Simultaneously Updating All Persistence Values in Reinforcement Learning

Luca Sabbioni[1], Luca Al Daire[1], Lorenzo Bisi[2], Alberto Maria Metelli[1], Marcello Restelli[1]

[1]Politecnico di Milano, Milan, Italy
[2]ML cube, Milan, Italy
luca.sabbioni@polimi.it

Abstract

In *reinforcement learning*, the performance of learning agents is highly sensitive to the choice of time discretization. Agents acting at high frequencies have the best control opportunities, along with some drawbacks, such as possible inefficient exploration and vanishing of the action advantages. The repetition of the actions, i.e., *action persistence*, comes into help, as it allows the agent to visit wider regions of the state space and improve the estimation of the action effects. In this work, we derive a novel *All-Persistence Bellman Operator*, which allows an effective use of both the low-persistence experience, by decomposition into sub-transition, and the high-persistence experience, thanks to the introduction of a suitable *bootstrap* procedure. In this way, we employ transitions collected at *any* time scale to update simultaneously the action values of the considered persistence set. We prove the contraction property of the All-Persistence Bellman Operator and, based on it, we extend classic Q-learning and DQN. After providing a study on the effects of persistence, we experimentally evaluate our approach in both tabular contexts and more challenging frameworks, including some Atari games.

Introduction

In recent years, Reinforcement Learning (RL, Sutton and Barto 2018) methods have proven to be successful in a wide variety of applications, including robotics (Kober and Peters 2014; Gu et al. 2017; Haarnoja et al. 2019; Kilinc, Hu, and Montana 2019), autonomous driving (Kiran et al. 2021) and continuous control (Lillicrap et al. 2016; Schulman et al. 2017). These sequential decision-making problems are typically modelled as a Markov Decision Process (MDP, Puterman 2014), 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 $\delta = \frac{1}{f}$ (Park, Kim, and Kim 2021). This represents an environment hyperparameter, which may have dramatic effects on the process of learning the optimal policy (Metelli et al. 2020; Kalyanakrishnan et al. 2021). 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 (Baird 1994; Tallec, Blier, and Ollivier 2019). Even policy gradient methods are shown to fail in this (near-)continuous-time setting, and the reason is related to the divergent variance of the gradient (Park, Kim, and Kim 2021). The consequences of each action might be detected if the dynamics of the environment has the time to evolve, hence with an agent acting with higher frequencies lead to higher sample complexity.

Another consequence of the use of high frequencies is related to the difficulty of exploration. A random uniform policy played at high frequency may not be adequate, as in some classes of environments, including the majority of real-world control problems, it tends to visit only a local neighborhood of the initial state (Amin et al. 2021; Park, Kim, and Kim 2021; Yu, Xu, and Zhang 2021). 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* or *action repetition* (Schoknecht and Riedmiller 2003; Braylan et al. 2015; Lakshminarayanan, Sharma, and Ravindran 2017; Metelli et al. 2020), which 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 fasten convergence to the optimal policy.

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 main contribution of this paper is a general approach in which information collected from the interaction with the environment at *one* persistence is used to improve the action value function estimates of *all* the considered possible persistences. On one hand, the $\overline{\kappa}$-step transitions can be decomposed in many sub-transitions of reduced length and used to update lower persistence $k \leq \overline{\kappa}$ value functions. On the other hand, they represent partial information for the estimation of the effects of higher persistence $k > \overline{\kappa}$ actions. Indeed, they can be employed to update the estimates by using a suitable *bootstrapping* procedure of the missing information. Thus, all value function estimates are updated simultaneously for each of the available persistences $k \in \mathcal{K}$. We formalize this procedure by introducing the *All-persistence Bellman Operator*. We prove that such an operator enjoys a contraction property analogous to that of the traditional optimal Bellman operator. Consequently, we embed the All-persistence Bellman operator into the classic Q-learning algorithm, obtaining *Persistent Q-learning* (Per$Q$-learning). This novel algorithm, through an effective use of the transitions sampled at different persistences, displays two main advantages. First, since each individual transition is employed to update the value function estimates at different persistences, we experience a faster convergence. Second, the execution of persistent actions, given the nature of a large class of environments, fosters exploration of the state space, with a direct effect on the learning speed. Furthermore, to deal with more complex domains, we move to the Deep RL scenario, extending the Deep Q-Network (DQN) algorithm to its persistent version, called *Persistent Deep Q-Network* (PerDQN). Finally, we evaluate the proposed algorithms, in comparison with state-of-the-art approaches, on illustrative and complex domains, highlighting strengths and weaknesses. The proofs and further results are reported in Appendix.[1]

Related Work

The first work extending RL agents with action repetition, go back to Schoknecht and Riedmiller 2003. In this paper, multi-step actions (MSAs) were introduced, reducing the number of decisions needed to reach the goal and making the time scale coarser. Action persistence has acquired practical relevance since the introduction of Deep RL (Mnih et al. 2013), by leveraging the *frame skip* parameter (Bellemare et al. 2013). Several works (Braylan et al. 2015; Khan et al. 2019; Metelli et al. 2020) had shown the importance of persistence for helping exploration and policy learning. Among these works, Dabney, Ostrovski, and Barreto 2020 introduced an $\epsilon z-$greedy exploration, with a random exploratory variable deciding the duration of each action. As explained in Metelli et al. 2020, changing frequency deeply modifies the underlying MDP, as a special instance of a configurable MDP (Metelli, Mutti, and Restelli 2018), where environmental parameters can be tuned to improve the performance. Indeed, in Grigsby, Yoo, and Qi 2021 the authors proposed an algorithm to automatically tune the control frequency, along with other learning hyperparameters. Mann, Mannor, and Precup 2015 illustrates that approximate value iteration techniques can converge faster with action persistence (seen as *options* with longer duration).

Action repetition has many advantages, but it may reduce the control opportunities. Consequently, researchers have been trying to include the possibility to *dynamically* change the control frequency during learning: in Augmented-DQN (Lakshminarayanan, Sharma, and Ravindran 2017), the action space is duplicated to be able to choose actions with two previously selected repetition rates.

A different approach is proposed in Sharma, Srinivas, and Ravindran 2017, where a *skip network* is used to the action persistence in a specific state, regardless of the chosen action. One way to differentiate persistences with actions is proposed by TempoRL (Biedenkapp et al. 2021), where the skip network depends on both state and action (and the estimated Q-value function) to evaluate the effects for different possible frequencies. In G. Bellemare et al. 2016, *persistent advantage learning* is proposed in which the advantage learning is overridden by a Q-learning update with repeated actions, when the latter promises better Q-values.

In the framework of policy-gradient methods, persistence is introduced in Yu, Xu, and Zhang 2021, with the introduction of a secondary policy for choosing whether to repeat the previous action or to change it according to the principal agent. A completely different approach is presented by Park, Kim, and Kim 2021: the authors claim that when $\delta \to 0$ policy-based methods tend to degrade. Thanks to the introduction of a *safe region*, the agent keeps repeating an action until the distance of the visited states overcomes a certain threshold. This state locality can guarantee reactivity, especially in some environments where the blind repetition of an action can be dangerous.

Preliminaries

In this section, we provide the necessary background employed in the following of the paper.

Mathematical Background Given a measurable space $(\mathcal{X}, \sigma_{\mathcal{X}})$, where $\mathcal{X}$ is a set and $\sigma_{\mathcal{X}}$ a σ-algebra, we denote with $\mathscr{P}(\mathcal{X})$ the set of probability measures and with $\mathscr{B}(\mathcal{X})$ the set of the bounded measurable functions. We denote with δ_x the Dirac measure centered in $x \in \mathcal{X}$. Let $f \in \mathscr{B}(\mathcal{X})$, the L_∞-norm is defined as $\|f\|_\infty = \sup_{x \in \mathcal{X}} f(x)$.

Markov Decision Processes A discrete-time Markov Decision Process (MDP, Puterman 2014) is defined as a tuple $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, 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, bounded as $\|r\|_\infty \le R_{\max} < +\infty$, 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:

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

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 $(s, a) \in \mathcal{S} \times \mathcal{A}$. A policy π is *greedy* w.r.t. a function $f \in \mathscr{B}(\mathcal{S} \times \mathcal{A})$ if it plays only greedy actions, i.e., $\pi(\cdot|s) \in \mathscr{P}\left(\arg\max_{a \in \mathcal{A}} f(s, a)\right)$. An *optimal policy* $\pi^\star \in \Pi$ is any policy which acts greedily w.r.t. $Q^\star$.

[1]The full version of the paper, including the appendix, is available at http://arxiv.org/abs/2211.11620

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 $Q$
 1: for $j{=}\overline{\kappa}_t,\overline{\kappa}_t{-}1...,1$ do
 2:   for $i{=}j{-}1,j{-}2,...,0$ do
 3:     $k{\leftarrow}j{-}i$
 4:     $Q(s_{t+i},a_t,k){\leftarrow}(1{-}\alpha)Q(s_{t+i},a_t,k)+$
 5:         $\alpha\widehat{T}^{\star}_{t+i}Q(s_{t+i},a_t,k)$
 6:     for $d{=}1,2,...,K_{\max}{-}k$ do
 7:       $Q(s_{t+i},a_t,k{+}d){\leftarrow}(1{-}\alpha)Q(s_{t+i},a_t,k{+}d)+$
 8:           $\alpha\widehat{T}^{k}_{t+i}Q(s_{t+i},a_t,k{+}d)$
 9:     end for
10:   end for
11: end for
```

Q-learning We make use of the *Bellman Optimal Operator* $T^\star : \mathscr{B}(\mathcal{S}\times\mathcal{A}) \to \mathscr{B}(\mathcal{S}\times\mathcal{A})$, defined as in (Bertsekas and Shreve 1996): $(T^\star f)(s,a) = r(s,a) + \gamma\int_{\mathcal{S}} P(\mathrm{d}s'|s,a)\max_{a'\in\mathcal{A}} f(s',a')$. $T^\star$ is a γ-contraction in L_∞-norm and its unique fixed point is the optimal Q-function. When P and r are known, value-iteration (Puterman 2014) allows computing $Q^\star$ via iterative application of $T^\star$. When the environment is unknown, Q-learning (Watkins 1989) collects samples with a *behavioral* policy (e.g., ϵ-greedy) and then updates Q-function estimate based on the updated rule: $Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha(r_{t+1} + \gamma\max_{a'\in\mathcal{A}} Q(s_{t+1},a'))$. where $\alpha > 0$ is the learning rate.

Deep Q-Networks Deep Q-Network (DQN, Mnih et al. 2013, 2015) employs a deep neural network with weights $\boldsymbol{\theta}$ to learn an approximation $Q_{\boldsymbol{\theta}}$ of $Q^\star$. The transitions are stored in the *replay buffer* $\mathcal{D} = \{(s_t,a_t,r_{t+1},s_{t+1})\}_{t=1}^n$ to mitigate temporal correlations. 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 $(s,a,r,s') \sim \mathcal{D}$.

Action Persistence The execution of actions with a persistence $k \in \mathbb{N}$ can be modeled by means of the k-persistent MDP (Metelli et al. 2020), 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 $P_k(\cdot|s,a) = \left((P^\delta)^{k-1}P\right)(\cdot|s,a)$ and $r_k(s,a) = \sum_{i=0}^{k-1}\gamma^i\left((P^\delta)^i r\right)(s,a)$. This framework eases the analysis of *fixed* persistences, but it does not allow action repetition for a *variable* number of steps.

All-Persistence Bellman Update

In this section, we introduce our approach to make effective use of the samples collected at *any* persistence. We first introduce the notion of *persistence option* and then, we present the *all-persistence Bellman operator*.

Algorithm 2: Persistent Q-learning (Per$Q$-learning)

```
Require: Learning rate $\alpha$, exploration coefficient $\epsilon$,
    number of episodes $N$
Ensure: $Q$-function
 1: Initialize $Q$ arbitrarily, $Q(terminal,\cdot,\cdot) = 0$
 2: for $episode = 1,\dots,N$ do
 3:   $t \leftarrow 0$
 4:   while $s_t$ is not $terminal$ do
 5:     $a_t,\overline{\kappa}_t \sim \psi_Q^\epsilon(s_t)$
 6:     for $\tau = 1,\dots,\overline{\kappa}_t$ do
 7:       Take action $a_t$, observe $s_{t+\tau}, r_{t+\tau}$
 8:     end for
 9:     Store partial history $H_t^{\overline{\kappa}_t}$
10:     Update $Q$ according to Alg.1
11:     $t \leftarrow t + \overline{\kappa}_t$
12:   end while
13: end for
```

Persistence Options

We formalize the decision process in which the agent chooses a primitive action a together with its persistence k. To this purpose, we introduce the *persistence option*.

Definition 0.1 *Let $\mathcal{A}$ be the space of* primitive *actions of an MDP $\mathcal{M}$ and $\mathcal{K} := \{1,\dots,K_{\max}\}$, where $K_{\max} \geq 1$, be the set of persistences. A* persistence option *$o := (a,k)$ is the decision of playing primitive action $a \in \mathcal{A}$ with persistence $k \in \mathcal{K}$. We denote with $\mathcal{O}^{(k)} := \{(a,k) : a \in \mathcal{A}\}$ the set of options with fixed persistence $k \in \mathcal{K}$ and $\mathcal{O} := \bigcup_{k\in\mathcal{K}}\mathcal{O}^{(k)} = \mathcal{A}\times\mathcal{K}$.*

The decision process works as follows. At time $t = 0$, the agent observes $s_0 \in \mathcal{S}$, selects a persistence option $o_0 = (a_0,k_0) \in \mathcal{O}$, observes the sequence of states $(s_1,\dots,s_{k_0})$ generated by repeating primitive action a_0 for k_0 times, i.e., $s_{i+1} \sim P(\cdot|s_i,a_0)$ for $i \in \{0,\dots,k_0-1\}$, and the sequence of rewards $(r_1,\dots,r_{k_0})$ with $r_{i+1} = r(s_i,a_0)$ for $i \in \{0,\dots,k_0-1\}$. Then, in state s_{k_0} the agent selects another option $o_1 = (a_1,k_1) \in \mathcal{O}$ and the process repeats. During the execution of the persistence option, the agent is not allowed to change the primitive action.[2]

Remark 0.1 (**Persistence and Options**) *The persistence option (Definition 0.1) is in all regards a* semi-Markov option *(Precup 2001), where the initiation set is the set of all states $\mathcal{S}$, the termination condition depends on time only, and the intra-option policy is constant. Indeed, the described process generates a* semi-Markov decision process *(Puterman 2014), fully determined by the behavior of $\mathcal{M}$, as shown in (Sutton, Precup, and Singh 1999).*

Remark 0.2 (**Persistence Options vs Augmented Action Space**) *There is an important difference between using persistence options $\mathcal{O}$ in the original MDP $\mathcal{M}$ and defining an augmented MDP $\mathcal{M}_{\mathcal{K}}$ with new action space $\mathcal{A}\times\mathcal{K}$ and properly redefined transition model and reward function (Lakshminarayanan, Sharma, and Ravindran 2017): when executing a persistence option $o_t = (a_t,k_t) \in \mathcal{O}$ at time t,*

[2]From this definition, it follows that $\mathcal{A}$ is isomorphic to $\mathcal{O}^{(1)}$.

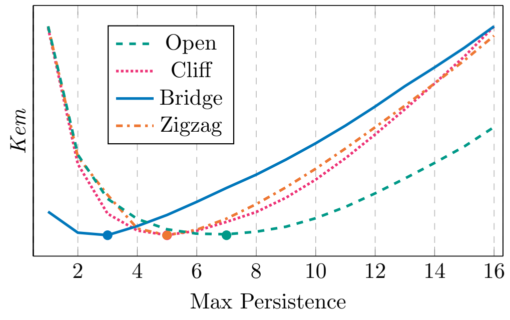

Figure 1: Normalized Kemeny's constant in tabular environments as function of $K_{\max}$. Bullets represent the minimum.

we observe the full *sequence of states* $(s_{t+1}, \dots, s_{t+k_t})$ *and rewards* $(r_{t+1}, \dots, r_{t+k_t})$. *Instead, in the augmented MDP* $\mathcal{M}_{\mathcal{K}}$ *we* only *observe the last state* s_{k_t} *and the cumulative reward* $r^k_{t+1} = \sum_{i=0}^{k-1} \gamma^i r_{t+i+1}$. *We will heavily exploit the particular option structure, re-using fragments of experience to perform intra-option learning.*

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 $Q^{\psi}(s,a,k) := \mathbb{E}_{\psi}\left[\sum_{t=0}^{+\infty} \gamma^t r_{t+1} | s_0 = s,\, a_0 = a, k_0 = k\right]$. In this context, the optimal action-value function is defined as: $Q^{\star}_{\mathcal{K}}(s,a,k) = \sup_{\psi \in \Psi} Q^{\psi}(s,a,k)$.

All-Persistence Bellman Operator

Our goal is to leverage any $\overline{\kappa}$-persistence transition to learn $Q^{\star}_{\mathcal{K}}(\cdot,\cdot,k)$ for *all* the possible action-persistences in $k \in \mathcal{K}$. 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 $T^{\star} : \mathscr{B}(\mathcal{S} \times \mathcal{O}) \to \mathscr{B}(\mathcal{S} \times \mathcal{O})$ 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 $f \in \mathscr{B}(\mathcal{S} \times \mathcal{O})$:

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

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

Theorem 0.1 *The all-persistence Bellman operator* $\mathcal{H}^{\overline{\kappa}}$ *fulfills the following properties:*

1. $\mathcal{H}^{\overline{\kappa}}$ *is a* γ*-contraction in* L_{∞} *norm;*
2. $Q^{\star}_{\mathcal{K}}$ *is its unique fixed point;*
3. $Q^{\star}_{\mathcal{K}}$ *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. This highlights that the primitive action space leads to the same optimal Q-function as with persistence options. Persistence, nevertheless, is helpful for exploration and learning, but for an optimal persistent policy ψ^*, there exists a primitive policy π^* with the same performance.

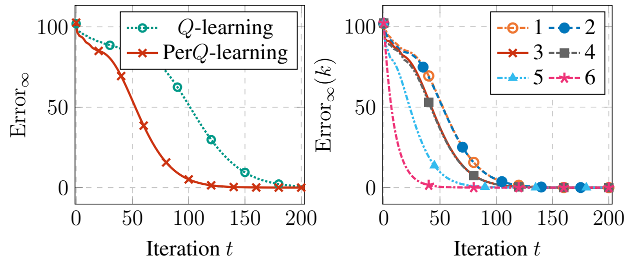

Figure 2: (Left) L_{∞} error on 6x6 grid-world between synchronous Q-learning and Per$Q$-learning. (Right) L_{∞} error of Per$Q$-learning for $k \in \{1, ..., 6\}$. (100 runs, avg $\pm$ 95 % c.i.)

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 $\widehat{\mathcal{H}}^{\overline{\kappa}}_t = \mathbb{1}_{k \leq \overline{\kappa}} \widehat{T}^{\star}_t + \mathbb{1}_{k > \overline{\kappa}} \widehat{T}^{\overline{\kappa}}_t$, where:

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

$$\left(\widehat{T}^{\overline{\kappa}}_t Q\right)(s_t,a_t,k) = r^{\overline{\kappa}}_{t+1} + \gamma^{\overline{\kappa}} Q(s_{t+k},a_t,k-\overline{\kappa}).$$

These empirical operators depend on the current *partial history*, which we define as: $H^{\overline{\kappa}}_t := (s_t, a_t, r_{t+1}, s_{t+1}, r_{t+2}, \dots, s_{t+\overline{\kappa}})$, used by Algorithm 1 to update each persistence in a backward fashion, as illustrated also in Appendix B. At timestep t, given a sampling persistence $\overline{\kappa}_t$, for all sub-transitions of $H^{\overline{\kappa}}_t$, starting at $t+i$ and ending in $t+j$, we apply $\widehat{\mathcal{H}}^{j-i}_t$ 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 extend Q-learning (Watkins 1989) to obtain the Persistent Q-learning algorithm (abbreviated as Per$Q$-learning), described in Algorithm 2. The agent follows a policy ψ^{ϵ}_{Q}, which is ϵ-greedy w.r.t. the option space and the current Q-function.

Algorithm 3: Multiple Replay Buffer Storing

```
Require: Maximum persistence $K_{\max}$, replay buffers $(\mathcal{D}_k)_{k=1}^{K_{\max}}$, transition tuple $(s_t, a_t, \overline{\kappa}_t, H_t^{\overline{\kappa}_t})$.
1: for $k = 1, \dots, K_{\max}$ do
2:     for $\tau = 0, \dots, \max\{\overline{\kappa}_t - k, 0\}$ do
3:         $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup (s_{t+\tau}, a_t, s_{t+\tau+k}, r^k_{t+1+\tau}, k)$
4:     end for
5:     for $\tau = 1, \dots, \min\{\overline{\kappa}_t, k-1\}$ do
6:         $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup (s_{t+\overline{\kappa}_t-\tau}, a_t, s_{t+\overline{\kappa}_t}, r^\tau_{t+1+\overline{\kappa}_t-\tau}, \tau)$
7:     end for
8: end for
```

This approach extends the MSA-Q-learning (Schoknecht and Riedmiller 2003), by bootstrapping higher persistence action values from lower ones. More precisely, both methods apply the update related to $\widehat{T}^\star$, but MSA-Q-learning does not use $\widehat{T}^{\overline{\kappa}}$ instead. As shown in the empirical analysis, in some domains this difference can be crucial to speed up the convergence. Similarly to MSA-Q-learning, we perform backwards updates to allow for an even faster propagation of values. The proposed approach also differs from TempoRL Q-learning (Biedenkapp et al. 2021), where action-persistence is selected using a dedicated value-function, learned separately from the Q-function. The asymptotic convergence of Persistent Q-learning to $Q^\star_{\mathcal{K}}$ directly follows (Singh et al. 2000), being $\mathcal{H}^{\overline{\kappa}}$ a contraction and since their (mild) assumptions are satisfied.

Empirical Advantages of Persistence

In this section, we provide some numerical simulations to highlight the benefits of our approach. The settings are illustrative, to ease the detection of the individual advantages of persistence, before presenting more complex applications.

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. 2021), 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 $\mathcal{O}$ (details in Appendix C of the complete version of the paper).

To this purpose, we compute the *Kemeny's constant* (Catral et al. 2010; Patel, Agharkar, and Bullo 2015), 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 (Biedenkapp et al. 2021), are depicted in the appendix. In Figure 1, 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

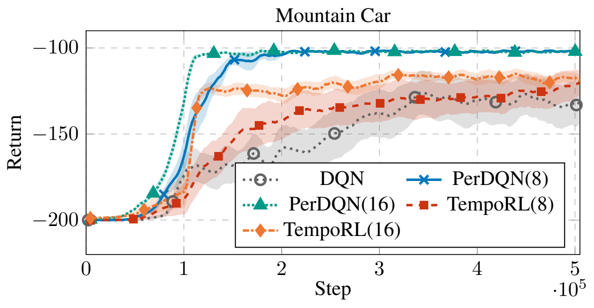

Figure 3: MountainCar results. Parenthesis in the legend denote $K_{\max}$. 20 runs (avg± 95% c.i.).

different $K_{\max}$ attaining the minimum value of the constant, due to the different complexity of the environments.

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, for which it is possible to obtain better convergence rates, as proved in (Metelli et al. 2020), in which the sample complexity in a $k-$persistent MDP is reduced by a factor $(1-\gamma^k)/(1-\gamma) > 1$. In order to evaluate the sample efficiency of Per$Q$-learning, separately from its effects on exploration, we considered a *synchronous* setting (Kearns and Singh 1999; Sidford et al. 2018) in a deterministic 6x6 Gridworld. At each iteration t, the agent has access to a set of independent samples for each state-action pair. In standard Q-learning, for each $(s, a) \in \mathcal{S} \times \mathcal{A}$, $Q(s, a)$ is updated. 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 Q function is updated.

In Figure 2 left, 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. In Figure 2 right, we plot the L_∞ error experienced by Per$Q$-learning for the different persistence options $\mathcal{O}^{(k)}$, i.e.,, for $k \in \mathcal{K}$:

$$\mathrm{Error}_\infty(k) := \max_{s,a\in\mathcal{S}\times\mathcal{A}} |Q_t(s, a, k) - Q^\star(s, a, k)|.$$

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 $Q_t(\cdot, \cdot, 1)$, the primitive actions Q-function, converges faster than classic Q-learning (details in Appendix E of the complete version of the paper).

Persistent Deep Networks

In this section, we develop the extension of Per$Q$-learning to high-dimensional settings. Deep RL methods are becoming

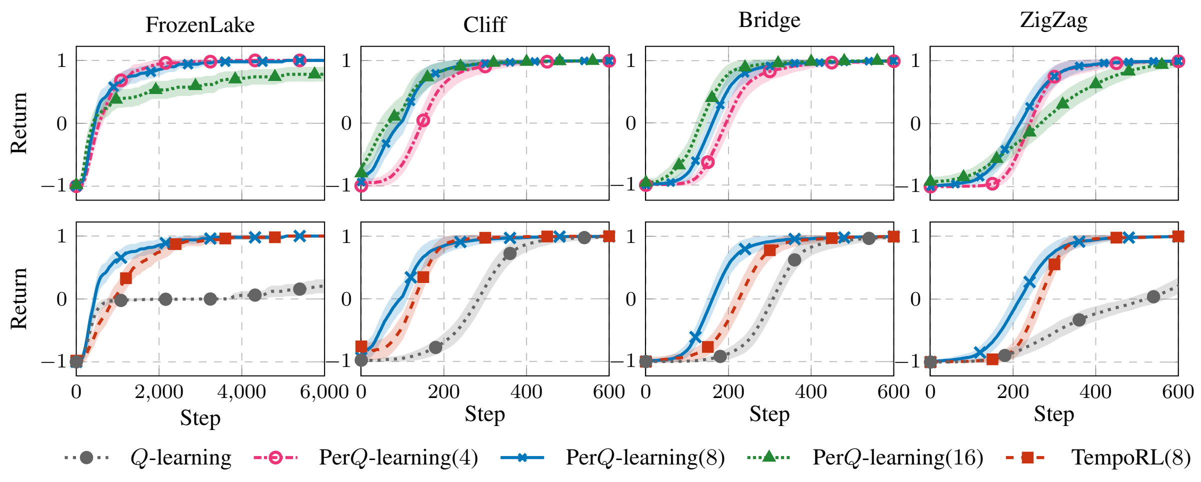

Figure 4: Results on tabular environments. Top row: performances with different maximum persistences. In the legend, parenthesis denote the selected $K_{\max}$. Bottom row: Per$Q$-learning and TempoRL comparison, $K_{\max} = 8$. 50 runs (avg$\pm$ 95% c.i.).

of fundamental importance when learning on real systems, as well as the research of methods to improve exploration and learning speed. It is straightforward to exploit Deep Q-Networks (DQN (Mnih et al. 2013, 2015)) 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, similarly to previous works (Arulkumaran et al. 2016; Lakshminarayanan, Sharma, and Ravindran 2017; Biedenkapp et al. 2021). 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 partial history $H_t^{\overline{\kappa}_t}$ is decomposed in all its sub-transitions, which are used to update Q-values at any persistence, as shown in the previous sections. The sub-transitions 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 $(s, a_t, s', r, \overline{\kappa})$, as summarized in Algorithm 3, 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 the true length of the sub-transition, which will then be used to suitably apply $\widehat{\mathcal{H}}_t^{\overline{\kappa}}$.

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 (Biedenkapp et al. 2021), 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.

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).

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 Biedenkapp et al. 2021. 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 (details in Appendix F.1). Moreover, we experiment the 16x16 FrozenLake, from OpenAI Gym benchmark (Brockman et al. 2016), 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 4. In the top row, we compare the results on the performance when applying Per$Q$-learning with different $K_{\max} \in \{4, 8, 16\}$. We can detect a faster convergence when passing from $K_{\max} = 4$ to 8. However, the largest value of $K_{\max}$ is not always the best one: while Bridge and Cliff show a slight improvement, performances in ZigZag and FrozenLake degrade. This is probably due to the nature of the environment. When there are many obstacles, high persistences might be inefficient, as the agent can get stuck or reach holes more easily. In the bottom plots of Figure 4 we selected the results with $K_{\max} = 8$, and compared them with TempoRL (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. exploration is very small. With this maximum persistence value Per$Q$-learning outperforms TempoRL. Further experiments with different values of $K_{\max}$ have been reported in Appendix G.1. In general, Per$Q$-learning shows faster rates of improvements than TempoRL, especially in the first learning iterations. However, this advantage may not be consistent for

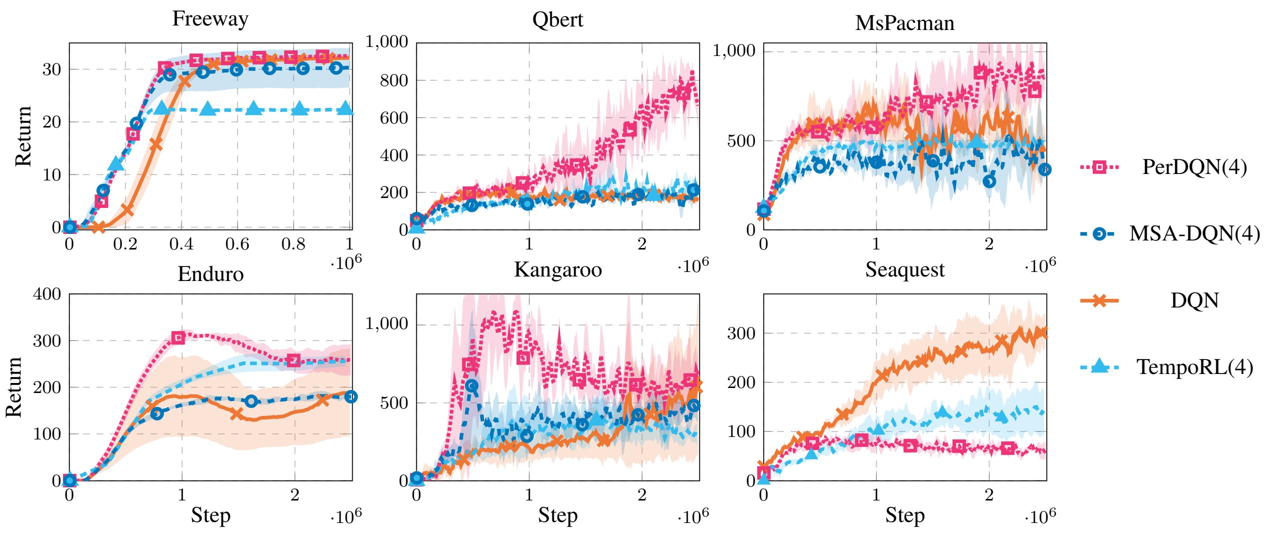

Figure 5: Atari games results. Parenthesis in the legend denote the maximum persistence $K_{\max}$. 5 runs (avg± 95% c.i.).

every value of $K_{\max}$, and every environment, as also shown in Appendix G.1.

PerDQN Our implementation of PerDQN is based on OpenAI Gym (Brockman et al. 2016) and Baselines (Dhariwal et al. 2017) Python toolkits. We start with MountainCar (Moore 1991), as it is perhaps the most suited to evaluate the performance of persistence options. As shown in (Metelli et al. 2020), 1-step explorative policies usually fail to reach the goal. Figure 3 shows that TempoRL and DQN cannot converge to the optimal policy, as already noticed in (Biedenkapp et al. 2021), while PerDQN attains the optimal solution, that reaches the top of the mountain with the minimum loss.

The algorithm is then tested in the challenging framework of Atari 2600 games, where we want to validate that action persistence is beneficial to speed up the initial phases of learning also in large environments.

The same architecture from (Mnih et al. 2013), suitably modified as in the previous section, is used for all environments. For a fair comparison with TempoRL and standard DQN, persistence is implemented on top of the *frame skip*. Thus, a one-step transition corresponds to 4 frame skips. In Figure 5 we compare PerDQN with TempoRL and classic DQN. In five games out of six, 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 (van Hasselt, Guez, and Silver 2016). In order to better understand which beneficial effects are provided by action persistence alone and which ones derive from the use of bootstrap operator, we run an ablation experiment on the same tasks removing the latter one. The resulting algorithm is then similar to the Deep RL version of MSA-Q-learning (Schoknecht and Riedmiller 2003), which we called MSA-DQN. The results show that PerDQN always dominates over its counterpart without bootstrap. The crucial importance of the bootstrap operator is confirmed also in the MountainCar setting where removing this feature causes a performance decrease, making its score comparable to TempoRL (see Appendix G.2). Finally, we notice that in Seaquest persistence seems to be detrimental for learning, as DQN clearly outperforms PerDQN. In this task, agents have to choose either to move or to shoot some moving targets. Persisting the shooting action, thus, may force the agent to stay still for a long time, hitting nothing. A possible solution could consist in the introduction of *interrupting persistence*, in a similar fashion to interrupting options (Sutton, Precup, and Singh 1999; Mankowitz, Mann, and Mannor 2014), which is an interesting future research direction.

Discussion and Conclusions

In this paper, we 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 at any time scale, as action-value function estimates can be updated simultaneously on the whole persistence set. In particular, low persistences (and primitive actions) can be updated by splitting the samples in their sub-transitions; high persistences can instead be improved by *bootstrap*, i.e. by estimating the partial missing information. After proving that the new operator is a contraction, we extended classic Q-learning and DQN to their persistent version. The empirical analysis underlines the benefits of the new operator for exploration and estimation. Furthermore, the experimental campaign on tabular and deep RL settings demonstrated the effectiveness of our approach and the importance of considering temporal extended actions, as well as some limitations. Future research directions include the introduction of persistence interruption and techniques to overcome the overestimation bias. Furthermore, one could investigate the us of the operator in the actor-critic framework to cope with continuous actions.

References

Amin, S.; Gomrokchi, M.; Aboutalebi, H.; Satija, H.; and Precup, D. 2021. Locally Persistent Exploration in Continuous Control Tasks with Sparse Rewards. In *International Conference on Machine Learning*, 275–285. PMLR.

Arulkumaran, K.; Dilokthanakul, N.; Shanahan, M.; and Bharath, A. A. 2016. Classifying options for deep reinforcement learning. *arXiv preprint arXiv:1604.08153*.

Baird, L. C. 1994. Reinforcement learning in continuous time: Advantage updating. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 4, 2448–2453. IEEE.

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

Bertsekas, D.; and Shreve, S. E. 1996. *Stochastic optimal control: the discrete-time case*, volume 5. Athena Scientific.

Biedenkapp, A.; Rajan, R.; Hutter, F.; and Lindauer, M. T. 2021. TempoRL: Learning When to Act. In *ICML*.

Braylan, A.; Hollenbeck, M.; Meyerson, E.; and Miikkulainen, R. 2015. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Catral, M.; Kirkland, S. J.; Neumann, M.; and Sze, N.-S. 2010. The Kemeny constant for finite homogeneous ergodic Markov chains. *Journal of Scientific Computing*, 45(1): 151–166.

Dabney, W.; Ostrovski, G.; and Barreto, A. 2020. Temporally-Extended ϵ-Greedy Exploration. In *International Conference on Learning Representations (ICLR)*.

Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; and Zhokhov, P. 2017. Openai baselines. https://github.com/openai/baselines.

G. Bellemare, M.; Ostrovski, G.; Guez, A.; Thomas, P.; and Munos, R. 2016. Increasing the Action Gap: New Operators for Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Grigsby, J.; Yoo, J. Y.; and Qi, Y. 2021. Towards Automatic Actor-Critic Solutions to Continuous Control. In *Deep RL Workshop NeurIPS 2021*.

Gu, S.; Holly, E.; Lillicrap, T.; and Levine, S. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, 3389–3396. IEEE.

Haarnoja, T.; Ha, S.; Zhou, A.; Tan, J.; Tucker, G.; and Levine, S. 2019. Learning to Walk Via Deep Reinforcement Learning. In Bicchi, A.; Kress-Gazit, H.; and Hutchinson, S., eds., *Robotics: Science and Systems XV*.

Kalyanakrishnan, S.; Aravindan, S.; Bagdawat, V.; Bhatt, V.; Goka, H.; Gupta, A.; Krishna, K.; and Piratla, V. 2021. An Analysis of Frame-skipping in Reinforcement Learning. *arXiv preprint arXiv:2102.03718*.

Kearns, M.; and Singh, S. 1999. Finite-sample convergence rates for Q-learning and indirect algorithms. *Advances in Neural Information Processing Systems (NIPS)*, 996–1002.

Khan, A.; Feng, J.; Liu, S.; and Asghar, M. Z. 2019. Optimal skipping rates: training agents with fine-grained control using deep reinforcement learning. *Journal of Robotics*, 2019.

Kilinc, O.; Hu, Y.; and Montana, G. 2019. Reinforcement Learning for Robotic Manipulation using Simulated Locomotion Demonstrations. *CoRR*, abs/1910.07294.

Kiran, B. R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A. A.; Yogamani, S.; and Pérez, P. 2021. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*.

Kober, J.; and Peters, J. 2014. *Learning Motor Skills - From Algorithms to Robot Experiments*, volume 97 of *Springer Tracts in Advanced Robotics*. Springer. ISBN 978-3-319-03193-4.

Lakshminarayanan, A. S.; Sharma, S.; and Ravindran, B. 2017. Dynamic Action Repetition for Deep Reinforcement Learning. In Singh, S. P.; and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*, 2133–2139. AAAI Press.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In Bengio, Y.; and LeCun, Y., eds., *4th International Conference on Learning Representations (ICLR)*.

Mankowitz, D. J.; Mann, T. A.; and Mannor, S. 2014. Time regularized interrupting options. In *Internation Conference on Machine Learning (ICML)*.

Mann, T. A.; Mannor, S.; and Precup, D. 2015. Approximate Value Iteration with Temporally Extended Actions. *J. Artif. Intell. Res.*, 53: 375–438.

Metelli, A. M.; Mazzolini, F.; Bisi, L.; Sabbioni, L.; and Restelli, M. 2020. Control frequency adaptation via action persistence in batch reinforcement learning. In *International Conference on Machine Learning (ICML)*, 6862–6873. PMLR.

Metelli, A. M.; Mutti, M.; and Restelli, M. 2018. Configurable Markov Decision Processes. In Dy, J. G.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, 3488–3497. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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

Moore, A. W. 1991. Efficient memory based learning for robot control. *PhD Thesis, Computer Laboratory, University of Cambridge*.

Park, S.; Kim, J.; and Kim, G. 2021. Time Discretization-Invariant Safe Action Repetition for Policy Gradient Methods. *Advances in Neural Information Processing Systems (NeurIPS)*, 34.

Patel, R.; Agharkar, P.; and Bullo, F. 2015. Robotic surveillance and Markov chains with minimal weighted Kemeny constant. *IEEE Transactions on Automatic Control*, 60(12): 3156–3167.

Precup, D. 2001. *Temporal abstraction in reinforcement learning.* Ph.D. thesis, University of Massachusetts Amherst.

Puterman, M. L. 2014. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Schoknecht, R.; and Riedmiller, M. 2003. Reinforcement learning on explicitly specified time scales. *Neural Computing & Applications*, 12(2): 61–80.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

Sharma, S.; Srinivas, A.; and Ravindran, B. 2017. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*.

Sidford, A.; Wang, M.; Wu, X.; Yang, L. F.; and Ye, Y. 2018. 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*, 5192–5202.

Singh, S.; Jaakkola, T.; Littman, M. L.; and Szepesvári, C. 2000. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3): 287–308.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.*, 112(1-2): 181–211.

Tallec, C.; Blier, L.; and Ollivier, Y. 2019. Making Deep Q-learning methods robust to time discretization. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, 6096–6104. PMLR.

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Watkins, C. J. C. H. 1989. *Learning from delayed rewards*. Ph.D. thesis, King's College, University of Cambridge.

Yu, H.; Xu, W.; and Zhang, H. 2021. TAAC: Temporally Abstract Actor-Critic for Continuous Control. *Advances in Neural Information Processing Systems (NeurIPS)*, 34.