Making Deep Q-learning Methods Robust to Time Discretization

Corentin Tallec [1] **Léonard Blier** [1 2] **Yann Ollivier** [2]

Abstract

Despite remarkable successes, *Deep Reinforcement Learning* (DRL) is not robust to hyperparameterization, implementation details, or small environment changes (Henderson et al. 2017, Zhang et al. 2018). Overcoming such sensitivity is key to making DRL applicable to real world problems. In this paper, we identify sensitivity to *time discretization* in near continuous-time environments as a critical factor; this covers, e.g., changing the number of frames per second, or the action frequency of the controller. Empirically, we find that Q-learning-based approaches such as *Deep Q-learning* (Mnih et al., 2015) and *Deep Deterministic Policy Gradient* (Lillicrap et al., 2015) collapse with small time steps. Formally, we prove that Q-learning does not exist in continuous time. We detail a principled way to build an off-policy RL algorithm that yields similar performances over a wide range of time discretizations, and confirm this robustness empirically.

1. Introduction

In recent years, *Deep Reinforcement Learning* (DRL) approaches have provided impressive results in a variety of domains, achieving superhuman performance with no expert knowledge in perfect information zero-sum games (Silver et al., 2017), reaching top player level in video games (OpenAI 2018b, Mnih et al. 2015), or learning dexterous manipulation from scratch without demonstrations (OpenAI, 2018a). In spite of those successes, DRL approaches are sensitive to a number of factors, including hyperparameterization, implementation details or small changes in the environment parameters (Henderson et al. 2017, Zhang et al. 2018). This sensitivity, along with sample inefficiency, largely prevents DRL from being applied in real world settings. Notably, high sensitivity to environment parameters prevents transfer from imperfect simulators to real world scenarios.

In this paper we focus on the sensitivity to time discretization of DRL approaches, such as what happens when an agent receives 50 observations and is expected to take 50 actions per second instead of 10. In principle, decreasing time discretization, or equivalently shortening reaction time, should only improve agent performance. Robustness to time discretization is especially relevant in *near-continuous* environments, which includes most continuous control environments, robotics, and many video games.

Standard approaches based on estimation of state-action value functions, such as *Deep Q-learning* (DQN, Mnih et al. 2015) and *Deep deterministic policy gradient* (DDPG, Lillicrap et al. 2015) are not at all robust to changes in time discretization. This is shown experimentally in Sec. 5. Intuitively, as the discretization timestep decreases, the effect of individual actions on the total return decreases too: $Q^*(s, a)$ is the value of playing action a then playing optimally, and if a is only maintained for a very short time its advantage over other actions will be accordingly small. (This occurs even with a suitably adjusted decay factor γ.) If the discretization timestep becomes infinitesimal, the effect of every individual action vanishes: there is no continuous-time Q-function (Thm. 2), hence the poor performance of Q-learning with small time steps. These statements can be fully formalized in the framework of continuous-time reinforcement learning (Sec. 3) (Doya, 2000; Baird, 1994).

We focus on continuous time because this leads to a clear theoretical framework, but our observations make sense in any setting in which the value results from taking a large number of small individual actions. Our results suggest standard Q-learning will fail in such settings without a delicate balance of hyperparameter scalings and reparameterizations.

We are looking for an algorithm that would be as invariant as possible to changing the discretization timestep. Such an algorithm should remain viable, in the sense that it should still learn, when this timestep is small, and in particular admit a continuous-time limit when the discretization timestep goes to 0. This leads to precise design choices in terms of agent architecture, exploration policy, and learning rates scalings. The resulting algorithm is shown to provide better invariance to time discretization than vanilla DQN or DDPG,

[1]TAckling the Underspecified, Université Paris Sud [2]Facebook Artificial Intelligence Research. Correspondence to: Corentin Tallec <corentin.tallec@inria.fr>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

on many environments (Sec. 5). On a new environment, as soon as the order of magnitude of the time discretization is known, our analysis readily provides relevant scalings for a number of hyperparameters.

Our contribution is threefold:

- Building on (Baird, 1994), we formally show that the Q-function collapses to the V-function in near-continuous time, and thus that standard Q-learning is ill-behaved in this setting.
- Our analysis of properties in the continuous-time limit leads to a robust off-policy algorithm. In particular, it provides insights on architecture design, and constrains exploration schemes and learning rates scalings.
- We empirically show that standard Q-learning methods are not robust to changes in time discretization, exhibiting degraded performance, while our algorithm demonstrates substantial robustness.

To the best of our knowledge, Thms 1 and 2 were known to (Doya, 2000; Baird, 1994), but not formally proven, while Thms 3, 4 and 5 are novel.

2. Related Work

Our approach builds on (Baird, 1994), who identified the collapse of Q-learning for small time steps and, as a solution, suggested the Advantage Updating algorithm, with proper scalings for the V and advantage parts depending on timescale δt; testing was only done on a quadratic-linear problem.

We expand on (Baird, 1994) in several directions. First, we modify the algorithm by using a different normalization step for A, which forgoes the need to learn the normalization itself, thanks to the parameterization (26). Second, we test Advantage Updating for the first time on a variety on RL environments using deep networks, establishing Deep Advantage Updating as a viable algorithm in this setting. Third, we provide formal proofs in a general setting for the collapse of Q-learning when the timescale δt tends to 0, and for the non-collapse of Advantage Updating with the proper scalings. Fourth, we also discuss how to obtain δt-invariant exploration. Fifth, we provide stringent experimental tests of the actual robustness to changing δt.

Our study focuses on off-policy algorithms. Some on-policy algorithms, such as A3C (Mnih et al., 2016), PPO (Schulman et al., 2017) or TRPO (Schulman et al., 2015) may be time discretization invariant with specific setups. This is out of the scope of our work and would require a separate study.

(Wang et al., 2015) also use a parameterization separating the value and advantage components of the Q-function. But contrary to (Baird, 1994)'s Advantage Updating, learning is still done in a standard way on the Q-function obtained from adding these two components. Thus this approach reparameterizes Q but does not change scalings and does not result in an invariant algorithm for small δt.

The problem studied here is a continuity effect quite distinct from multiscale RL approaches: indeed the issue arises even if there is only one timescale in the environment. Arguably, a small δt can be seen as a mismatch between the algorithm's timescale and the physical system's timescale, but the collapse of the Q function to the V function is an intrinsic mathematical phenomenon arising from time continuity.

Reinforcement learning has been studied from a mathematical perspective when time and space are both continuous, in connection with optimal control and the Hamilton–Jacobi–Bellman (HJB) equation (a PDE which characterizes the value function for continuous space-time). Explicit algorithms for continuous space-time can be found in (Doya, 2000; Munos & Bourgine, 1998) (see also the references therein). (Munos & Bourgine, 1998) use a grid approach to provably solve the HJB equation when discretization tends to 0 (assuming every state in the grid is visited a large number of times). However, the resulting algorithms are impractical (Doya, 2000) for larger-dimensional problems. (Doya, 2000) focusses on algorithms specific to the continuous space-time case, including advantage updating and modelling the time derivative of the environment.

Here on the other hand we focus on generic deep RL algorithms that can handle both discrete and continuous time and space, without collapsing in continuous time, thus being robust to arbitrary timesteps.

3. Near Continuous-Time Reinforcement Learning

Many reinforcement learning environments are not intrinsically time-discrete, but discretizations of an underlying continuous-time environment. For instance, many simulated control environments, such as the Mujoco environments (Lillicrap et al., 2015) or OpenAI Gym classic control environments (Brockman et al., 2016), are discretizations of continuous-time control problems. In simulated environments, the time discretization is fixed by the simulator, and is often used to approximate an underlying differential equation. In this case, the timestep may correspond to the number of frames generated by second. In real world environments, sensors and actuators have a fixed time precision: cameras can only capture a fixed amount of frames per second, and physical limitations prevent actuators from responding instantaneously. The quality of these components thus imposes a lower bound on the discretization timestep. As the timestep δt is a constraint imposed by computational ressources, we would expect that decreasing δt would only improve the performance of RL agents (though it might

make optimization harder). RL algorithms should, at least, be resilient to a change of δt, and remain viable when $\delta t \to 0$. Besides, designing a time discretization invariant algorithm could alleviate tedious hyperparameterization by providing better defaults for time-horizon-related parameters.

We are thus interested in the behavior of RL algorithms in discretized environments, when the discretization timestep is small. We will refer to such environments as *near-continuous environments*. A formalized view of near-continuous environments is given below, along with δt-dependent definitions of return, discount factor and value functions, that converge to well defined continous-time limits. The state-action value function is shown to collapse to the value function as δt goes to 0. Consequently there is no Q-learning in continuous time, foreshadowing problematic behavior of Q-learning with small timesteps.

3.1. Framework

Let $\mathcal{S} = \mathbb{R}^d$ be a set of states, and $\mathcal{A}$ be a set of actions. Consider the continuous-time *Markov Decision Process* (MDP) defined by the differential equation

$$ds_t/dt = F(s_t, a_t) \tag{1}$$

where $F\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ describes the dynamics of the environment. The agent interacts with the environment through a deterministic policy function $\pi\colon \mathcal{S} \to \mathcal{A}$, so that $a_t = \pi(s_t)$. Actions can be discrete or continuous. For simplicity we assume here that both the dynamics and exploitation policy are deterministic; the exploration policy will be random, but care must be taken to define proper random policies in continuous time, especially with discrete actions (Sec. 4.2).

For any timestep $\delta t > 0$, we can define an MDP $\mathcal{M}_{\delta t} = \langle \mathcal{S}, \mathcal{A}, T_{\delta t}, r_{\delta t}, \gamma_{\delta t} \rangle$ as a discretization of the continuous-time MDP with *time discretization* δt. The transition function of a state s is the state obtained when starting at $s_0 = s$ and maintaining $a_t = a$ constant for a time δt. This corresponds to an agent evolving in the continuous environment (1), but only making observations and choosing actions every δt. The rewards and decay factor are specified below. We call such an MDP $\mathcal{M}_{\delta t}$ *near-continuous*.

A necessary condition for robustness of an algorithm for near-continuous time MDPs is to remain viable when $\delta t \to 0$. Thus we will try to make sure the various quantities involved converge to meaningful limits when $\delta t \to 0$.

We give semi-formal statements below; the full statements, proofs, and technical assumptions (typically, differentiability assumptions) can be found in the supplementary material.

Return and discount factor. Suitable δt-dependent scalings of the discount factor $\gamma_{\delta t}$ and reward $r_{\delta t}$ are as follows. These definitions fit the discrete case when $\delta t = 1$, and provide well-defined, non-trivial returns and value functions when δt goes to 0.

For a continuous MDP and a continuous trajectory $\tau = (s_t, a_t)_t$, the return is defined as (Doya, 2000)

$$R(\tau) := \int_{t=0}^{\infty} \gamma^t \, r(s_t, a_t) \, dt. \tag{2}$$

A natural time discretization is obtained by defining the discretized return $R_{\delta t}$ of the MDP $\mathcal{M}_{\delta t}$ as

$$R_{\delta t}(\tau) := \sum_{k=0}^{\infty} \gamma^{k\delta t} \, r(s_{k\delta t}, a_{k\delta t}) \, \delta t \tag{3}$$

and the discretized return will correspond to the continuous-time return if we set the decay factor $\gamma_{\delta t}$ and rewards $r_{\delta t}$ of the discretized MDP $\mathcal{M}_{\delta t}$ to

$$\gamma_{\delta t} := \gamma^{\delta t}, \qquad r_{\delta t} := \delta t . r. \tag{4}$$

Physical time vs algorithmic time, time horizon. In near-continuous environments, there are two notions of time: the algorithmic time k (number of steps or actions taken), and the physical time t (time spent in the underlying continuous time environment), related via $t = k.\delta t$.

The time horizon is, informally, the time range over which the agent optimizes its return. As a rule of thumb, the time horizon of an agent with discount factor γ is of order $\frac{1}{1-\gamma}$ steps; beyond that, the decay factor kicks in and the influence of further rewards becomes small.

The definition (4) of the decay factor $\gamma_{\delta t}$ in near-continuous environments keeps the time horizon constant in *physical* time, by making $\gamma_{\delta t}$ close to 1 in algorithmic time. Indeed, physical time horizon is δt times the algorithmic time horizon, namely

$$\frac{\delta t}{1 - \gamma^{\delta t}} = -\frac{1}{\log \gamma} + O(\delta t) \approx \frac{1}{1-\gamma}, \tag{5}$$

which is thus stable when $\delta t \to 0$. On the other hand, if $\gamma_{\delta t}$ was left constant as δt goes to 0, the corresponding time horizon in physical time would be $\approx \frac{\delta t}{1-\gamma^{\delta t}}$ which goes to 0 when δt goes to 0: such an agent would be increasingly short-sighted as $\delta t \to 0$.

In the following, we use the suitably-scaled decay factor (4) both for Deep Advantage Updating and for the classical deep Q-learning baselines.

Value function. The return (2) leads to the following continuous-time value function

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} \left[R(\tau) \mid s_0 = s \right] \tag{6}$$

$$= \mathbb{E}_{\tau \sim \pi} \left[\int_0^{\infty} \gamma^t \, r(s_t, a_t) \, dt \mid s_0 = s \right]. \tag{7}$$

Meanwhile, the value function (in the ordinary sense) of the discrete MDP $\mathcal{M}_{\delta t}$ is

$$V^\pi_{\delta t}(s) = \mathbb{E}_{\tau\sim\pi}\left[R_{\delta t}(\tau) \mid s_0 = s\right] \tag{8}$$

$$= \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=0}^{\infty} \gamma^{k\delta t}\, r(s_{k\delta t}, a_{k\delta t})\, \delta t \mid s_0 = s\right] \tag{9}$$

which obeys the Bellman equation

$$V^\pi_{\delta t}(s) = r(s,\pi(s))\,\delta t + \gamma^{\delta t}\, \mathbb{E}_{s_{(k+1)\delta t}|s_{k\delta t}=s} V^\pi_{\delta t}(s_{(k+1)\delta t}) \tag{10}$$

When the timestep tends to 0, one converges to the other.

Theorem 1. *Under suitable smoothness assumptions, $V^\pi_{\delta t}(s)$ converges to $V^\pi(s)$ when $\delta t \to 0$.*

3.2. There is No Q-Function in Continuous Time

Contrary to the value function, the action-value function Q is ill-defined for continuous-time MDPs. More precisely, the Q-function collapses to the V-function when $\delta t \to 0$. In near continuous time, the effect of individual actions on the Q-function is of order $O(\delta t)$. This will make ranking of actions difficult, especially with an approximate Q-function. This argument appears informally in (Baird, 1994). Formally:

Theorem 2. *Under suitable smoothness assumptions, The action-value function of a near-continuous MDP is related to its value function via*

$$Q^\pi_{\delta t}(s,a) = V^\pi_{\delta t}(s) + O\left(\delta t\right) \tag{11}$$

when $\delta t \to 0$, for every (s,a).

As a consequence, in exact continuous time, Q^π is equal to V^π: it does not bear *any* information on the ranking of actions, and thus cannot be used to select actions with higher returns. There is no continuous-time Q-learning.

Proof. The discrete-time Q-function of the MDP $\mathcal{M}_{\delta t}$ satisfies the Bellman equation

$$Q^\pi_{\delta t}(s,a) = r(s,a)\,\delta t + \gamma^{\delta t}\mathbb{E}_{s'|s,a}\left[V^\pi_{\delta t}(s')\right]. \tag{12}$$

The dynamics (1) of the environment yields

$$s' = s + F(s,a)\,\delta t + o(\delta t). \tag{13}$$

Assuming that $V^\pi_{\delta t}$ is continuously differentiable with respect to the state, and that its derivatives are uniformly bounded, we obtain,

$$V^\pi_{\delta t}(s') = V^\pi_{\delta t}(s) + \nabla_s V^\pi_{\delta t}(s)\cdot F(s,a)\,\delta t + o(\delta t) \tag{14}$$

$$= V^\pi_{\delta t}(s) + O(\delta t) \tag{15}$$

Expanding $V^\pi_{\delta t}(s')$ into $Q^\pi_{\delta t}$ yields

$$Q^\pi_{\delta t}(s,a) = r(s,a)\,\delta t + \gamma^{\delta t}(V^\pi_{\delta t}(s) + O(\delta t)) \tag{16}$$

$$= O(\delta t) + (1 + O(\delta t))(V^\pi_{\delta t}(s) + O(\delta t)) \tag{17}$$

$$= V^\pi_{\delta t}(s) + O(\delta t). \tag{18}$$

which ends the proof. □

4. Reinforcement Learning with a Continuous-Time Limit

We now define a discrete algorithm with a well-defined continuous-time limit. It relies on three elements: defining and learning a quantity that still contains information on action rankings in the limit, using exploration methods with a meaningful limit, and scaling learning rates to induce well-behaved parameter trajectories when δt goes to 0.

4.1. Advantage Updating

As seen above, there is no continuous time limit to Q-learning, because Q^π becomes independent of actions and thus cannot be used to select actions. With small but nonzero δt, $Q^\pi_{\delta t}$ still depends on actions, and could still be used to choose actions. However, when approximating $Q^\pi_{\delta t}$, if the approximation error is much larger than $O(\delta t)$, this error dominates, the ranking of actions given by the approximated $Q^\pi_{\delta t}$ is likely to be erroneous.

To define an object which contains the same information on actions as $Q^\pi_{\delta t}$, but admits a learnable action-dependent limit, it is natural to define (Baird, 1994)

$$A^\pi_{\delta t}(s,a) := \frac{Q^\pi_{\delta t}(s,a) - V^\pi_{\delta t}(s)}{\delta t}, \tag{19}$$

a rescaled version of the advantage function, as the difference between between $Q^\pi_{\delta t}(s,a)$ and $V^\pi_{\delta t}(s)$ is of order $O(\delta t)$. This amounts to splitting Q into value and advantage, and observing that these scale very differently when $\delta t \to 0$.

Contrary to the Q-function, this rescaled advantage function converges when $\delta t \to 0$ to a non-degenerate action-dependent quantity.

Theorem 3. *Under suitable smoothness assumptions, $A^\pi_{\delta t}(s,a)$ has a limit $A^\pi(s,a)$ when $\delta t \to 0$. The limit keeps information about actions: namely, if a policy π' strictly dominates π, then $A^\pi(s,\pi'(s)) > 0$ for some state s.*

Learning A^π. The discretized Q-function rewrites as

$$Q^\pi_{\delta t}(s,a) = V^\pi_{\delta t}(s) + \delta t A^\pi_{\delta t}(s,a). \tag{20}$$

A natural way to approximate $V^\pi_{\delta t}$ and $A^\pi_{\delta t}$ is to apply *Sarsa* or Q-learning to a reparameterized Q-function approximator

$$Q_\Theta(s,a) := V_\theta(s) + \delta t A_\psi(s,a). \tag{21}$$

with $\Theta := (\theta, \psi)$. At initialization, if both V_θ and A_ψ are initialized independently of δt, this parameterization provides reasonable scaling of the contribution of actions versus states in Q. Our goal is for V_θ to approximate $V^\pi_{\delta t}$ and for A_ψ to approximate $A^\pi_{\delta t}$.

Still, this reparameterization does not, on its own, guarantee that A correctly approximates $A^\pi_{\delta t}$ if Q_Θ approximates $Q^\pi_{\delta t}$. Indeed, for any given pair $(V_\theta,\ A_\psi)$, the pair $(V_\theta(s) - f(s),\ A_\psi(s,a) + f(s)/\delta t)$ (for an arbitrary f) yields the exact same function Q_Θ. This new A_ψ still defines the same ranking of actions, yet this phenomenon might cause numerical problems or instability of A_ψ when $\delta t \to 0$, and prevents direct interpretation of the learned A_ψ. To enforce identifiability of A_ψ, one must enforce the consistency equation

$$V^\pi_{\delta t}(s) = Q^\pi_{\delta t}(s, \pi(s)) \tag{22}$$

on the approximate A_ψ and V_θ. This translates to

$$A_\psi(s, \pi(s)) = 0. \tag{23}$$

With this additional constraint, if $Q_\Theta = Q^\pi_{\delta t}$, then $A_\psi = A^\pi_{\delta t}$ and $V_\theta = V^\pi_{\delta t}$: indeed

$$A^\pi_{\delta t}(s,a) = \frac{Q^\pi_{\delta t}(s,a) - V^\pi_{\delta t}(s)}{\delta t} \tag{24}$$

$$= \frac{Q_\Theta(s,a) - Q_\Theta(s,\pi(s))}{\delta t} = A_\psi(s,a). \tag{25}$$

In the spirit of (Wang et al., 2015), instead of directly parameterizing A_ψ, we define a parametric function $\bar{A}_\psi$ (typically a neural network), and use $\bar{A}_\psi$ to define A_ψ as

$$A_\psi(s,a) := \bar{A}_\psi(s,a) - \bar{A}_\psi(s, \pi(s)) \tag{26}$$

so that A_ψ directly verifies the consistency condition.

This approach will lead to δt-robust algorithms for approximating $A^\pi_{\delta t}$, from which a ranking of actions can be derived.

4.2. Timestep-Invariant Exploration

To obtain a timestep-invariant RL algorithm, a timestep-invariant exploration scheme is required. For *continuous* actions, (Lillicrap et al., 2015) already introduced such a scheme, by adding an *Ornstein–Uhlenbeck* (Uhlenbeck & Ornstein, 1930) (OU) random process to the actions. Formally, this is defined as

$$\pi^{\text{explore}}(s_{k\delta t}, z_{k\delta t}) := \pi(s_{k\delta t}) + z_{k\delta t} \tag{27}$$

with $z_{k\delta t}$ the discretization of a continuous-time OU process,

$$dz_t = -z_t\,\kappa\,dt + \sigma\,dB_t. \tag{28}$$

where B_t is a brownian motion, κ a stiffness parameter and σ a noise scaling parameter. The discretized trajectories of z converge to nontrivial continuous-time trajectories, exhibiting Brownian behavior with a recall force towards 0.

This exploration can be extended to schemes of the form

$$a_{k\delta t} = \pi^{\text{explore}}_{\delta t}(s_{k\delta t}, z_{k\delta t}) \tag{29}$$

with $(z_{k\delta t})_{k\geq 0}$ a sequence of random variables independent from the a's and s's. A sufficient condition for this policy to admit a continuous-time limit is for the sequence $z_{k\delta t}$ to converge in law to a well-defined continuous stochastic process z_t as δt goes to 0.

Thus, for *discrete* actions we can obtain a consistent exploration scheme by taking $z_{\delta t}$ to be a discretization of an $(\#\mathcal{A})$-dimensional continuous OU process, and setting

$$\pi^{\text{explore}}(s_{k\delta t}, z_{k\delta t}) := \operatorname*{argmax}_a \left(A_\psi(s_{k\delta t}, a) + z_{k\delta t}[a]\right)$$

where $z_{k\delta t}[a]$ denotes the a-th component of $z_{k\delta t}$. Namely, we perturb the *advantage values* by a random process before selecting an action. The resulting scheme converges in continuous time to a nontrivial exploration scheme.

On the other hand, ε-greedy exploration is likely *not* to explore, i.e., to collapse to a deterministic policy, when δt goes to 0. Intuitively, with very small δt, changing the action at random every δt time step just averages out the randomness due to the law of large numbers. More precisely:

Theorem 4. *Consider a near-continuous MDP in which an agent selects an action according to an ε-greedy policy that mixes a deterministic exploitation policy π with an action taken from a noise policy $\pi^{noise}(a|s)$ with probability ε at each step. Then the agent's trajectories converge when $\delta t \to 0$ to the solutions of the* deterministic *equation*

$$ds_t/dt = (1-\varepsilon)F(s_t, \pi(s_t)) + \varepsilon \mathbb{E}_{a\sim\pi^{noise}(a|s)} F(s_t, a)$$

4.3. Algorithms for Deep Advantage Updating

We learn V_θ and A_ψ via suitable variants of Q-learning for continuous and discrete action spaces. Namely, the true $A^\pi_{\delta t}$ and $V^\pi_{\delta t}$ of a near-continuous MDP with greedy exploitation policy $\pi(s) := \text{argmax}_{a'} A^\pi_{\delta t}(s,a')$ are the unique solution to the Bellman and consistency equations

$$V^\pi_{\delta t}(s) + \delta t\, A^\pi_{\delta t}(s,a) = r\,\delta t + \gamma^{\delta t}\,\mathbb{E}_{s'} V^\pi_{\delta t}(s') \tag{30}$$

$$A^\pi_{\delta t}(s, \pi(s)) = 0. \tag{31}$$

as seen in 4.1. Thus V_θ and A_ψ are trained to approximately solve these equations.

Maximization over actions for π is implemented exactly for discrete actions, and, for continuous actions, approximated by a policy neural network $\pi_\phi(s)$ trained to maximize $A_\psi(s, \pi_\phi(s))$, similarly to (Lillicrap et al., 2015).

Eq. (31) is directly verified by A_ψ, owing to the reparametrization $A_\psi(s,a) = \bar{A}_\psi(s,a) - \bar{A}_\psi(s,\pi(s))$, described in 4.1. To approximately verify (30), the corresponding squared Bellman residual is minimized by an approximate gradient descent. The update equations when learning

Algorithm 1 DAU (Continuous actions)

Inputs:
θ, ψ and ϕ, parameters of V_θ, $\bar{A}_\psi$ and π_ϕ.
π^{explore} and $\nu_{\delta t}$ defining an exploration policy.
$\mathbf{opt}_V$, $\mathbf{opt}_A$, $\mathbf{opt}_\pi$, $\alpha^V \delta t$, $\alpha^A \delta t$ and $\alpha^\pi \delta t$, optimizers and learning rates.
$\mathcal{D}$, buffer of transitions (s, a, r, d, s'), with d the episode termination signal.
δt and γ, time discretization and discount factor.
nb_epochs, **nb_steps**, **nb_learn** number of epochs, steps and learning steps per epoch.
N, training batch size

Observe initial state s^0, set $t \leftarrow 0$
for $e = 0$, **nb_epochs do**
  for $j = 1$, **nb_steps do**
    $a^t \leftarrow \pi^{\text{explore}}(s^t, \nu^t_{\delta t})$.
    Perform a^t and observe $(r^{t+1}, d^{t+1}, s^{t+1})$.
    Store $(s^t, a^t, r^{t+1}, d^{t+1}, s^{t+1})$ in $\mathcal{D}$.
    $t \leftarrow t + 1$
  end for
  for $k = 0$, **nb_learn do**
    Sample a batch of N random transitions from $\mathcal{D}$
    for i=0, N **do**
      $\delta Q^i \leftarrow \delta t\big(\bar{A}_\psi(s^i, a^i) - \bar{A}_\psi(s^i, \pi_\phi(s^i))\big)$
        $\quad - \big(r^i \delta t + (1 - d^i)\gamma^{\delta t} V_\theta(s'^i) - V_\theta(s^i)\big)$
    end for
    $\Delta\theta \leftarrow \frac{1}{N} \sum_{i=1}^{N} \frac{\delta Q^i \partial_\theta V_\theta(s^i)}{\delta t}$
    $\Delta\psi \leftarrow \frac{1}{N} \sum_{i=1}^{N} \frac{\delta Q^i \partial_\psi \big(\bar{A}_\psi(s^i, a^i) - \bar{A}_\psi(s^i, \pi_\phi(s^i))\big)}{\delta t}$
    $\Delta\phi \leftarrow \frac{1}{N} \sum_{i=1}^{N} \partial_a \bar{A}_\psi(s^i, \pi_\phi(s^i)) \partial_\phi \pi_\phi(s^i)$
    Update θ with $\mathbf{opt}_V$, $\Delta\theta$ and learning rate $\alpha^V \delta t$.
    Update ψ with $\mathbf{opt}_A$, $\Delta\psi$ and learning rate $\alpha^A \delta t$.
    Update ϕ with $\mathbf{opt}_\pi$, $\Delta\phi$ and learning rate $\alpha^\pi \delta t$.
  end for
end for

from a transition (s, a, r, s'), either from an exploratory trajectory or from a replay buffer (Mnih et al., 2015), are

$$\delta Q_{\delta t} \leftarrow A_\psi(s, a)\, \delta t - \big(r\, \delta t + \gamma^{\delta t}\, V_\theta(s') - V_\theta(s)\big) \quad (32)$$

$$\theta_{\delta t} \leftarrow \theta_{\delta t} - \eta^V_{\delta t}\, \partial_\theta V_\theta(s)\, \frac{\delta Q_{\delta t}}{\delta t} \quad (33)$$

$$\psi_{\delta t} \leftarrow \psi_{\delta t} - \eta^A_{\delta t}\, \partial_\psi A_\psi(s, a)\, \frac{\delta Q_{\delta t}}{\delta t}. \quad (34)$$

where the η's are learning rates. Appropriate scalings for the learning rates $\eta^V_{\delta t}$ and $\eta^A_{\delta t}$ in terms of δt to obtain a well defined continuous limit are derived next.

4.4. Scaling the Learning Rates

For the algorithm to admit a continuous-time limit, the discrete-time trajectories of parameters must converge to well-defined trajectories as δt goes to 0. This in turn imposes precise conditions on the scalings of the learning rates.

Informally, in the parameter updates (32)–(34), the quantity $\delta Q_{\delta t}$ is of order $O(\delta t)$, because $s' = s + O(\delta t)$ in a near-continuous system. Therefore, $\delta Q_{\delta t}/\delta t$ is $O(1)$, so that the gradients used to update θ and ψ are $O(1)$. Therefore, if the learning rates are of order δt, one would expect the parameters θ and ψ to change by $O(\delta t)$ in each time interval δt, thus hopefully converging to smooth continuous-time trajectories. The next theorem formally confirms that learning rates of order δt are the only possibility.

Theorem 5. *Let (s_t, a_t) be some exploration trajectory in a near-continuous MDP. Set the learning rates to $\eta^V_{\delta t} = \alpha^V \delta t^\beta$ and $\eta^A = \alpha^A \delta t^\beta$ for some $\beta \geq 0$, and learn the parameters θ and ψ by iterating (32)–(34) along the trajectory (s_t, a_t). Then, when $\delta t \to 0$:*

- *If $\beta = 1$ the discrete parameter trajectories converge to continuous parameter trajectories.*
- *If $\beta > 1$ the parameters stay at their initial values.*
- *If $\beta < 1$, the parameters can reach infinity in arbitrarily small physical time.*

The resulting algorithm with suitable scalings, *Deep Advantage Updating* (DAU), is specified in Alg. 1 for continuous actions (in the Supplementary for continuous ones). With $\delta t = 1$, DAU is similar to dueling architectures (Wang et al., 2015), but weighs contributions differently for $\delta t \neq 1$.

5. Experiments

The experiments provided here are specifically aimed at showing that the proposed method, DAU, works efficiently over a wide range of time discretizations, without specific tuning, while standard deep Q-learning approaches do not. DAU is compared to DDPG for continuous actions and to DQN for discrete actions. As mentionned earlier, we do not study the time discretization invariance of on-policy methods (A3C, PPO, TRPO...).

In all setups, we use the algorithms described in Alg. 1 and Supplementary Alg. 1. The variants of DDPG and DQN used are described in the Supplementary, as well as all hyperparameters. We tested two different setups for DDPG and DQN. In one, we scaled the discount factor (to avoid shortsightedness with small δt), but left all other hyperparameters constant across time discretizations. In the other, we used the properly rescaled discount factor and reward from Eq. (4), as well as $O(\delta t)$ learning rates

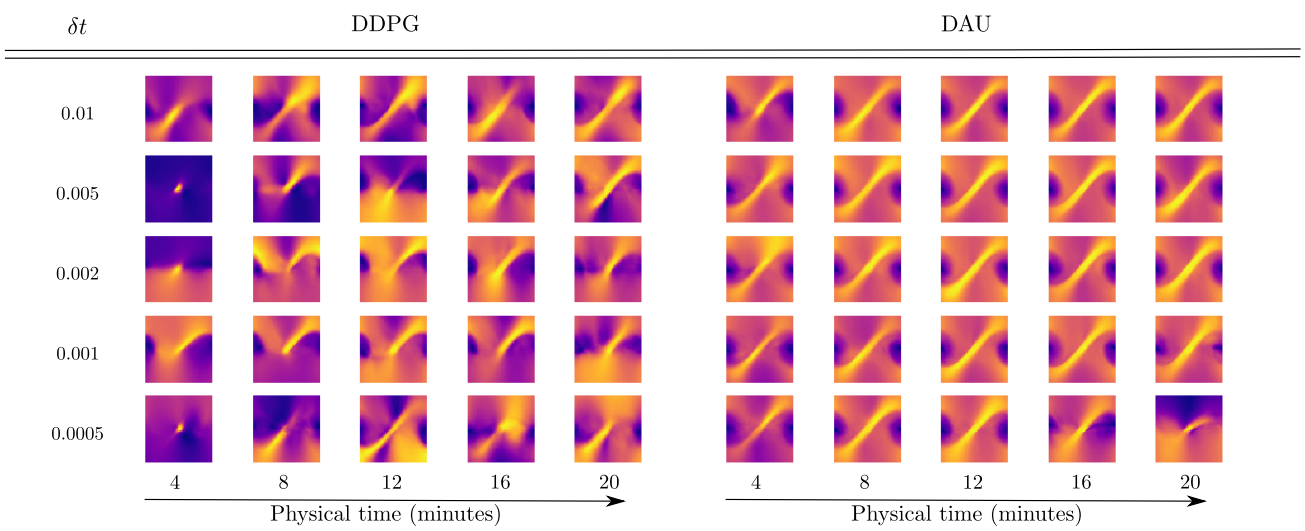

Figure 1. Value functions obtained by DDPG (unscaled version) and DAU at different instants in physical time of training on the pendulum swing-up environment. Each image represents the learnt value function (the x-axis is the angle, and the y-axis the angular velocity). The lighter the pixel, the higher the value.

for RMSProp. The first variant empirically yields slightly better results, and is presented here, with the second variant in the Supplementary. For all setups, quantitative results are averaged over five runs.

Let us stress that the quantities plotted are rescaled to make comparison possible across different timesteps. For example, returns are given in terms of the discretized return $R_{\delta t}$ as defined in (3),[1] and, most notably, time elapsed is always given in *physical* time, i.e., the amount of time that the agent spent interacting with the environment (this is not the number of steps).

Qualitative experiments: Visualizing policies and values. To provide qualitative results, and check robustness to time discretization both in terms of returns and in terms of convergence of the approximate value function and policies, we first provide results on the simple pendulum environment from the OpenAI Gym classic control suite. The state space is of dimension 2. We visualize both the learnt value and policy functions by plotting, for each point of the phase diagram $(\theta, \dot{\theta})$, its value and policy. The results are presented in Fig. 1 (value function) and Figs. 1, 2, 3 in Supplementary.

We plot the learnt policy at several instants in physical time during training, for various time discretizations δt, for both DAU and DDPG. With DAU, the agent's policy and value function quickly converge for every time discretization without specific tuning. On the contrary, with DDPG, learning of both value function and policy vary greatly from one discretization to another.

[1]This amounts to scaling rewards by a factor δt when this scaling is not naturally done in the environment. Environment-specific details are given in the Supplementary.

Quantitative experiments. We benchmark DAU against DDPG on classic control benchmarks[2]: Pendulum, Cartpole, BipedalWalker, Ant, and Half-Cheetah environments from OpenAI Gym (Fig. 2). On Pendulum, Bipedal Walker and Ant, DAU is quite robust to variations of δt and displays reasonable performance in all cases. On the other hand, DDPG's performance varies with δt; performance either degrades as δt decreases (Ant, Cheetah), or the standard deviation across runs increases as learning progresses (Pendulum) for small δt. On Cartpole, noise dominates, making interpretation difficult. On Half-Cheetah, DAU is not clearly invariant to time discretization. This could be explained by the multiple suboptimal regimes that coexist in the Half-Cheetah environment (walking on the head, walking on the back), which create discontinuities in the value function (see Discussion).

6. Discussion

The method derived in this work is theoretically invariant to time discretization, and indeed seems to yield improved timestep robustness on various environments, e.g., simple locomotion tasks. However, on some environments there is still room for improvment. We discuss some of the issues that could explain this theoretical/practical discrepancy.

Note that Alg. 1 requires knowledge of the timestep δt. In most environments, this is readily available, or even directly set by the practitioner: depending on the environment it is given by the frame rate, the maximum frequency of actu-

[2] We also experimented with a continuous action variant of dueling architectures as a baseline, but found that the results were not substantially different from those of vanilla DDPG.

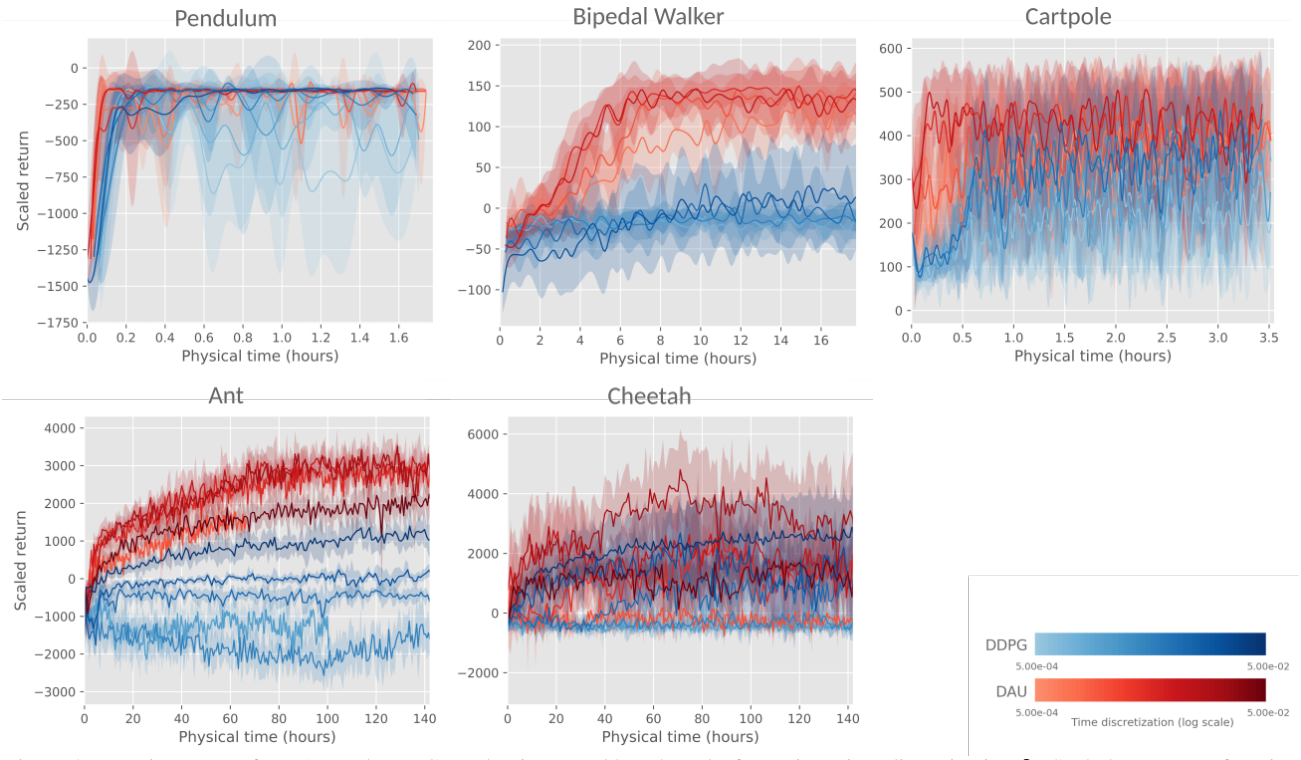

Figure 2. Learning curves for DAU and DDPG on classic control benchmarks for various time discretization δt: Scaled return as a function of the physical time spent in the environment.

ators or observation acquisition, the timestep of a physics simulator, etc.

Smoothness of the value function. In our proofs, V^π is assumed to be continuously differentiable. This hypothesis is not always satisfied in practice. For instance, in the pendulum swing-up environment, depending on initial position and momentum, the optimal policy may need to oscillate before reaching the target state. The optimal value function is discontinuous at the boundary between states where oscillations are needed and those where they are not. This results in non-infinitesimal advantages for actions on the boundary. In such environments where a given policy has different regimes depending on the initial state, the continuous-time analysis only holds almost-everywhere.

Memory buffer size. Thm. 5 is stated for transitions sampled sequentially from a fixed trajectory. In practice, transitions are sampled from a memory replay buffer, to prevent excessive correlations. We used a fixed-size circular buffer, filled with samples from a single growing exploratory trajectory. In our experiments, the same buffer size was used for all time discretizations. Thus the physical-time freshness of samples in the buffer varies with the time discretization, and in the strictest sense using a fixed-size buffer breaks timestep invariance. A memory-intensive option would be to use a buffer of size $\frac{1}{\delta t}$ (fixed memory in physical time).

Near-continuous reinforcement learning and RMSProp. RMSProp (Tieleman & Hinton, 2012) divides gradient steps by the square root of a moving average estimate of the second moment of gradients. This may interact with the learning rate scaling discussed above. In deterministic environments, gradients typically scale as $O(1)$ in terms of δt, as seen in (34). In that case, RMSProp preconditioning has no effect on the suitable order of magnitude for learning rates. However, in near continuous *stochastic* environments, variance of $\delta Q_{\delta t}/\delta t$ and of the gradients typically scales as $O\left(1/\delta t\right)$. With a fixed batch size, RMSProp will multiply gradients by a factor $O(\sqrt{\delta t})$. In that case, learning rates need only be scaled as $\sqrt{\delta t}$ instead of δt.

More generally, the continuous-time analysis should in principle be repeated for every component of a system. For instance, if a recurrent model is used to handle state memory or partial observability, care should be taken that the model is able to maintain memory for a non-infinitesimal physical time when $\delta t \to 0$ (see e.g. Tallec & Ollivier 2018).

7. Conclusion

Q-learning methods have been found to fail to learn with small time steps, both theoretically and empirically. A theoretical analysis help in building a practical off-policy deep RL algorithm with better robustness to time discretization. This robustness is confirmed empirically.

Acknowledgements

We would like to thank Harsh Satija, Thomas Lucas and Joelle Pineau for their useful remarks and comments.

References

Baird, L. C. Reinforcement learning in continuous time: Advantage updating. In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, volume 4, pp. 2448–2453. IEEE, 1994.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Doya, K. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017. URL http://arxiv.org/abs/1709.06560.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL http://arxiv.org/abs/1509.02971.

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. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Munos, R. and Bourgine, P. Reinforcement learning for continuous stochastic control problems. In *Advances in neural information processing systems*, pp. 1029–1035, 1998.

OpenAI. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018a. URL http://arxiv.org/abs/1808.00177.

OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018b.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Tallec, C. and Ollivier, Y. Can recurrent neural networks warp time? *arXiv preprint arXiv:1804.11188*, 2018.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012.

Uhlenbeck, G. E. and Ornstein, L. S. On the theory of the Brownian motion. *Physical review*, 36(5):823, 1930.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Zhang, A., Wu, Y., and Pineau, J. Natural environment benchmarks for reinforcement learning. *CoRR*, abs/1811.06032, 2018. URL http://arxiv.org/abs/1811.06032.