathcal{H}^{\overline{\kappa}}$ contracts to the optimal action-value function $Q^{\star}_{\mathcal{K}}$, which, thanks to monotonicity, has its highest value at the lowest possible persistence. In particular, it is simple to show that $Q^{\star}_{\mathcal{K}}(s, a, 1) = Q^{\star}(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, i.e., by fixing the persistence to $k = 1$ we retrieve the optimal Q-function in the original MDP, and consequently, we can reconstruct a greedy optimal policy.

Algorithm 1 All Persistence Bellman Update

Require: Sampling persistence $\overline{\kappa}_t$, partial history $H_t^{\overline{\kappa}_t}$, Q-function Q.
Ensure: Updated Q-function Q'

$Q' \leftarrow Q$
for $j = \overline{\kappa}_t, \overline{\kappa}_t - 1, \ldots, 1$ **do**
  for $i = j-1, j-2, \ldots, 0$ **do**
    $k \leftarrow j - i$
    $Q'(s_{t+i}, a_t, k) \leftarrow (1-\alpha) Q(s_{t+i}, a_t, k) + \alpha \widehat{T}^{\star}_{t+i} Q(s_{t+i}, a_t, k)$
    for $d = 1, 2, \ldots, K_{\max} - k$ **do**
      $Q'(s_{t+i}, a_t, k+d) \leftarrow (1-\alpha) Q(s_{t+i}, a, k+d) + \alpha \widehat{T}^{k}_{t+i} Q(s_{t+i}, a_t, k+d)$
    end for
  end for
end for

5. Persistent Q-learning

It may not be immediately clear what are the advantages of $\mathcal{H}^{\overline{\kappa}}$ over traditional updates. These become apparent with its empirical counterpart $\widehat{\mathcal{H}}_t^{\overline{\kappa}} = \mathbb{1}_{k \leq \overline{\kappa}} \widehat{T}_t^{\star} + \mathbb{1}_{k > \overline{\kappa}} \widehat{T}_t^{\overline{\kappa}}$, where:

$$\left(\widehat{T}_t^{\star} Q\right)(s_t, a_t, k) = r_{t+1}^{k} + \gamma^{k} \max_{(a', k') \in \mathcal{O}} Q(s_{t+k}, a', k'),$$

$$\left(\widehat{T}_t^{\overline{\kappa}} Q\right)(s_t, a_t, k) = r_{t+1}^{\overline{\kappa}} + \gamma^{\overline{\kappa}} Q(s_{t+k}, a', k - \overline{\kappa}).$$

These empirical operators depend on the current *partial history*, which we define as: $H_t^{\overline{\kappa}} := (s_t, a_t, r_{t+1}, s_{t+1}, r_{t+2}, \ldots, s_{t+\overline{\kappa}})$, used by Algorithm 1 to update each persistence in a backward fashion, to allow for an even faster propagation of values. At timestep t, given a sampling persistence $\overline{\kappa}_t$, for all sub-transitions of $H_t^{\overline{\kappa}}$, starting at $t+i$ and ending in $t+j$, we apply $\widehat{\mathcal{H}}_t^{j-i}$ to $Q(s_{t+i}, a_t, k+d)$, for all $d \leq K_{\max} - k$, where $k = j - i$. With these tools, it is possible to obtain the Persistent Q-learning algorithm (abbreviated as Per$Q$-learning), a persistent extension of Q-learning [27]: the agent follows a policy ψ_Q^{ϵ}, which is ϵ-greedy w.r.t. the option space and the current Q-function. This approach extends the MSA-Q-learning algorithm presented in [20], by bootstrapping higher persistence action values from lower ones.

The asymptotic convergence of Persistent Q-learning to $Q_{\mathcal{K}}^{\star}$ directly follows from the application of the results in [23], thanks to the fact that $\mathcal{H}^{\overline{\kappa}}$ is a contraction, provided that their (mild) assumptions are satisfied.

In Figure 1 it is presented the interaction between the persistent agent and the environment.

After the agent chooses the persistence option $o := (a, k)$, it executes the action a for an amount of k time steps. When the end of current persistence is reached, i.e. the agent has executed action a for k times, we can start the train process with the current persistence history (see Algorithm 1). After that, the agent can select the new persistence option.

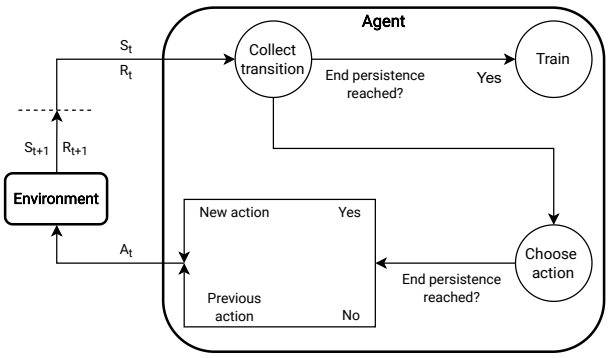

Figure 1: Interaction between persistent agent and the environment.

6. Persistent Deep Networks

In order to deal with high-dimensional settings, we develop an extension of Per$Q$-learning. It is straightforward to exploit Deep Q-Networks for learning in the options space $\mathcal{O}$. Standard DQN is augmented with $K_{\max}$ distinct sets of action outputs, to represent Q-value of the options space $\mathcal{O} = \mathcal{A} \times \mathcal{K}$, while the first layers are shared. The resulting algorithm, *Persistent Deep Q-Network* (PerDQN) is obtained by exploiting the application of the empirical all-persistence Bellman operator. The main differences between PerDQN and standard DQN consist in: (i) a modified ϵ-greedy strategy, which is equivalent to the one described for its tabular version; (ii) the use of *multiple* replay buffers accounting for persistence.

Persistence Replay Buffers Whenever an option $o_t = (a_t, \overline{\kappa}_t)$ is executed, the generated partial history $H_t^{\overline{\kappa}_t}$ is decomposed in all its sub-transitions, which are stored in multiple replay buffers $\mathcal{D}_k$, one for each persistence $k \in \mathcal{K}$. Specifically, $\mathcal{D}_k$ stores tuples in the form $(s, a_t, s', r, \overline{\kappa})$, where s and s' are the first and the last state of the sub-transition, r is the $\overline{\kappa}$-persistent reward, and $\overline{\kappa}$ is a parameter to denote true length of the sub-transition, which will then be used to suitably apply $\widehat{\mathcal{H}}_t^{\overline{\kappa}}$.

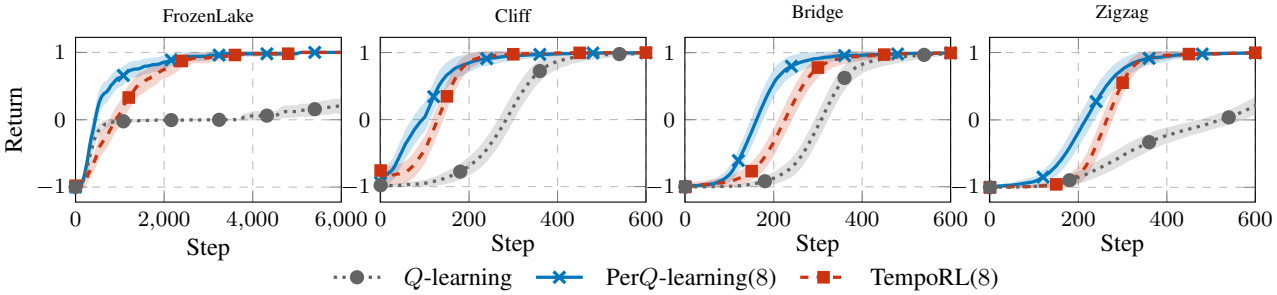

Figure 2: Performance evaluation on tabular environments, $K_{\max} = 8$. 50 runs (95% c.i.).

Finally, the gradient update is computed by sampling a mini-batch of experience tuples from each replay buffer $\mathcal{D}_k$, in equal proportion. Given the current network and target parametrizations $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^-$, the temporal difference error of a sample $(s, a, r, s', \overline{\kappa})$ is computed as $\widehat{\mathcal{H}}^{\overline{\kappa}} Q_{\boldsymbol{\theta}^-}(s, a, k) - Q_{\boldsymbol{\theta}}(s, a, k)$. Our approach differs from TempoRL DQN [4], which uses a dedicated network to learn the persistence at each state and employs a standard replay buffer, ignoring the persistence at which samples have been collected.

7. Empirical Advantages of Persistence

In this section, we provide some numerical simulations to highlight the benefits of using action persistence.

7.1. Exploration

One of the main advantages of persistence is related to faster exploration, especially in goal-based environments (e.g., robotics and locomotion tasks). Indeed, persisting an action allows reaching faster states far from the starting point and, consequently, propagating faster the reward. The reason is due to the increased chances of 1-persistent policies to get stuck in specific regions. As explained in Amin et al. [1], persistence helps to achieve *self-avoiding* trajectories, by increasing the expected return time in previously visited states. Hence, we study the effects of a persisted exploratory policy on the MDP, i.e., a policy $\psi \in \Psi$ over persistence options $\mathcal{O}$.

To this purpose, we compute the *Kemeny's constant* [6, 18], which corresponds to the expected first passage time from an arbitrary starting state s to another one s' under the stationary distribution induced by ψ. We consider four discrete tabular environments: *Open* is a 10x10 grid with no obstacles, while the others, presented in [4], are described in Section 8. In Figure 4, we plot the variations of Kemeny's constant as a function of the maximum persistence $K_{\max}$, while following a uniform policy ψ over $\mathcal{O}$. We observe that increasing $K_{\max}$ promotes exploration, and highlights the different values of $K_{\max}$ attaining the minimum value of the constant, due to the different complexity of the environments.

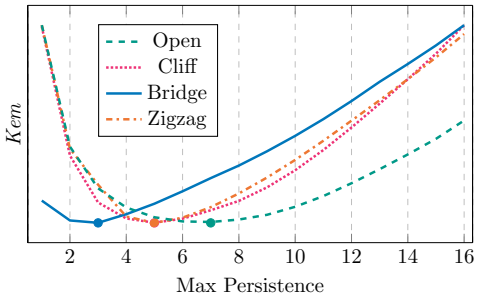

Figure 4: Normalized Kemeny's constant in tabular environments as function of the maximum persistence $K_{\max}$. Bullets represent the minimum value of the constant.

7.2. Sample Complexity

The second relevant effect of persistence concerns with the sample complexity. The intuition behind persistence relies on the fact that the most relevant information propagates faster through the state-action space, thanks to multi-step updates. Moreover, these updates are associated to a lower discount factor, which allows for better convergence rates. In order to evaluate the sample efficiency of Per$Q$-learning, separately from its effects on exploration, we considered a *synchronous* setting [10, 22] in a deterministic 6x6 Gridworld. At each itera-

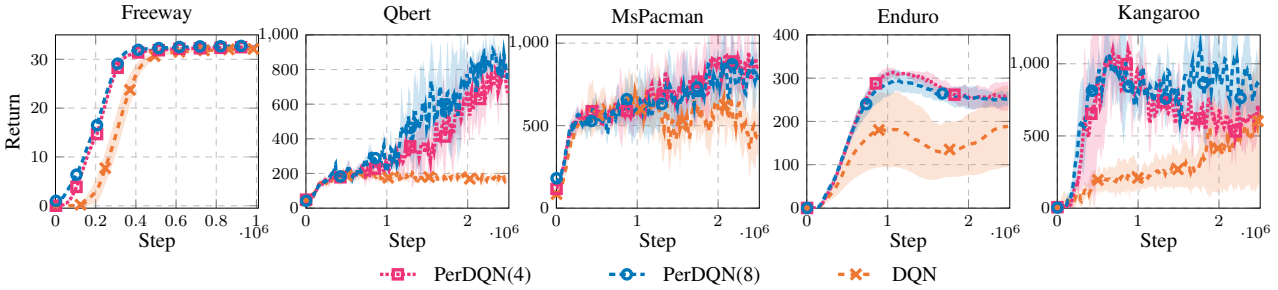

Figure 3: Atari games results for DQN and PerDQN, with $K_{\max} = 4$ and 8. 5 runs (avg± 95% c.i.).

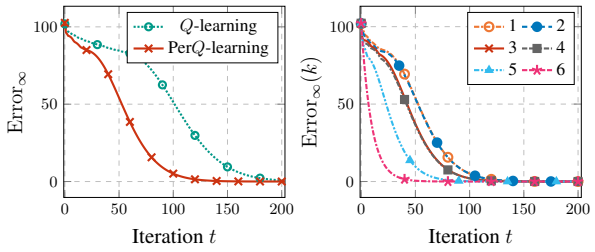

Figure 5: L_∞ error on 6x6 grid-world between synchronous Q-learning and Per$Q$-learning (left) and for different persistence options $k \in \{1, ..., 6\}$ of Per$Q$-learning (right). (100 runs, avg ± 95 % c.i.)

tion t, the agent has access to a set of independent samples for each state-action pair. In standard Q-learning the samples are used to update the action value function $Q(s, a)$ for each $(s, a) \in \mathcal{S} \times \mathcal{A}$. In Per$Q$-learning, the samples are combined to obtain each possible set of $\overline{\kappa}$-persistent transitions, i.e., the tuples related to each possible $(s, a, k) \in \mathcal{S} \times \mathcal{O}$, with $K_{\max} = 6$; finally, the persistent action value function is updated. On the left side of Figure 5, we compare the L_∞ error of Q-learning estimating $Q^\star(s, a)$, i.e., $\max_{s,a \in \mathcal{S} \times \mathcal{A}} |Q_t(s, a) - Q^\star(s, a)|$, and that of Per$Q$-learning estimating $Q^\star_{\mathcal{K}}(s, a, k)$, i.e., $\max_{s,a,k \in \mathcal{S} \times \mathcal{O}} |Q_t(s, a, k) - Q^\star_{\mathcal{K}}(s, a, k)|$, as a function of the number of iterations t. We observe that, although estimating a higher-dimensional function (as $Q^\star_{\mathcal{K}}(s, a, k)$ is a function of the persistence k too), Per$Q$-learning converges faster than Q-learning. On the right side of Figure 5, we plot the L_∞ error experienced by Per$Q$-learning for the different persistence options $\mathcal{O}^{(k)}$, i.e., $\text{Error}_\infty(k) \coloneqq \max_{s,a \in \mathcal{S} \times \mathcal{A}} |Q_t(s, a, k) - Q^\star(s, a, k)|$ for $k \in \mathcal{K}$. We note that, as expected, higher values of k lead to faster convergence; consequently, the persistent Bellman operator helps improving the estimations also for the lower option sets. Indeed, we can see that also $Q_t(\cdot, \cdot, 1)$, which represents the action value function for the primitive actions, converges faster than classic Q-learning.

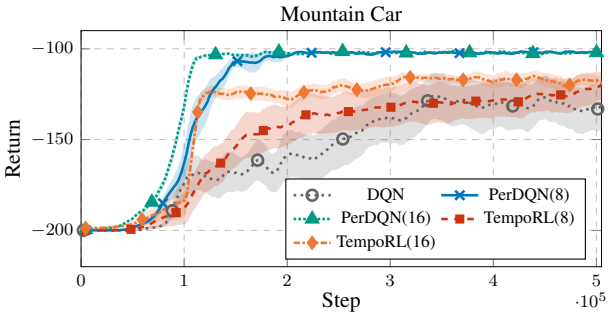

Figure 7: Performance on MountainCar. Parenthesis denote $K_{\max}$. 20 runs (avg± 95% c.i.).

8. Experimental Evaluation

In this section, we show the empirical analysis of our approach on both the tabular setting (Per$Q$-learning) and the function approximation one (PerDQN).

8.1. Per$Q$-learning

We present the results on the experiments in tabular environments, particularly suited for testing Per$Q$-learning because of the sparsity of rewards. We start with the deterministic 6x10 grid-worlds introduced by [4]. In these environments, the episode ends if either the goal or a hole is reached, with +1 or −1 points respectively. In all the other cases, the reward is 0, and the episode continues. Moreover, we experiment the 16x16 FrozenLake, with rewards and transition process analogous to the previous case, but with randomly generated holes at the beginning of the episode. The results are shown in Figure 2, where we compared Per$Q$-learning with Tem-

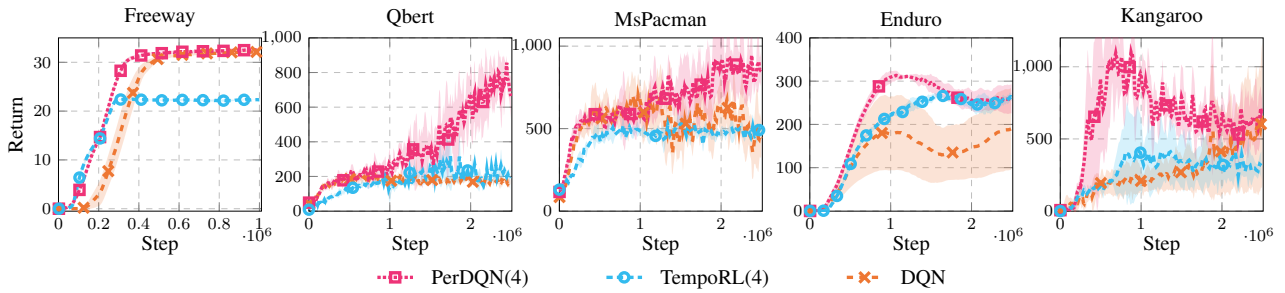

Figure 6: Atari games results for DQN, PerDQN and TempoRL, with $K_{\max} = 4$. 5 runs (avg± 95% c.i.).

poRL (with the same maximum *skip-length* $J = 8$) and classic Q-learning. In all cases, Per$Q$-learning outperforms the other methods, especially Q-learning, whose convergence is significantly slower. In general, Per$Q$-learning shows faster rates of improvements than TempoRL, especially in the first learning iterations.

8.2. PerDQN

Our implementation of PerDQN is based on OpenAI Gym and Baselines [8] Python toolkits. We start with MountainCar, as it is perhaps the most suited to evaluate the performance of persistence options, since 1-step explorative policies usually fail to reach the goal because of their low probability to commit to an action for long times [14]. Figure 7 shows that TempoRL and standard DQN cannot converge to the optimal policy, while PerDQN can reach the optimal solution, which consists in obtaining the minimum loss required to reach the top of the mountain. The algorithm is then tested in the challenging framework of Atari 2600 games. In Figure 3 we compare PerDQN and classic DQN. Our PerDQN displays a faster learning curve thanks to its ability of reusing experience, although in some cases (e.g. Kangaroo) PerDQN seems to inherit the same instability issues of DQN, we conjecture due to the overestimation bias [26]. We have also compared our method with the one described by TempoRL ([4]): the results are shown in Figures 6. As we can see, PerDQN outperforms TempoRL in all the environments evaluated.

9. Conclusion

In this paper, we have considered RL policies that implement action persistence, modeled as *persistence options*, selecting a primitive action and its duration. We defined the *all-persistence* Bellman operator, which allows for an effective use of the experience collected from the interaction with the environment at any time scale. Thanks to this operator, action-value function estimates can be updated simultaneously on the selected persistence set: low persistences (and primitive actions) can be updated by splitting the samples in their sub-transitions; action value functions for high persistences can instead be improved by *bootstrap*, a procedure that takes into account the estimation of the partial missing information. After proving that the new operator is a contraction, we applied it to extend classic Q-learning and DQN with their persistent version. We performed an experimental campaign on tabular and deep RL settings demonstrating the effectiveness of our approach and the importance of considering temporal extended actions. Future research directions include, the introduction of criteria for persistence interruptions (like for interrupting options). Furthermore, one could investigate the possibility of employing the operator in the actor-critic framework to cope with continuous action spaces.

References

[1] Susan Amin et al. "Locally Persistent Exploration in Continuous Control Tasks with Sparse Rewards". In: *arXiv preprint arXiv:2012.13658* (2020).

[2] Marc G Bellemare et al. "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[3] Dimitir P Bertsekas and Steven Shreve. *Stochastic optimal control: the discrete-time case*. Academic Press, 2004.

[4] André Biedenkapp et al. "TempoRL: Learning When to Act". In: *ICML*. 2021.

[5] Alex Braylan et al. "Frame skip is a powerful parameter for learning to play atari". In: *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[6] Minerva Catral et al. "The Kemeny constant for finite homogeneous ergodic Markov chains". In: *Journal of Scientific Computing* 45.1 (2010), pp. 151–166.

[7] Will Dabney et al. "Temporally-Extended ϵ-Greedy Exploration". In: *International Conference on Learning Representations (ICLR)*. 2020.

[8] Prafulla Dhariwal et al. *OpenAI Baselines*. `https://github.com/openai/baselines`. 2017.

[9] Jake Grigsby et al. "Towards Automatic Actor-Critic Solutions to Continuous Control". In: *arXiv preprint arXiv:2106.08918* (2021).

[10] Michael Kearns and Satinder Singh. "Finite-sample convergence rates for Q-learning and indirect algorithms". In: *Advances in Neural Information Processing Systems (NIPS)* (1999), pp. 996–1002.

[11] Aravind S. Lakshminarayanan et al. "Dynamic Action Repetition for Deep Reinforcement Learning". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*. Ed. by Satinder P. Singh and Shaul Markovitch. AAAI Press, 2017, pp. 2133–2139.

[12] Timothy A. Mann et al. "Approximate Value Iteration with Temporally Extended Actions". In: *J. Artif. Intell. Res.* 53 (2015), pp. 375–438.

[13] Alberto Maria Metelli et al. "Configurable Markov Decision Processes". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 3488–3497.

[14] Alberto Maria Metelli et al. "Control frequency adaptation via action persistence in batch reinforcement learning". In: *International Conference on Machine Learning (ICML)*. PMLR. 2020, pp. 6862–6873.

[15] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[16] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[17] Seohong Park et al. "Time Discretization-Invariant Safe Action Repetition for Policy Gradient Methods". In: *Advances in Neural Information Processing Systems (NeurIPS)* 34 (2021).

[18] Rushabh Patel et al. "Robotic surveillance and Markov chains with minimal weighted Kemeny constant". In: *IEEE Transactions on Automatic Control* 60.12 (2015), pp. 3156–3167.

[19] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[20] Ralf Schoknecht and Martin Riedmiller. "Reinforcement learning on explicitly specified time scales". In: *Neural Computing & Applications* 12.2 (2003), pp. 61–80.

[21] Sahil Sharma et al. "Learning to repeat: Fine grained action repetition for deep reinforcement learning". In: *arXiv preprint arXiv:1702.06054* (2017).

[22] Aaron Sidford et al. "Near-optimal time and sample complexities for solving Markov decision processes with a generative model". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 5192–5202.

[23] Satinder Singh et al. "Convergence results for single-step on-policy reinforcement-learning algorithms". In: *Machine learning* 38.3 (2000), pp. 287–308.

[24] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[25] Corentin Tallec et al. "Making Deep Q-learning methods robust to time discretization". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6096–6104.

[26] Hado Van Hasselt et al. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on Artificial Intelligence (AAAI)*. Vol. 30. 2016.

[27] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". PhD thesis. King's College, University of Cambridge, 1989.

[28] Haonan Yu et al. "TAAC: Temporally Abstract Actor-Critic for Continuous Control". In: *Advances in Neural Information Processing Systems (NeurIPS)* 34 (2021).